Основы Веб-программирования

Свинцов Дмитрий

05-12-2020

Оглавление

Глава 1

Содержание

1.1 Описание курса

1.1.1 Обзор курса

Курс объемом 140 учебных часов рассчитан на 6-ой семестр. Состоит из 70 часов лекционных занятий, 70 часов практической работы. В качестве самостоятельной работы предусмотрены домашние задания и курсовая работа. По окончанию обучения студенты сдают экзамен. Допуском к экзамену является выполнение всех домашних работ и сдача курсовой работы.

Инструменты

Операционная система

Операционная система в данном курсе не имеет значения, подойдет любая распространенная ОС с графическим интерфейсом. Например *Linux*, *MacOS* или *Windows*. Но в примерах будет использоваться ОС *Linux*.

Текстовый редактор

См.также:

Текстовые редакторы

За работой в текстовом редакторе Веб-программист проводит 90% времени, поэтому нужно ответственно подойти к этому выбору. Можно использовать любой понятный вам и удобный в использовании текстовый редактор.

Критериями должны стать:

- простота использования
- удобный интерфейс
- возможность гибкой настройки
- кроссплатформенность
- подсветка синтаксиса
- автодополнение кода

Всем этим критериям удовлетворяют такие редакторы как *Vim* и *Emacs*. Также программисты используют менее функциональные *Bred3*, *Notepad++*, *SublimeText* и другие. Если нет времени на изучение редактора, отличным выбором будет Visual Studio Code, в котором из коробки можно отлаживать Python, управлять git и писать код с автодополнением и проверкой синтаксиса.

Веб-браузер

Можно выбрать один из самых популярных браузеров (на сегодняшний день это *Mozilla Firefox* или *Google Chrome*) или любой другой, соответствующий Веб-стандартам.

Система контроля версий

Примечание: Git - самая популярная система контроля версий, по сути это уже стандарт в отрасли.

В данном курсе для выполнения самостоятельных работ потребуются знания системы контроля версий *git* и учетная запись в сервисе *GitHub*.

Системы контроля версий:

- git
- mercurial (hg)
- subversion (svn)

Социальные сети для разработчиков:

- GitHub - использует *git*, исходный код закрыт
- GitLab - opensource аналог github
- BitBucket - использует *git*, *mercurial*, исходный код закрыт
- SourceForge - использует *subversion*, один из первых подобных сервисов
- RhodeCode - opensource проект, позволяет использовать в проектах любую систему контроля версий, на выбор (*git*, *hg*, *svn*).

Git

- http://progit.org/book/ru/ - основная документация по Git. Нас будут интересовать первые три главы: введение, основы Git, ветвления в Git (а также слияние веток). Данный учебник является репозитарием на github и хостится как статический сайт при помощи сервиса Pages.
- Git для начинающих
- http://githowto.com/ru

github.com

- Установка Git и активация открытых ключей шифрования
- Создание репозитария
- Как скопировать чужой репозитарий
- Внесение исправлений в чужие репозитарии
- Социальные функции в Github
- Pages - хостинг статического сайта. Сервис Pages позволяет хостить статический сайт на github. Причем сам сайт будет обычным репозитарием.

1.1.2 Студенты

3 курс

Студенты должны:

- **УМЕТЬ**
  - верстать сайты с помощью (X)HTML и CSS
  - программировать на языках высокого уровня (Си, OCaml, Python, JavaScript)
  - составлять SQL запросы

  - пользоваться системой контроля версий git

- **ЗНАТЬ**
  - ОС подобные Unix
  - HTML, CSS
  - языки программирования высокого уровня
  - РСУБД SQLite, PostgreSQL
  - основы алгоритмизации и программирования
  - основы каналов передачи данных
  - систему контроля версий git
- **ИМЕТЬ**
  - компьютер с подключением к сети Интернет
  - ОС Linux или виртуальный образ

1.2 Введение

1.2.1 История развития Интернет

Хронология событий по годам.[1]

1969 - сеанс связи ARPANET
1971 - отправка первого Email
1983 - ARPANET переходит на TCP/IP
1984 - запущена система DNS
1989 - появление WWW, HTTP, HTML
1993 - первый браузер NCSA Mosaic
1995 - Yahoo, Hotmail, Amazon.com

1.2.2 История развития Веб

[1] https://ru.wikipedia.org/wiki/\T2A\CYRI\T2A\cyrn\T2A\cyrt\T2A\cyre\T2A\cyrr\T2A\cyrn\T2A\cyre\T2A\cyrt

Примечание: Интернет — это глобальная компьютерная сеть, объединяющая сотни миллионов компьютеров в общее информационное пространство. Интернет представляет свою инфраструктуру для прикладных сервисов различного назначения, самым популярным из которых является Всемирная Паутина – World Wide Web (www).[2]

World Wide Web (www, web, рус.: веб, Всемирная Паутина) — распределенная информационная система, предоставляющая доступ к гипертекстовым документам по протоколу HTTP.

WWW — сетевая технология прикладного уровня стека TCP/IP, построенная на клиент-серверной архитектуре и использующая инфраструктуру Интернет для взаимодействия между сервером и клиентом (`www`).

Серверы www (веб-серверы) — это хранилища гипертекстовой (в общем случае) информации, управляемые специальным программным обеспечением.

Документы, представленные в виде гипертекста, называются веб-страницами. Несколько веб-страниц, объединенных общей тематикой, оформлением, связанных гипертекстовыми ссылками и обычно находящихся на одном и том же веб-сервере, называются веб-сайтом.

Для загрузки и просмотра информации с веб-сайтов используются специальные программы — браузеры, способные обрабатывать гипертектовую разметку и отображать содержимое веб-страниц.

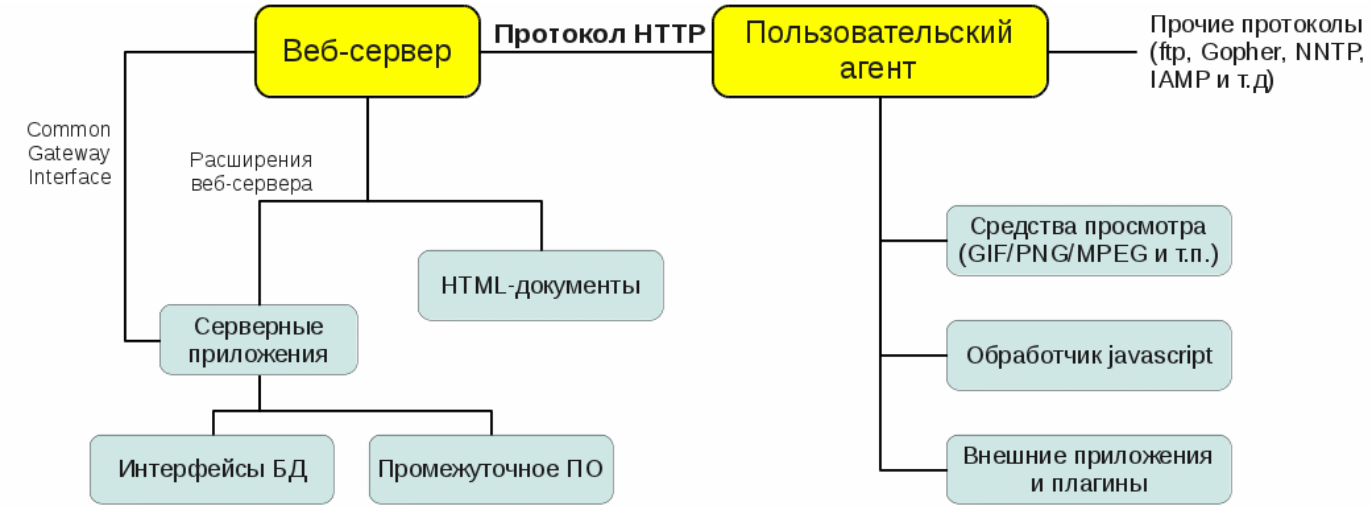


Рис. 1: Архитектура сервиса WWW

В основе www — взаимодействие между веб-сервером и браузерами по протоколу HTTP (HyperText Transfer Protocol). Веб-сервер — это программа, запущенная на сетевом компьютере и ожидающая клиентские запросы по протоколу HTTP. Браузер может

[2] http://www.4stud.info/web-programming/lecture1.html

обратиться к веб-серверу по доменному имени или по ip-адресу, передавая в запросе идентификатор требуемого ресурса. Получив запрос от клиента, сервер находит соответствующий ресурс на локальном устройстве хранения и отправляет его как ответ. Браузер принимает ответ и обрабатывает его соответствующим образом, в зависимости от типа ресурса (отображает гипертекст, показывает изображения, сохраняет полученные файлы и т.п.).

Основной тип ресурсов Всемирной паутины — гипертекстовые страницы. Гипертекст — это обычный текст, размеченный специальными управляющими конструкциями — тегами. Браузер считывает теги и интерпретирует их как команды форматирования при выводе информации. Теги описывают структуру документа, а специальные теги, якоря и гиперссылки, позволяют установить связи между веб-страницами и перемещаться как внутри веб-сайта, так и между сайтами.

Примечание: Т. Дж. Бернерс-Ли — «отец» Всемирной паутины

Сэр Тимоти Джон Бернерс-Ли — британский учёный-физик, изобретатель Всемирной паутины (совместно с Робертом Кайо), автор URI, HTTP и HTML. Действующий глава Консорциума Всемирной паутины (W3C). Автор концепции семантической паутины и множества других разработок в области информационных технологий. 16 июля 2004 года Королева Великобритании Елизавета II произвела Тима Бернерса-Ли в Рыцари-Командоры за «службу во благо глобального развития Интернета».

Компоненты WWW

Функционирование сервиса обеспечивается четырьмя составляющими:

- URL/URI — унифицированный способ адресации и идентификации сетевых ресурсов;
- HTML — язык гипертекстовой разметки веб-документов;
- HTTP — протокол передачи гипертекста;
- CGI — общий шлюзовый интерфейс, представляющий доступ к серверным приложениям.

Адресация веб-ресурсов. URL, URN, URI

Для доступа к любым сетевым ресурсам необходимо знать, где они размещены, и как к ним можно обратиться. Во Всемирной паутине для обращения к веб-документам изначально используется стандартизированная схема адресации и идентификации, учитывающая опыт адресации и идентификации таких сетевых сервисов, как e-mail, telnet, ftp и т.п. — URL, Uniform Resource Locator.

URL (RFC 1738) — унифицированный локатор (указатель) ресурсов, стандартизированный способ записи адреса ресурса в www и сети Интернет. Адрес URL имеет гибкую и расширяемую структуру для максимально естественного указания местонахождения ресурсов в сети. Для записи адреса используется ограниченный набор символов ASCII. Общий вид адреса можно представить так:

<схема>://<логин>:<пароль>@<хост>:<порт>/<полный-путь-к-ресурсу>

Где:

схема
: схема обращения к ресурсу: http, ftp, gopher, mailto, news, telnet, file, man, info, whatis, ldap, wais и т.п.

логин:пароль
: имя пользователя и его пароль, используемые для доступа к ресурсу

хост
: доменное имя хоста или его IP-адрес

порт
: порт хоста для подключения

полный-путь-к-ресурсу
: уточняющая информация о месте нахождения ресурса (зависит от протокола).

Примеры URL:

1. http://example.com # запрос стартовой страницы по умолчанию
2. http://www.example.com/site/map.html # запрос страницы в указанном каталоге
3. http://example.com:81/script.php # подключение на нестандартный порт
4. http://example.org/script.php?key=value # передача параметров скрипту
5. ftp://user:pass@ftp.example.org # авторизация на ftp-сервере
6. http://192.168.0.1/example/www # подключение по ip-адресу
7. file:///srv/www/htdocs/index.html # открытие локального файла
8. gopher://example.com/1 # подключение к серверу gopher
9. mailto://user@example.org # ссылка на адрес эл.почты

В августе 2002 года RFC 3305 анонсировал устаревание URL в пользу URI (Uniform Resource Identifier), еще более гибкого способа адресации, вобравшего возможности как URL, так и URN (Uniform Resource Name, унифицированное имя ресурса). URI позволяет не только указывать местонахождение ресурса (как URL), но и идентифицировать

его в заданном пространстве имен (как URN). Если в URI не указывать местонахождение, то с его помощью можно описывать ресурсы, которые не могут быть получены непосредственно из Интернета (автомобили, персоны и т.п.). Текущая структура и синтаксис URI регулируется стандартом RFC 3986, вышедшим в январе 2005 года.

Язык гипертекстовой разметки HTML

HTML (*HyperText Markup Language <https://ru.wikipedia.org/wiki/HTML>*) — стандартный язык разметки документов во Всемирной паутине. Большинство веб-страниц созданы при помощи языка HTML. Язык HTML интерпретируется браузером и отображается в виде документа в удобной для человека форме. HTML является приложением SGML (стандартного обобщённого языка разметки) и соответствует международному стандарту ISO 8879.

HTML создавался как язык для обмена научной и технической документацией, пригодный для использования людьми, не являющимися специалистами в области вёрстки. Для этого он представляет небольшой (сравнительно) набор структурных и семантических элементов — тегов. С помощью HTML можно легко создать относительно простой, но красиво оформленный документ. Изначально язык HTML был задуман и создан как средство структурирования и форматирования документов без их привязки к средствам воспроизведения (отображения). В идеале, текст с разметкой HTML должен единообразно воспроизводиться на различном оборудовании (монитор ПК, экран планшета, ограниченный по размерам экран мобильного телефона, медиа-проектор). Однако современное применение HTML очень далеко от его изначальной задачи. Со временем основная идея платформонезависимости языка HTML стала жертвой коммерциализации www и потребностей в мультимедийном и графическом оформлении.

Протокол HTTP

HTTP (HyperText Transfer Protocol) — протокол передачи гипертекста, текущая версия HTTP/1.1 (RFC 2616). Этот протокол изначально был предназначен для обмена гипертекстовыми документами, но сейчас его возможности существенно расширены в сторону передачи двоичной информации.

HTTP — типичный клиент-серверный протокол, обмен сообщениями идёт по схеме «запрос-ответ» в виде ASCII-команд. Особенностью протокола HTTP является возможность указать в запросе и ответе способ представления одного и того же ресурса по различным параметрам: формату, кодировке, языку и т. д. Именно благодаря возможности указания способа кодирования сообщения клиент и сервер могут обмениваться двоичными данными, хотя данный протокол является символьно-ориентированным.

HTTP — протокол прикладного уровня, но используется также в качестве «транспорта» для других прикладных протоколов, в первую очередь, основанных на языке XML (SOAP, XML-RPC, SiteMap, RSS и проч.).

Общий шлюзовый интерфейс CGI

CGI (Common Gateway Interface) — механизм доступа к программам на стороне веб-сервера. Спецификация CGI была разработана для расширения возможностей сервиса www за счет подключения различного внешнего программного обеспечения. При использовании CGI веб-сервер представляет браузеру доступ к исполнимым программам, запускаемым на его (серверной) стороне через стандартные потоки ввода и вывода.

Интерфейс CGI применяется для создания динамических веб-сайтов, например, когда веб-страницы формируются из результатов запроса к базе данных. Сейчас популярность CGI снизилась, т.к. появились более совершенные альтернативные решения (например, модульные расширения веб-серверов).

Программное обеспечение сервиса WWW

Веб-серверы

Веб-сервер — это сетевое приложение, обслуживающее HTTP-запросы от клиентов, обычно веб-браузеров. Веб-сервер принимает запросы и возвращает ответы, обычно вместе с HTML-страницей, изображением, файлом, медиа-потоком или другими данными. Веб-серверы — основа Всемирной паутины. С расширением спектра сетевых сервисов веб-серверы все чаще используются в качестве шлюзов для серверов приложений или сами представляют такие функции (например, Apache Tomcat).

Созданием программного обеспечения веб-серверов занимаются многие разработчики, но наибольшую популярность (по статистике http://netcraft.com) имеют такие программные продукты, как Apache (Apache Software Foundation), IIS (Microsoft), Google Web Server (GWS, Google Inc.) и nginx.

Apache — свободное программное обеспечение, распространяется под совместимой с GPL лицензией. Apache уже многие годы является лидером по распространенности во Всемирной паутине в силу своей надежности, гибкости, масштабируемости и безопасности.

IIS (Internet Information Services) — проприетарный набор серверов для нескольких служб Интернета, разработанный Майкрософт и распространяемый с серверными операционными системами семейства Windows. Основным компонентом IIS является веб-сервер, также поддерживаются протоколы FTP, POP3, SMTP, NNTP.

Google Web Server (GWS) — разработка компании Google на основе веб-сервера Apache. GWS оптимизирован для выполнения приложений сервиса Google Applications.

nginx [engine x] — это HTTP-сервер, совмещенный с кэширующим прокси-сервером. Разработан И. Сысоевым для компании Рамблер. Осенью 2004 года вышел первый публично доступный релиз, сейчас nginx используется на 9-12% веб-серверов.

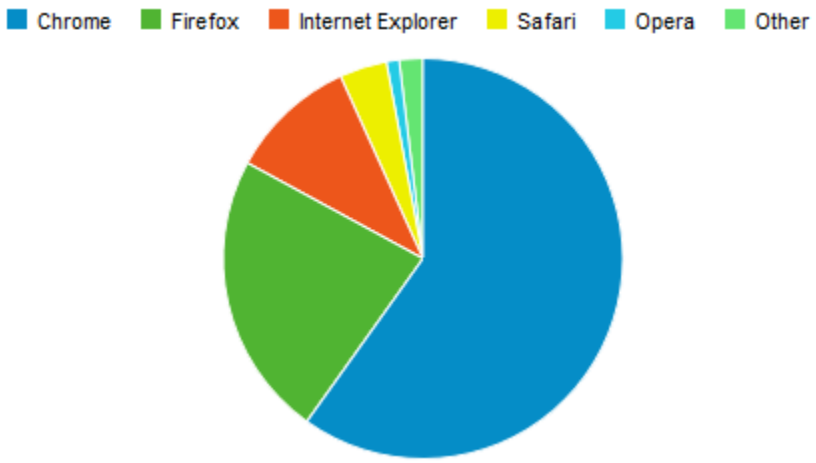
Браузеры

Браузер, веб-обозреватель (web-browser) — клиентское приложение для доступа к веб-серверам по протоколу HTTP и просмотра веб-страниц. Как правило браузеры дополнительно поддерживают и ряд других протоколов (например ftp, file, mms, pop3).

Первые HTTP-клиенты были консольными и работали в текстовом режиме, позволяя читать гипертекст и перемещаться по ссылкам. Сейчас консольные браузеры (такие, как lynx, w3m или links) практически не используются рядовыми посетителями веб-сайтов. Тем не менее такие браузеры весьма полезны для веб-разработчиков, так как позволяют «увидеть» веб-страницу «глазами» поискового робота.

Исторически первым браузером в современном понимании (т.е. с графическим интерфейсом и т.д.) была программа NCSA Mosaic, разработанная Марком Андерисеном и Эриком Бина. Mosaic имел довольно ограниченные возможности, но его открытый исходный код стал основой для многих последующих разработок.

Существует большое число программ-браузеров, но наибольшей популярностью пользуются следующие[43]:



Internet Explorer (IE) — браузер, разработанный компанией Майкрософт и тесно интегрированный с ОС Windows. Платформозависим (поддержка сторонних ОС прекращена начиная с версии 5). Единственный браузер, напрямую поддерживающий технологию ActiveX. Не полностью совместим со стандартами W3C, в связи с чем требует дополнительных затрат от веб-разработчиков.

Firefox — свободный кроссплатформенный браузер, разрабатываемый Mozilla Foundation и распространяемый под тройной лицензией GPL/LGPL/MPL. В основе браузера — движок Gekko, который изначально создавался для Netscape Communicator. Однако, вместо того, чтобы предоставить все возможности движка в стандартной поставке, Firefox реализует лишь основную его функциональность, предоставляя пользователям возможность модифицировать браузер в соответствии с их требованиями через поддержку расширений (add-ons), тем оформления и плагинов.

[4] http://www.w3schools.com/browsers/default.asp
[3] http://evolutionofweb.appspot.com/

Safari — проприетарный браузер, разработанный корпорацией Apple и входящий в состав операционной системы Mac OS X. Бесплатно распространяется для операционных систем семейства Microsoft Windows. В браузере используется уникальный по производительности интерпретатор JavaScript и еще ряд интересных для пользователя решений, которые отсутствуют или не развиты в других браузерах.

Chrome — кроссплатформенный браузер с открытым исходным кодом, разрабатываемый компанией Google. Первая стабильная версия вышла 11 декабря 2008 года. В отличие от многих других браузеров, в Chrome каждая вкладка является отдельным процессом. В случае если процесс обработки содержимого вкладки зависнет, его можно будет завершить без риска потери данных других вкладок. Еще одна особенность — интеллектуальная адресная строка (Omnibox). К возможности автозаполнения она добавляет поисковые функции с учетом популярности сайта, релевантности и пользовательских предпочтений (истории переходов).

Opera — кроссплатформенный многофункциональный веб-браузер, впервые представленный в 1994 году группой исследователей из норвежской компании Telenor. Дальнейшая разработка ведется Opera Software ASA. Этот браузер обладает высокой скоростью работы и совместим с основными стандартами. Отличительными особенностями Opera долгое время являлись многостраничный интерфейс и возможность масштабирования веб-страниц целиком. На разных этапах развития в Opera были интегрированы возможности почтового/новостного клиента, адресной книги, клиента сети BitTorrent, агрегатора RSS, клиента IRC, менеджера закачек, WAP-браузера, а также поддержка виджетов — графических модулей, работающих вне окна браузера.

Роботы-«пауки»

Наряду с браузерами, ориентированными на пользователя, существуют и специализированные клиенты-роботы («пауки», «боты»), подключающиеся к веб-серверам и выполняющие различные задачи автоматической обработки гипертекстовой информации. Сюда относятся, в первую очередь, роботы поисковых систем, таких как google.com, yandex.ru, yahoo.com и т.п., выполняющие обход веб-сайтов для последующего построения поискового индекса.

Эволюция Веб сайтов

Web 1.0 - до .com bubble. Статичное содержание страниц, аскетичный дизайн, чаты, форумы, гостевые книги.

Web 2.0 - новое поколение сайтов (после 2001) User-generated content. Предоставление и потребление API. RSS. Обновление страниц «на лету» (ajax).

Web 3.0 - ??? Community-generated content. Семантическая паутина. Уникальные идентификаторы и микроформаты.

1.3 Веб сервер

См.также:

Презентация с лекций

1.3.1 Что такое Веб-сервер

См.также:

- https://developer.mozilla.org/en-US/Learn/What_is_a_web_server
- https://ru.wikipedia.org/wiki/\T2A\CYRV\T2A\cyre\T2A\cyrb-\T2A\cyrs\T2A\cyre\T2A\cyrr\T2A\cyrv\T2A\cyre\T2A\cyrr
- https://docs.python.org/3.5/howto/webservers.html
- https://gist.github.com/willurd/5720255

Описание

Понятие *Веб-сервер* может относиться как к железу, так и к программному обеспечению (ПО).

1. С точки зрения железа *Веб-сервер* — это компьютер, который хранит ресурсы сайта (HTML документы, CSS стили, JavaScript файлы и другое) и доставляет их на устройство конечного пользователя (веб-браузер и т.д.). Обычно он подключен к сети Интернет и может быть доступен через доменное имя, например, mozilla.org.
2. С точки зрения ПО, *Веб-сервер* включает в себя некоторые вещи, которые контролируют доступ Веб-пользователей к размещенным на сервере файлам, это минимум *HTTP сервера*. *HTTP сервер* это часть ПО, которая понимает URL'ы (веб-адреса) и HTTP (протокол который использует ваш браузер для просмотра веб-станиц).

Простыми словами, когда браузеру нужен файл, размещенный на веб-сервере, браузер запрашивает его через HTTP. Когда запрос достигает нужного веб-сервера (железо), сервер HTTP (ПО) передает запрашиваемый документ обратно, также через HTTP.

См.также:

- https://ru.wikipedia.org/wiki/\T2A\CYRS\T2A\cyrt\T2A\cyra\T2A\cyrt\T2A\cyri\T2A\cyrch\T2A\cyre\T2A\cyrs\T2A\cyrk\T2A\cyri\T2A\cyrishrt_\T2A\cyrs\T2A\cyra\T2A\cyrishrt\T2A\cyrt

- https://ru.wikipedia.org/wiki/\T2A\CYRD\T2A\cyri\T2A\cyrn\T2A\cyra\T2A\cyrm\T2A\cyri\T2A\cyrch\T2A\cyre\T2A\cyrs\T2A\cyrk\T2A\cyri\T2A\cyrishrt_\T2A\cyrs\T2A\cyra\T2A\cyrishrt\T2A\cyrt
- https://ru.wikipedia.org/wiki/\T2A\CYRS\T2A\cyre\T2A\cyrr\T2A\cyrv\T2A\cyre\T2A\cyrr_\T2A\cyrp\T2A\cyrr\T2A\cyri\T2A\cyrl\T2A\cyro\T2A\cyrzh\T2A\cyre\T2A\cyrn\T2A\cyri\T2A\cyrishrt

Чтобы опубликовать веб-сайт, нужен либо статический, либо динамический веб-сервер.

Статический веб-сервер или стек состоит из компьютера (железо) с сервером HTTP (ПО). Мы называем это «статикой», потому что сервер посылает размещенные на нем файлы в браузер не изменяя их.

Динамических веб-сервер состоит из **статического веб-сервера** плюс дополнительного программного обеспечения, наиболее часто **сервером приложений** и **базы данных**. Мы называем его «динамический», потому что **сервер приложений** изменяет исходные файлы перед отправкой в ваш браузер по HTTP.

Примечание: Сервера приложений для Python

- CherryPy
- Gunicorn
- uWSGI
- Waitress
- Tornado
- Zope
- Werkzeug

Например, для получения итоговой страницы, которую вы видите в браузере, сервер приложений может заполнить HTML шаблон данными из базы данных. Такие сайты, как MDN (Mozilla Developer Network) или Википедия состоят из тысяч веб-страниц, но они не являются реальными HTML документами, лишь несколько HTML шаблонов и гигантские базы данных. Эта структура упрощает и ускоряет сопровождение веб-приложений и доставку контента.

Более детально

Чтобы загрузить веб-страницу, как мы уже говорили, браузер отправляет запрос к веб-серверу, который приступает к поиску запрашиваемого файла в своем собственном пространстве памяти. Найдя файл, сервер считывает его, обрабатывает так, как ему это необходимо, и направляет его в браузер. Давайте рассмотрим эти шаги более подробно.

Хостинг файлов

Во-первых, веб-сервер хранит файлы веб-сайта, а именно все HTML документы и связанные с ними ресурсы, включая изображения, CSS стили, JavaScript файлы, шрифты и видео.

См.также:

- https://ru.wikipedia.org/wiki/\T2A\CYRV\T2A\cyrery\T2A\cyrd\T2A\cyre\T2A\cyrl\T2A\cyre\T2A\cyrn\T2A\cyrn\T2A\cyrery\T2A\cyrishrt_\T2A\cyrs\T2A\cyre\T2A\cyrr\T2A\cyrv\T2A\cyre\T2A\cyrr
- Статический IP адрес

Технически, вы можете разместить все эти файлы на своем компьютере, но гораздо удобнее хранить их на выделенном веб-сервере, который:

- всегда запущен и работает
- постоянно в сети Интернет
- имеет один и тот же IP адрес все время (не все провайдеры предоставляют статический IP адрес для домашнего подключения)
- обслуживается на стороне

Таким образом, выбор хорошего хостинг-провайдера является важной частью создания сайта. Рассмотрите различные предложения компаний и выберите то, что соответствует вашим потребностям и бюджету (предложения варьируются от бесплатных до тысяч долларов в месяц).

Связь по HTTP

Во-вторых, веб-сервер обеспечивает поддержку HTTP (hypertext transfer protocol). Как следует из названия, HTTP указывает, как передавать гипертекст (т.е. связанные веб-документы) между двумя компьютерами.

Протокол представляет собой набор правил для связи между двумя компьютерами. HTTP является текстовым протоколом без сохранения состояния.

Текстовый

Все команды это человеко-читаемый текст.

Не сохраняет состояние

Ни клиент, ни сервер, не помнят о предыдущих соединениях. Например, опираясь только на HTTP, сервер не сможет вспомнить введенный вами пароль, или на каком шаге транзакции вы находитесь. Для таких задач вам потребуется сервер приложений.

HTTP задает строгие правила, как клиент и сервер должны общаться. Более подробно смотри http-protocol. Вот некоторые из них:

Примечание:

- https://ru.wikipedia.org/wiki/File_(\T2A\cyrs\T2A\cyrh\T2A\cyre\T2A\cyrm\T2A\cyra_URI)

- Только клиенты могут отправлять HTTP запросы, и только на сервера. Сервера отвечают только на HTTP запросы клиента.
- Когда запрашивается физический файл, клиент должен сформировать file URL (*file:///var/log/syslog*)
- Веб-сервер должен ответить на каждый HTTP запрос, по крайней мере с сообщением об ошибке.

На веб-сервере, HTTP сервер отвечает за обработку входящих запросов и ответ на них.

1. При получении запроса, HTTP сервер сначала проверяет существует ли ресурс по данному URL.
2. Если это так, веб-сервер отправляет содержимое файла обратно в браузер. Если нет, сервер приложений создает необходимый ресурс.
3. Если это не возможно, веб-сервер возвращает сообщение об ошибке в браузер, чаще всего «404 Not Found». (Эта ошибка настолько распространена, что многие веб-дизайнеры тратят большое количество времени на разработку 404 страниц об ошибках.)

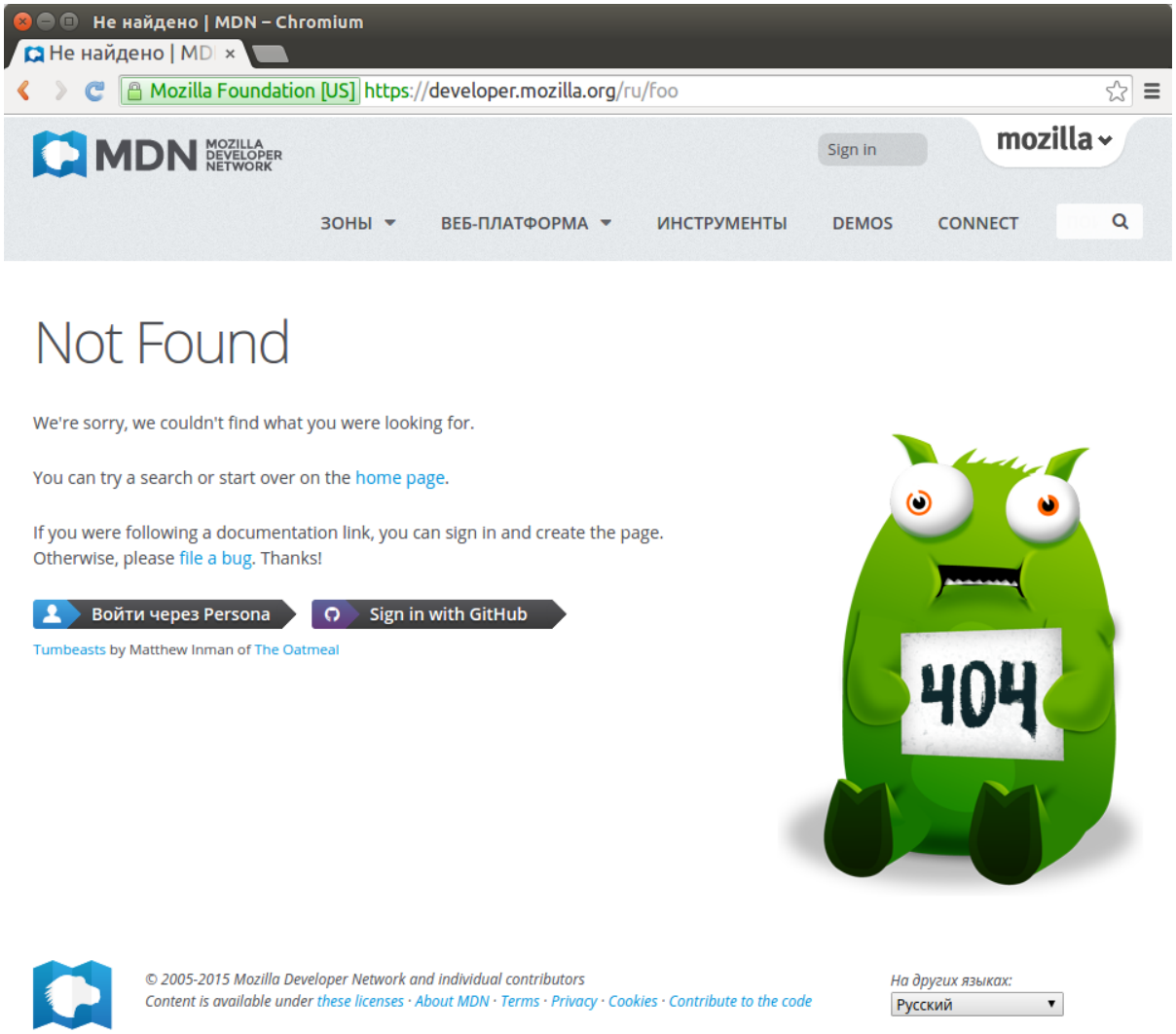
Статика vs Динамика

Грубо говоря, сервер может отдавать статическое или динамическое содержимое.

«Статическое» означает «отдается как есть». Статические веб-сайты проще всего установить, поэтому мы предлагаем вам сделать свой первый сайт статическим.

«Динамическое» означает, что сервер обрабатывает данные или даже генерирует их на лету из базы данных. Это обеспечивает больше гибкости, но технически сложнее в обслуживании, что делает его более сложным для создания веб-сайта.

Возьмем к примеру страницу What is web server, перевод которой вы читаете. На веб-сервере, где это хостится, есть сервер приложений, который извлекает содержимое статьи из базы данных, форматирует его, добавляет в HTML шаблоны и отправляет вам результат. В нашем случае, сервер приложений называется *Kuma*, написан он на языке программирования *Python* (используя фреймворк *Django*). Команда Mozilla создали *Kuma* для конкретных нужд MDN, но есть много подобных приложений, построенных на многих других технологий.



Существует много серверов приложений для разных запросов, поэтому довольно трудно выбрать какой-то один универсальный. Некоторые серверы приложений удовлетворяют определенной категории веб-сайтов, такие как блоги, вики или интернет-магазины; другие, называемые CMS (системы управления контентом), являются более общими. Если вы создаете динамический сайт, потратьте немного времени на выбор инструмента, который соответствует вашим потребностям. Если вы не хотите изучать веб-программирование (хотя это захватывающая область сама по себе!), то вам не нужно создавать свой собственный сервер приложений. Это будет очередной велосипед.

1.3.2 CGI

См.также:

- https://ru.wikipedia.org/wiki/CGI
- http://www.ietf.org/rfc/rfc3875.txt
- Лекции ОмГТУ, кафедра АСОИУ
- http://webpython.codepoint.net/cgi_tutorial

CGI (от англ. Common Gateway Interface — «общий интерфейс шлюза») — стандарт интерфейса, используемого для связи внешней программы с веб-сервером. Программу, которая работает по такому интерфейсу совместно с веб-сервером, принято называть шлюзом, хотя многие предпочитают названия «скрипт» (сценарий) или «CGI-программа».

Поскольку гипертекст статичен по своей природе, веб-страница не может непосредственно взаимодействовать с пользователем. До появления *JavaScript*, не было иной возможности отреагировать на действия пользователя, кроме как передать введенные им данные на веб-сервер для дальнейшей обработки. В случае CGI эта обработка осуществляется с помощью внешних программ и скриптов, обращение к которым выполняется через стандартизованный (см. RFC 3875: CGI Version 1.1) интерфейс — общий шлюз.

Упрощенная модель, иллюстрирующая работу CGI:

Сам интерфейс разработан таким образом, чтобы можно было использовать любой язык программирования, который может работать со стандартными устройствами ввода-вывода. Такими возможностями обладают даже скрипты для встроенных командных интерпретаторов операционных систем, поэтому в простых случаях могут использоваться даже командные скрипты.

Как работает CGI?

Обобщенный алгоритм работы через CGI можно представить в следующем виде:

1. Клиент запрашивает CGI-приложение по его URI.
2. Веб-сервер принимает запрос и устанавливает переменные окружения, через них приложению передаются данные и служебная информация.
3. Веб-сервер перенаправляет запросы через стандартный поток ввода (stdin) на вход вызываемой программы.
4. CGI-приложение выполняет все необходимые операции и формирует результаты в виде HTML.

5. Сформированный гипертекст возвращается веб-серверу через стандартный поток вывода (stdout). Сообщения об ошибках передаются через stderr.
6. Веб-сервер передает результаты запроса клиенту.

Области применения CGI

Наиболее частая задача, для решения которой применяется CGI — создание интерактивных страниц, содержание которых зависит от действий пользователя. Типичными примерами таких веб-страниц является форма регистрации на сайте или форма для отправки комментария. Другая область применения CGI, остающаяся за кулисами взаимодействия с пользователем, связана со сбором и обработкой информации о клиенте: установка и чтение «печенюшек»-cookies; получение данных о браузере и операционной системе; подсчет количества посещений веб-страницы; мониторинг веб-трафика и т.п.

Это обеспечивается возможностью подключения CGI-скрипта к базе данных, а также возможностью обращаться к файловой системе сервера. Таким образом CGI-скрипт может сохранять информацию в таблицах БД или файлах и получать ее оттуда по запросу, чего нельзя сделать средствами HTML.

Предупреждение: CGI — это не язык программирования! Это простой протокол, позволяющий веб-серверу передавать данные через stdin и читать их из stdout. Поэтому в качестве CGI-обработчика может использоваться любая серверная программа, способная работать со стандартными потоками ввода-вывода.

Примеры

Пример на Python:

```
#!/usr/bin/python
print("""Content-Type: text/plain

Hello, world!""")
```

В этом коде строка `#!/usr/bin/python` указывает полный путь к интерпретатору Python.

Пример на Си:

```
#include <stdio.h>
int main(void) {
  printf("Content-Type: text/plain\n\n");
  printf("Hello, world!\n\n");
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  return 0;
}
```

Строка `Content-type: text/html\n\n` — http-заголовок, задающий тип содержимого (mime-type). Удвоенный символ разрыва строки (\n\n) — обязателен, он отделяет заголовки от тела сообщения.

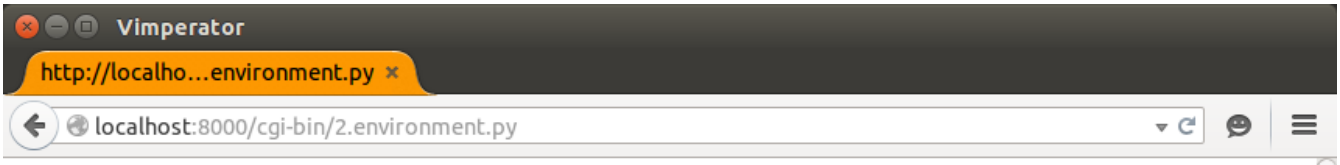
Все скрипты, как правило, помещают в каталог cgi (или cgi-bin) сервера, но это необязательно: скрипт может располагаться где угодно, но при этом большинство веб-серверов требуют специальной настройки. В веб-сервере Apache, например, такая настройка может производиться при помощи общего файла настроек httpd.conf или с помощью файла .htaccess в том каталоге, где содержится этот скрипт. Также скрипты должны иметь права на исполнение (`chmod +x hello.py`).

Переменные окружения

Все CGI-приложения имеют доступ к переменным окружения, устанавливаемым веб-сервером. Эти переменные играют важную роль при написании CGI-программ. В таблице перечислены некоторые из переменных, доступных CGI.

Пример вывода переменных окружения CGI-скрипта:

```
1 #!/usr/bin/python
2 import os
3
4 print("Content-type: text/html\r\n\r\n")
5 print("<font size=+10>Environment</font><br>")
6
7 for param in os.environ.keys():
8     print("<b>%20s</b>: %s<br>" % (param, os.environ[param]))
```



Environment

HTTP_REFERER:
WORKON_HOME: /home/vagrant/.virtualenvs
SERVER_SOFTWARE: SimpleHTTP/0.6 Python/2.7.8
SCRIPT_NAME: /cgi-bin/2.environment.py
LESSOPEN: | /usr/bin/lesspipe %s
SSH_CLIENT: 172.17.42.1 43481 22
REQUEST_METHOD: GET
LOGNAME: vagrant
USER: vagrant
SERVER_PROTOCOL: HTTP/1.0
QUERY_STRING:
PATH: /home/vagrant/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:
/usr/local/games
VIRTUALENVWRAPPER_SCRIPT: /usr/local/bin/virtualenvwrapper.sh
LANG: ru_RU.UTF-8
TERM: screen-256color
SHELL: /bin/bash
HTTP_COOKIE:
SERVER_NAME: 9d5157200f01
REMOTE_ADDR: 172.17.42.1
SHLVL: 1
EDITOR: vim
SERVER_PORT: 8000
HOME: /home/vagrant
PATH_TRANSLATED: /home/vagrant/sourcecode
CONTENT_LENGTH:
HTTP_USER_AGENT: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:36.0) Gecko/20100101 Firefox/36.0
_: /usr/bin/python
HTTP_ACCEPT: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
VIRTUALENVWRAPPER_PROJECT_FILENAME: .project
SSH_CONNECTION: 172.17.42.1 43481 172.17.0.10 22
LESSCLOSE: /usr/bin/lesspipe %s %s
VIRTUALENVWRAPPER_HOOK_DIR: /home/vagrant/.virtualenvs
GATEWAY_INTERFACE: CGI/1.1
SSH_TTY: /dev/pts/0
OLDPWD: /home/vagrant
PWD: /home/vagrant/sourcecode
CONTENT_TYPE: text/plain
MAIL: /var/mail/vagrant
LS_COLORS: rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;
33:01:or=40:31:01:su=37:41:sg=30:43:ca=30:41:tw=30:42:ow=34:42:st=37:44:ex=01:32:*.tar=01:

Преимущества CGI

- Процесс CGI скрипта не зависит от Веб-сервера и, в случае падения, никак не отразится на работе последнего
- Может быть написан на любом языке программирования
- Поддерживается большинством Веб-серверов

Недостатки CGI

Самым большим недостатком этой технологии являются повышенные требования к производительности веб-сервера. Дело в том, что каждое обращение к CGI-приложению вызывает порождение нового процесса, со всеми вытекающими отсюда накладными расходами. Если же приложение написано с ошибками, то возможна ситуация, когда оно, например, зациклится. Браузер прервет соединение по истечении тайм-аута, но на серверной стороне процесс будет продолжаться, пока администратор не снимет его принудительно.

Альтернативы

- FastCGI — дальнейшее развитие технологии CGI. Поддерживается многими Веб-серверами, например Nginx.
- Веб-сервера, в которые уже встроена поддержка дополнительных стандартов и протоколов, таких как WSGI (Gunicorn, waitress, uwsgi)
- Веб-сервер, функционал которого расширяется через модули, например, Apache (mod_wsgi, mod_php, mod_fastcgi)

Практика

См.также:

- https://docs.python.org/2/library/cgihttpserver.html
- https://docs.python.org/3/library/http.server.html

Для запуска CGI сервера необходимо перейти в директорию `sourcecode` и выполнить команду:

```
python -m CGIHTTPServer 8000
```

или

```
python3 -m http.server --cgi 8000
```

или cgiserver.py

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-
# vim:fenc=utf-8
#
# Copyright © 2014 uralbash <root@uralbash.ru>
#
# Distributed under terms of the MIT license.

"""
Demo CGI Server
"""
try:
    import BaseHTTPServer
    import CGIHTTPServer
except ImportError:
    import http.server as BaseHTTPServer
    import http.server as CGIHTTPServer
import cgitb

cgitb.enable()  # This line enables CGI error reporting

server = BaseHTTPServer.HTTPServer
handler = CGIHTTPServer.CGIHTTPRequestHandler
server_address = ("", 8000)
handler.cgi_directories = ["/cgi-bin", "/wsgi"]

httpd = server(server_address, handler)
httpd.serve_forever()
```

```
python cgiserver.py
```

Теперь CGI-скрипты доступны на 8000 порту, например по адресу http://localhost:8000/cgi-bin/1.hello.py

- Пример CGI скриптов на Python
- Пример CGI скриптов на C++

Примечание: Для компиляции кода на C++ необходимо установить библиотеку cgicc:

```
sudo apt-get install libcgicc5-dev
```

Пример компиляции:

```
g++ -o 3.get.post.cgi 3.get.post.cpp -lcgicc
```

Hello World!

Примечание:

- http://localhost:8000/cgi-bin/1.hello.cgi
- http://localhost:8000/cgi-bin/1.hello.go.cgi
- http://localhost:8000/cgi-bin/1.hello.py
- http://localhost:8000/cgi-bin/1.hello.rb

Python

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-
# vim:fenc=utf-8
#
# Copyright © 2014 uralbash <root@uralbash.ru>
#
# Distributed under terms of the MIT license.
#
# 1.hello.py
# http://www.tutorialspoint.com/python/python_cgi_programming.htm

print("Content-type:text/html\r\n\r\n")
print('<html>')
print('<head>')
print('<title>Hello Word - First CGI Program</title>')
print('</head>')
print('<body>')
print('<h2>Hello Word! This is my first CGI program</h2>')
print('</body>')
print('</html>')
```

Ruby

```
#!/usr/bin/env ruby
#
# 1.hello.rb
# Copyright (C) 2015 uralbash <root@uralbash.ru>
```

(continues on next page)

(продолжение с предыдущей страницы)

```
#
# Distributed under terms of the MIT license.

puts "Content-type:text/html\r\n\r\n"
puts '<html>'
puts '<head>'
puts '<title>Hello Word - First CGI Program</title>'
puts '</head>'
puts '<body>'
puts '<h2>Hello Word! This is my first CGI program</h2>'
puts '</body>'
puts '</html>'
```

C++

Для компиляции: `make 1_hello`

```
/*
 * 1.hello.cpp
 * Copyright (C) 2015 uralbash <root@uralbash.ru>
 *
 * Distributed under terms of the MIT license.
 */

#include <iostream>

using namespace std;

int main()
{
    cout << "Content-type:text/html\r\n\r\n";
    cout << "<html>\n";
    cout << "<head>\n";
    cout << "<title>Hello World - First CGI Program</title>\n";
    cout << "</head>\n";
    cout << "<body>\n";
    cout << "<h2>Hello World! This is my first CGI program</h2>\n";
    cout << "</body>\n";
    cout << "</html>\n";

    return 0;
}
```

Go

Для компиляции: `make 1_hello_go`

```
package main

import "fmt"

func main() {
        html :=
                `Content-type:text/html

<html>
<head>
        <title>Hello Word - First CGI Program</title>
</head>
<body>
        <h2>Hello Word! This is my first CGI program</h2>
</body>
</html>`
        fmt.Println(html)
}
```

Вывод переменных окружения

Примечание:

- http://localhost:8000/cgi-bin/2.environment.cgi
- http://localhost:8000/cgi-bin/2.environment.py
- http://localhost:8000/cgi-bin/2.environment.rb

Python

```
#!/usr/bin/python
import os

print("Content-type: text/html\r\n\r\n")
print("<font size=+10>Environment</font><br>")

for param in os.environ.keys():
    print("<b>%20s</b>: %s<br>" % (param, os.environ[param]))
```

Ruby

```
#!/usr/bin/env ruby
#
```

(continues on next page)

(продолжение с предыдущей страницы)

```ruby
# 2.environment.rb
# Copyright (C) 2015 uralbash <root@uralbash.ru>
#
# Distributed under terms of the MIT license.
#

print "Content-type: text/html\r\n\r\n"
print "<font size=+10>Environment</font><br>"

for param in ENV
    print "<b>%20s</b>: %s<br>" % param
end
```

C++

Для компиляции: `make 2_environment`

```cpp
/*
 * 2.environment.cpp
 * Copyright (C) 2015 uralbash <root@uralbash.ru>
 *
 * Distributed under terms of the MIT license.
 */

#include <iostream>

using namespace std;

int main(int argc, char **argv, char** env)
{
    cout << "Content-type:text/html\r\n\r\n";
    cout << "<html>\n";
    cout << "<head>\n";
    cout << "<title>CGI Envrionment Variables</title>\n";
    cout << "</head>\n";
    cout << "<body>\n";

    while (*env)
        cout << *env++ << "<br/>";

    cout << "</body>\n";
    cout << "</html>\n";

    return 0;
}
```

GET и POST запросы

Примечание:

- http://localhost:8000/cgi-bin/3.get.post.cgi?first_name=Lev&last_name=Tolstoy
- http://localhost:8000/cgi-bin/3.get.post.py?first_name=Lev&last_name=Tolstoy
- http://localhost:8000/cgi-bin/3.get.post.rb?first_name=Lev&last_name=Tolstoy

---

- **GET** (action=»http://localhost:8000/cgi-bin/3.get.post.cgi» method=»get»)

- **POST** (action=»http://localhost:8000/cgi-bin/3.get.post.cgi» method=»post»)

Python

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-
# vim:fenc=utf-8
#
# Copyright © 2014 uralbash <root@uralbash.ru>
#
# Distributed under terms of the MIT license.
#
# CGI get method
# http://www.tutorialspoint.com/python/python_cgi_programming.htm

import cgi

# Create instance of FieldStorage
form = cgi.FieldStorage()

# Get data from fields
first_name = form.getvalue('first_name')
last_name = form.getvalue('last_name')

print("Content-type:text/html\r\n\r\n")
print("<html>")
print("<head>")
print("<title>Hello - Second CGI Program</title>")
print("</head>")
print("<body>")
print("<h2>Hello %s %s</h2>" % (first_name, last_name))
```

(continues on next page)

(продолжение с предыдущей страницы)

```
print("</body>")
print("</html>")
```

Ruby

```ruby
#!/usr/bin/env ruby
#
# 3.get.rb
# Copyright (C) 2015 uralbash <root@uralbash.ru>
#
# Distributed under terms of the MIT license.
#

require 'cgi'

cgi = CGI.new

# Get data from fields
first_name = cgi['first_name']
last_name = cgi['last_name']

print "Content-type:text/html\r\n\r\n"
print "<html>"
print "<head>"
print "<title>Hello - Second CGI Program</title>"
print "</head>"
print "<body>"
print "<h2>Hello %s %s</h2>" % [first_name, last_name]
print "</body>"
print "</html>"
```

C++

Для компиляции: `make 3_get_post`

```cpp
/*
 * 3.get.cpp
 * Copyright (C) 2015 uralbash <root@uralbash.ru>
 *
 * Distributed under terms of the MIT license.
 */

#include <iostream>
#include <cgicc/Cgicc.h>

using namespace std;
```

(continues on next page)

(продолжение с предыдущей страницы)

```
12 using namespace cgicc;
13
14 int main()
15 {
16     Cgicc formData;
17
18     cout << "Content-type:text/html\r\n\r\n";
19     cout << "<html>\n";
20     cout << "<head>\n";
21     cout << "<title>Using GET and POST Methods</title>\n";
22     cout << "</head>\n";
23     cout << "<body>\n";
24
25     form_iterator fi = formData.getElement("first_name");
26     if(fi != (*formData).end()) {
27         cout << "First name: " << **fi << endl;
28     }else{
29         cout << "No text entered for first name" << endl;
30     }
31     cout << "<br/>\n";
32     fi = formData.getElement("last_name");
33     if(fi != (*formData).end()) {
34         cout << "Last name: " << **fi << endl;
35     }else{
36         cout << "No text entered for last name" << endl;
37     }
38     cout << "<br/>\n";
39
40     cout << "</body>\n";
41     cout << "</html>\n";
42
43     return 0;
44 }
```

Checkbox

Python

```
1 #! /usr/bin/env python
2 # -*- coding: utf-8 -*-
3 # vim:fenc=utf-8
4 #
5 # Copyright © 2014 uralbash <root@uralbash.ru>
6 #
```

(continues on next page)

(продолжение с предыдущей страницы)

```
# Distributed under terms of the MIT license.
#
# CGI checkbox
# http://www.tutorialspoint.com/python/python_cgi_programming.htm

import cgi

# Create instance of FieldStorage
form = cgi.FieldStorage()

# Get data from fields
maths = form.getvalue('maths')
physics = form.getvalue('physics')

print("Content-type:text/html\r\n\r\n")
print("<html>")
print("<head>")
print("<title>Checkbox - Third CGI Program</title>")
print("</head>")
print("<body>")

if maths:
    print("Maths Flag: ON")
else:
    print("Maths Flag: OFF")

print("<br>")

if physics:
    print("Physics Flag: ON")
else:
    print("Physics Flag: OFF")

print("</body>")
print("</html>")
```

C++

Для компиляции: `make 4_checkbox`

```
/*
 * 4.checkbox.cpp
 * Copyright (C) 2015 uralbash <root@uralbash.ru>
 *
 * Distributed under terms of the MIT license.
 */
```

(continues on next page)

(продолжение с предыдущей страницы)

```
7  #include <iostream>
8  #include <cgicc/Cgicc.h>
9
10 using namespace std;
11 using namespace cgicc;
12
13 int main()
14 {
15     Cgicc formData;
16     bool maths_flag, physics_flag;
17
18     cout << "Content-type:text/html\r\n\r\n";
19     cout << "<html>\n";
20     cout << "<head>\n";
21     cout << "<title>Checkbox Data to CGI</title>\n";
22     cout << "</head>\n";
23     cout << "<body>\n";
24
25     maths_flag = formData.queryCheckbox("maths");
26     if( maths_flag ) {
27         cout << "Maths Flag: ON " << endl;
28     }else{
29         cout << "Maths Flag: OFF " << endl;
30     }
31     cout << "<br/>\n";
32
33     physics_flag = formData.queryCheckbox("physics");
34     if( physics_flag ) {
35         cout << "Physics Flag: ON " << endl;
36     }else{
37         cout << "Physics Flag: OFF " << endl;
38     }
39     cout << "<br/>\n";
40     cout << "</body>\n";
41     cout << "</html>\n";
42
43     return 0;
44 }
```

Radio

Python

```python
#! /usr/bin/env python
# -*- coding: utf-8 -*-
# vim:fenc=utf-8
#
# Copyright © 2014 uralbash <root@uralbash.ru>
#
# Distributed under terms of the MIT license.
#
# CGI radio
# http://www.tutorialspoint.com/python/python_cgi_programming.htm

import cgi

# Create instance of FieldStorage
form = cgi.FieldStorage()

# Get data from fields
if form.getvalue('subject'):
    subject = form.getvalue('subject')
else:
    subject = "Not set"

print("Content-type:text/html\r\n\r\n")
print("<html>")
print("<head>")
print("<title>Radio - Fourth CGI Program</title>")
print("</head>")
print("<body>")
print("<h2> Selected Subject is %s</h2>" % subject)
print("</body>")
print("</html>")
```

C++

Для компиляции: `make 5_radio`

```cpp
/*
 * 5.radio.cpp
 * Copyright (C) 2015 uralbash <root@uralbash.ru>
 *
 * Distributed under terms of the MIT license.
 */
#include <iostream>
#include <cgicc/Cgicc.h>

using namespace std;
```

(continues on next page)

(продолжение с предыдущей страницы)

```
using namespace cgicc;

int main()
{
    Cgicc formData;

    cout << "Content-type:text/html\r\n\r\n";
    cout << "<html>\n";
    cout << "<head>\n";
    cout << "<title>Radio Button Data to CGI</title>\n";
    cout << "</head>\n";
    cout << "<body>\n";

    form_iterator fi = formData.getElement("subject");
    if( !fi->isEmpty() && fi != (*formData).end()) {
        cout << "Radio box selected: " << **fi << endl;
    }

    cout << "<br/>\n";
    cout << "</body>\n";
    cout << "</html>\n";

    return 0;
}
```

TextArea

Python

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-
# vim:fenc=utf-8
#
# Copyright © 2015 uralbash <root@uralbash.ru>
#
# Distributed under terms of the MIT license.
import cgi

# Create instance of FieldStorage
form = cgi.FieldStorage()

# Get data from fields
if form.getvalue('textcontent'):
    text_content = form.getvalue('textcontent')
```

(continues on next page)

(продолжение с предыдущей страницы)

```
else:
    text_content = "Not entered"

print("Content-type:text/html\r\n\r\n")
print("<html>")
print("<head>")
print("<title>Text Area - Fifth CGI Program</title>")
print("</head>")
print("<body>")
print("<h2> Entered Text Content is %s</h2>" % text_content)
print("</body>")
```

C++

Для компиляции: `make 6_textarea`

```
/*
 * 6.textarea.cpp
 * Copyright (C) 2015 uralbash <root@uralbash.ru>
 *
 * Distributed under terms of the MIT license.
 */
#include <iostream>
#include <cgicc/Cgicc.h>

using namespace std;
using namespace cgicc;

int main()
{
    Cgicc formData;

    cout << "Content-type:text/html\r\n\r\n";
    cout << "<html>\n";
    cout << "<head>\n";
    cout << "<title>Text Area Data to CGI</title>\n";
    cout << "</head>\n";
    cout << "<body>\n";

    form_iterator fi = formData.getElement("textcontent");
    if( !fi->isEmpty() && fi != (*formData).end()) {
        cout << "Text Content: " << **fi << endl;
    }else{
        cout << "No text entered" << endl;
    }

```

(continues on next page)

(продолжение с предыдущей страницы)

```
31      cout << "<br/>\n";
32      cout << "</body>\n";
33      cout << "</html>\n";
34
35      return 0;
36  }
```

Drop Down Box

Python

```
1   #! /usr/bin/env python
2   # -*- coding: utf-8 -*-
3   # vim:fenc=utf-8
4   #
5   # Copyright © 2015 uralbash <root@uralbash.ru>
6   #
7   # Distributed under terms of the MIT license.
8
9   import cgi
10
11  # Create instance of FieldStorage
12  form = cgi.FieldStorage()
13
14  # Get data from fields
15  if form.getvalue('dropdown'):
16      subject = form.getvalue('dropdown')
17  else:
18      subject = "Not entered"
19
20  print("Content-type:text/html\r\n\r\n")
21  print("<html>")
22  print("<head>")
23  print("<title>Dropdown Box - Sixth CGI Program</title>")
24  print("</head>")
25  print("<body>")
26  print("<h2> Selected Subject is %s</h2>" % subject)
27  print("</body>")
28  print("</html>")
```

C++

Для компиляции: `make 7_dropdown`

```cpp
/*
 * 7.dropdown.cpp
 * Copyright (C) 2015 uralbash <root@uralbash.ru>
 *
 * Distributed under terms of the MIT license.
 */
#include <iostream>
#include <cgicc/Cgicc.h>

using namespace std;
using namespace cgicc;

int main()
{
    Cgicc formData;

    cout << "Content-type:text/html\r\n\r\n";
    cout << "<html>\n";
    cout << "<head>\n";
    cout << "<title>Drop Down Box Data to CGI</title>\n";
    cout << "</head>\n";
    cout << "<body>\n";

    form_iterator fi = formData.getElement("dropdown");
    if( !fi->isEmpty() && fi != (*formData).end()) {
        cout << "Value Selected: " << **fi << endl;
    }

    cout << "<br/>\n";
    cout << "</body>\n";
    cout << "</html>\n";

    return 0;
}
```

Печать Cookie

Примечание:

- http://localhost:8000/cgi-bin/8.getcookie.cgi
- http://localhost:8000/cgi-bin/8.getcookie.py

Python

```python
#! /usr/bin/env python
# -*- coding: utf-8 -*-
# vim:fenc=utf-8
#
# Copyright © 2014 uralbash <root@uralbash.ru>
#
# Distributed under terms of the MIT license.

# Import modules for CGI handling
from os import environ

print("Content-type:text/html\r\n\r\n")
print("<html>")
print("<head>")
print("<title>Get Cookie</title>")
print("</head>")
print("<body>")

if 'HTTP_COOKIE' in environ:
    for cookie in environ['HTTP_COOKIE'].split(';'):
        (key, value) = cookie.split('=')
        print("%s: %s" % (key, value))
        print("<br/>")

print("</body>")
print("</html>")
```

C++

Для компиляции: `make 8_getcookie`

```cpp
/*
 * 8.getcookie.cpp
 * Copyright (C) 2015 uralbash <root@uralbash.ru>
 *
 * Distributed under terms of the MIT license.
 */
#include <iostream>
#include <cgicc/Cgicc.h>

using namespace std;
using namespace cgicc;

int main()
{
    Cgicc cgi;
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    const_cookie_iterator cci;

    cout << "Content-type:text/html\r\n\r\n";
    cout << "<html>\n";
    cout << "<head>\n";
    cout << "<title>Cookies in CGI</title>\n";
    cout << "</head>\n";
    cout << "<body>\n";
    cout << "<table border = \"0\" cellspacing = \"2\">";

    // get environment variables
    const CgiEnvironment& env = cgi.getEnvironment();

    for( cci = env.getCookieList().begin();
            cci != env.getCookieList().end();
            ++cci )
    {
        cout << "<tr><td>" << cci->getName() << "</td><td>";
        cout << cci->getValue();
        cout << "</td></tr>\n";
    }
    cout << "</table>\n";

    cout << "<br/>\n";
    cout << "</body>\n";
    cout << "</html>\n";

    return 0;
}
```

Установка Cookie

Примечание:

- http://localhost:8000/cgi-bin/9.setcookie.cgi
- http://localhost:8000/cgi-bin/9.setcookie.py

Python

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-
# vim:fenc=utf-8
```

(continues on next page)

(продолжение с предыдущей страницы)

```
#
# Copyright © 2014 uralbash <root@uralbash.ru>
#
# Distributed under terms of the MIT license.

print("Set-Cookie:UserID=XYZ;")
print("Set-Cookie:Password=XYZ123;")
print("Set-Cookie:Expires=Tuesday, 31-Dec-2007 23:12:40 GMT;")
print("Set-Cookie:Domain=www.tutorialspoint.com;")
print("Set-Cookie:Path=/perl;")
print("Content-type:text/html\r\n")

print("<html>")
print("<head>")
print("<title>Cookies in CGI</title>")
print("</head>")
print("<body>")
print("Setting cookies")
print("<br/>")
print("</body>")
print("</html>")
```

C++

Для компиляции: `make 9_setcookie`

```
/*
 * 9.setcookie.cpp
 * Copyright (C) 2015 uralbash <root@uralbash.ru>
 *
 * Distributed under terms of the MIT license.
 */
#include <iostream>

using namespace std;

int main()
{
    cout << "Set-Cookie:UserID=XYZ;\r\n";
    cout << "Set-Cookie:Password=XYZ123;\r\n";
    cout << "Set-Cookie:Domain=www.tutorialspoint.com;\r\n";
    cout << "Set-Cookie:Path=/perl;\n";
    cout << "Content-type:text/html\r\n\r\n";

    cout << "<html>\n";
    cout << "<head>\n";
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    cout << "<title>Cookies in CGI</title>\n";
    cout << "</head>\n";
    cout << "<body>\n";

    cout << "Setting cookies" << endl;

    cout << "<br/>\n";
    cout << "</body>\n";
    cout << "</html>\n";

    return 0;
}
```

Загрузка файлов

Python

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-
# vim:fenc=utf-8
#
# Copyright © 2015 uralbash <root@uralbash.ru>
#
# Distributed under terms of the MIT license.
import cgi
import os

form = cgi.FieldStorage()

# Get filename here.
fileitem = form['filename']

# Test if the file was uploaded
if fileitem.filename:
    # strip leading path from file name to avoid
    # directory traversal attacks
    fn = os.path.basename(fileitem.filename)
    open('/tmp/' + fn, 'wb').write(fileitem.file.read())

    message = 'The file "' + fn + '" was uploaded successfully'

else:
    message = 'No file was uploaded'

```

(continues on next page)

(продолжение с предыдущей страницы)

```
print("""\
    Content-Type: text/html\n
<html>
<body>
<p>%s</p>
</body>
</html>""" % (message,))
```

C++

Для компиляции: `make 10_fileupload`

```
/*
 * 10.fileupload.cpp
 * Copyright (C) 2015 uralbash <root@uralbash.ru>
 *
 * Distributed under terms of the MIT license.
 */
#include <iostream>
#include <cgicc/Cgicc.h>
#include <cgicc/HTTPHTMLHeader.h>

using namespace std;
using namespace cgicc;

int main()
{
    Cgicc cgi;

    // get list of files to be uploaded
    const_file_iterator file = cgi.getFile("filename");
    if(file != cgi.getFiles().end()) {
        // send data type at cout.
        cout << HTTPContentHeader(file->getDataType());
        // write content at cout.
        file->writeToStream(cout);
        cout << "<br><br>";
        cout << "File uploaded successfully!\n";
    } else
        cout << "No file :(";

    return 0;
}
```

Отладка

См.также:

- http://pymotw.com/2/cgitb/
- https://docs.python.org/2/library/cgitb.html

Примечание:

- http://localhost:8000/cgi-bin/test.py

Python

```python
#! /usr/bin/env python
# -*- coding: utf-8 -*-
# vim:fenc=utf-8
#
# Copyright © 2015 uralbash <root@uralbash.ru>
#
# Distributed under terms of the MIT license.

import cgitb
cgitb.enable()


def func1(arg1):
    local_var = arg1 * 2
    return func2(local_var)


def func2(arg2):
    local_var = arg2 + 2
    return func3(local_var)


def func3(arg3):
    local_var = arg2 / 2    # noqa
    return local_var

func1(1)
```

1.3.3 FastCGI

См.также:

- http://dev-lab.info/2012/10/\T2A\cyrp\T2A\cyrr\T2A\cyri\T2A\cyrm\T2A\cyre\T2A\cyrr-\T2A\cyrp\T2A\cyrr\T2A\cyro\T2A\cyrs\T2A\cyrt\T2A\cyro\T2A\cyrg\T2A\cyro-fastcgi-\T2A\cyrs\T2A\cyre\T2A\cyrr\T2A\cyrv\T2A\cyre\T2A\cyrr\T2A\cyra-\T2A\cyri-fastcgi-\T2A\cyrk\T2A\cyrl\T2A\cyri\T2A\cyre\T2A\cyrn\T2A\cyrt\T2A\cyra/
- https://ru.wikipedia.org/wiki/FastCGI

Что такое FastCGI

В отличие от **CGI**, **FastCGI** использует постоянно запущенные процессы для обработки множества запросов.

CGI-программы взаимодействуют с сервером через **STDIN** и **STDOUT** запущенного процесса.

FastCGI-процессы используют для связи с сервером **Unix Domain Sockets** или **TCP/IP** . Это даёт следующее преимущество над обычными CGI-программами: FastCGI-программы могут быть запущены не только на этом же сервере, но и где угодно в сети. Также возможна обработка запросов несколькими FastCGI-процессами, работающими параллельно. Можно использовать несколько FastCGI-серверов, распределяя нагрузку между ними с помощью **nginx** или **lighttpd**.

После установления соединения **FastCGI-процесса** с **web-сервером**, между ними начинается обмен данными с использованием простого протокола, решающего две задачи: организация двунаправленного обмена в рамках одного соединения (для эмуляции **STDIN**, **STDOUT**, **STDERR**) и организация нескольких независимых FastCGI-сессий в рамках одного соединения.

Все передаваемые данные оборачиваются в **FastCGI-записи** — единицу данных протокола. **FastCGI-записи** служат для организации двунаправленного обмена и мультиплексирования нескольких сессий в рамках одного соединения.

FastCGI-запись состоит из заголовка фиксированной длины, следующего за ним содержимого и выравнивающих данных переменной длины. Каждая запись содержит 7 элементов.

Пример

См.также:

- http://chriswu.me/blog/writing-hello-world-in-fcgi-with-c-plus-plus/

Nginx

Примечание: *Nginx* доступен по адресу http://localhost:8080/

```
# default.nginx

server {
    listen 80 default_server;

    root /usr/share/nginx/html;
    index index.html index.htm;

    include includes/fcgi.nginx;
    include includes/static.nginx;
}
```

```
# fcgi.nginx

location /fastcgi_hello {
    # host and port to fastcgi server
    include         fastcgi.conf;
    fastcgi_pass 172.17.0.89:5000;
}
```

fastcgi_param

```
fastcgi_param  GATEWAY_INTERFACE  CGI/1.1;
fastcgi_param  SERVER_SOFTWARE    nginx;
fastcgi_param  QUERY_STRING       $query_string;
fastcgi_param  REQUEST_METHOD     $request_method;
fastcgi_param  CONTENT_TYPE       $content_type;
fastcgi_param  CONTENT_LENGTH     $content_length;
fastcgi_param  SCRIPT_FILENAME    $document_root$fastcgi_script_name;
fastcgi_param  SCRIPT_NAME        $fastcgi_script_name;
fastcgi_param  REQUEST_URI        $request_uri;
fastcgi_param  DOCUMENT_URI       $document_uri;
fastcgi_param  DOCUMENT_ROOT      $document_root;
fastcgi_param  SERVER_PROTOCOL    $server_protocol;
fastcgi_param  REMOTE_ADDR        $remote_addr;
fastcgi_param  REMOTE_PORT        $remote_port;
fastcgi_param  SERVER_ADDR        $server_addr;
fastcgi_param  SERVER_PORT        $server_port;
fastcgi_param  SERVER_NAME        $server_name;
```

C

Примечание: Компиляция `gcc -o hello.fcgi hello.cpp -lfcgi`

```
/*
 * hello.cpp
 * Copyright (C) 2015 uralbash <root@uralbash.ru>
 *
 * Distributed under terms of the MIT license.
 */
#include "fcgi_stdio.h"
#include <stdlib.h>

int main(void)
{
    while(FCGI_Accept() >= 0)
    {
        printf("Content-type: text/html\r\nStatus: 200 OK\r\n\r\nHello World!");
    }

    return 0;
}
```

Запуск fcgi сервера на 5000 порту

```
spawn-fcgi -p 5000 -n hello.fcgi
```

Примечание: Пример доступен по адресу http://localhost:8080/fastcgi_hello

1.3.4 Встроенный сервер

См.также:

- http://golang.org/pkg/net/http/

В некоторых языках, например в *Go*, уже существует встроенный Веб-сервер, который можно использовать в вашем приложении.

Go FastCGI

См.также:

- http://golang.org/pkg/net/http/fcgi/

В этом случае не нужно запускать отдельно fcgi сервер, например `spawn-fcgi`.

Примечание: Компиляция `go build -o hello.go.fcgi hello.go`

```
package main

import (
        "fmt"
        "net"
        "net/http"
        "net/http/fcgi"
)

func handler(res http.ResponseWriter, req *http.Request) {
        fmt.Fprint(res, "Hello World!")
}

func main() {
        // For local machine
        // l, _ := net.Listen("unix", "/var/run/go-fcgi.sock")

        l, err := net.Listen("tcp", "0.0.0.0:5000") // TCP 5000 listen
        if err != nil {
                return
        }
        http.HandleFunc("/", handler)
        fcgi.Serve(l, nil)
}
```

Запуск go fcgi сервера на 5000 порту (Без компиляции).

```
go run hello.go
```

Или скомпилированный файл.

```
./hello.go.fcgi
```

Настройка Nginx

```
# fcgi.nginx

location /fastcgi_hello {
    # host and port to fastcgi server
    include         fastcgi.conf;
    fastcgi_pass 172.17.0.89:5000;
}
```

```
Client Request ----> Nginx (Reverse-Proxy) ----> App. FastCGI Server I. 127.0.0.
↪1:5000
```

либо с балансировкой на несколько серверов:

```
# Nginx
upstream myapp1 {
    server 127.0.0.1:5000;
    server 127.0.0.1:5001;
    server 127.0.0.1:5002;
}

server {
    listen 80;

    location /some/path {
        fastcgi_pass http://myapp1;
    }
}
```

```
Client Request ----> Nginx (Reverse-Proxy)
                       |
                      /|\
                     | | `-> App. FastCGI Server I.   127.0.0.1:5000
                     |  `--> App. FastCGI Server II.  127.0.0.1:5001
                      `----> App. FastCGI Server III. 127.0.0.1:5002
```

Go HTTP

Запуск напрямую без CGI и FastCGI.

Примечание: Компиляция `go build -o hello.go.http hello.go`

```
package main

import (
        "fmt"
        "net/http"
)

func handler(w http.ResponseWriter, r *http.Request) {
        fmt.Fprintln(w, "Hello World!")
}
```

(continues on next page)

(продолжение с предыдущей страницы)

```
11
12 func main() {
13         http.HandleFunc("/", handler)
14         http.ListenAndServe(":8000", nil)
15 }
```

Запуск go http сервера на 8000 порту (Без компиляции).

```
go run hello.go
```

Или скомпилированный файл.

```
./hello.go.http
```

На такой сервер можно зайти напрямую по адресу http://localhost:8000/, либо настроить обратный прокси сервер:

```
1 # Nginx
2
3 location /some/path/ {
4     proxy_pass http://127.0.0.1:8000;
5 }
```

```
Client Request ----> Nginx (Reverse-Proxy) ----> App. HTTP Server I. 127.0.0.1:8000
```

либо с балансировкой на несколько серверов:

```
1  # Nginx
2  upstream myapp1 {
3      server 127.0.0.1:8000;
4      server 127.0.0.1:8001;
5      server 127.0.0.1:8002;
6  }
7
8  server {
9      listen 80;
10
11     location /some/path {
12         proxy_pass http://myapp1;
13     }
14 }
```

```
Client Request ----> Nginx (Reverse-Proxy)
                          |
                         /|\
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                    | | `-> App. HTTP Server I.   127.0.0.1:8000
                    |  `--> App. HTTP Server II.  127.0.0.1:8001
                     `----> App. HTTP Server III. 127.0.0.1:8002
```

1.3.5 WSGI (pep-333)

См.также:

- https://www.python.org/dev/peps/pep-0333/
- https://www.python.org/dev/peps/pep-3333/
- https://ru.wikipedia.org/wiki/WSGI
- http://docs.repoze.org/moonshining/pep333.html
- http://pylonsbook.com/en/1.1/the-web-server-gateway-interface-wsgi.html
- http://pylons-webframework.readthedocs.org/en/latest/concepts.html
- http://www.docstoc.com/docs/69863691/WSGI-from-Start-to-Finish
- Презентация с лекций

WSGI — стандарт взаимодействия между Python-программой, выполняющейся на стороне сервера, и самим веб-сервером, например Apache.

Идея:

В Python существует большое количество различного рода веб-фреймворков, тулкитов и библиотек. У каждого из них собственный метод установки и настройки, они не умеют взаимодействовать между собой. Это может стать затруднением для тех, кто только начинает изучать Python, так как, например, выбор определённого фреймворка может ограничить выбор веб-сервера и наоборот.

WSGI предоставляет простой и универсальный интерфейс между большинством веб-серверов и веб-приложениями или фреймворками.

Application

По стандарту, **WSGI-приложение** должно удовлетворять следующим требованиям:

- должно быть вызываемым (callable) объектом (обычно это функция или метод)
- принимать два параметра:
  - словарь переменных окружения (**environ**)
  - обработчик запроса (**start_response**)

- вызывать обработчик запроса с кодом HTTP-ответа и HTTP-заголовками
- возвращать итерируемый объект с телом ответа

Простейшим примером WSGI-приложения может служить такая функция-генератор:

```
def simple_app(environ, start_response):
    """
    (dict, callable( status: str,
                     headers: list[(header_name: str, header_value: str)]))
                  -> body: iterable of strings
    """
    status = '200 OK'
    response_headers = [('Content-type', 'text/plain')]
    start_response(status, response_headers)
    return 'Hello world!\n'
```

или то же самое в виде класса:

```
class AppClass(object):
    def __init__(self, environ, start_response):
        self.environ = environ
        self.start = start_response

    def __iter__(self):
        status = '200 OK'
        response_headers = [('Content-type', 'text/plain')]
        self.start(status, response_headers)
        yield "Hello world!\n"
```

Server/Gateway

Чтобы запустить наше WSGI приложение, нужен **WSGI сервер**. Он запускает *WSGI приложение* один раз при каждом HTTP запросе от клиента.

Задачи WSGI сервера:

- Сформировать переменные окружения (**environment**)
- Описать функцию обработчик запроса (**start_response**)
- Передать их в **WSGI приложение**
- Результат **WSGI сервер** отправляет по HTTP клиенту
- а **WSGI шлюз** приводит к формату клиент-серверного протокола (CGI, FastCGI, SCGI, uWSGI, ...) и передает их на Веб-сервер (например выводит в stdout, stderr).

Пример WSGI-шлюза к CGI-серверу.

```
import os
import sys


def run_with_cgi(application):

    environ = dict(os.environ.items())
    environ['wsgi.input'] = sys.stdin
    environ['wsgi.errors'] = sys.stderr
    environ['wsgi.version'] = (1, 0)
    environ['wsgi.multithread'] = False
    environ['wsgi.multiprocess'] = True
    environ['wsgi.run_once'] = True

    if environ.get('HTTPS', 'off') in ('on', '1'):
        environ['wsgi.url_scheme'] = 'https'
    else:
        environ['wsgi.url_scheme'] = 'http'

    headers_set = []
    headers_sent = []

    def write(data):
        if not headers_set:
            raise AssertionError("write() before start_response()")

        elif not headers_sent:
            # Before the first output, send the stored headers
            status, response_headers = headers_sent[:] = headers_set
            sys.stdout.write('Status: %s\r\n' % status)
            for header in response_headers:
                sys.stdout.write('%s: %s\r\n' % header)
                sys.stdout.write('\r\n')

        sys.stdout.write(data)
        sys.stdout.flush()

    def start_response(status, response_headers, exc_info=None):
        if exc_info:
            try:
                if headers_sent:
                    # Re-raise original exception if headers sent
                    raise Exception(exc_info[0], exc_info[1], exc_info[2])
            finally:
                exc_info = None     # avoid dangling circular ref
        elif headers_set:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            raise AssertionError("Headers already set!")

        headers_set[:] = [status, response_headers]
        return write

    result = application(environ, start_response)
    try:
        for data in result:
            if data:    # don't send headers until body appears
                write(data)
        if not headers_sent:
            write('')   # send headers now if body was empty
    finally:
        if hasattr(result, 'close'):
            result.close()
```

Environment

Всегда словарь

- `environ` это обычно копия переменных окружения ОС `os.environ` + стандартные CGI переменные.

  `SCRIPT_NAME` - содержит имя вызванного скрипта. Например: myapp.py

  `PATH_INFO` - путь к файлу `/cgi-bin/myapp.py`

- также включает в себя дополнительные WSGI-специфичные переменные, наиболее важные из них:

  `wsgi.input` - представляет тело (body) HTTP запроса.

  `wsgi.errors` - указывает поток куда нужно выводить ошибки.

  `wsgi.url_scheme` - это просто «http» или «https».

start_response

См.также:

- Реализация в mod_wsgi для Apache

Функция `start_response` принимает два обязательных аргумента:

- `status` - строка содержащая статус HTTP ответа, например `200 OK`.

- `response_headers` - список кортежей, которые содержат заголовки ответа, например `[('Content-Type', 'text/html'), ('Content-Length', '15')]`.

```
def simple_app(environ, start_response):
    """
    (dict, callable( status: str,
                     headers: list[(header_name: str, header_value: str)]))
                  -> body: iterable of strings
    """
    status = '200 OK'
    response_headers = [('Content-type', 'text/plain')]
    start_response(status, response_headers)
    return 'Hello world!\n'
```

`start_response` возвращает вызываемый объект, обычно «write». `write` выводит тело ответа в поток вывода, используется при необычных обстоятельствах.

Предупреждение: Обратный вызов `write` плохо поддерживается серверами и веб-фреймворками, поэтому рекомендуется проектировать свои приложения без его вызова.

Обычно данные возвращаются таким образом:

```
def application(environ, start_response):
    start_response(status, headers)
    return ['content block 1',
            'content block 2',
            'content block 3']
```

Но можно делать и так:

```
def application(environ, start_response):
    write = start_response(status, headers)
    write('content block 1')
    return ['content block 2',
            'content block 3']
```

Запуск нашего приложения через WSGI-шлюз к CGI

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-
# vim:fenc=utf-8
#
# Copyright © 2015 uralbash <root@uralbash.ru>
#
```

(continues on next page)

(продолжение с предыдущей страницы)

```
# Distributed under terms of the MIT license.

"""
Example from pep-333
"""
from cgi_gateway import run_with_cgi


def simple_app(environ, start_response):
    """
    (dict, callable( status: str,
                     headers: list[(header_name: str, header_value: str)]))
                  -> body: iterable of strings
    """
    status = '200 OK'
    response_headers = [('Content-type', 'text/plain')]
    start_response(status, response_headers)
    return 'Hello world!\n'


class AppClass(object):
    def __init__(self, environ, start_response):
        self.environ = environ
        self.start = start_response

    def __iter__(self):
        status = '200 OK'
        response_headers = [('Content-type', 'text/plain')]
        self.start(status, response_headers)
        yield "Hello world!\n"


if __name__ == '__main__':
    run_with_cgi(AppClass)
```

Результат выполнения:

```
$ python 1.cgi.app.py
Status: 200 OK
Content-type: text/plain

Hello world!
```

Примечание: В исходных кодах к лекциям `cgiserver.py` делает этот пример доступ-

ным по адресу http://localhost:8000/wsgi/1.cgi.app.py

Middleware

Помимо приложений и серверов, стандарт дает определение **middleware-компонентов**, предоставляющих интерфейсы как приложению, так и серверу. То есть для сервера middleware является приложением, а для приложения сервером. Это позволяет составлять «цепочки» WSGI-совместимых middleware.

Middleware могут брать на себя следующие функции (но не ограничиваются этим):

- обработка сессий
- аутентификация/авторизация
- управление URL (маршрутизация запросов)
- балансировка нагрузки
- пост-обработка выходных данных (например, проверка на валидность)

Мы рассмотрим пример приложения, которое считает количество обращений и использует следующие middleware:

- Обработчик исключений
- Сессии
- Сжатие Gzip
- Пони

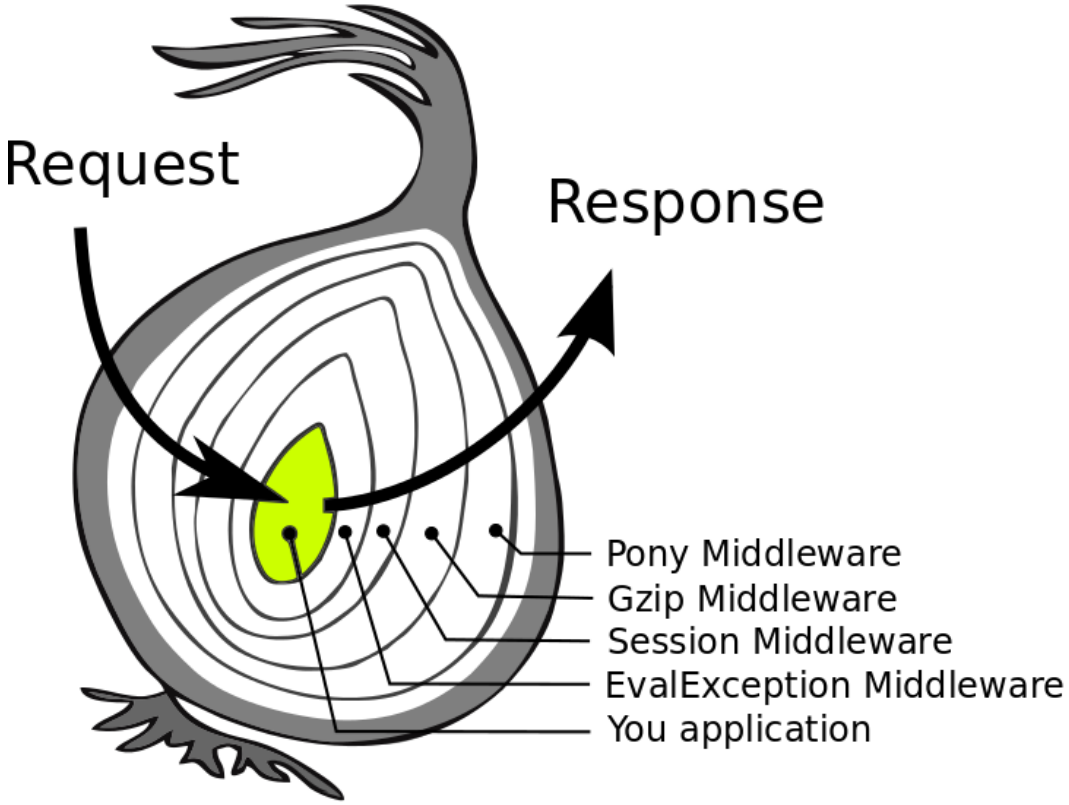
Приложение

```
1                       def app(environ, start_response):
2                           # Except error
3                           if 'error' in environ['PATH_INFO'].lower():
4                               raise Exception('Detect "error" in URL path')
5
6                           # Session
7                           session␣
                        ↪= environ.get('paste.session.factory', lambda: {})()
8                           if 'count' in session:
9                               count = session['count']
10                          else:
11                              count = 1
12                          session['count'] = count + 1
```

(continues on next page)

(продолжение с предыдущей страницы)

```
13
14                      # Generate response
15                      start_
                    ↪response('200 OK', [('Content-Type', 'text/plain')])
16                      return [b'You have been here %d times!\n' % count, ]
```

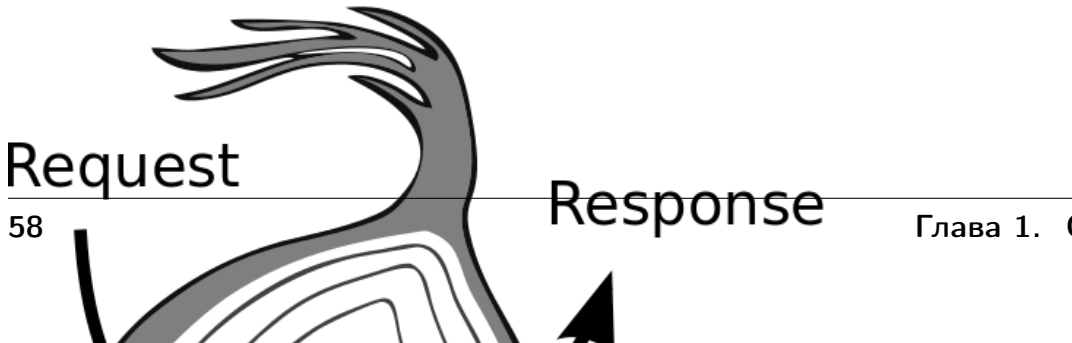


Приложение выводит число 1 при первом обращении, записывает его в сессию и при каждом последующем обращении увеличивает число на 1.

Т.к. протокол HTTP не сохраняет предыдущего состояния, то при обновлении страницы число не увеличится. Чтобы это произошло, нужно реализовать механизм сессий.

Обработчик исключений

```
1 from paste.evalexception.middleware import EvalException
2 app = EvalException(app)
```

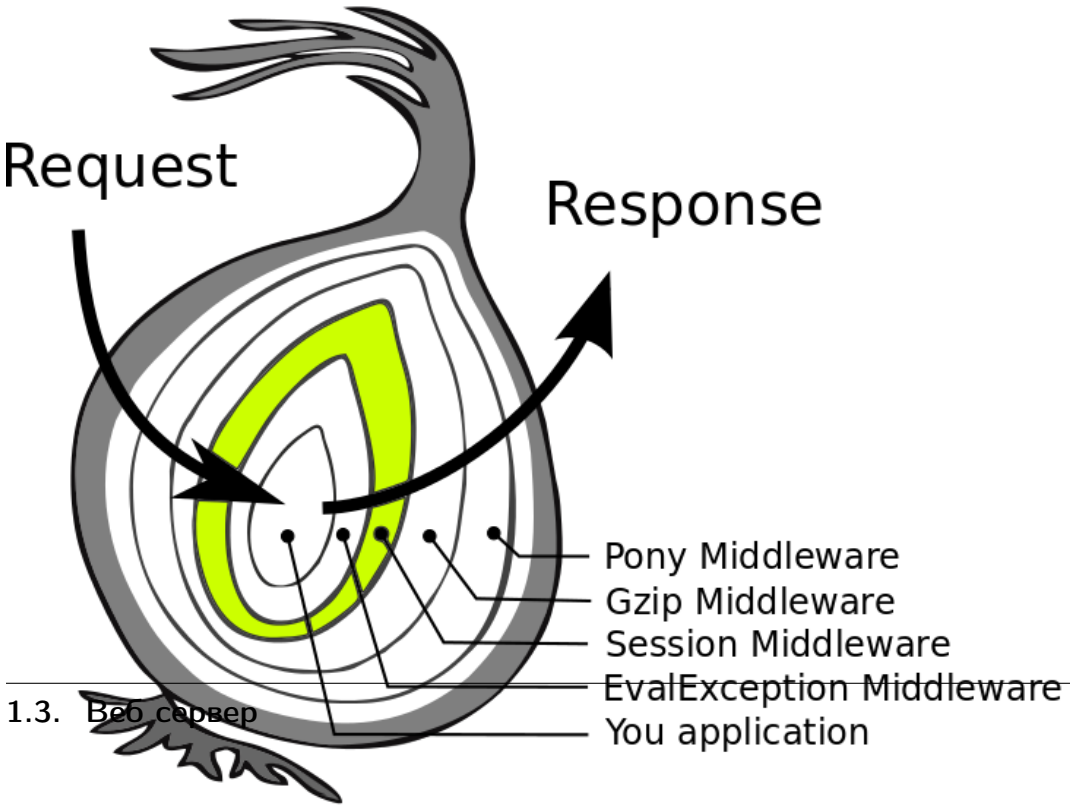


EvalException позволяет нам

отлавливать ошибки и выводить их в браузере. Если мы перейдем по адресу http://localhost:8000/Errors_500, наше приложение найдет слово *error* в пути и искусственно вызовет исключение.

Сессии

```
1 from paste.session import SessionMiddleware
2 app = SessionMiddleware(app)
```



`SessionMiddleware` добавляет cookie клиенту с ключом *_SID_* и номером сессии.

Например `_SID_=20150313142600-`

Для каждой сессии на сервере в директории `/tmp/` (по умолчанию)

создается файл с таким же именем.

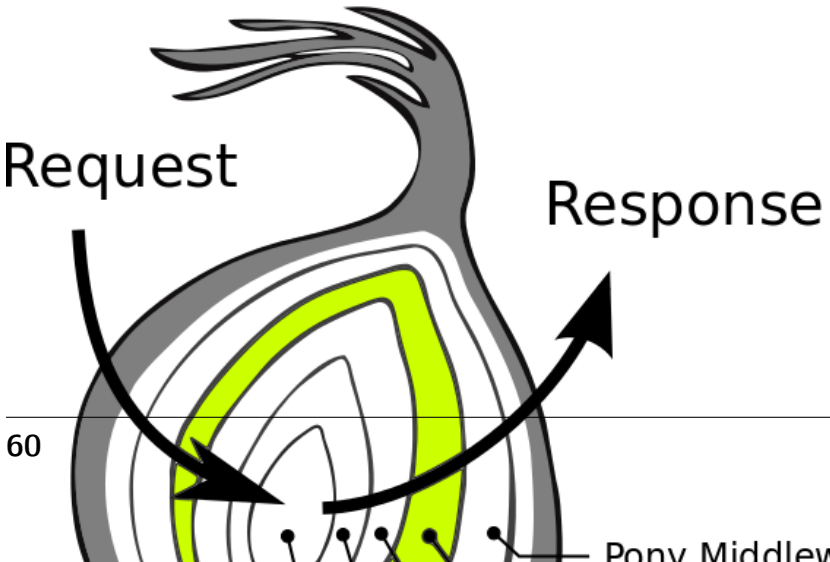
```
↪20150313094744-
↪5d2e448000e6312d7c0b8

↪20150313142600-
↪d18ec118fff970ad4fb36

↪ 2 files
```

В этот файл записывается значение `count` для нашей сессии. При каждом обращении клиента `SessionMiddleware` находит файл с таким же именем как у cookie `_SID_` десереализует объекты в нем и присваивает переменной окружения `paste.session.factory`. Таким образом мы можем хранить состояние сессии и при каждом обновлении будет отдаваться значение, увеличенное на 1.

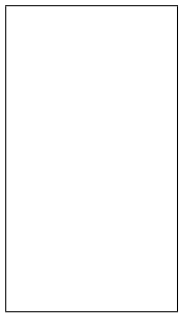
Сжатие Gzip

```
1 from paste.gzipper import middleware as GzipMiddleware
2 app = GzipMiddleware(app)
```

`GzipMiddleware` сжимает ответ методом gzip



Pony



Это самое важное расширение в WSGI. Доступно по адресу http://localhost:8000/pony.

Полный пример

```
#! /usr/bin/env nix-shell

#! nix-shell -i python3 -p python3Packages.paste
```

(continues on next page)

are
ddleware

(продолжение с предыдущей страницы)

```python
# -*- coding: utf-8 -*-
# vim:fenc=utf-8
#
# Copyright © 2015 uralbash <root@uralbash.ru>
#
# Distributed under terms of the MIT license.

"""
3.http.middleware.py
"""

from paste.evalexception.middleware import EvalException
from paste.gzipper import middleware as gzipmiddleware
```

(continues on next page)

(продолжение с предыдущей страницы)

```
from paste.pony import PonyMiddleware
from paste.session import SessionMiddleware


def app(environ, start_response):
    # Except error
    if 'error' in environ['PATH_INFO'].lower():
        raise Exception('Detect "error" in URL path')

    # Session
    session = environ.get('paste.session.factory
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    if␣
↪'count
↪' in␣
↪session:
        ␣
↪count =␣
↪session[
↪'count']
    else:
     ␣
↪  count␣
↪= 1
   ␣
↪session[
↪'count
↪']␣
↪= count␣
↪+ 1


␣
↪␣
↪␣
↪␣
↪
↪#␣
↪Generate␣
↪response
␣
↪␣
↪␣
↪␣
↪start_
↪response(
↪'200␣
↪OK
↪',
↪␣
↪[(
↪'Content-
↪Type',
↪ 'text/
↪plain
↪')])
  ␣
↪ return␣
↪[b'You␣
↪have␣
↪been␣
↪here␣
↪%d␣
↪times!
```

(continues on next page)

(продолжение с предыдущей страницы)

```


app = EvalException(app)     # go to http://localhost:8000/Errors
app = SessionMiddleware(app
app = GzipMiddleware(app)
app = PonyMiddleware(app)   # go to http://localhost:8000/pony

if __name__ == '__main__':
    from paste import reloader
    from paste.httpserver import serve
```

(continues on next page)

(продолжение с предыдущей страницы)

```

        reloader.install()
        serve(app, host='0.0.0.0', port=8000)
```

Свой middleware

```
class GoogleRefMiddleware(o
    def __init__(self, app):
        self.app = app

    def __call__(self, environ, start_response):
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                environ['google'] = False
            if 'HTTP_REFERER' in environ:
                if environ['HTTP_REFERER'].startswith('http://google.com'):
                    environ['google'] = True
            return self.app(environ, start_
```

(continues on next page)

(продолжение с предыдущей страницы)

```
11
12                                                                  app␣
                                                                    ↪=␣
                                                                    ↪GoogleRefMiddleware(a
```

`GoogleRefMiddleware` добавляет переменную окружения `google`, и если бы мы перешли на наш сайт из поиска `google.com`, то это значение было бы `True`.

Кто использует WSGI?

- BlueBream
- bobo
- Bottle
- CherryPy
- Django
- Eventlet
- Flask
- Google App Engine's webapp2
- Gunicorn
- prestans

- mod_wsgi для Apache
- MoinMoin
- netius
- Plone
- Pylons
- Pyramid
- repoze
- restlite
- Tornado
- Trac
- TurboGears
- Uliweb
- webpy
- Falcon
- web2py
- weblayer
- Werkzeug
- Zope
- и многие другие

Аналоги

- Rack – Ruby web server interface
- PSGI – Perl Web Server Gateway Interface
- JSGI – JavaScript web server gateway interface
- WAI - Web Application Interface (Haskell)
- Ring - Clojure

1.4 Веб-программирова

См.также:

- Веб-разработка без фреймворков
- A Do-It-Yourself Framework
- Another Do-It-Yourself Framework

В этом разделе мы напишем *еще один блог*, используя популярные инструменты языка программирования *Python*.

1.4.1 Paste

См.также:

- https://ru.wikipedia.org/wiki/Python_Paste

Примечание: Исходный код доступен по адресу:
https://github.com/iitwebdev/lectures_wsgi_example

Python *Paste*, или просто *Paste* — набор

программ для веб-разработки. Включает в себя множество различных *middleware*, *WSGI-сервер* и другое. Был разработан *Яном Бикингом*, чтобы показать всю красоту спецификации *WSGI*, которая на тот момент была еще в черновиках. Проект больше академический и до недавнего времени не имел даже поддержки *Python 3*, но несмотря на это, многие современные фреймворки взяли за основу примеры из *Paste* (TurboGears, Zope, Pylons, Pyramid)

В нем есть готовая поддержка самых разных способов аутентификации (Basic, Digest, form, signed cookie, auth_tkt), поддержка корректной и удобной генерации ответов и заголовков (к примеру редиректы, Cache-control, Expires, gzipper и прочие). Различные базовые средства комбинации приложений (URLMap, Cascade, Recursive), статических данных (с учетом

Etag, If-Modified итп).

Некоторые возможности `paste` мы рассмотрели в разделе *WSGI (pep-333)*.

Предупреждение: Примеры работают только в Python3

HTTP server

См.также:

- http://pythonpaste.org/modules/httpserver.html#module-paste.httpserver

Встроенный *WSGI* сервер `wsgiref` появился в *Python* начиная с версии 2.5, частично вобрав наработки из мо-

дуля `paste.httpserver`. На данный момент целесообразно использовать встроенный в Python модуль `wsgiref` или сторонние более производительные реализации waitress и gunicorn.

```
def blog(environ, start_response):
    start_response('200 OK', [('Content-Type', 'text/plain')])
    return [b'Simple Blog', ]


if __name__ == '__main__':
    from wsgiref.simple_server import make_server
```

(continues on next page)

(продолжение с предыдущей страницы)

```
11                                                    httpd␣
                                                    ↪= make_
                                                    ↪server(
                                                    ↪'0.0.0.0
                                                    ↪', 8000,
                                                    ↪ blog)
12                                                    ␣
                                                    ↪␣
                                                    ↪␣
                                                    ↪␣
                                                    ↪httpd.
                                                    ↪serve_
                                                    ↪forever()
```

В нашем случае подойдет любая реализация сервера отвечающего стандарту *WSGI*, поэтому в примерах используется `paste.httpserver`. С его помощью запустим простое *WSGI-приложение*:

```
1                                                     def␣
                                                    ↪blog(environ,
                                                    ↪␣
                                                    ↪start_
                                                    ↪response):
```

(continues on next page)

(продолжение с предыдущей страницы)

```
␣
↪␣
↪␣
↪␣
↪start_
↪response(
   ␣
↪   '200␣
↪OK',
␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪[(
↪'Content-
↪Type',
↪ 'text/
↪plain')]
   )
  ␣
↪ return␣
↪[b
↪'Simple␣
↪Blog', ]


if __name_
↪_ == '__
↪main__':
␣
↪␣
↪␣
↪␣
↪from␣
↪paste.
↪httpserver␣
↪import␣
↪serve
␣
↪␣
↪␣
↪␣
↪serve(blog,
↪␣
↪host= 75
↪'0.
↪0.
```

(continues on next page)

(продолжение с предыдущей страницы)

Теперь приложение доступно по адресу http://localhost:8000/.

Примечание: Стоит отметить, что приложение будет доступно по любому пути этого адреса, например:

- http://localhost:8000/
- http://localhost:8000/foo
- http://localhost:8000/foo/bar/
- http://localhost:8000/foo/bar/baz
- http://localhost:8000/no_good

URL диспетчеризация

Доступ к *WSGI* приложению обычно осуществляется по конкретным *URL* адресам (ресурсам). В нашем примере приложение *blog* должно

быть доступно только по корневому *URL* адресу, иные адреса должны выдавать страницу с ошибкой *404*.

Для разделения путей напишем *WSGI-middleware* `URLDispatch`.

```
class URLDispatch(object):

    def __init__(self, app_list):
        self.app_list = app_list


    def __call__(self, environ, start_response):
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        path_info = environ.get('PATH_INFO', '')
        for prefix, app in self.app_list:
            if path_info == prefix or path_info == prefix+'/':
                return app(environ, start_response)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        start_response('404 Not Found',
                       [('content-type', 'text/plain')])
        return [b'not found']
```

Добавим настройки в наше приложе-

ние:

```
from middlewares.urldispatch import URLDispatch


def blog(environ, start_response):
    start_response('200 OK', [('Content-Type', 'text/plain')])
    return [b'Simple Blog', ]


# URL dispatching middleware
```

(continues on next page)

(продолжение с предыдущей страницы)

```
app_list = [
    ('/', blog),
]
dispatch = URLDispatch(app_list)

if __name__ == '__main__':
    from paste.httpserver import serve
    serve(dispatch, host='0.0.0.0', port=8000)
```

Любой путь, отличающийся от корневого (http://localhost:8000/), по

которому доступно приложение `blog`, будет инициализировать код ошибки *404*.

Такой механизм в Веб-разработке называется *URL* маршрутизация или диспетчеризация, более подробно об этом будет говориться в разделе *Маршруты*.

Структура нашего блога будет состоять из следующих страниц:

Название	URL	Описание
Главная	/	Показывает все записи в блоге, отсортированные по дате
(**CREATE**) Добавление	/article/add	Форма добавления новой статьи
(**READ**) Просмотр	/article/{id}	Показывает конкретную статью, соответствующую {id}
(**UPDATE**) Редактирование	/article/{id}/edit	Редактирование статьи по {id}
(**DELETE**) Удаление	/article/{id}/delete	Удаление статьи по {id}

По сути блог является стандартным *CRUD* (CREATE-READ-UPDATE-DELETE) интерфейсом, каждую часть которого будет реализовывать свое отдельное *WSGI* приложение, связанное со своим *URL* адресом.

```
1
```

```
from␣
↪middlewares.
↪urldispatch␣
↪import␣
↪URLDispatch
```

(continues on next page)

(продолжение с предыдущей страницы)

```
class␣
↪BaseBlog(object):
␣
↪␣
↪␣
↪␣
↪def␣
↪_
↪_
↪init_
↪_
↪(self,
↪␣
↪environ,
↪␣
↪start_
↪response):
    ␣
↪   self.
↪environ␣
↪=␣
↪environ
    ␣
↪   self.
↪start␣
↪= start_
↪response


class␣
↪BlogIndex(BaseBlog):

    def␣
↪__iter_
↪_(self):
␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪self.
↪start(
↪200␣
↪OK
↪',
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        yield b'Simple Blog'


class BlogCreate(BaseBlog):

    def __iter__(self):
        self.start('200 OK', [('Content-Type', 'text/plain')])
        yield b'Simple Blog -> CREATE'


class BlogRead(BaseBlog):

    def __iter__(self):
```

(continues on next page)

(продолжение с предыдущей страницы)

```
28                                                                 ␣
                                                                   ↪␣
                                                                   ↪␣
                                                                   ↪␣
                                                                   ↪␣
                                                                   ↪␣
                                                                   ↪␣
                                                                   ↪self.
                                                                   ↪start(
                                                                   ↪'200␣
                                                                   ↪OK
                                                                   ↪',
                                                                   ↪␣
                                                                   ↪[(
                                                                   ↪'Content-
                                                                   ↪Type',
                                                                   ↪ 'text/
                                                                   ↪plain
                                                                   ↪')])
29                                                                     ␣
                                                                   ↪ yield b
                                                                   ↪'Simple␣
                                                                   ↪Blog␣
                                                                   ↪-> READ'
30
31
32                                                                 class␣
                                                                   ↪BlogUpdate(BaseBlog):
33
34                                                                     def␣
                                                                   ↪__iter_
                                                                   ↪_(self):
35                                                                 ␣
                                                                   ↪␣
                                                                   ↪␣
                                                                   ↪␣
                                                                   ↪␣
                                                                   ↪␣
                                                                   ↪␣
                                                                   ↪self.
                                                                   ↪start(
                                                                   ↪'200␣
                                                                   ↪OK
                                                                   ↪',
                                                                   ↪␣
                                                                   ↪[(
                                                                   ↪'Content-
                                                                   ↪Type',
                                                                   ↪ 'text/
```

(continues on next page)

(продолжение с предыдущей страницы)

```
36                                                                  ␣
                                                                  ↪ yield b
                                                                  ↪'Simple␣
                                                                  ↪Blog ->
                                                                  ↪ UPDATE'
37
38
39                                                                class␣
                                                                  ↪BlogDelete(BaseBlog):
40
41                                                                    def␣
                                                                  ↪__iter_
                                                                  ↪_(self):
42                                                                ␣
                                                                  ↪␣
                                                                  ↪␣
                                                                  ↪␣
                                                                  ↪␣
                                                                  ↪␣
                                                                  ↪␣
                                                                  ↪self.
                                                                  ↪start(
                                                                  ↪'200␣
                                                                  ↪OK
                                                                  ↪',
                                                                  ↪␣
                                                                  ↪[(
                                                                  ↪'Content-
                                                                  ↪Type',
                                                                  ↪ 'text/
                                                                  ↪plain
                                                                  ↪')])
43                                                                      ␣
                                                                  ↪ yield b
                                                                  ↪'Simple␣
                                                                  ↪Blog ->
                                                                  ↪ DELETE'
44
45
46
                                                                  ↪#␣
                                                                  ↪URL␣
                                                                  ↪dispatching␣
                                                                  ↪middleware
```

(continues on next page)

(продолжение с предыдущей страницы)

```
app_list = [
    ('/', BlogIndex),
    ('/article/add', BlogCreate),
    ('/article/{id}', BlogRead),
    ('/article/{id}/edit',
```

(continues on next page)

(продолжение с предыдущей страницы)

```
52                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪(
                                                                    ↪'/
                                                                    ↪article/
                                                                    ↪
                                                                    ↪{id}
                                                                    ↪/
                                                                    ↪delete
                                                                    ↪',
                                                                    ↪␣
                                                                    ↪BlogDelete),
                                                                    ↪
53                                                                  ]
54                                                                  dispatch␣
                                                                    ↪=␣
                                                                    ↪URLDispatch(app_
                                                                    ↪list)
55
56                                                                  if __name_
                                                                    ↪_ == '__
                                                                    ↪main__':
57                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪from␣
                                                                    ↪paste.
                                                                    ↪httpserver␣
                                                                    ↪import␣
                                                                    ↪serve
58                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪serve(dispatch,
                                                                    ↪␣
                                                                    ↪host=
                                                                    ↪'0.
                                                                    ↪0.
                                                                    ↪0.
                                                                    ↪0
                                                                    ↪',
                                                                    ↪␣
                                                                    ↪port=8000)
```

(continues on next page)

(продолжение с предыдущей страницы)

Примечание: Обратите внимание, что адреса доступны по следующим ссылкам:

- http://localhost:8000/article/add
- http://localhost:8000/article/\protect\T2A\textbraceleftid\protect\T2A\textbraceright
- http://localhost:8000/article/\protect\T2A\textbraceleftid\protect\T2A\textbraceright/edit
- http://localhost:8000/article/\protect\T2A\textbraceleftid\protect\T2A\textbraceright/delete

Если вместо `{id}` подставить цифру, то вернется *404 ошибка*.

Пока наша реализация роутов ничего не знает про символы типа «{id}», поэтому мы не можем заменять их числом. Чтобы это испра-

вить
научим

`URLDispatch` *middleware* понимать регулярные выражения.

Название	URL	Описание
Главная	/	Показывает все записи в блоге, отсортированные по дате
(**CREATE**) Добавление	/article/add	Форма добавления новой статьи
(**READ**) Просмотр	^/article/(?P<id>d+)/$	Показывает конкретную статью, соответствующую {id}
(**UPDATE**) Редактирование	^/article/(?P<id>d+)/edit$	Редактирование статьи по {id}
(**DELETE**) Удаление	^/article/(?P<id>d+)/delete$	Удаление статьи по {id}

```
1  class␣
   ↪RegexDispatch(object)
2
3      def␣
   ↪__init_
   ↪_(self,
   ↪ app_
   ↪list):
4      ␣
   ↪   self.
   ↪app_
   ↪list =␣
   ↪app_list
5
6  ␣
   ↪␣
   ↪␣
   ↪␣
   ↪def␣
   ↪_
   ↪_
   ↪call_
   ↪_
   ↪(self,
   ↪␣
   ↪environ,
   ↪␣
   ↪start_
   ↪response):
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        path_info = environ.get('PATH_INFO', '')
        for prefix, app in self.app_list:
            if path_info == prefix or path_info == prefix+'/':
                return app(environ, start_response)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            match = re.match(prefix, path_info) or\
                    re.match(prefix, path_info+'/')

```

(continues on next page)

(продолжение с предыдущей страницы)

```
environ['url_params'] = match.groupdict()
return app(environ, start_response)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        return [b'not found']
```

Подменим символы «{id}» в адресах на регулярные выражения:

```
from middlewares.urldispatch import RegexDispatch


class BaseBlog(object):

    def __init__(self, environ, start_response):
        self.environ = environ
        self.start = start_response
```

(continues on next page)

(продолжение с предыдущей страницы)

```
class BlogIndex(BaseBlog):

    def __iter__(self):
        self.start('200 OK', [('Content-Type', 'text/plain')])
        yield b'Simple Blog'


class BlogCreate(BaseBlog):

    def __iter__(self):
        self.start('200
```

(continues on next page)

(продолжение с предыдущей страницы)

```
22                                                                  ␣
                                                                  ↪ yield b
                                                                  ↪'Simple␣
                                                                  ↪Blog ->
                                                                  ↪ CREATE'
23
24
25                                                                class␣
                                                                  ↪BlogRead(BaseBlog):
26
27                                                                    def␣
                                                                  ↪__iter_
                                                                  ↪_(self):
28                                                                ␣
                                                                  ↪␣
                                                                  ↪␣
                                                                  ↪␣
                                                                  ↪␣
                                                                  ↪␣
                                                                  ↪␣
                                                                  ↪␣
                                                                  ↪self.
                                                                  ↪start(
                                                                  ↪'200␣
                                                                  ↪OK
                                                                  ↪',
                                                                  ↪␣
                                                                  ↪[(
                                                                  ↪'Content-
                                                                  ↪Type',
                                                                  ↪ 'text/
                                                                  ↪plain
                                                                  ↪')])
29                                                                  ␣
                                                                  ↪ yield b
                                                                  ↪'Simple␣
                                                                  ↪Blog␣
                                                                  ↪-> READ'
30
31
32                                                                class␣
                                                                  ↪BlogUpdate(BaseBlog):
33
34                                                                    def␣
                                                                  ↪__iter_
                                                                  ↪_(self):
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        self.start('200 OK', [('Content-Type', 'text/plain')])
        yield b'Simple Blog -> UPDATE'


class BlogDelete(BaseBlog):

    def __iter__(self):
        self.start('200 OK', [('Content-Type', 'text/plain
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        yield b'Simple Blog -> DELETE'



# URL dispatching middleware
app_list = [
    ('/', BlogIndex),
    ('/article/add', BlogCreate),
    (r'^/article/(?P<id>\d+)/$'
```

(continues on next page)

(продолжение с предыдущей страницы)

```
51                                                                  ␣
                                                                     ↪␣
                                                                     ↪␣
                                                                     ↪␣
                                                                     ↪(r
                                                                     ↪'^
                                                                     ↪/
                                                                     ↪article/
                                                                     ↪(?
                                                                     ↪P
                                                                     ↪<id>
                                                                     ↪\d+)/
                                                                     ↪edit/
                                                                     ↪
                                                                     ↪$
                                                                     ↪',
                                                                     ↪␣
                                                                     ↪BlogUpdate),
                                                                     ↪
52                                                                  ␣
                                                                     ↪␣
                                                                     ↪␣
                                                                     ↪␣
                                                                     ↪(r
                                                                     ↪'^
                                                                     ↪/
                                                                     ↪article/
                                                                     ↪(?
                                                                     ↪P
                                                                     ↪<id>
                                                                     ↪\d+)/
                                                                     ↪delete/
                                                                     ↪
                                                                     ↪$
                                                                     ↪',
                                                                     ↪␣
                                                                     ↪BlogDelete),
                                                                     ↪
53                                                                  ]
54                                                                  dispatch␣
                                                                     ↪=␣
                                                                     ↪RegexDispatch(app_
                                                                     ↪list)
55
56                                                                  if __name_
                                                                     ↪_ == '__
                                                                     ↪main__':
```

(continues on next page)

(продолжение с предыдущей страницы)

```
57                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪from␣
                                                                    ↪paste.
                                                                    ↪httpserver␣
                                                                    ↪import␣
                                                                    ↪serve
58                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪serve(dispatch,
                                                                    ↪␣
                                                                    ↪host=
                                                                    ↪'0.
                                                                    ↪0.
                                                                    ↪0.
                                                                    ↪0
                                                                    ↪',
                                                                    ↪␣
                                                                    ↪port=8000)
```

Примечание: Теперь можно переходить по URL'ам с числами вместо {id}, например:

- http://localhost:8000/article/1
- http://localhost:8000/article/13/
- http://localhost:8000/article/100500
- http://localhost:8000/article/100500/edit
- http://localhost:8000/article/100500/delete

Данные

Приложения обычно пред-ставляют данные, которые

собирают из разных мест (БД, память, файлы, и т.д.), для блога такими данными как раз являются статьи и возможно комментарии к ним.

Оформим данные в виде списка, каждая запись которого будет являться статьей со следующими ключами `id`, `title`, `content`.

```
ARTICLES = [
    {
'id': 1,
 'title':
'Lorem ipsum dolor sid amet!',

'content':
```

(continues on next page)

(продолжение с предыдущей страницы)

```
␣
↪␣
↪␣
↪␣
↪␣
↪Curabitur␣
↪vel␣
↪tortor␣
↪eleifend,
↪␣
↪sollicitudin␣
↪nisl␣
↪quis,␣
↪lacinia␣
↪augue.
    Duis␣
↪quam␣
↪est,␣
↪laoreet␣
↪sit␣
↪amet␣
↪justo␣
↪vitae,␣
↪viverra␣
↪egestas␣
↪sem.
␣
↪␣
↪␣
↪␣
↪␣
↪Maecenas␣
↪pellentesque␣
↪augue␣
↪in␣
↪nibh␣
↪feugiat␣
↪tincidunt.
↪ Nunc␣
↪magna␣
↪ante,
␣
↪␣
↪␣
↪␣
↪␣
↪mollis␣
↪vitae␣
↪ultricies␣
↪eu,
↪␣
```

(continues on next page)

(продолжение с предыдущей страницы)

```
     blandit ipsum a, ullamcorper nunc. Sed bibendum eget odio eget
     pellentesque. Curabitur elit felis, pellentesque id feugiat et, tincidunt
     ut mauris. Integer vitae vehicula nunc. Integer ullamcorper, nunc in
```

(continues on next page)

(продолжение с предыдущей страницы)

```
11                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪volutpat␣
                                                                    ↪auctor,
                                                                    ↪␣
                                                                    ↪elit␣
                                                                    ↪leo␣
                                                                    ↪convallis␣
                                                                    ↪nulla,
                                                                    ↪ vitae␣
                                                                    ↪varius␣
                                                                    ↪mi␣
                                                                    ↪nisl ac␣
                                                                    ↪lorem.
12                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪Sed␣
                                                                    ↪a␣
                                                                    ↪lacus␣
                                                                    ↪mi.
                                                                    ↪␣
                                                                    ↪In␣
                                                                    ↪hac␣
                                                                    ↪habitasse␣
                                                                    ↪platea␣
                                                                    ↪dictumst.
                                                                    ↪␣
                                                                    ↪Cras in␣
                                                                    ↪posuere␣
                                                                    ↪velit,
13                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪id␣
                                                                    ↪dignissim␣
                                                                    ↪nisl.
                                                                    ↪␣
                                                                    ↪Interdum␣
                                                                    ↪et␣
                                                                    ↪malesuada␣
                                                                    ↪fames␣
                                                                    ↪ac ante␣
                                                                    ↪ipsum␣
                                                                    ↪primis
```

(continues on next page)

(продолжение с предыдущей страницы)

```
14                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪faucibus.
                                                                    ↪␣
                                                                    ↪Nulla␣
                                                                    ↪bibendum␣
                                                                    ↪suscipit␣
                                                                    ↪convallis.
                                                                    ↪'''},
15                                                                      {
                                                                    ↪'id': 2,
                                                                    ↪ 'title
                                                                    ↪':␣
                                                                    ↪'Hello
                                                                    ↪',␣
                                                                    ↪'content
                                                                    ↪':␣
                                                                    ↪'Test2
                                                                    ↪'},
16                                                                      {
                                                                    ↪'id': 3,
                                                                    ↪ 'title
                                                                    ↪':␣
                                                                    ↪'World
                                                                    ↪',␣
                                                                    ↪'content
                                                                    ↪':␣
                                                                    ↪'Test2
                                                                    ↪'}, ]
```

Главная страница формируется перебором статей в списке:

```
1                                                                   class␣
                                                                    ↪BlogIndex(BaseBlog):
2
3                                                                       def␣
                                                                    ↪__iter__(self):
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        self.start('200 OK', [('Content-Type', 'text/html')])
        yield b'<h1>Simple Blog</h1>'
        for article in ARTICLES:
            yield str.encode(
                '''
                {0} - <a href="/article/{0}">{1}</a>
```

(continues on next page)

(продолжение с предыдущей страницы)

```
(<a href="/article/{0}/delete">delete</a>)<br/>
'''.format(
article['id'],
article['title']
)
)
```

WSGI-приложения BlogRead, BlogUpdate и BlogDelete теперь наследуются от специ-

альноклассаBaseArticle, онберетidстатьи (переменная окружения, которую добавляет *middlware* RegexDispatch) и находит ее среди списка данных.

```
class␣
↪BaseArticle(BaseBlog)

    def␣
↪__init_
↪_(self,
↪ *args):
␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪super(BaseArticle,
↪ self).
↪__init_
↪_(*args)
       ␣
↪article_
↪id␣
↪= self.
↪environ[
↪'url_
↪params
↪']['id']
     ␣
↪  (self.
↪index,
␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪self.
↪article)␣
↪=␣
↪next(((i
```

(continues on next page)

(продолжение с предыдущей страницы)

```
if art['id'] == int(article_id)),
(None, None))
```

Приложение `BlogRead`, отвечающее за чтение статьи, выводит его содержимое или отдает *404 ошибку*:

```
class BlogRead(BaseArticle)
```

(continues on next page)

(продолжение с предыдущей страницы)

```

    def __iter__(self):
        if not self.article:
            self.start('404 Not Found', [('content-type', 'text/plain')])
            yield b'not found'
            return

        self.start('200 OK
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            yield b'<h1><a href="/">Simple Blog</a> -> READ</h1>'
        yield str.encode('<h2>{}</h2>'.format(self.article['title']))
        yield str.encode(self.article['content'])
```

(continues on next page)

(продолжение с предыдущей страницы)

Приложение, удаляющее статью — `BlogDelete`, удаляет объект из списка данных и возвращает статус ответа *302 Fount* с заголовком *Location: /*, указывающий браузеру, что нужно перейти на главную страницу (перенаправление).

```
class␣
↪BlogDelete(BaseArticl

    def␣
↪__iter_
↪_(self):
␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪self.
↪start(
↪'302␣
↪Found
↪',
↪␣
↪␣
↪
↪#␣
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                                        [('Content-Type', 'text/html'),
                                         ('Location', '/')])
```

(continues on next page)

(продолжение с предыдущей страницы)

```
␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪ARTICLES.
↪pop(self.
↪index)
    ␣
↪  yield␣
↪b''
```

Полный код с изменениями:

```
from␣
↪middlewares.
↪urldispatch␣
↪import␣
↪RegexDispatch

ARTICLES␣
↪= [
    {
↪'id': 1,
↪ 'title
↪':␣
↪'Lorem␣
↪ipsum␣
↪dolor␣
↪sid␣
↪amet!',
␣
↪␣
↪␣
↪␣
↪
↪'content
↪':␣
↪
↪'
↪'
↪'Lorem␣
↪ipsum␣
↪dolor␣
↪sit␣
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    Curabitur vel tortor eleifend, sollicitudin nisl quis, lacinia augue. Duis
    quam est, laoreet sit amet justo vitae, viverra egestas sem.
    Maecenas pellentesque augue in nibh feugiat tincidunt. Nunc magna ante,
    mollis vitae ultricies eu,
```

(continues on next page)

(продолжение с предыдущей страницы)

```
10                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪blandit␣
                                                                    ↪ipsum␣
                                                                    ↪a,
                                                                    ↪␣
                                                                    ↪ullamcorper␣
                                                                    ↪nunc.
                                                                    ↪␣
                                                                    ↪Sed␣
                                                                    ↪bibendum␣
                                                                    ↪eget␣
                                                                    ↪odio␣
                                                                    ↪eget
11                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪pellentesque.
                                                                    ↪␣
                                                                    ↪Curabitur␣
                                                                    ↪elit␣
                                                                    ↪felis,
                                                                    ↪␣
                                                                    ↪pellentesque␣
                                                                    ↪id␣
                                                                    ↪feugiat␣
                                                                    ↪et,
                                                                    ↪␣
                                                                    ↪tincidunt
12                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪ut␣
                                                                    ↪mauris.
                                                                    ↪␣
                                                                    ↪Integer␣
                                                                    ↪vitae␣
                                                                    ↪vehicula␣
                                                                    ↪nunc.
                                                                    ↪␣
                                                                    ↪Integer␣
                                                                    ↪ullamcorper,
                                                                    ↪ nunc in
```

(continues on next page)

(продолжение с предыдущей страницы)

```
13                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪volutpat␣
                                                                    ↪auctor,
                                                                    ↪␣
                                                                    ↪elit␣
                                                                    ↪leo␣
                                                                    ↪convallis␣
                                                                    ↪nulla,
                                                                    ↪ vitae␣
                                                                    ↪varius␣
                                                                    ↪mi␣
                                                                    ↪nisl ac␣
                                                                    ↪lorem.
14                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪Sed␣
                                                                    ↪a␣
                                                                    ↪lacus␣
                                                                    ↪mi.
                                                                    ↪␣
                                                                    ↪In␣
                                                                    ↪hac␣
                                                                    ↪habitasse␣
                                                                    ↪platea␣
                                                                    ↪dictumst.
                                                                    ↪␣
                                                                    ↪Cras in␣
                                                                    ↪posuere␣
                                                                    ↪velit,
15                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪id␣
                                                                    ↪dignissim␣
                                                                    ↪nisl.
                                                                    ↪␣
                                                                    ↪Interdum␣
                                                                    ↪et␣
                                                                    ↪malesuada␣
                                                                    ↪fames␣
                                                                    ↪ac ante␣
                                                                    ↪ipsum␣
                                                                    ↪primis
```

(continues on next page)

(продолжение с предыдущей страницы)

```
16                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪faucibus.
                                                                    ↪␣
                                                                    ↪Nulla␣
                                                                    ↪bibendum␣
                                                                    ↪suscipit␣
                                                                    ↪convallis.
                                                                    ↪'''},
17                                                                      {
                                                                    ↪'id': 2,
                                                                    ↪ 'title
                                                                    ↪':␣
                                                                    ↪'Hello
                                                                    ↪',␣
                                                                    ↪'content
                                                                    ↪':␣
                                                                    ↪'Test2
                                                                    ↪'},
18                                                                      {
                                                                    ↪'id': 3,
                                                                    ↪ 'title
                                                                    ↪':␣
                                                                    ↪'World
                                                                    ↪',␣
                                                                    ↪'content
                                                                    ↪':␣
                                                                    ↪'Test2
                                                                    ↪'}, ]
19
20
21                                                                  class␣
                                                                    ↪BaseBlog(object):
22
23                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪def␣
                                                                    ↪_
                                                                    ↪_
                                                                    ↪init_
                                                                    ↪_
                                                                    ↪(self,
                                                                    ↪␣
                                                                    ↪environ,
                                                                    ↪␣
                                                                    ↪start_
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        self.environ = environ
        self.start = start_response


class BaseArticle(BaseBlog):

    def __init__(self, *args):
        super(BaseArticle, self).__init__(*args)
        article_id = self.environ['url_params']['id']
        (self.index,
         self
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                    if art['id'] == int(article_id)),
                    (None, None))


class BlogIndex(BaseBlog):

    def __iter__(self):
        self.start('200 OK',
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        yield b'<h1>Simple Blog</h1>'
        for article in ARTICLES:
            yield str.encode(
                '''
                {0} - <a href="/article/{0}">{1}</a>
                (<a href="/article/{0}/delete">delete</a>)<br/>
                '''.format(
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                                   article['id'],
                                   article['title']
                               )
                           )


class BlogCreate(BaseBlog):

    def __iter__(self):
        self.start('200 OK', [('Content-Type', 'text/plain')])
        yield b'Simple Blog -> CREATE'
```

(continues on next page)

(продолжение с предыдущей страницы)

```


class BlogRead(BaseArticle):

    def __iter__(self):
        if not self.article:
            self.start('404 Not Found', [('content-type', 'text/plain')])
            yield b'not found'
            return

```

(continues on next page)

(продолжение с предыдущей страницы)

```
72                                                                  ␣
                                                                ↪  yield␣
                                                                ↪b'<h1>
                                                                ↪<a href=
                                                                ↪"/">
                                                                ↪Simple␣
                                                                ↪Blog</a>
                                                                ↪ -> READ
                                                                ↪</h1>'
73                                                              ␣
                                                                ↪␣
                                                                ↪␣
                                                                ↪␣
                                                                ↪␣
                                                                ↪␣
                                                                ↪␣
                                                                ↪␣
                                                                ↪yield␣
                                                                ↪str.
                                                                ↪encode(
                                                                ↪'
                                                                ↪<h2>
                                                                ↪
                                                                ↪{}
                                                                ↪
                                                                ↪</
                                                                ↪h2>
                                                                ↪
                                                                ↪'.
                                                                ↪format(self.
                                                                ↪article[
                                                                ↪'title
↪']))
74                                                              ␣
                                                                ↪␣
                                                                ↪␣
                                                                ↪␣
                                                                ↪␣
                                                                ↪␣
                                                                ↪␣
                                                                ↪␣
                                                                ↪yield␣
                                                                ↪str.
                                                                ↪encode(self.
                                                                ↪article[
                                                                ↪'content
                                                                ↪'])
```

(continues on next page)

(продолжение с предыдущей страницы)

```
75
76
77                                                                  class␣
                                                                    ↪BlogUpdate(BaseArticl
78
79                                                                      def␣
                                                                    ↪__iter_
                                                                    ↪_(self):
80                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪self.
                                                                    ↪start(
                                                                    ↪'200␣
                                                                    ↪OK
                                                                    ↪',
                                                                    ↪␣
                                                                    ↪[(
                                                                    ↪'Content-
                                                                    ↪Type',
                                                                    ↪ 'text/
                                                                    ↪plain
                                                                    ↪')])
81                                                                      ␣
                                                                    ↪ yield b
                                                                    ↪'Simple␣
                                                                    ↪Blog ->
                                                                    ↪ UPDATE'
82
83
84                                                                  class␣
                                                                    ↪BlogDelete(BaseArticl
85
86                                                                      def␣
                                                                    ↪__iter_
                                                                    ↪_(self):
87                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪self.
                                                                    ↪start(
                                                                    ↪'302
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                   [('Content-Type', 'text/html'),
                   ('Location', '/')])
```

(continues on next page)

(продолжение с предыдущей страницы)

```
90                                                                 ␣
                                                                   ↪␣
                                                                   ↪␣
                                                                   ↪␣
                                                                   ↪␣
                                                                   ↪␣
                                                                   ↪␣
                                                                   ↪␣
                                                                   ↪ARTICLES.
                                                                   ↪pop(self.
                                                                   ↪index)
91                                                                     ␣
                                                                   ↪  yield␣
                                                                   ↪b''
92
93
94                                                                 ↪#␣
                                                                   ↪URL␣
                                                                   ↪dispatching␣
                                                                   ↪middleware
95                                                                 app_
                                                                   ↪list = [
96                                                                 ␣
                                                                   ↪␣
                                                                   ↪␣
                                                                   ↪␣
                                                                   ↪(
                                                                   ↪'/
                                                                   ↪
                                                                   ↪',
                                                                   ↪␣
                                                                   ↪BlogIndex),
                                                                   ↪
97                                                                 ␣
                                                                   ↪␣
                                                                   ↪␣
                                                                   ↪␣
                                                                   ↪(
                                                                   ↪'/
                                                                   ↪article/
                                                                   ↪add
                                                                   ↪',
                                                                   ↪␣
                                                                   ↪BlogCreate),
                                                                   ↪
```

(continues on next page)

(продолжение с предыдущей страницы)

```
98                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪(r
                                                                    ↪'^
                                                                    ↪/
                                                                    ↪article/
                                                                    ↪(?
                                                                    ↪P
                                                                    ↪<id>
                                                                    ↪\d+)/
                                                                    ↪
                                                                    ↪$
                                                                    ↪',
                                                                    ↪␣
                                                                    ↪BlogRead),
                                                                    ↪
99                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪(r
                                                                    ↪'^
                                                                    ↪/
                                                                    ↪article/
                                                                    ↪(?
                                                                    ↪P
                                                                    ↪<id>
                                                                    ↪\d+)/
                                                                    ↪edit/
                                                                    ↪
                                                                    ↪$
                                                                    ↪',
                                                                    ↪␣
                                                                    ↪BlogUpdate),
                                                                    ↪
100                                                                 ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪(r
                                                                    ↪'^
                                                                    ↪/
                                                                    ↪article/
                                                                    ↪(?
                                                                    ↪P
                                                                    ↪<id>
                                                                    ↪\d+)/
                                                                    ↪delete/
                                                                    ↪
                                                                    ↪$
```

(continues on next page)

(продолжение с предыдущей страницы)

```
101                                                                 ]
102                                                                 dispatch␣
                                                                    ↪=␣
                                                                    ↪RegexDispatch(app_
                                                                    ↪list)
103
104                                                                 if __name_
                                                                    ↪_ == '__
                                                                    ↪main__':
105                                                                 ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪from␣
                                                                    ↪paste.
                                                                    ↪httpserver␣
                                                                    ↪import␣
                                                                    ↪serve
106                                                                 ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪serve(dispatch,
                                                                    ↪␣
                                                                    ↪host=
                                                                    ↪'0.
                                                                    ↪0.
                                                                    ↪0.
                                                                    ↪0
                                                                    ↪',
                                                                    ↪␣
                                                                    ↪port=8000)
```

Формы

Для создания статьи требуется *HTML* фор-

ма, где указываются заголовок и содержание. В *WSGI* приложении `BlogCreate`, запрос с методом *GET* возвращает *HTML* форму, а *POST* записывает данные в список `ARTICLES`, после чего перенаправляет на главную страницу.

```
class BlogCreate(BaseBlog):

    def __iter__(self):
        if self.environ['REQUEST_METHOD'].upper() == 'POST':
            from urllib.parse import parse_qs
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        values = parse_qs(self.environ['wsgi.input'].read())
        max_id = max([art['id'] for art in ARTICLES])
        ARTICLES.append(
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                {'id': max_id+1,
                 'title': values[b'title'].pop().decode(),
                 'content': values[b'content'].pop().decode()
                }
            )
            self.start('302 Found',
```

(continues on next page)

(продолжение с предыдущей страницы)

```
[('Content-Type', 'text/html'),
('Location', '/')])
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        return

        self.start('200 OK', [('Content-Type', 'text/html')])
        yield b'<h1><a href="/">Simple Blog</a> -> CREATE</h1>'
        yield b'''
<form action="" method="POST">
    Title:<br>
    <input type="text" name="title"><br>
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    Content: <br>
                <textarea name="content"></textarea><br><br>
     <input type="submit" value="Submit">
</form>'''
```

Обновление статей происходит схожим образом, за исключением того, что в форму подставляются уже существующие значения и вместо добавления нового объекта в список

ARTICLES, обновляется уже существующий.

```
class BlogUpdate(BaseArticl

    def __iter__(self):
        if self.environ['REQUEST_METHOD'].upper() == 'POST':
            from urllib.parse import parse_qs
            values = parse_qs(self.environ['wsgi.input'].read())
            self.article['title'] = values[b'title'].pop().decode()
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            self.article['content'] = values[b'content'].pop().decode()
            self.start('302 Found',
                       [('Content-Type', 'text/html'),
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        return
    self.start('200 OK', [('Content-Type', 'text/html')])
    yield b'<h1><a href="/">Simple Blog</a> -> UPDATE</h1>'
    yield str.encode(
        '''
        <form action="" method="POST">
            Title:<br>
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                <input type="text" name="title" value="{0}"><br>
                Content:<br>
                <textarea name="content">{1}</textarea><br><br>
                <input type="submit" value="Submit">
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            </form>
        '''.format(
            self.article['title'],
            self.article['content']
        )
    )
```

Полный код с изменениями:

```
# third-party
from middlewares.urldispatch import RegexDispatch

ARTICLES = [
    {'id': 1, 'title': 'Lorem ipsum dolor sid amet!',
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        'content': '''Lorem ipsum dolor sit amet, consectetur adipiscing elit.
        Curabitur vel tortor eleifend, sollicitudin nisl quis, lacinia augue. Duis
quam est, laoreet sit amet justo vitae, viverra egestas sem.
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                    Maecenas pellentesque augue in nibh feugiat tincidunt.  Nunc magna ante,
                    mollis vitae ultricies eu,  consectetur id ante.  In ut libero eleifend,
                    blandit ipsum a,  ullamcorper nunc.  Sed bibendum eget odio eget
```

(continues on next page)

(продолжение с предыдущей страницы)

```
12                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪pellentesque.
                                                                    ↪␣
                                                                    ↪Curabitur␣
                                                                    ↪elit␣
                                                                    ↪felis,
                                                                    ↪␣
                                                                    ↪pellentesque␣
                                                                    ↪id␣
                                                                    ↪feugiat␣
                                                                    ↪et,
                                                                    ↪␣
                                                                    ↪tincidunt
13                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪ut␣
                                                                    ↪mauris.
                                                                    ↪␣
                                                                    ↪Integer␣
                                                                    ↪vitae␣
                                                                    ↪vehicula␣
                                                                    ↪nunc.
                                                                    ↪␣
                                                                    ↪Integer␣
                                                                    ↪ullamcorper,
                                                                    ↪ nunc in
14                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪volutpat␣
                                                                    ↪auctor,
                                                                    ↪␣
                                                                    ↪elit␣
                                                                    ↪leo␣
                                                                    ↪convallis␣
                                                                    ↪nulla,
                                                                    ↪ vitae␣
                                                                    ↪varius␣
                                                                    ↪mi␣
                                                                    ↪nisl ac␣
                                                                    ↪lorem.
```

(continues on next page)

(продолжение с предыдущей страницы)

```
15                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪Sed␣
                                                                    ↪a␣
                                                                    ↪lacus␣
                                                                    ↪mi.
                                                                    ↪␣
                                                                    ↪In␣
                                                                    ↪hac␣
                                                                    ↪habitasse␣
                                                                    ↪platea␣
                                                                    ↪dictumst.
                                                                    ↪␣
                                                                    ↪Cras in␣
                                                                    ↪posuere␣
                                                                    ↪velit,
16                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪id␣
                                                                    ↪dignissim␣
                                                                    ↪nisl.
                                                                    ↪␣
                                                                    ↪Interdum␣
                                                                    ↪et␣
                                                                    ↪malesuada␣
                                                                    ↪fames␣
                                                                    ↪ac ante␣
                                                                    ↪ipsum␣
                                                                    ↪primis␣
                                                                    ↪in
17                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪faucibus.
                                                                    ↪␣
                                                                    ↪Nulla␣
                                                                    ↪bibendum␣
                                                                    ↪suscipit␣
                                                                    ↪convallis.
                                                                    ↪'''},
```

(continues on next page)

(продолжение с предыдущей страницы)

```
18                                                                  {
                                                                ↪'id': 2,
                                                                ↪ 'title
                                                                ↪':␣
                                                                ↪'Hello
                                                                ↪',␣
                                                                ↪'content
                                                                ↪':␣
                                                                ↪'Test2
                                                                ↪'},
19                                                                  {
                                                                ↪'id': 3,
                                                                ↪ 'title
                                                                ↪':␣
                                                                ↪'World
                                                                ↪',␣
                                                                ↪'content
                                                                ↪':␣
                                                                ↪'Test2
                                                                ↪'}, ]
20
21
22                                                              class␣
                                                                ↪BaseBlog(object):
23
24                                                              ␣
                                                                ↪␣
                                                                ↪␣
                                                                ↪␣
                                                                ↪def␣
                                                                ↪_
                                                                ↪_
                                                                ↪init_
                                                                ↪_
                                                                ↪(self,
                                                                ↪␣
                                                                ↪environ,
                                                                ↪␣
                                                                ↪start_
                                                                ↪response):
25                                                                  ␣
                                                                ↪   self.
                                                                ↪environ␣
                                                                ↪=␣
                                                                ↪environ
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        self.start = start_response


class BaseArticle(BaseBlog):

    def __init__(self, *args):
        super(BaseArticle, self).__init__(*args)
        article_id = self.environ['url_params']['id']
        (self.index,
         self.article) = next(((i, art)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                                    if art['id'] == int(article_id)),
                                   (None, None))


class BlogIndex(BaseBlog):

    def __iter__(self):
        self.start('200 OK',
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        yield b'<h1>Simple Blog</h1>'
        yield b'<a href="/article/add">Add article</a>'
        yield b'<br />'
        yield b'<br />'
        for article in ARTICLES:
            yield str.encode(
                '''
                {0} - (<a href="/article/{0}/delete">delete</a> |
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                    <a href="/article/{0}/edit">edit</a>)
                    <a href="/article/{0}">{1}</a><br />
                '''.format(
                    article['id'],
                    article['title']
                )
            )


class BlogCreate(BaseBlog):

    def __iter__(self):
        if
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            from urllib.parse import parse_qs
            values = parse_qs(self.environ['wsgi.input'].read())
            max_id = max([art['id'] for art in ARTICLES])
            ARTICLES.append(
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                {'id': max_id+1,
                 'title': values[b'title'].pop().decode(),
                 'content': values[b'content'].pop().decode()
                }
            )
            self.start('302 Found',
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                                                                    ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪ [(
↪'Content-Type', 'text/html'),
                                                                    ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪  (
↪'Location', '/')])
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            return

        self.start('200 OK', [('Content-Type', 'text/html')])
        yield b'<h1><a href="/">Simple Blog</a> -> CREATE</h1>'
        yield b'''
<form action="" method="POST">
    Title:<br>
    <input type="text" name="title"><br>
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    Content: <br>
    <textarea name="content"></textarea><br><br>
    <input type="submit" value="Submit">
</form>'''


class BlogRead(BaseArticle):

    def __iter__(self):
        if not self.article:
            self.start('404 Not
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                yield b'not found'
                return

            self.start('200 OK', [('Content-Type', 'text/html')])
            yield b'<h1><a href="/">Simple Blog</a> -> READ</h1>'
            yield str.encode('<h2>{}</
```

(continues on next page)

(продолжение с предыдущей страницы)

```
102                                                                 ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪yield␣
                                                                    ↪str.
                                                                    ↪encode(self.
                                                                    ↪article[
                                                                    ↪'content
                                                                    ↪'])
103
104
105                                                                 class␣
                                                                    ↪BlogUpdate(BaseArticl
106
107                                                                     def␣
                                                                    ↪__iter_
                                                                    ↪_(self):
108                                                                 ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪if␣
                                                                    ↪self.
                                                                    ↪environ[
                                                                    ↪'REQUEST_
                                                                    ↪METHOD
                                                                    ↪'].
                                                                    ↪upper()␣
                                                                    ↪==␣
                                                                    ↪'POST':
109                                                                             ␣
                                                                    ↪    from␣
                                                                    ↪urllib.
                                                                    ↪parse␣
                                                                    ↪import␣
                                                                    ↪parse_qs
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            values = parse_qs(self.environ['wsgi.input'].read())
            self.article['title'] = values[b'title'].pop().decode()
            self.article['content'] = values[b'content'].pop().decode()
            self.start('302 Found',
                       
```

(continues on next page)

(продолжение с предыдущей страницы)

```
115                                                                 ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪  (
↪'Location', '/')])
116                                                                             ␣
                                                                    ↪  return
117                                                                 ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪self.
                                                                    ↪start(
                                                                    ↪'200␣
                                                                    ↪OK
                                                                    ↪',
                                                                    ↪␣
                                                                    ↪[(
                                                                    ↪'Content-
                                                                    ↪Type',
                                                                    ↪ 'text/
                                                                    ↪html')])
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                yield b'<h1><a href="/">Simple Blog</a> -> UPDATE</h1>'
                yield str.encode('''
                    <form action="" method="POST">
                        Title: <br>
                        <input type="text" name="title" value="{0}"><br>
                        Content: <br>
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                <input type="submit" value="Submit">
            </form>
        '''.format(
            self.article['title'],
            self.article['content']
        )
    )


class BlogDelete(BaseArticl

    def __iter__(self):
        self.start('302 Found',
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                          [('Content-Type', 'text/html'),
                           ('Location', '/')])
```

(continues on next page)

(продолжение с предыдущей страницы)

```
141                                                                 ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪ARTICLES.
                                                                    ↪pop(self.
                                                                    ↪index)
142                                                                     ␣
                                                                    ↪  yield␣
                                                                    ↪b''
143
144
145                                                                 ↪#␣
                                                                    ↪URL␣
                                                                    ↪dispatching␣
                                                                    ↪middleware
146                                                                 app_
                                                                    ↪list = [
147                                                                 ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪(
                                                                    ↪'/
                                                                    ↪
                                                                    ↪',
                                                                    ↪␣
                                                                    ↪BlogIndex),
                                                                    ↪
148                                                                 ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪(
                                                                    ↪'/
                                                                    ↪article/
                                                                    ↪add
                                                                    ↪',
                                                                    ↪␣
                                                                    ↪BlogCreate),
                                                                    ↪
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    (r'^/article/(?P<id>\d+)/$', BlogRead),
    (r'^/article/(?P<id>\d+)/edit/$', BlogUpdate),
    (r'^/article/(?P<id>\d+)/delete/$
```

(continues on next page)

(продолжение с предыдущей страницы)

```
152                                                                 ]
153                                                                 dispatch␣
                                                                    ↪=␣
                                                                    ↪RegexDispatch(app_
                                                                    ↪list)
154
155                                                                 if __name_
                                                                    ↪_ == '__
                                                                    ↪main__':
156                                                                 ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪from␣
                                                                    ↪paste.
                                                                    ↪httpserver␣
                                                                    ↪import␣
                                                                    ↪serve
157                                                                 ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪serve(dispatch,
                                                                    ↪␣
                                                                    ↪host=
                                                                    ↪'0.
                                                                    ↪0.
                                                                    ↪0.
                                                                    ↪0
                                                                    ↪',
                                                                    ↪␣
                                                                    ↪port=8000)
```

Авторизация

См.также:

- http://pythonpaste.org/modules/auth.basic.html#module-paste.auth.basic

Авторизация поможет защитить ресурсы от

сторонних пользователей, в первую очередь это касается операций, которые изменяют данные (`BlogCreate`, `BlogUpdate`, `BlogDelete`). В эти *WSGI* приложения необходимо будет добавить проверку пользователя.

В нашем примере используется алгоритм `BasicAuth` и *WSGI-middleware* `middlewares.basicauth.BasicAuth`.

```
class BasicAuth(object):

```

(continues on next page)

(продолжение с предыдущей страницы)

```
    def __init__(self, app, users, realm='www'):
        self.app = app
        self.users = users
        self.realm = realm

    def __call__(self, environ, start_response):
        auth = environ.get('HTTP_AUTHORIZATION')
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        if not auth:
            return self.auth_required(environ, start_response)
        auth_type, enc_auth_info = auth.split(None, 1)
        assert auth_type.lower() == 'basic'
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        auth_info = base64.b64decode(enc_auth_info)
        username, password = auth_info.decode().split(':', 1)
        if self.users.get(username) != password:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                return self.auth_required(environ, start_response)
            environ['REMOTE_USER'] = username
        return self.app(environ, start_response)

    def auth_required(self, environ, start_response):
```

(continues on next page)

(продолжение с предыдущей страницы)

```
␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪status␣
↪=␣
↪
↪'401␣
↪Authorization␣
↪Required
↪'
      ␣
↪headers␣
↪= [
␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪(
↪'content-
↪type',
↪ 'text/
↪plain'),
␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪(
↪'WWW-
↪Authenticate
↪',
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                ]
        start_response(status, headers)
        return [b'authentication required']
```

Полный код с изменениями:

```
from middlewares.urldispatch import RegexDispatch
from middlewares.basicauth import BasicAuth

ARTICLES = [
    {'id': 1, 'title': 'Lorem ipsum dolor sid
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        'content': '' 'Lorem ipsum dolor sit amet, consectetur adipiscing elit.
        Curabitur vel tortor eleifend, sollicitudin nisl quis, lacinia augue.
        Duis quam est, laoreet sit amet justo vitae, viverra egestas sem.
```

(continues on next page)

(продолжение с предыдущей страницы)

```
     Maecenas pellentesque augue in nibh feugiat tincidunt. Nunc magna ante,
     mollis vitae ultricies eu,  consectetur id ante.  In ut libero eleifend,
     blandit ipsum a,  ullamcorper nunc.  Sed bibendum eget odio eget
```

(continues on next page)

(продолжение с предыдущей страницы)

```
␣
↪␣
↪␣
↪␣
↪␣
↪pellentesque.
↪␣
↪Curabitur␣
↪elit␣
↪felis,
↪␣
↪pellentesque␣
↪id␣
↪feugiat␣
↪et,
↪␣
↪tincidunt
␣
↪␣
↪␣
↪␣
↪␣
↪ut␣
↪mauris.
↪␣
↪Integer␣
↪vitae␣
↪vehicula␣
↪nunc.
↪␣
↪Integer␣
↪ullamcorper,
↪ nunc in
␣
↪␣
↪␣
↪␣
↪␣
↪volutpat␣
↪auctor,
↪␣
↪elit␣
↪leo␣
↪convallis␣
↪nulla,
↪ vitae␣
↪varius␣
↪mi␣
↪nisl ac␣
↪lorem.
```

(continues on next page)

(продолжение с предыдущей страницы)

```
15                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪Sed␣
                                                                    ↪a␣
                                                                    ↪lacus␣
                                                                    ↪mi.
                                                                    ↪␣
                                                                    ↪In␣
                                                                    ↪hac␣
                                                                    ↪habitasse␣
                                                                    ↪platea␣
                                                                    ↪dictumst.
                                                                    ↪␣
                                                                    ↪Cras in␣
                                                                    ↪posuere␣
                                                                    ↪velit,
16                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪id␣
                                                                    ↪dignissim␣
                                                                    ↪nisl.
                                                                    ↪␣
                                                                    ↪Interdum␣
                                                                    ↪et␣
                                                                    ↪malesuada␣
                                                                    ↪fames␣
                                                                    ↪ac ante␣
                                                                    ↪ipsum␣
                                                                    ↪primis␣
                                                                    ↪in
17                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪faucibus.
                                                                    ↪␣
                                                                    ↪Nulla␣
                                                                    ↪bibendum␣
                                                                    ↪suscipit␣
                                                                    ↪convallis.
                                                                    ↪'''},
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    {'id': 2, 'title': 'Hello', 'content': 'Test2'},
    {'id': 3, 'title': 'World', 'content': 'Test2'}, ]


class BaseBlog(object):

    def __init__(self, environ, start_response):
        self.environ = environ
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        self.start = start_response


class BaseArticle(BaseBlog):

    def __init__(self, *args):
        super(BaseArticle, self).__init__(*args)
        article_id = self.environ['url_params']['id']
        (self.index,
         self.article) = next(((i, art)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                                if art['id'] == int(article_id)),
                               (None, None))


class BlogIndex(BaseBlog):

    def __iter__(self):
        self.start('200 OK',
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        yield b'<h1>Simple Blog</h1>'
        yield b'<a href="/article/add">Add article</a>'
        yield b'<br />'
        yield b'<br />'
        for article in ARTICLES:
            yield str.encode(
                '''
                    {0} - (<a href="/article/{0}/delete">delete</a> |
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            <a href="/article/{0}/edit">edit</a>)
            <a href="/article/{0}">{1}</a><br />
        '''.format(
            article['id'],
            article['title']
        )
    )


class BlogCreate(BaseBlog):

    def __iter__(self):
        if self
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                from urllib.parse import parse_qs
                values = parse_qs(self.environ['wsgi.input'].read())
                max_id = max([art['id'] for art in ARTICLES])
                ARTICLES.append(
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                {'id': max_id+1,
                 'title': values[b'title'].pop().decode(),
                 'content': values[b'content'].pop().decode()
                }
            )
            self.start('302 Found',
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                                        [('Content-Type', 'text/html'),
                                         ('Location', '/')])
```

(continues on next page)

(продолжение с предыдущей страницы)

```
77                                                                  ␣
                                                                ↪  return
78
79                                                              ␣
                                                                ↪␣
                                                                ↪␣
                                                                ↪␣
                                                                ↪␣
                                                                ↪␣
                                                                ↪␣
                                                                ↪␣
                                                                ↪self.
                                                                ↪start(
                                                                ↪'200␣
                                                                ↪OK
                                                                ↪',
                                                                ↪␣
                                                                ↪[(
                                                                ↪'Content-
                                                                ↪Type',
                                                                ↪ 'text/
                                                                ↪html')])
80                                                                  ␣
                                                                ↪  yield␣
                                                                ↪b'<h1>
                                                                ↪<a href=
                                                                ↪"/">
                                                                ↪Simple␣
                                                                ↪Blog
                                                                ↪</a> -
                                                                ↪> CREATE
                                                                ↪</h1>'
81                                                                  ␣
                                                                ↪  yield␣
                                                                ↪b'''
82                                                                  ␣
                                                                ↪  <form␣
                                                                ↪action=
                                                                ↪""␣
                                                                ↪method=
                                                                ↪"POST">
83                                                                  ␣
                                                                ↪       ␣
                                                                ↪  Title:
                                                                ↪<br>
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            <input type="text" name="title"><br>
            Content:<br>
            <textarea name="content"></textarea><br><br>
            <input type="submit" value="Submit">
        </form>
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        '''


class BlogRead(BaseArticle):

    def __iter__(self):
        if not self.article:
            self.start('404 Not Found', [('content-type', 'text/plain')])
            yield b'not found'
            return

```

(continues on next page)

(продолжение с предыдущей страницы)

```
100                                                                 ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪self.
                                                                    ↪start(
                                                                    ↪'200␣
                                                                    ↪OK
                                                                    ↪',
                                                                    ↪␣
                                                                    ↪[(
                                                                    ↪'Content-
                                                                    ↪Type',
                                                                    ↪ 'text/
                                                                    ↪html')])
101                                                                     ␣
                                                                    ↪  yield␣
                                                                    ↪b'<h1>
                                                                    ↪<a href=
                                                                    ↪"/">
                                                                    ↪Simple␣
                                                                    ↪Blog</a>
                                                                    ↪ -> READ
                                                                    ↪</h1>'
102                                                                 ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪yield␣
                                                                    ↪str.
                                                                    ↪encode(
                                                                    ↪'
                                                                    ↪<h2>
                                                                    ↪
                                                                    ↪{}
                                                                    ↪
                                                                    ↪</
                                                                    ↪h2>
                                                                    ↪
                                                                    ↪'.
                                                                    ↪format(self.
                                                                    ↪article[
                                                                    ↪'title
    ']))
```

(continues on next page)

(продолжение с предыдущей страницы)

```
103                                                                 ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪yield␣
                                                                    ↪str.
                                                                    ↪encode(self.
                                                                    ↪article[
                                                                    ↪'content
                                                                    ↪'])
104
105
106                                                                 class␣
                                                                    ↪BlogUpdate(BaseArticl
107
108                                                                     def␣
                                                                    ↪__iter_
                                                                    ↪_(self):
109                                                                 ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪if␣
                                                                    ↪self.
                                                                    ↪environ[
                                                                    ↪'REQUEST_
                                                                    ↪METHOD
                                                                    ↪'].
                                                                    ↪upper()␣
                                                                    ↪==␣
                                                                    ↪'POST':
110                                                                         ␣
                                                                    ↪   from␣
                                                                    ↪urllib.
                                                                    ↪parse␣
                                                                    ↪import␣
                                                                    ↪parse_qs
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            values = parse_qs(self.environ['wsgi.input'].read())
            self.article['title'] = values[b'title'].pop().decode()
            self.article['content'] = values[b'content'].pop().decode()
            self.start('302 Found',
                       
```

(continues on next page)

(продолжение с предыдущей страницы)

```
116                                                                 ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪  (
↪'Location', '/')])
117                                                                             ␣
                                                                    ↪  return
118                                                                 ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪self.
                                                                    ↪start(
                                                                    ↪'200␣
                                                                    ↪OK
                                                                    ↪',
                                                                    ↪␣
                                                                    ↪[(
                                                                    ↪'Content-
                                                                    ↪Type',
                                                                    ↪  'text/
                                                                    ↪html')])
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        yield b'<h1><a href="/">Simple Blog</a> -> UPDATE</h1>'
        yield str.encode(
            '''
                <form action="" method="POST">
                    Title: <br>
                    <input type="text" name="title" value="{0}"><br>
                    Content: <br>

```

(continues on next page)

(продолжение с предыдущей страницы)

```
                ␣
↪       ␣
↪ <input␣
↪type=
↪"submit
↪" value=
↪"Submit
↪">
  ␣
↪       ␣
↪</form>
        ␣
↪   '''.
↪format(
  ␣
↪       ␣
↪   self.
↪article[
↪'title
↪'],
  ␣
↪       ␣
↪   self.
↪article[
↪'content
↪']
  ␣
↪       )
       )


class␣
↪BlogDelete(BaseArticl

   def␣
↪__iter_
↪_(self):
␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪self.
↪start(
↪'302␣
↪Found
↪',
↪␣
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                                        [('Content-Type', 'text/html'),
                                        ('Location', '/')])
```

(continues on next page)

(продолжение с предыдущей страницы)

```
142                                                                 ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪ARTICLES.
                                                                    ↪pop(self.
                                                                    ↪index)
143                                                                     ␣
                                                                    ↪  yield␣
                                                                    ↪b''
144
145
146                                                                 ↪#␣
                                                                    ↪BasicAuth␣
                                                                    ↪applications
                                                                    passwd =␣
                                                                    ↪{'admin
                                                                    ↪':␣
                                                                    ↪'123'}
147                                                                 create␣
                                                                    ↪=␣
                                                                    ↪BasicAuth(BlogCreate,
                                                                    ↪ passwd)
148                                                                 update␣
                                                                    ↪=␣
                                                                    ↪BasicAuth(BlogUpdate,
                                                                    ↪ passwd)
149                                                                 delete␣
                                                                    ↪=␣
                                                                    ↪BasicAuth(BlogDelete,
                                                                    ↪ passwd)
150
151
152                                                                 ↪#␣
                                                                    ↪URL␣
                                                                    ↪dispatching␣
                                                                    ↪middleware
153                                                                 app_
                                                                    ↪list = [
154                                                                 ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪(
                                                                    ↪'/
                                                                    ↪
```

(continues on next page)

(продолжение с предыдущей страницы)

```
155                                                     ('/
                                                    ↪article/
                                                    ↪add',␣
                                                    ↪create),
156                                                 ␣
                                                    ↪␣
                                                    ↪␣
                                                    ↪␣
                                                    ↪(r
                                                    ↪'^
                                                    ↪/
                                                    ↪article/
                                                    ↪(?
                                                    ↪P
                                                    ↪<id>
                                                    ↪\d+)/
                                                    ↪
                                                    ↪$
                                                    ↪',
                                                    ↪␣
                                                    ↪BlogRead),
                                                    ↪
157                                                     (r'^/
                                                    ↪article/
                                                    ↪(?P<id>
                                                    ↪\d+)/
                                                    ↪edit/
                                                    ↪$',␣
                                                    ↪update),
158                                                    (r'^/
                                                    ↪article/
                                                    ↪(?P<id>
                                                    ↪\d+)/
                                                    ↪delete/
                                                    ↪$',␣
                                                    ↪delete),
159                                                 ]
160                                                 dispatch␣
                                                    ↪=␣
                                                    ↪RegexDispatch(app_
                                                    ↪list)
161
162                                                 if __name_
                                                    ↪_ == '__
                                                    ↪main__':
```

(continues on next page)

(продолжение с предыдущей страницы)

```
163                                                                 ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪from␣
                                                                    ↪paste.
                                                                    ↪httpserver␣
                                                                    ↪import␣
                                                                    ↪serve
164                                                                 ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪serve(dispatch,
                                                                    ↪␣
                                                                    ↪host=
                                                                    ↪'0.
                                                                    ↪0.
                                                                    ↪0.
                                                                    ↪0
                                                                    ↪',
                                                                    ↪␣
                                                                    ↪port=8000)
```

1.4.2 Разделение кода

См.также:

- Сравнение скорости фреймворков и чистого WSGI

В программном обеспечении принято разделять: программную логику, код, который относится к данным, настройки

и шаблоны. Таким образом, код становится более структурированным, и его легче развивать, а также сопровождать в дальнейшем.

MVC

См.также:

- Статья о фреймворке Ruby on Rails
- Концепция MVC для чайников

MVC (Model-View-Controller: модель-вид-контроллер) — шаблон архитектуры ПО, который подразумевает разделение программы на 3 слабосвязанных компонента, каждый из которых отвечает за свою сферу деятельности.

Бешеная

популярность данной структуры в Веб-приложениях сложилась благодаря её включению в две среды разработки, которые стали очень востребованными: Struts и Ruby on Rails. Эти среды разработки наметили пути развития для сотен рабочих сред, созданных позже.

- **Model** - модель, предоставляющая доступ к данным. Позволяет извлекать данные и менять их состояние;
- **View** - представление, отоб-

ра-
жа-
ю-
щее
дан-
ные
кли-
ен-
ту.
В
веб-
программировании
суще-
ствует в виде конечных данных (*HTML*, *JSON*, ...), которые получает клиент. Может формироваться при помощи генераторов по заданному шаблону, например *Jinja2*, *Mako*; или систем для построения интерфейсов по разметке, таких, как *Windows Presentation Foundation* (WPF), либо *Qt Widgets*; или описываться декларативно, как это делается в *QML* и *ReactJs*.

- **Controller** - контроллер, отслеживающий различные события (действия пользователя) и по заданной логике оповещающий модель о необходимости изменить состояние системы.

  Классические *MVC* фреймворки:

- Ruby on Rails
- Pylons

MTV

Фреймворк Django ввел новую терминологию **MTV**.

Примечание: https://docs.djangoproject.com/en/dev/faq/general/#django-appears-to-be-a-mvc-framework-but-you-call-the-controller-the-view-and-the-view-the-template

В *Django* функции, отвечающие за обработку логики, соответствуют части *Controller* из MVC, но называются *View*, а отображение соответствует части *View* из MVC, но называется *Template*.

Получилось, что:

- **M -> M** Модели

остались неизменными

- **V -> T** Представление назвали *Templates*
- **C -> V** Контроллеры назвали *Views*

Так появилась аббревиатура *MTV*.

Вся логика при таком подходе вынесена во *View*, а то, как будут отображаться данные в *Template*. Из-за ограничений *HTTP* протокола, *View* в *Django* описывает, **какие данные** будут представленны по запросу на определенный URL. *View*, как и протокол *HTTP*, не хранит состояний и по факту является обычной функцией обратного вызова, которая

запускается вновь при каждом запросе по *URL*. *Шаблоны* (*Templates*), в свою очередь, описывают, **как данные** представить пользователю.

MTV фреймворки:

- Django

RV

См.также:

- RV Pyramid
- Что не так в терминологии MVC
- Pyramid wikipedia

В защиту своего дизайна авторы Pyramid написали довольно большой документ, который призван развеять мифы о фреймворке. Например, на критику модели *MVC* в *Pyramid* следует подробное объяснение, что *MVC* «притянут за уши» к веб-приложениям. Следующая цитата хорошо характеризует подход к терминологии в *Pyramid*:

«Мы считаем,

что есть только две вещи: **ресурсы** (*Resource*) и **виды** (*View*). Дерево ресурсов представляет структуру сайта, а вид представляет ресурс.

«Шаблоны» (*Template*) в реальности лишь деталь реализации некоторого вида: строго говоря, они не обязательны, и вид может вернуть ответ (*Response*) и без них.

Нет никакого **«контроллера»** (*Controller*): его просто

не существует.

«Модель» (*Model*) же либо представлена деревом ресурсов, либо «доменной моделью» (*domain model*) (например, моделью SQLAlchemy), которая вообще не является частью каркаса.

Нам кажется, что наша терминология более разумна при существу-

ющих ограничениях веб-технологий.»

Веб ограничен *URL*, который и представляет из себя дерево ресурсов или структуру сайта.

Также протокол *HTTP* не позволяет хранить состояние и отправлять/принимать оповещения клиенту от сервера, что ограничивает возможность отслеживания действий клиента для последующего уведомления модели на изменение состояния.

Поэтому

данные часто используются на «frontend»-е (например в связке *React/Redux*), а на стороне сервера формируются только один раз во время ответа, либо загружаются отдельным запросом при помощи *AJAX*, или даже с помощью других протоколов, например *WebSocket*.

RV фреймворки:

- Pyramid

Пример MVC блога

Структура файлов

Приведем структуру нашего блога к следующему виду:

```
.
├─┐
↪ __init__.py
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                                                                ├──␣
                                                                ↪models.
                                                                ↪py
                                                                └──␣
                                                                ↪views.py

                                                                0␣
                                                                ↪directories,
                                                                ↪ 3 files
```

Примечание: Исходный код доступен по адресу:

- https://github.com/iitwebdev/lectures_wsgi_example/tree/master/1.mvc

Где:

- `__init__.py` - входная точка программы, которая содержит основные настройки и запуск Веб-сервера
- `models.py` - код, который представляет данные, обычно называется *модели*
- `views.py` -

ло-
ги-
ка
про-
грам-
мы
(в
на-
шем
слу-
чае
*WSGI-
приложения*)

Предупреждение: Примеры работают только в Python3

Данные

`models.py`:

```
ARTICLES␣
↪= [
    {
        ␣
↪'id': 1,
     ␣
↪  'title
↪':␣
↪'Lorem␣
↪ipsum␣
↪dolor␣
↪sid␣
↪amet!',
        ␣
↪'content
↪': '''
␣
↪␣
↪␣
↪␣
↪Lorem␣
↪ipsum␣
↪dolor␣
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    Curabitur vel tortor eleifend, sollicitudin nisl quis, lacinia augue. Duis quam est, laoreet sit amet justo vitae, viverra egestas sem.
    Maecenas pellentesque augue in nibh feugiat tincidunt. Nunc magna ante,
    mollis vitae ultricies eu, consectetur
```

(continues on next page)

(продолжение с предыдущей страницы)

```
11                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪blandit␣
                                                                    ↪ipsum␣
                                                                    ↪a,
                                                                    ↪␣
                                                                    ↪ullamcorper␣
                                                                    ↪nunc.
                                                                    ↪␣
                                                                    ↪Sed␣
                                                                    ↪bibendum␣
                                                                    ↪eget␣
                                                                    ↪odio␣
                                                                    ↪eget
12                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪pellentesque.
                                                                    ↪␣
                                                                    ↪Curabitur␣
                                                                    ↪elit␣
                                                                    ↪felis,
                                                                    ↪␣
                                                                    ↪pellentesque␣
                                                                    ↪id␣
                                                                    ↪feugiat␣
                                                                    ↪et,
                                                                    ↪␣
                                                                    ↪tincidunt
13                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪ut␣
                                                                    ↪mauris.
                                                                    ↪␣
                                                                    ↪Integer␣
                                                                    ↪vitae␣
                                                                    ↪vehicula␣
                                                                    ↪nunc.
                                                                    ↪␣
                                                                    ↪Integer␣
                                                                    ↪ullamcorper,
                                                                    ↪ nunc in
```

(continues on next page)

(продолжение с предыдущей страницы)

```
14                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪volutpat␣
                                                                    ↪auctor,
                                                                    ↪␣
                                                                    ↪elit␣
                                                                    ↪leo␣
                                                                    ↪convallis␣
                                                                    ↪nulla,
                                                                    ↪ vitae␣
                                                                    ↪varius␣
                                                                    ↪mi␣
                                                                    ↪nisl ac␣
                                                                    ↪lorem.
15                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪Sed␣
                                                                    ↪a␣
                                                                    ↪lacus␣
                                                                    ↪mi.
                                                                    ↪␣
                                                                    ↪In␣
                                                                    ↪hac␣
                                                                    ↪habitasse␣
                                                                    ↪platea␣
                                                                    ↪dictumst.
                                                                    ↪␣
                                                                    ↪Cras in␣
                                                                    ↪posuere␣
                                                                    ↪velit,
16                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪id␣
                                                                    ↪dignissim␣
                                                                    ↪nisl.
                                                                    ↪␣
                                                                    ↪Interdum␣
                                                                    ↪et␣
                                                                    ↪malesuada␣
                                                                    ↪fames␣
                                                                    ↪ac ante␣
                                                                    ↪ipsum␣
                                                                    ↪primis␣
```

(continues on next page)

(продолжение с предыдущей страницы)

```
faucibus. Nulla bibendum suscipit convallis.
    '''
    },
    {
        'id': 2,
        'title': 'Hello',
        'content': 'Test2'
    },
    {
        'id': 3,
        'title': 'World',
        'content': 'Test2'
    },
]
```

Авторизация

Мы использовали самописные *WSGI-middleware*, которые решают стандартные задачи. Заменим их на уже существующие:

- selector - URL-диспетчеризация
- wsgi-basic-auth - авторизация по методу *Basic Auth*

Настройки авторизации `__init__.py`:

```
from
↪views
↪import
↪BlogRead,
↪
↪BlogIndex,
↪
↪BlogCreate,
↪
↪BlogDelete,
↪
↪BlogUpdate
```

(continues on next page)

(продолжение с предыдущей страницы)

```
from␣
↪wsgi_
↪basic_
↪auth␣
↪import␣
↪BasicAuth

# third-
↪party
import␣
↪selector


def make_
↪wsgi_
↪app():
  ␣
↪ passwd␣
↪= {
    ␣
↪  'admin
↪': '123'
    }
␣
↪␣
↪␣
↪␣
↪
↪#␣
↪BasicAuth␣
↪applications
␣
↪␣
↪␣
↪␣
↪create␣
↪=␣
↪BasicAuth(BlogCreate,
↪ 'www',
↪ passwd)
␣
↪␣
↪␣
↪␣
↪update␣
↪=␣
↪BasicAuth(BlogUpdate,
↪ 'www',
↪ passwd)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
15                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪delete␣
                                                                    ↪=␣
                                                                    ↪BasicAuth(BlogDelete,
                                                                    ↪ 'www',
                                                                    ↪ passwd)
16
17                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪
                                                                    ↪#␣
                                                                    ↪URL␣
                                                                    ↪dispatching␣
                                                                    ↪middleware
18                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪dispatch␣
                                                                    ↪=␣
                                                                    ↪selector.
                                                                    ↪Selector()
19                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪dispatch.
                                                                    ↪add(
                                                                    ↪'/
                                                                    ↪
                                                                    ↪',
                                                                    ↪␣
                                                                    ↪GET=BlogIndex)
20                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪dispatch.
                                                                    ↪prefix␣
                                                                    ↪= '/
                                                                    ↪article'
```

(continues on next page)

(продолжение с предыдущей страницы)

```
21                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪dispatch.
                                                                    ↪add(
                                                                    ↪'/
                                                                    ↪add
                                                                    ↪',
                                                                    ↪␣
                                                                    ↪GET=create,
                                                                    ↪␣
                                                                    ↪POST=create)
22                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪dispatch.
                                                                    ↪add(
                                                                    ↪'/
                                                                    ↪
                                                                    ↪{id:digits}
                                                                    ↪
                                                                    ↪',
                                                                    ↪␣
                                                                    ↪GET=BlogRead)
23                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪dispatch.
                                                                    ↪add(
                                                                    ↪'/
                                                                    ↪
                                                                    ↪{id:digits}
                                                                    ↪/
                                                                    ↪edit
                                                                    ↪',
                                                                    ↪␣
                                                                    ↪GET=update,
                                                                    ↪␣
                                                                    ↪POST=update)
24                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪dispatch.
                                                                    ↪add(
                                                                    ↪'/
                                                                    ↪
                                                                    ↪{id:digits}
                                                                    ↪/
```

(continues on next page)

(продолжение с предыдущей страницы)

```
25                                                                  ␣
                                                                   ↪ return␣
                                                                   ↪dispatch
26
27                                                                 if __name_
                                                                   ↪_ == '__
                                                                   ↪main__':
28                                                                 ␣
                                                                   ↪␣
                                                                   ↪␣
                                                                   ↪␣
                                                                   ↪from␣
                                                                   ↪paste.
                                                                   ↪httpserver␣
                                                                   ↪import␣
                                                                   ↪serve
29                                                                     app␣
                                                                   ↪= make_
                                                                   ↪wsgi_
                                                                   ↪app()
30                                                                 ␣
                                                                   ↪␣
                                                                   ↪␣
                                                                   ↪␣
                                                                   ↪serve(app,
                                                                   ↪␣
                                                                   ↪host=
                                                                   ↪'0.
                                                                   ↪0.
                                                                   ↪0.
                                                                   ↪0
                                                                   ↪',
                                                                   ↪␣
                                                                   ↪port=8000)
```

URL-диспетчеризация

Настройки URL-диспетчеризации `__init__.py`:

```
from␣
↪views␣
↪import␣
↪BlogRead,
↪␣
↪BlogIndex,
↪␣
↪BlogCreate,
↪␣
↪BlogDelete,
↪␣
↪BlogUpdate
from␣
↪wsgi_
↪basic_
↪auth␣
↪import␣
↪BasicAuth

# third-
↪party
import␣
↪selector


def make_
↪wsgi_
↪app():
  ␣
↪ passwd␣
↪= {
     ␣
↪  'admin
↪': '123'
    }
␣
↪␣
↪␣
↪␣
↪
↪#␣
↪BasicAuth␣
↪applications
␣
↪␣
↪␣
↪create␣
↪=␣
↪BasicAuth(BlogCreate,
↪ 'www',
↪ passwd)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
14                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪update␣
                                                                    ↪=␣
                                                                    ↪BasicAuth(BlogUpdate,
                                                                    ↪ 'www',
                                                                    ↪ passwd)
15                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪delete␣
                                                                    ↪=␣
                                                                    ↪BasicAuth(BlogDelete,
                                                                    ↪ 'www',
                                                                    ↪ passwd)
16
17                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪
                                                                    ↪#␣
                                                                    ↪URL␣
                                                                    ↪dispatching␣
                                                                    ↪middleware
18                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪dispatch␣
                                                                    ↪=␣
                                                                    ↪selector.
                                                                    ↪Selector()
19                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪dispatch.
                                                                    ↪add(
                                                                    ↪'/
                                                                    ↪
                                                                    ↪',
                                                                    ↪␣
                                                                    ↪GET=BlogIndex)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
20                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪dispatch.
                                                                    ↪prefix␣
                                                                    ↪= '/
                                                                    ↪article'
21                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪dispatch.
                                                                    ↪add(
                                                                    ↪'/
                                                                    ↪add
                                                                    ↪',
                                                                    ↪␣
                                                                    ↪GET=create,
                                                                    ↪␣
                                                                    ↪POST=create)
22                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪dispatch.
                                                                    ↪add(
                                                                    ↪'/
                                                                    ↪
                                                                    ↪{id:digits}
                                                                    ↪
                                                                    ↪',
                                                                    ↪␣
                                                                    ↪GET=BlogRead)
23                                                                  ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪dispatch.
                                                                    ↪add(
                                                                    ↪'/
                                                                    ↪
                                                                    ↪{id:digits}
                                                                    ↪/
                                                                    ↪edit
                                                                    ↪',
                                                                    ↪␣
                                                                    ↪GET=update,
                                                                    ↪␣
                                                                    ↪POST=update)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                                                                ␣
                                                                ↪␣
                                                                ↪␣
                                                                ↪␣
                                                                ↪dispatch.
                                                                ↪add(
                                                                ↪'/
                                                                ↪
                                                                ↪{id:digits}
                                                                ↪/
                                                                ↪delete
                                                                ↪',
                                                                ↪␣
                                                                ↪GET=delete)
                                                                  ␣
                                                                ↪ return␣
                                                                ↪dispatch


                                                                if __name_
                                                                ↪_ == '__
                                                                ↪main__':
                                                                ␣
                                                                ↪␣
                                                                ↪␣
                                                                ↪␣
                                                                ↪from␣
                                                                ↪paste.
                                                                ↪httpserver␣
                                                                ↪import␣
                                                                ↪serve
                                                                    app␣
                                                                ↪= make_
                                                                ↪wsgi_
                                                                ↪app()
                                                                ␣
                                                                ↪␣
                                                                ↪␣
                                                                ↪␣
                                                                ↪serve(app,
                                                                ↪␣
                                                                ↪host=
                                                                ↪'0.
                                                                ↪0.
                                                                ↪0.
                                                                ↪0
                                                                ↪'
                                                                ↪␣
                                                                ↪port=8000)
```

(continues on next page)

(продолжение с предыдущей страницы)

WSGI-приложение можно указывать как объект (`BlogRead`) или как строку импорта (`"views.BlogIndex"`).

`views.py`:

```
class BaseArticle(BaseBlog)

    def __init__(self, *args):
 
        super(BaseArticle, self).__init__(*args)
 
```

(continues on next page)

(продолжение с предыдущей страницы)

```
(self.index,
self.article) = next(((i, art) for i, art in enumerate(ARTICLES)
if art['id'] == int(article_id)),
```

(continues on next page)

(продолжение с предыдущей страницы)

```
(None, None))
```

urlrelay добавляет результат поиска в переменную с названием wsgiorg.routing_args.

WSGI-приложения

Практически не изменились.

views.py:

```
from models import ARTICLES
```

(continues on next page)

(продолжение с предыдущей страницы)

```
class␣
↪BaseBlog(object):
␣
↪␣
↪␣
↪␣
↪def␣
↪_
↪_
↪init_
↪_
↪(self,
↪␣
↪environ,
↪␣
↪start_
↪response):
   ␣
↪   self.
↪environ␣
↪=␣
↪environ
   ␣
↪   self.
↪start␣
↪= start_
↪response


class␣
↪BaseArticle(BaseBlog)

   def␣
↪__init_
↪_(self,
↪ *args):
␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪super(BaseArticle,
↪self).
↪__init_
↪_(*args)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                article_id = self.environ['wsgiorg.routing_args'][1]['id']
            (self.index,
                self.article) = next(((i, art) for i, art in enumerate(ARTICLES)
                
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                                             (None, None))


class BlogIndex(BaseBlog):

    def __iter__(self):
        self.start('200 OK', [('Content-Type', 'text/html')])
        yield b'<h1>Simple Blog</h1>'
        yield b'<a href="/article/add">Add article</a>'
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        yield b'<br />'
        yield b'<br />'
        for article in ARTICLES:
            yield str.encode(
                '''
                {0} - (<a href="/article/{0}/delete">delete</a> |
                <a href="/article/{0}/edit">edit</a>)
                <a href="/article/{0}">{1}</a><br />
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            '''.format(
                article['id'],
                article['title']
            )
        )


class BlogCreate(BaseBlog):

    def __iter__(self):
        if self.environ['REQUEST_METHOD'].upper() == 'POST':
            from urllib.parse import parse_qs
```

(continues on next page)

(продолжение с предыдущей страницы)

```
values = parse_qs(self.environ['wsgi.input'].read())
max_id = max([art['id'] for art in ARTICLES])
ARTICLES.append(
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                {'id': max_id+1,
                 'title': values[b'title'].pop().decode(),
                 'content': values[b'content'].pop().decode()
                }
            )
            self.start('302 Found',
```

(continues on next page)

(продолжение с предыдущей страницы)

```
[(
'Content-Type', 'text/html'),
(
'Location', '/')])
```

(continues on next page)

(продолжение с предыдущей страницы)

```
59                                                                  ␣
                                                                ↪  return
60
61                                                              ␣
                                                                ↪␣
                                                                ↪␣
                                                                ↪␣
                                                                ↪␣
                                                                ↪␣
                                                                ↪␣
                                                                ↪␣
                                                                ↪self.
                                                                ↪start(
                                                                ↪'200␣
                                                                ↪OK
                                                                ↪',
                                                                ↪␣
                                                                ↪[(
                                                                ↪'Content-
                                                                ↪Type',
                                                                ↪ 'text/
                                                                ↪html')])
62                                                                  ␣
                                                                ↪  yield␣
                                                                ↪b'<h1>
                                                                ↪<a href=
                                                                ↪"/">
                                                                ↪Simple␣
                                                                ↪Blog
                                                                ↪</a> -
                                                                ↪> CREATE
                                                                ↪</h1>'
63                                                                  ␣
                                                                ↪  yield␣
                                                                ↪b'''
64                                                                  ␣
                                                                ↪  <form␣
                                                                ↪action=
                                                                ↪""␣
                                                                ↪method=
                                                                ↪"POST">
65                                                                  ␣
                                                                ↪       ␣
                                                                ↪  Title:
                                                                ↪<br>
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            <input type="text" name="title"><br>
            Content:<br>
            <textarea name="content"></textarea><br><br>
            <input type="submit" value="Submit">
        </form>
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    '''


class BlogRead(BaseArticle)

    def __iter__(self):
        if not self.article:
            self.start('404 Not Found', [('content-type', 'text/plain')])
            yield b'not found'
            return

```

(continues on next page)

(продолжение с предыдущей страницы)

```
␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪self.
↪start(
↪'200␣
↪OK
↪',
↪␣
↪[(
↪'Content-
↪Type',
↪ 'text/
↪html')])
    ␣
↪  yield␣
↪b'<h1>
↪<a href=
↪"/">
↪Simple␣
↪Blog</a>
↪ -> READ
↪</h1>'
    ␣
↪  yield␣
↪str.
↪encode(
␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪␣
↪
↪'
↪<h2>
↪
↪{}
↪
↪</
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        )
        yield str.encode(self.article['content'])


class BlogUpdate(BaseArticl

    def __iter__(self):
        if self.environ['REQUEST_METHOD'].upper() == 'POST':
            from urllib.parse import parse_qs
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            values = parse_qs(self.environ['wsgi.input'].read())
            self.article['title'] = values[b'title'].pop().decode()
            self.article['content'] = values[b'content'].pop().decode()
            self.start('302 Found',
                       
```

(continues on next page)

(продолжение с предыдущей страницы)

```
('Location', '/')])
return
self.start('200 OK', [('Content-Type', 'text/html')])
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        yield b'<h1><a href="/">Simple Blog</a> -> UPDATE</h1>'
        yield str.encode(
            '''
                <form action="" method="POST">
                    Title:<br>
                    <input type="text" name="title" value="{0}"><br>
                    Content:<br>

```

(continues on next page)

(продолжение с предыдущей страницы)

```
<input type="submit" value="Submit">
</form>
'''.format(
self.article['title'],
self.article['content']
)
)


class BlogDelete(BaseArticl

def __iter__(self):
self.start('302 Found',
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                                                                                      [('Content-Type', 'text/html'),
                                                                                      ('Location', '/')])
```

(continues on next page)

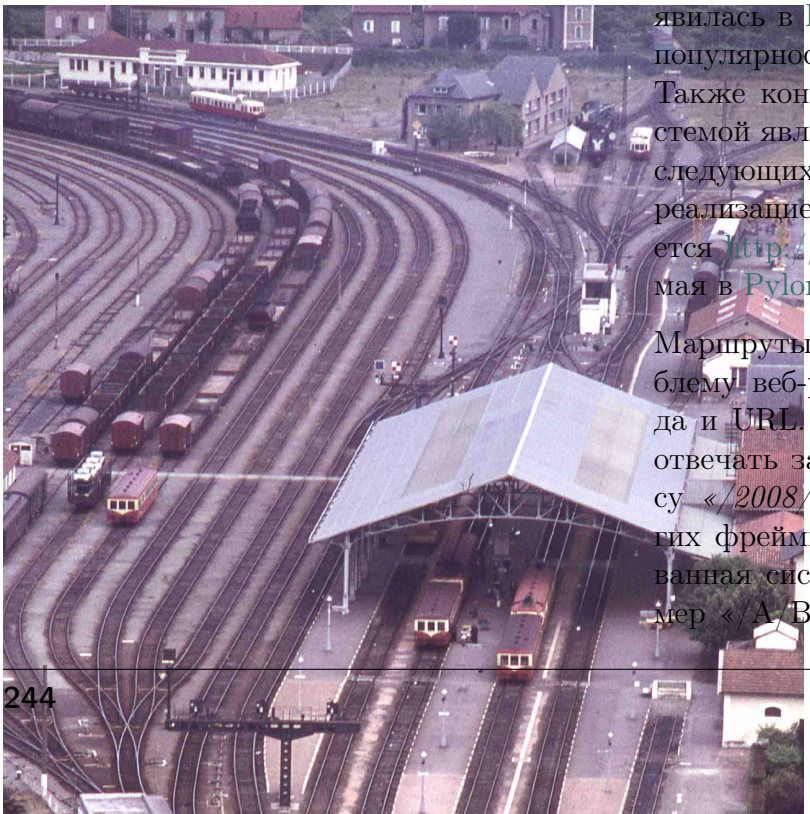
(продолжение с предыдущей страницы)

```
126                                                                 ␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪␣
                                                                    ↪ARTICLES.
                                                                    ↪pop(self.
                                                                    ↪index)
127                                                                     ␣
                                                                    ↪   yield␣
                                                                    ↪b''
```

1.4.3 Маршруты

См.также:

- http://restas.lisper.ru/ru/manual/routes.html
- Ryby on Rails -> Routes
- Python -> Routes
- https://ru.wikipedia.org/wiki/\T2A\CYRCH\T2A\CYRP\T2A\CYRU_(\T2A\CYRI\T2A\cyrn\T2A\cyrt\T2A\cyre\T2A\cyrr\T2A\cyrn\T2A\cyre\T2A\cyrt)



Идея маршрута (англ. - route) впервые появилась в Ruby on Rails и быстро обрела популярность в других веб-фреймворках. Также концептуально очень близкой системой является URLConf в Django. В последующих разработках наиболее мощной реализацией данной идеи, вероятно, является http://routes.groovie.org/, используемая в Pylons.

Маршруты отвечают за ключевую проблему веб-разработки: сопоставление кода и URL. Например, какой код должен отвечать за обработку запросов по адресу *«/2008/01/08»* или *«/login»*? Во многих фреймворках используется фиксированная система диспетчеризации, например «/A/B/C» означает прочитать файл

«C» в каталоге «B» (например `/auth/login.php` или `/cgi-bin/hello.cgi`), или вызвать метод «C» класса «B» в модуле «A».

Это работает прекрасно до тех пор, пока не возникает необходимости в реорганизации кода, и выясняется, что закладки пользователей стали недействительны. Кроме того, если вы хотите переделать адреса (например, создать раздел в подразделе), то нужно изменить уже отлаженную логику по генерации ссылок внутри сайта.

Маршруты предлагают иной подход. Вы определяете шаблоны *URL* и связываете их со своим кодом. Если вы измените свое решение по поводу конкретного *URL*, то просто поменяйте шаблон *URL* - код по-прежнему будет работать отлично, и не понадобится менять какую-либо логику.

Сопоставление с образом

См.также:

- Сопоставление с образом
- Python -> Selector

Регулярные выражения дают огромные возможности для обработки URL-путей, но из-за ограничений, описанных в стандарте RFC 1738, большинство из них не нужны, при этом использование *регулярных выражений* затрудняет читабельность кода. Более современный подход придуманный Ruby on Rails это использовать технологию *сопоставление с образом*. Рассмотрим отличия на примере нашего блога:

Примечание: Исходный код доступен по адресу:

- https://github.com/iitwebdev/lectures_wsgi_example/tree/master/1.mvc

---

Предупреждение: Примеры работают только в Python3

```
from views import BlogRead, BlogIndex, BlogCreate, BlogDelete, BlogUpdate
from wsgi_basic_auth import BasicAuth

# third-party
import selector


def make_wsgi_app():
    passwd = {
        'admin': '123'
    }
    # BasicAuth applications
    create = BasicAuth(BlogCreate, 'www', passwd)
    update = BasicAuth(BlogUpdate, 'www', passwd)
    delete = BasicAuth(BlogDelete, 'www', passwd)

    # URL dispatching middleware
    dispatch = selector.Selector()
    dispatch.add('/', GET=BlogIndex)
    dispatch.prefix = '/article'
    dispatch.add('/add', GET=create, POST=create)
    dispatch.add('/{id:digits}', GET=BlogRead)
    dispatch.add('/{id:digits}/edit', GET=update, POST=update)
    dispatch.add('/{id:digits}/delete', GET=delete)
    return dispatch

if __name__ == '__main__':
    from paste.httpserver import serve
    app = make_wsgi_app()
    serve(app, host='0.0.0.0', port=8000)
```

Таблица 3: Сравнение URL

Регулярные выражения	Сопоставление с образом
/	/
/article/add	/article/add
^/article/(?P<id>d+)/$	/article/{id:digits}
^/article/(?P<id>d+)/edit$	/article/{id:digits}/edit
^/article/(?P<id>d+)/delete$	/article/{id:digits}/delete

1.4.4 Шаблоны

См.также:

- https://ru.wikipedia.org/wiki/\T2A\CYRSH\T2A\cyra\T2A\cyrb\T2A\cyrl\T2A\cyro\T2A\cyrn\T2A\cyri\T2A\cyrz\T2A\cyra\T2A\cyrt\T2A\cyro\T2A\cyrr

Шаблонизатор (в web) — это программное обеспечение, позволяющее использовать *html-шаблоны* для генерации конечных *html-страниц*.

Основная цель использования **шаблонизаторов** — это отделение *представления данных* от *исполняемого кода*. Часто это необходимо для обеспечения возможности параллельной работы *программиста* и *дизайнера-верстальщика*. Такой подход значительно ускоряет время разработки и прототипирования приложения, дизайнеру не нужно вникать в программирование, а программисту беспокоиться об интерфейсе.

Использование **шаблонизаторов** улучшает читаемость кода и упрощает внесение изменений во внешний вид, когда проект целиком выполняет один человек.

Свою популярность шаблоны обрели с приходом фреймворка Ruby On Rails и стали популярны не только в Вебе, современные десктопные приложения тоже идут по пути отделения логики программы от интерфейса, например библиотека Electron позволяет создавать GUI приложения с интерфейсом, написанном на *HTML + JavaScript* и логикой на NodeJS, по сути встраивая движок *Chromium* в ваш исполняемый файл.

Другим примером является фреймворк Qt, в котором интерфейс может быть написан на *Qml + JavaScript* и запускаться независимо от основного приложения при помощи утилиты `qmlscene`. Компания *Microsoft* также продвигает эту идею в *.Net*, предоставляя технологию WPF. Как мы видим, некоторые принципы, ранее встречающиеся преимущественно в Вебе, перенимаются другими областями программирования. Тем самым с развитием Интернет Веб-технологии будут все больше влиять на программирование в целом.

Шаблоны имеют очень простое определение — в статические файлы вставляются куски кода, а при прогоне таких файлов через специальный транслятор (препроцессор), код заменяется результатом его выполнения. Например, при компиляции шаблоны в *C++* заменяются определенными значениями:

```
template< typename T >
T min( T a, T b )
{
  return a < b ? a : b;
}
```

Перед компиляцией этот шаблон может принять такой вид:

```
int min( int a, int b )
{
  return a < b ? a : b;
}

long min( long a, long b )
```

(continues on next page)

(продолжение с предыдущей страницы)

```
{
  return a < b ? a : b;
}
```

Если *Ruby* развивался как полноценный, интерпретируемый язык общего назначения, который потом обрел фреймворк Ruby On Rails и наконец систему шаблонов, то *PHP* изначально был языком шаблонов, т.е. препроцессором, через который можно прогнать любой файл (например, *HTML* со вставками *PHP*) и получить результат.

Простой пример на *PHP*:

```
<html>
  <head>
     <title>
        Тестируем PHP
     </title>
  </head>
  <body>

  <?php
    echo '<h1>Hello, world!</h1>';
  ?>

  <br />

  <?php
    $colors = array("red", "green", "blue", "yellow");

    foreach ($colors as $value) {
        echo "* $value <br />\n";
    }
  ?>

  </body>
</html>
```

Результат выполнения программы:

Код 1: php index.php

```
<html>
  <head>
     <title>
        Тестируем PHP
     </title>
  </head>
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  <body>

  <h1>Hello, world!</h1>
  <br />

  * red <br />
* green <br />
* blue <br />
* yellow <br />

  </body>
</html>
```

Jinja2

См.также:

- http://jinja.pocoo.org/
- https://ru.wikipedia.org/wiki/Jinja

Jinja2 — самый популярный шаблонизатор в языке программирования Python. Автор *Armin Ronacher* из команды http://www.pocoo.org/ не раз приезжал на конференции в *Екатеринбург* с докладами о своих продуктах.

Синтаксис *Jinja2* сильно похож на *Django*-шаблонизатор, но при этом дает возможность использовать чистые *Python* выражения и поддерживает гибкую систему расширений.

Hello {{ name }}!

```
1 # -*- coding: utf-8 -*-
2 from jinja2 import Template
3
4 template = Template('Hello {{ name }}!')
5 print(template.render(name=u'Вася'))
```

Hello Вася!

{# Комментарии #}

```
{# Это кусок кода, который стал временно не нужен, но удалять жалко
    {% for user in users %}
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        ...
    {% endfor %}
#}
```

{{ Выражения }}

См.также:

- https://ru.wikipedia.org/wiki/\T2A\CYRV\T2A\cyrery\T2A\cyrr\T2A\cyra\T2A\cyrzh\T2A\cyre\T2A\cyrn\T2A\cyri\T2A\cyre_(\T2A\cyri\T2A\cyrn\T2A\cyrf\T2A\cyro\T2A\cyrr\T2A\cyrm\T2A\cyra\T2A\cyrt\T2A\cyri\T2A\cyrk\T2A\cyra)

{% Операторы %}

См.также:

- https://ru.wikipedia.org/wiki/\T2A\CYRO\T2A\cyrp\T2A\cyre\T2A\cyrr\T2A\cyra\T2A\cyrt\T2A\cyro\T2A\cyrr_(\T2A\cyrp\T2A\cyrr\T2A\cyro\T2A\cyrg\T2A\cyrr\T2A\cyra\T2A\cyrm\T2A\cyrm\T2A\cyri\T2A\cyrr\T2A\cyro\T2A\cyrv\T2A\cyra\T2A\cyrn\T2A\cyri\T2A\cyre)

```
1 # -*- coding: utf-8 -*-
2 from jinja2 import Template
3 text = '{% for item in range(5) %}Hello {{ name }}! {% endfor %}'
4 template = Template(text)
5 print(template.render(name=u'Вася'))
```

Hello Вася! Hello Вася! Hello Вася! Hello Вася! Hello Вася!

Модули

```
1 # -*- coding: utf-8 -*-
2 from jinja2 import Template
3 template = Template("{% set a, b, c = 'foo', 'ффуу', 'фёё' %}")
4 m = template.module
5 print(m.a)
6 print(m.b)
7 print(m.c)
```

foo

фуу
föö

Макросы

```
# -*- coding: utf-8 -*-
from jinja2 import Template

template = Template('{% macro foo() %}42{% endmacro %}23')
m = template.module
print(m)
print(m.foo())
```

23
42

Чтение из файла

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  </head>
  <body>
    {% for item in range(5) %}
      Hello {{ name }}!
    {% endfor %}
  </body>
</html>
```

Код 2: `jinja2/3.loader/0.file.py` — чтение шаблона из файла

```
# -*- coding: utf-8 -*-
from jinja2 import Template

html = open('foopkg/templates/0.hello.html').read()
template = Template(html)
print(template.render(name=u'Петя'))
```

Код 3: Результат рендеринга шаблона `jinja2/3.loader/foopkg/templates/0.hello.html`

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  </head>
  <body>

      Hello Петя!

      Hello Петя!

      Hello Петя!

      Hello Петя!

      Hello Петя!

  </body>
</html>
```

Окружение (Environment)

См.также:

- http://jinja.pocoo.org/docs/dev/api/#loaders
- http://jinja.pocoo.org/docs/dev/api/#jinja2.Environment

Настройки

Загрузчики шаблонов (Loaders)

FileSystemLoader

```
# -*- coding: utf-8 -*-
from jinja2 import Environment, FileSystemLoader

env = Environment(loader=FileSystemLoader('foopkg/templates'))
template = env.get_template('0.hello.html')
print(template.render(name=u'Петя'))
```

Результат рендеринга шаблона jinja2/3.loader/foopkg/templates/0.hello.html

PackageLoader

```
# -*- coding: utf-8 -*-
from jinja2 import Environment, PackageLoader

env = Environment(loader=PackageLoader('foopkg', 'templates'))

template = env.get_template('0.hello.html')
print(template.render(name=u'Петя'))
```

Результат рендеринга шаблона jinja2/3.loader/foopkg/templates/0.hello.html

DictLoader

```
# -*- coding: utf-8 -*-
from jinja2 import Environment, DictLoader

html = '''<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  </head>
  <body>
    {% for item in range(5) %}
      Hello {{ name }}!
    {% endfor %}
  </body>
</html>
'''

env = Environment(loader=DictLoader({'index.html': html}))
template = env.get_template('index.html')
print(template.render(name=u'Петя'))
```

Результат рендеринга шаблона jinja2/3.loader/foopkg/templates/0.hello.html

FunctionLoader

```
# -*- coding: utf-8 -*-
from jinja2 import Environment, FunctionLoader
```

(continues on next page)

(продолжение с предыдущей страницы)

```
3
4  html = '''<!DOCTYPE html>
5  <html>
6    <head>
7      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
8    </head>
9    <body>
10     {% for item in range(5) %}
11       Hello {{ name }}!
12     {% endfor %}
13   </body>
14 </html>
15 '''
16
17
18 def myloader(name):
19     if name == 'index.html':
20         return html
21
22 env = Environment(loader=FunctionLoader(myloader))
23 template = env.get_template('index.html')
24 print(template.render(name=u'Петя'))
```

Результат рендеринга шаблона jinja2/3.loader/foopkg/templates/0.hello.html

PrefixLoader

```
1  # -*- coding: utf-8 -*-
2  from jinja2 import Environment, FunctionLoader, PackageLoader, PrefixLoader
3
4  html = '''<!DOCTYPE html>
5  <html>
6    <head>
7      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
8    </head>
9    <body>
10     {% for item in range(5) %}
11       Hello {{ name }}!
12     {% endfor %}
13   </body>
14 </html>
15 '''
16
17
```

(continues on next page)

(продолжение с предыдущей страницы)

```
def myloader(name):
    if name == 'index.html':
        return html

env = Environment(loader=PrefixLoader({
    'foo': FunctionLoader(myloader),
    'bar': PackageLoader('foopkg', 'templates')
}))
template1 = env.get_template('foo/index.html')
template2 = env.get_template('bar/0.hello.html')
print(template1.render(name=u'Петя'))
print(template2.render(name=u'Петя'))
```

Результат рендеринга шаблона jinja2/3.loader/foopkg/templates/0.hello.html

ChoiceLoader

```
# -*- coding: utf-8 -*-
from jinja2 import Environment, FunctionLoader, PackageLoader, ChoiceLoader

html = '''<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  </head>
  <body>
    {% for item in range(5) %}
      Hello {{ name }}!
    {% endfor %}
  </body>
</html>
'''


def myloader(name):
    if name == 'index.html':
        return html

env = Environment(loader=ChoiceLoader({
    FunctionLoader(myloader),
    PackageLoader('foopkg', 'templates')
}))
template1 = env.get_template('index.html')
template2 = env.get_template('0.hello.html')
```

(continues on next page)

(продолжение с предыдущей страницы)

```
print(template1.render(name=u'Петя'))
print(template2.render(name=u'Петя'))
```

Результат рендеринга шаблона jinja2/3.loader/foopkg/templates/0.hello.html

ModuleLoader

```
# -*- coding: utf-8 -*-
from jinja2 import Environment, FileSystemLoader, ModuleLoader

# Compile template
Environment(loader=FileSystemLoader('foopkg/templates'))\
    .compile_templates("foopkg/compiled/foopkg.zip",
                       py_compile=True)  # pyc generate, only for python2

# Environment
env = Environment(loader=ModuleLoader("foopkg/compiled/foopkg.zip"))
template = env.get_template('0.hello.html')
print(template.render(name=u'Петя'))
```

Результат рендеринга шаблона jinja2/3.loader/foopkg/templates/0.hello.html

BaseLoader

```
# -*- coding: utf-8 -*-
from os.path import exists, getmtime, join

from jinja2 import BaseLoader, Environment, TemplateNotFound


class FoopkgLoader(BaseLoader):

    def __init__(self, path="foopkg/templates"):
        self.path = path

    def get_source(self, environment, template):
        path = join(self.path, template)
        if not exists(path):
            raise TemplateNotFound(template)
        mtime = getmtime(path)
        with open(path) as f:
            source = f.read()
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        return source, path, lambda: mtime == getmtime(path)

# Environment
env = Environment(loader=FoopkgLoader())
template = env.get_template('0.hello.html')
print(template.render(name=u'Петя'))
```

Результат рендеринга шаблона jinja2/3.loader/foopkg/templates/0.hello.html

Шаблон конфига Nginx

```
server {
         listen {{ SRC_SERVER_PUB_IP }}:80;
         servern_name {{ FQDN }} www.{{ FQDN }}

          location / {
              proxy_pass         http://{{ SRC_SERVER_LOCAL_IP }}:80/;
              proxy_redirect     off;

              proxy_set_header   Host             $host;
              proxy_set_header   X-Real-IP        $remote_addr;
              proxy_set_header   X-Forwarded-For  $proxy_add_x_forwarded_for;
         }
}
```

```
#!/usr/bin/env python
from jinja2 import Environment, FileSystemLoader

env = Environment(loader=FileSystemLoader('.'))

template = env.get_template('nginx_proxy_conf.tpl')

data = {
    "SRC_SERVER_PUB_IP": "192.168.0.100",
    "SRC_SERVER_LOCAL_IP": "10.0.3.100",
    "FQDN": "example.com"
}

conf = template.render(**data)
print(conf)

open("proxy.nginx.conf", "w").write(conf)
```

```
server {
         listen 192.168.0.100:80;
         servern_name example.com www.example.com

          location / {
              proxy_pass         http://10.0.3.100:80/;
              proxy_redirect     off;

              proxy_set_header   Host             $host;
              proxy_set_header   X-Real-IP        $remote_addr;
              proxy_set_header   X-Forwarded-For  $proxy_add_x_forwarded_for;
         }
}
```

{% extends «Наследование» %}

```
<!DOCTYPE html>
<html lang="en">
<head>
    {% block head %}
      <link rel="stylesheet" href="style.css" />
      <title>{% block title %}{% endblock %} - My Webpage</title>
      <meta charset='utf-8'>
    {% endblock %}
</head>
<body>
    <div id="content">{% block content %}{% endblock %}</div>
    <div id="footer">
        {% block footer %}
          &copy; Copyright 2008 by <a href="http://domain.invalid/">you</a>.
        {% endblock %}
    </div>
</body>
</html>
```

```
{% extends "base.html" %}
{% block title %}Index{% endblock %}
{% block head %}
    {{ super() }}
    <style type="text/css">
        .important { color: #336699; }
    </style>
{% endblock %}
{% block content %}
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    <h1>Index</h1>
    <p class="important">
      Welcome {{ name }} to my awesome homepage.
    </p>
{% endblock %}
```

```
# -*- coding: utf-8 -*-
from jinja2 import Environment, FileSystemLoader

env = Environment(loader=FileSystemLoader('.'))
template = env.get_template('index.html')
print(template.render(name=u'Петя'))
```

```
<!DOCTYPE html>
<html lang="en">
<head>


    <link rel="stylesheet" href="style.css" />
    <title>Index - My Webpage</title>
    <meta charset='utf-8'>

    <style type="text/css">
        .important { color: #336699; }
    </style>

</head>
<body>
    <div id="content">
    <h1>Index</h1>
    <p class="important">
      Welcome Петя to my awesome homepage.
    </p>
</div>
    <div id="footer">

        &copy; Copyright 2008 by <a href="http://domain.invalid/">you</a>.

    </div>
</body>
</html>
```

Блог

Код 4: templates/base.html — базовый шаблон.

```
<!DOCTYPE html>
<html lang="en">
<head>
    {% block head %}
      <link rel="stylesheet" href="style.css" />
      <title>{% block title %}{% endblock %} - My Blog</title>
      <meta charset='utf-8'>
    {% endblock %}
</head>
<body>
    <div id="content">{% block content %}{% endblock %}</div>
    <br />
    <br />
    <br />
    <div id="footer">
        {% block footer %}
          &copy; Copyright 2015 by <a href="http://domain.invalid/">you</a>.
        {% endblock %}
    </div>
</body>
</html>
```

Код 5: Главная страница templates/index.html наследуется от templates/base.html

```
{% extends "base.html" %}

{% block title %}Index{% endblock %}
{% block content %}
    <h1>Simple Blog</h1>
    <a href="/article/add">Add article</a>
    <br />
    <br />
    {% for article in articles %}
      {{ article.id }} - (<a href="/article/{{ article.id }}/delete">delete</a> |
      <a href="/article/{{ article.id }}/edit">edit</a>)
      <a href="/article/{{ article.id }}">{{ article.title }}</a><br />
    {% endfor %}
{% endblock %}
```

Код 6: templates/create.html наследуется от базового шаблона.

```
{% extends "base.html" %}

{% block title %}Create{% endblock %}
{% block content %}
    <h1>Simple Blog -> EDIT</h1>
    <form action="" method="POST">
        Title:<br>
        <input type="text" name="title" value="{{ article.title }}"><br>
        Content:<br>
        <textarea name="content">{{ article.content }}</textarea><br><br>
        <input type="submit" value="Submit">
    </form>
{% endblock %}
```

Код 7: templates/read.html наследуется от базового шаблона.

```
{% extends "base.html" %}

{% block title %}Index{% endblock %}
{% block content %}
    <h1><a href="/">Simple Blog</a> -> READ</h1>
    <h2>{{ article.title }}</h2>
    {{  article.content }}
{% endblock %}
```

Код 8: views.py — окружение Jinja2.

```
from models import ARTICLES

from jinja2 import Environment, FileSystemLoader

env = Environment(loader=FileSystemLoader('templates'))


class BaseBlog(object):

    def __init__(self, environ, start_response):
        self.environ = environ
        self.start = start_response


```

(continues on next page)

(продолжение с предыдущей страницы)

```
class BaseArticle(BaseBlog):

    def __init__(self, *args):
        super(BaseArticle, self).__init__(*args)
        article_id = self.environ['wsgiorg.routing_args'][1]['id']
        (self.index,
         self.article) = next(((i, art) for i, art in enumerate(ARTICLES)
                               if art['id'] == int(article_id)),
                              (None, None))


class BlogIndex(BaseBlog):

    def __iter__(self):
        self.start('200 OK', [('Content-Type', 'text/html')])
        yield str.encode(
            env.get_template('index.html').render(articles=ARTICLES)
        )


class BlogCreate(BaseBlog):

    def __iter__(self):
        if self.environ['REQUEST_METHOD'].upper() == 'POST':
            from urllib.parse import parse_qs
            values = parse_qs(self.environ['wsgi.input'].read())
            max_id = max([art['id'] for art in ARTICLES])
            ARTICLES.append(
                {'id': max_id+1,
                 'title': values[b'title'].pop().decode(),
                 'content': values[b'content'].pop().decode()
                 }
            )
            self.start('302 Found',
                       [('Content-Type', 'text/html'),
                        ('Location', '/')])
            return

        self.start('200 OK', [('Content-Type', 'text/html')])
        yield str.encode(
            env.get_template('create.html').render(article=None)
        )


class BlogRead(BaseArticle):
```

(continues on next page)

(продолжение с предыдущей страницы)

```
60
61      def __iter__(self):
62          if not self.article:
63              self.start('404 Not Found', [('content-type', 'text/plain')])
64              yield b'not found'
65              return
66
67          self.start('200 OK', [('Content-Type', 'text/html')])
68          yield str.encode(
69              env.get_template('read.html').render(article=self.article)
70          )
71
72
73  class BlogUpdate(BaseArticle):
74
75      def __iter__(self):
76          if self.environ['REQUEST_METHOD'].upper() == 'POST':
77              from urllib.parse import parse_qs
78              values = parse_qs(self.environ['wsgi.input'].read())
79              self.article['title'] = values[b'title'].pop().decode()
80              self.article['content'] = values[b'content'].pop().decode()
81              self.start('302 Found',
82                         [('Content-Type', 'text/html'),
83                          ('Location', '/')])
84              return
85          self.start('200 OK', [('Content-Type', 'text/html')])
86          yield str.encode(
87              env.get_template('create.html').render(article=self.article)
88          )
89
90
91  class BlogDelete(BaseArticle):
92
93      def __iter__(self):
94          self.start('302 Found',  # '301 Moved Permanently',
95                     [('Content-Type', 'text/html'),
96                      ('Location', '/')])
97          ARTICLES.pop(self.index)
98          yield b''
```

Mako

Mako — это стандартный шаблонизатор для фреймворка Pylons, написанный Майком Байером (автор SQLAlchemy). Используется на таких сайтах как https://python.org и http://reddit.com. Преимуществом является высокая скорость работы.

Hello ${ name }!

```
# -*- coding: utf-8 -*-
from mako.template import Template

template = Template('Hello ${ name }!')
print(template.render(name=u'Вася'))
```

Hello Вася!

Комментарии

```
## Однострочный коммент

<%doc> Это кусок кода, который стал временно не нужен, но удалять жалко
    % for user in users:
        ...
    % endfor
</%doc>
```

${ Выражения }

```
Это foo: ${foo}
```

```
Теорема Пифагора:  ${pow(x,2) + pow(y,2)}
```

Bash

sed шаблонизатор

```
server {
        listen {{ SRC_SERVER_PUB_IP }}:80;
        servern_name {{ FQDN }} www.{{ FQDN }}

         location / {
             proxy_pass         http://{{ SRC_SERVER_LOCAL_IP }}:80/;
             proxy_redirect     off;

             proxy_set_header   Host             $host;
             proxy_set_header   X-Real-IP        $remote_addr;
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                proxy_set_header   X-Forwarded-For  $proxy_add_x_forwarded_for;
        }
}
```

```
#! /usr/bin/env bash

SRC_SERVER_PUB_IP=192.168.0.100
SRC_SERVER_LOCAL_IP=127.0.0.1
FQDN=example.com

sed -e "s/{{ SRC_SERVER_PUB_IP }}/${SRC_SERVER_PUB_IP}/"\
  -e "s/{{ SRC_SERVER_LOCAL_IP }}/${SRC_SERVER_LOCAL_IP}/"\
  -e "s/{{ FQDN }}/${FQDN}/g" < 0.nginx_proxy_conf.tpl > proxy.nginx.conf
```

Код 9: bash/proxy.nginx.conf - результат рендеринга шаблона

```
server {
        listen 192.168.0.100:80;
        servern_name example.com www.example.com

         location / {
             proxy_pass         http://127.0.0.1:80/;
             proxy_redirect     off;

             proxy_set_header   Host             ;
             proxy_set_header   X-Real-IP        ;
             proxy_set_header   X-Forwarded-For  ;
        }
}
```

eval шаблонизатор

```
server {
        listen ${SRC_SERVER_PUB_IP}:80;
        servern_name ${FQDN} www.${FQDN}

         location / {
             proxy_pass         http://${SRC_SERVER_LOCAL_IP}:80/;
             proxy_redirect     off;

             proxy_set_header   Host             $host;
             proxy_set_header   X-Real-IP        $remote_addr;
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                proxy_set_header   X-Forwarded-For  $proxy_add_x_forwarded_for;
        }
}
```

```
#! /usr/bin/env bash

# render a template configuration file
# expand variables + preserve formatting
render_template() {
  eval "echo \"$(cat $1)\""
}

SRC_SERVER_PUB_IP=192.168.0.100
SRC_SERVER_LOCAL_IP=127.0.0.1
FQDN=example.com

render_template 1.nginx_proxy_conf.tpl > proxy.nginx.conf
```

bash/proxy.nginx.conf - результат рендеринга шаблона

1.4.5 Статика

При загрузке *HTML* страницы браузер ищет все недостающие медиа-файлы (*css*, *js*, *swf*, *png*, *svg*, *gif*, *jpg*, *avi*, *mp3*, *mp4*, ...) и подгружает их отдельными *HTTP* запросами (см. `itcase`).

На картинке выше видно, что *FireFox* нашел 14 дополнительных ресурсов для этого сайта. Браузер автоматически установит 14 *TCP* соединений, по одному на каждый ресурс, и попытается получить данные, отправив *HTTP* запросы. Т.к. установка соединения и сетевые задержки — очень дорогие операции, лучшей практикой является объединение ресурсов в один файл, например можно объединить все *JavaScript* файлы в один при помощи RequireJS, то же можно проделать и для *CSS* файлов или для картинок (спрайты), помимо прочего Webpack вообще позволяет запаковывать *JavaScript* файлы совместно с *CSS*.

Примечание: Стоит отметить, что протокол *HTTP2* лишен этого недостатка, так как загружает ресурсы асинхронно в рамках одного соединения.

Отдачу статических файлов можно настроить через отдельный сервер статики, например Nginx, или реализовать средствами самого Веб-приложения (гораздо медленнее).

Nginx

См.также:

- http://nginx.org/ru/docs/beginners_guide.html#static

Для примера возьмем следующую структуру файлов:

```
/usr/share/nginx/html/
|-- index.html
|-- index2.html
`-- static_example
    `-- static
        |-- html-css-js.png
        |-- jquery.min.js
        |-- script.js
        `-- style.css

2 directories, 6 files
```

`index2.html` страница, которая ссылается на другие статические файлы.

Код 10: index2.html

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Untitled Document</title>
  <link href="/static/style.css" media="all" rel="stylesheet" type="text/css" />
  <script src="/static/jquery.min.js"></script>
  <script src="/static/script.js"></script>
</head>
<body>
<table width="100%" border="1" cellspacing="10" cellpadding="10">
  <tr>
    <td width="60%">
    <h1>HTML</h1>
    <a href="">HTML</a>
    <a href="">JS</a>
    <a href="" style="color: green">CSS</a>
    <hr/>
    <p class="replace-text">HTML (от англ. HyperText Markup Language - «язык
    гипертекстовой разметки») - стандартный язык разметки документов во
    Всемирной паутине. Большинство веб-страниц содержат описание разметки на
    языке HTML (или XHTML). Язык HTML интерпретируется браузерами и
    отображается в виде документа в удобной для человека форме.</p> <p
    style="color: black; background-color: red; color: #fff"> Язык HTML
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    является приложением («частным случаем») SGML (стандартного обобщённого
    языка разметки) и соответствует международному стандарту ISO 8879.</p>
    <p class="hide"> Язык XHTML является более строгим вариантом HTML, он
    следует всем ограничениям XML и, фактически, XHTML можно воспринимать
    как приложение языка XML к области разметки гипертекста.</p> <p> Во
    всемирной паутине HTML-страницы, как правило, передаются браузерам от
    сервера по протоколам HTTP или HTTPS, в виде простого текста или с
    использованием сжатия.</p>
    <hr/>
    </td>
    <td width="40%"><img src="/static/html-css-js.png" class="jquery-image"
      width="500" height="293" alt="HTML JS CSS"></td> </tr>
</table>
</body>
</html>
```

Сервер Nginx настроен таким образом, что по адресу `/example` отдается страница `index2.html`, а по `/static` файлы из директории `/usr/share/nginx/html/static_example/static`.

Код 11: /etc/nginx/sites-enabled/default.nginx

```
# default.nginx

server {
    listen 80 default_server;

    root /usr/share/nginx/html;
    index index.html index.htm;

    include includes/fcgi.nginx;
    include includes/static.nginx;
}
```

Код 12: /etc/nginx/includes/static.nginx

```
location  /example {
    try_files ../$uri ../$uri/ /index2.html;
}

location /static {
    alias /usr/share/nginx/html/static_example/static;
}
```

Если скопировать файлы статики в директорию `/usr/share/nginx/html/static_example/static`, то сервер начнет их отдавать:

Paste

См.также:

- http://pythonpaste.org/modules/urlparser.html

Приведем предыдущий пример к следующей структуре файлов:

```
.
├── app.py
├── index.html
└── static
    ├── html-css-js.png
    ├── jquery.min.js
    ├── script.js
    └── style.css

1 directory, 6 files
```

Для отдачи статики используется WSGI-приложение `StaticURLParser` из модуля paste.

Код 13: app.py

```
1 from paste.urlparser import StaticURLParser
2 from paste import httpserver
3
4 static_app = StaticURLParser(".")
5
6 if __name__ == '__main__':
7     httpserver.serve(static_app, host='0.0.0.0', port='8000')
```

По адресу http://localhost:8000 будет открыта страница *index.html* (действие по умолчанию). Остальные файлы доступны по адресам:

- http://localhost:8000/index.html
- http://localhost:8000/static/html-css-js.png
- http://localhost:8000/static/jquery.min.js
- http://localhost:8000/static/script.js
- http://localhost:8000/static/style.css

Блог

В нашем примере блога определенно не хватает стилей и динамики. Чтобы это исправить, добавим файлы *css* и *js* в директорию *static*, как показано ниже:

```
.
├── __init__.py
├── models.py
├── requirements.txt
├── static
│   ├── main.css
│   └── main.js
├── templates
│   ├── base.html
│   ├── create.html
│   ├── index.html
│   └── read.html
└── views.py

2 directories, 12 files
```

Код 14: templates/base.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    {% block head %}
      <title>{% block title %}{% endblock %} - My Blog</title>
      <meta charset='utf-8'>
    {% endblock %}


    {% block css %}
      <link href="/static/main.css" media="all" rel="stylesheet" type="text/css" />
    {% endblock %}
</head>
<body>
    <div class="wrapper">
      <div class="blog">
        {% block content %}
        {% endblock %}
      </div>
      <div class="footer">
          {% block footer %}
            &copy; Copyright 2015 by <a href="http://domain.invalid/">you</a>.
          {% endblock %}
      </div>
    </div>
    {% block js %}
        <script src="/static/main.js" type="text/javascript"></script>
    {% endblock %}
```

(continues on next page)

(продолжение с предыдущей страницы)

```
</body>
</html>
```

Код 15: static/main.css

```
.wrapper
       {
         width: 50%;
         max-width: 600px;
         margin: 0 auto;
         padding: 100px 0 0 0;
         min-width: 550px;
       }
       .blog
       {
         position: relative;
       }
         .blog__title
         {
           font: 1.8em Arial;
         }
           .blog__title-link
           {
             color: blue;
           }
           .blog__title-link:hover
           {
             color: red;
           }
           .blog__title-text
           {
             color: #000;
             position: relative;
             display: inline-block;
             padding: 0 0 0 0.9em;
           }
           .blog__title-text:after
           {
             display: block;
             content: ">";
             position: absolute;
             left: 0;
             top: 0;
           }
         .blog__button
```

(continues on next page)

(продолжение с предыдущей страницы)

```
41          {
42            font: 1.1em Arial;
43            display: inline-block;
44            background: green;
45            color: #fff;
46            padding: 0.3em 0.5em;
47            text-decoration: none;
48            position: absolute;
49            right: 3.5%;
50            top: 0;
51          }
52          .blog__button:hover
53          {
54            background: darkgreen;
55          }
56          .blog-list
57          {
58            padding: 1em 0 0 0;
59            border-top: solid 1px #ccc;
60            margin: 1em 0 0 0;
61          }
62            .blog-list__item
63            {
64              padding: 0.5em 1em;
65            }
66            .blog-list__item:hover
67            {
68              background: #efefef;
69            }
70              .blog-list__item-id
71              {
72                width: 5%;
73                font: 0.9em Arial;
74                display: inline-block;
75              }
76              .blog-list__item-link
77              {
78                width: 65%;
79                font: 0.9em Arial;
80                display: inline-block;
81                color: blue;
82              }
83              .blog-list__item-link:hover
84              {
85                color: red;
```

(continues on next page)

(продолжение с предыдущей страницы)

```
86              }
87              .blog-list__item-action
88              {
89                width: 28%;
90                display: inline-block;
91                text-align: right;
92              }
93                .blog-list__item-edit
94                {
95                  font: 0.9em Arial;
96                  display: inline-block;
97                  background: blue;
98                  color: #fff;
99                  padding: 0.3em 0.6em;
100                 text-decoration: none;
101               }
102               .blog-list__item-edit:hover
103               {
104                 background: darkblue;
105               }
106               .blog-list__item-delete
107               {
108                 font: 0.9em Arial;
109                 display: inline-block;
110                 background: red;
111                 color: #fff;
112                 padding: 0.3em 0.6em;
113                 text-decoration: none;
114               }
115               .blog-list__item-delete:hover
116               {
117                 background: darkred;
118               }
119           .blog-item
120           {
121             padding: 1em 0 0 0;
122             border-top: solid 1px #ccc;
123             margin: 1em 0 0 0;
124           }
125             .blog-item__title
126             {
127                font: 1.8em Arial;
128                color: #000;
129                padding: 0 0 0.5em 0;
130             }
```

(continues on next page)

(продолжение с предыдущей страницы)

```
131             .blog-item__text
132             {
133               font: 0.9em Arial;
134               color: #000;
135             }
136           .blog-form
137           {
138             padding: 1em 0 0 0;
139             border-top: solid 1px #ccc;
140             margin: 1em 0 0 0;
141           }
142             .blog-form-field
143             {
144               padding: 0 0 1em 0;
145             }
146               .blog-form-field__title
147               {
148                 font: 0.9em Arial;
149                 color: #000;
150                 padding: 0 0 0.5em;
151               }
152               .blog-form-field__input
153               {
154                 width: 100%;
155                 border: solid 1px #ccc;
156                 font: 0.9em Arial;
157                 padding: 0.3em 0.5em;
158                 box-sizing: border-box;
159               }
160               .blog-form-field__textarea
161               {
162                 width: 100%;
163                 border: solid 1px #ccc;
164                 font: 0.9em Arial;
165                 padding: 0.3em 0.5em;
166                 min-height: 200px;
167                 box-sizing: border-box;
168               }
169             .blog-form__button
170             {
171               font: 1.2em Arial;
172               display: inline-block;
173               background: blue;
174               color: #fff;
175               padding: 0.3em 0.6em 0.25em 0.6em;
```

(continues on next page)

(продолжение с предыдущей страницы)

```
176             text-decoration: none;
177             border: solid 0px red;
178             cursor: pointer;
179           }
180           .blog-form__button:hover
181           {
182             background: darkblue;
183           }
184         .footer
185         {
186           padding: 3em 0 0 1em;
187           font: 0.8em Arial;
188         }
```

Функция `confirm_delete()` выводит окно подтверждения при нажатии на кнопку удаления статьи.

Код 16: static/main.js

```
1  function confirm_delete() {
2    return confirm("Are you sure to delete entry?");
3  }
```

Добавим классы в остальных шаблонах, чтобы наши стили применились.

Код 17: templates/index.html со стилями.

```
1  {% extends "base.html" %}
2
3  {% block title %}Index{% endblock %}
4
5  {% block content %}
6      <div class="blog__title">Simple Blog</div>
7      <a href="/article/add" class="blog__button">add article</a>
8      <div class="blog-list">
9          {% for article in articles %}
10             <div class="blog-list__item">
11                 <div class="blog-list__item-id">{{ article.id }}</div>
12                 <a href="/article/{{ article.id }}" class="blog-list__item-link">{
   ↪{ article.title }}</a>
13                 <div class="blog-list__item-action">
14                     <a href="/article/{{ article.id }}/edit" class="blog-list__
   ↪item-edit">edit</a>
15                     <a href="/article/{{ article.id }}/delete" onclick="return␣
   ↪confirm_delete();"
16                         class="blog-list__item-delete">delete</a>
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                </div>
            </div>
        {% endfor %}
    </div>
{% endblock %}
```

Код 18: templates/read.html со стилями.

```
{% extends "base.html" %}

{% block title %}{{ article.title }}{% endblock %}

{% block content %}
    <div class="blog__title">
        <a href="/" class="blog__title-link">Simple Blog</a>
        <span class="blog__title-text">{{ article.title }}</span>
    </div>
    <div class="blog-item">
        <div class="blog-item__title">{{ article.title }}</div>
        <div class="blog-item__text">{{ article.content }}</div>
    </div>
{% endblock %}
```

Код 19: templates/create.html со стлями.

```
{% extends "base.html" %}

{% block title %}Create{% endblock %}

{% block content %}
    <div class="blog__title">
        <a href="/" class="blog__title-link">Simple Blog</a>
        <span class="blog__title-text">Edit</span>
    </div>
    <form action="" method="POST" class="blog-form">
        <div class="blog-form-field">
            <div class="blog-form-field__title">Title:</div>
            <input type="text" class="blog-form-field__input" name="title" value="{
↪{ article.title }}"><br>
        </div>
        <div class="blog-form-field">
            <div class="blog-form-field__title">Content:</div>
            <textarea class="blog-form-field__textarea" name="content">{{ article.
↪content }}</textarea><br><br>
        </div>
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        <input class="blog-form__button" type="submit" value="Submit">
    </form>
{% endblock %}
```

В результате получим новое отображение блога, но пока мы не укажем откуда брать статику, он ее не сможет найти.

Статику будет отдавать WSGI-приложение `StaticURLParser`.

Код 20: __init__.py

```
from paste.auth.basic import AuthBasicHandler

import selector
from views import BlogCreate, BlogDelete, BlogIndex, BlogRead, BlogUpdate


def authfunc(environ, username, password):
    return username == 'admin' and password == '123'


def make_wsgi_app():
    # BasicAuth applications
    create = AuthBasicHandler(BlogCreate, 'www', authfunc)
    update = AuthBasicHandler(BlogUpdate, 'www', authfunc)
    delete = AuthBasicHandler(BlogDelete, 'www', authfunc)

    # URL dispatching middleware
    dispatch = selector.Selector()
    dispatch.add('/', GET=BlogIndex)
    dispatch.prefix = '/article'
    dispatch.add('/add', GET=create, POST=create)
    dispatch.add('/{id:digits}', GET=BlogRead)
    dispatch.add('/{id:digits}/edit', GET=update, POST=update)
    dispatch.add('/{id:digits}/delete', GET=delete)

    # Static files
    from paste.urlparser import StaticURLParser
    static_app = StaticURLParser("static/")

    from paste import urlmap
    mapping = urlmap.URLMap()
    mapping['/static'] = static_app

    from paste.cascade import Cascade
    app = Cascade([mapping, dispatch])
```

(continues on next page)

(продолжение с предыдущей страницы)

```
36
37      return app
38
39  if __name__ == '__main__':
40      from paste.httpserver import serve
41      app = make_wsgi_app()
42      serve(app, host='0.0.0.0', port=8000)
```

При помощи `paste.urlmap.URLMap` добавляется префикс */static/* для наших файлов из директории *static*. `paste.cascade.Cascade` позволяет запускать несколько WSGI-приложений одновременно. Таким образом наш блог принял следующую архитектуру:

И стал по-другому выглядеть:

Для реальных проектов лучше использовать библиотеку Whitenoise, она поддерживает *Python3* и *CDN*.

Код 21: __init__.py файл блога

```
1   from views import BlogRead, BlogIndex, BlogCreate, BlogDelete, BlogUpdate
2   from wsgi_basic_auth import BasicAuth
3
4   # third-party
5   import selector
6
7
8   def make_wsgi_app():
9       passwd = {
10          'admin': '123'
11      }
12      # BasicAuth applications
13      create = BasicAuth(BlogCreate, 'www', passwd)
14      update = BasicAuth(BlogUpdate, 'www', passwd)
15      delete = BasicAuth(BlogDelete, 'www', passwd)
16
17      # URL dispatching middleware
18      dispatch = selector.Selector()
19      dispatch.add('/', GET=BlogIndex)
20      dispatch.prefix = '/article'
21      dispatch.add('/add', GET=create, POST=create)
22      dispatch.add('/{id:digits}', GET=BlogRead)
23      dispatch.add('/{id:digits}/edit', GET=update, POST=update)
24      dispatch.add('/{id:digits}/delete', GET=delete)
25
26      return dispatch
27
```

(continues on next page)

(продолжение с предыдущей страницы)

```
28 if __name__ == '__main__':
29
30     app = make_wsgi_app()
31
32     from whitenoise import WhiteNoise
33     app = WhiteNoise(app)
34     app.add_files('./static/', prefix='static/')
35
36     from paste.httpserver import serve
37     serve(app, host='0.0.0.0', port=8000)
```

1.4.6 Пагинация

См.также:

- wikipedia пагинация

В Интернете под пагинацией понимают показ ограниченной части информации на одной веб-странице (например, 10 результатов поиска или 20 форумных трэдов). Она повсеместно используется в веб-приложениях для разбиения большого массива данных на страницы и включает в себя навигационный блок для перехода на другие страницы.

Paginate

См.также:

- webhelpers.paginate
- https://github.com/Pylons/paginate
- https://v4-alpha.getbootstrap.com/components/pagination/

Модуль `paginate` делит список статей на страницы. Номер страницы передается методом *GET*, в параметре *page*. По умолчанию берется первая страница.

```
p = paginate.Page(
    items,
    page=1,
    items_per_page=42
)
```

Пример *Mako* шаблона, использующего *Bootstrap4* для пагинации.

```
<%inherit file="base.mako"/>
```

(continues on next page)

(продолжение с предыдущей страницы)

```
<%block name="content">
  <h2>${tag.title()}</h2>
  <br/>
  <div class="row">
    %for item in p:
      <div class="col">
        <div class="row">
          <a href="${_static_prefix}/item/${item.id}.html">  </a>
        </div>
        <br/>
        <div class="row">
          <pre id='id-${item.id}' width=100%>
            ${item.text}
          </pre>
        </div>
      </div>
    %endfor
  </div>

  # https://v4-alpha.getbootstrap.com/components/pagination/
  <div class="row">
    <nav aria-label="Page navigation example">
      <ul class="pagination">
        ${p.pager(
          url="../$page/index.html".format(tag),
          link_attr={'class': 'page_link'},
          link_tag=lambda page: '<li class="page-item {} {}"><a class="page-link"␣
↪href="{}">{}</a></li>'.format(
          'active' if page['type'] == 'current_page' else '',
          'disabled' if not len(page['href'].strip()) else '',
          page['href'],
          page['value']
          )
        )}
      </ul>
    </nav>
  </div>
</%block>
```

Блог

Данные

См.также:

- http://ru.lipsum.com/

Для начала наполним блог случайными статьями при помощи функции `generate_lorem_ipsum` из пакета `jinja2.utils`.

Код 22: models.py

```
from jinja2.utils import generate_lorem_ipsum

ARTICLES = []

for id, article in enumerate(range(100), start=1):
    title = generate_lorem_ipsum(
        n=1,          # Одно предложение
        html=False,   # В виде обычного текста
        min=2,        # Минимум 2 слова
        max=5         # Максимум 5
    )
    content = generate_lorem_ipsum()
    ARTICLES.append(
        {'id': id, 'title': title, 'content': content}
    )
```

Paginate

Код 23: views.py

```
class BlogIndex(BaseBlog):

    def __iter__(self):
        self.start('200 OK', [('Content-Type', 'text/html')])

        # Get page number
        from urllib.parse import parse_qs
        values = parse_qs(self.environ['QUERY_STRING'])

        # Wrap articles to paginated list
        from paginate import Page
        page = values.get('page', ['1', ]).pop()
        paged_articles = Page(
            ARTICLES,
            page=page,
            items_per_page=8,
        )

```

(continues on next page)

(продолжение с предыдущей страницы)

```
        yield str.encode(
            env.get_template('index.html').render(
                articles=paged_articles
            )
        )
```

Код 24: templates/index.html

```
{% extends "base.html" %}

{% block title %}Index{% endblock %}

{% block content %}
    <div class="blog__title">Simple Blog</div>
    <a href="/article/add" class="blog__button">add article</a>
    <div class="blog-list">
        {% for article in articles %}
            <div class="blog-list__item">
                <div class="blog-list__item-id">{{ article.id }}</div>
                <a href="/article/{{ article.id }}" class="blog-list__item-link">{
↪{ article.title }}</a>
                <div class="blog-list__item-action">
                    <a href="/article/{{ article.id }}/edit" class="blog-list__
↪item-edit">edit</a>
                    <a href="/article/{{ article.id }}/delete" onclick="return␣
↪confirm_delete();"
                       class="blog-list__item-delete">delete</a>
                </div>
            </div>
        {% endfor %}
    </div>
    <div class="paginator">
        {{ articles.pager(url="?page=$page") }}
    </div>
{% endblock %}
```

В результате на каждой странице отображаются только 8 статей.

1.4.7 WebOb

См.также:

- WebOb
- http://maluke.com/old/webdev#part2

WebOb — это библиотека, сериализующая *HTTP* запрос (текст) в объект и позволяющая генерировать *HTTP* ответы. В частности работает с окружение *WSGI*.

Изначально библиотеку написал *Ян Бикинг*, затем разработкой занимался *Сергей Щетинин*, а теперь она перешла в руки *Pylons Foundation*.

Вместо странных конструкций вида:

```
from urlparse import parse_qs

values = parse_qs(environ['QUERY_STRING'])
page = values.get('page', ['1', ]).pop()
```

Мы можем использовать:

```
from webob import Request

req = Request(environ)
page = req.params.get('page', '1')
```

Request

Класс `Request` оборачивает окружение, пришедшее от Веб-сервера, в случае HTTP-запроса.

Мы можем сами создать окружение для класса `Request` и получить объект запроса, как в примере ниже.

Код 25: 0.request.py

```
1  environ = {
2      'HTTP_HOST': 'localhost:80',
3      'PATH_INFO': '/article',
4      'QUERY_STRING': 'id=1',
5      'REQUEST_METHOD': 'GET',
6      'SCRIPT_NAME': ''
7  }
8
9  from webob import Request
10 req = Request(environ)
11
12 from pprint import pprint
13 pprint(req.environ)
```

```
1  {'HTTP_HOST': 'localhost:80',
2   'PATH_INFO': '/article',
```

(continues on next page)

(продолжение с предыдущей страницы)

```
 'QUERY_STRING': 'id=1',
 'REQUEST_METHOD': 'GET',
 'SCRIPT_NAME': ''}
```

Mock запрос

`Request` имеет конструктор, который создает минимальное окружение запроса. При помощи метода `blank` можно имитировать HTTP запрос:

Код 26: 1.request.py

```
from webob import Request
req = Request.blank('/blog?page=4')

from pprint import pprint
pprint(req.environ)
```

```
{'HTTP_HOST': 'localhost:80',
 'PATH_INFO': '/blog',
 'QUERY_STRING': 'page=4',
 'REQUEST_METHOD': 'GET',
 'SCRIPT_NAME': '',
 'SERVER_NAME': 'localhost',
 'SERVER_PORT': '80',
 'SERVER_PROTOCOL': 'HTTP/1.0',
 'wsgi.errors': <open file '<stderr>', mode 'w' at 0x7f4ff5d111e0>,
 'wsgi.input': <_io.BytesIO object at 0x7f4ff3b622f0>,
 'wsgi.multiprocess': False,
 'wsgi.multithread': False,
 'wsgi.run_once': False,
 'wsgi.url_scheme': 'http',
 'wsgi.version': (1, 0)}
```

Методы объекта Request

Код 27: 2.request.py

```
from webob import Request
req = Request.blank('/blog?page=4')

print(req.method)
print(req.scheme)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
print(req.path_info)
print(req.host)
print(req.host_url)
print(req.application_url)
print(req.path_url)
print(req.url)
print(req.path)
print(req.path_qs)
print(req.query_string)
```

```
GET
http
/blog
localhost:80
http://localhost
http://localhost
http://localhost/blog
http://localhost/blog?page=4
/blog
/blog?page=4
page=4
```

GET

Код 28: 3.request.py

```
from webob import Request
req = Request.blank('/test?check=a&check=b&name=Bob')

print(req.GET)
print(req.GET['check'])
print(req.GET.getall('check'))
print(list(req.GET.items()))
```

```
GET([('check', 'a'), ('check', 'b'), ('name', 'Bob')])
b
['a', 'b']
[('check', 'a'), ('check', 'b'), ('name', 'Bob')]
```

POST

Код 29: 4.request.py

```
from webob import Request
req = Request.blank('/test')

print(req.POST)  # empty
print(list(req.POST.items()))

print()

# Set POST
req.method = 'POST'
req.body = b'name=Vasya&email=vasya@example.com'

print(req.POST)  # not empty
print(req.POST['name'])
print(req.POST['email'])
```

```
<NoVars: Not a form request>
[]

MultiDict([('name', 'Vasya'), ('email', 'vasya@example.com')])
Vasya
vasya@example.com
```

GET & POST & PUT & DELETE ...

Если вы не уверенны, каким методом были отправлены данные, можно воспользоваться атрибутом `params`.

Код 30: 5.request.py

```
from webob import Request
req = Request.blank('/test?check=a&check=b&name=Bob')

# Set POST
req.method = 'POST'
req.body = b'name=Vasya&email=vasya@example.com'

print(req.params)
print(req.params.getall('check'))
print(req.params['email'])
print(req.params['name'])
```

```
NestedMultiDict([('check', 'a'), ('check', 'b'), ('name', 'Bob'), ('name', 'Vasya
↪'), ('email', 'vasya@example.com')])
['a', 'b']
vasya@example.com
Bob
```

Cookie

Код 31: 6.request.py

```
from webob import Request
req = Request.blank('/test')

# Set Cookie
req.headers['Cookie'] = 'session_id=9999999;foo=abcdef;bar=2'

print(req.cookies)
print(req.cookies['foo'])
```

```
<RequestCookies (dict-like) with values {'bar': '2', 'foo': 'abcdef', 'session_id
↪': '9999999'}>
abcdef
```

Запуск WSGI-приложений

`webob.request.Request` умеет запускать *WSGI*-приложения. Это может понадобиться, например, при написании тестов.

Код 32: 7.request.py

```
from webob import Request


def wsgi_app(environ, start_response):
    request = Request(environ)
    if request.path == '/test':
        start_response('200 OK', [('Content-type', 'text/plain')])
        return ['Hi!']
    start_response('404 Not Found', [('Content-type', 'text/plain')])

req = Request.blank('/test')
status, headers, app_iter = req.call_application(wsgi_app)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
print(status)
print(headers)
print(app_iter)
print

req = Request.blank('/bar')
status, headers, app_iter = req.call_application(wsgi_app)
print(status)
print(headers)
print(app_iter)
```

```
200 OK
[('Content-type', 'text/plain')]
['Hi!']

404 Not Found
[('Content-type', 'text/plain')]
None
```

Response

Класс, который содержит все необходимое для создания ответа *WSGI*-приложения.

Конструктор класса `Response` имеет минимальный набор для HTTP ответа:

```
>>> from webob import Response
>>> res = Response()
>>> res.status
'200 OK'
>>> res.headerlist
[('Content-Type', 'text/html; charset=UTF-8'), ('Content-Length', '0')]
>>> res.body
''
```

В процессе выполнения программы ответ можно переопределить:

```
>>> res.status = 404
>>> res.status
'404 Not Found'
>>> res.status_code
404
>>> res.headerlist = [('Content-type', 'text/html')]
>>> res.body = b'test'
>>> print res
```

(continues on next page)

(продолжение с предыдущей страницы)

```
404 Not Found
Content-type: text/html
Content-Length: 4

test
>>> res.body = u"test"
Traceback (most recent call last):
    ...
TypeError: You cannot set Response.body to a unicode object (use Response.text)
>>> res.text = u"test"
Traceback (most recent call last):
    ...
AttributeError: You cannot access Response.text unless charset is set
>>> res.charset = 'utf8'
>>> res.text = u"test"
>>> res.body
'test'
```

Также можно задать значения передав их в конструктор, например `Response(charset='utf8')`.

```
>>> from webob import Response
>>> resp = Response(body=b'Hello World!')
>>> resp.content_type
'text/html'
>>> resp.content_type = 'text/plain'
>>> print resp
200 OK
Content-Length: 12
Content-Type: text/plain; charset=UTF-8

Hello World!
```

get_response

`get_response` генерирует HTTP ответ.

Код 33: 8.response.py

```
1 from webob import Request, Response
2
3
4 def wsgi_app(environ, start_response):
5     response = Response()
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    response.content_type = 'text/plain'

    parts = []
    for name, value in sorted(environ.items()):
        parts.append('%s: %r' % (name, value))

    response.body = str.encode(
        '\n'.join(parts)
    )
    return response(environ, start_response)

req = Request.blank('/test')
print(req.call_application(wsgi_app))  # WSGI-application response
print()
print(req.get_response(wsgi_app))  # HTTP response
```

```
('200 OK', [('Content-Type', 'text/plain; charset=UTF-8'), ('Content-Length', '411
↪')], [b"HTTP_HOST: 'localhost:80'\nPATH_INFO: '/test'\nQUERY_STRING: ''\nREQUEST_
↪METHOD: 'GET'\nSCRIPT_NAME: ''\nSERVER_NAME: 'localhost'\nSERVER_PORT: '80
↪'\nSERVER_PROTOCOL: 'HTTP/1.0'\nwsgi.errors: <_io.TextIOWrapper name='<stderr>'␣
↪mode='w' encoding='UTF-8'>\nwsgi.input: <_io.BytesIO object at 0x7f692e219048>
↪\nwsgi.multiprocess: False\nwsgi.multithread: False\nwsgi.run_once: False\nwsgi.
↪url_scheme: 'http'\nwsgi.version: (1, 0)"])

200 OK
Content-Type: text/plain; charset=UTF-8
Content-Length: 411

HTTP_HOST: 'localhost:80'
PATH_INFO: '/test'
QUERY_STRING: ''
REQUEST_METHOD: 'GET'
SCRIPT_NAME: ''
SERVER_NAME: 'localhost'
SERVER_PORT: '80'
SERVER_PROTOCOL: 'HTTP/1.0'
wsgi.errors: <_io.TextIOWrapper name='<stderr>' mode='w' encoding='UTF-8'>
wsgi.input: <_io.BytesIO object at 0x7f692e219048>
wsgi.multiprocess: False
wsgi.multithread: False
wsgi.run_once: False
wsgi.url_scheme: 'http'
wsgi.version: (1, 0)
```

Exceptions

См.также:

- http://maluke.com/old/webdev#webobexchttpexception

```
>>> from webob.exc import *
>>> exc = HTTPTemporaryRedirect(location='foo')
>>> req = Request.blank('/path/to/something')
>>> print str(req.get_response(exc)).strip()
307 Temporary Redirect
Location: http://localhost/path/to/foo
Content-Length: 126
Content-Type: text/plain; charset=UTF-8

307 Temporary Redirect

The resource has been moved to http://localhost/path/to/foo; you should be␣
↪redirected automatically.
```

Блог

Добавим декоратор `wsgify`, который будет делать для каждого «вида» всю *WSGI*-магию и добавлять объект `request`.

Код 34: views.py декоротор wsgify

```
1 def wsgify(view):
2     from webob import Request
3
4     def wrapped(environ, start_response):
5         request = Request(environ)
6         app = view(request).response()
7         return app(environ, start_response)
8     return wrapped
```

Index

В самих представлениях `request` передается как параметр конструктора, а ответ реализуется в виде метода класса `response`.

Код 35: views.py класс BlogIndex

```
@wsgify
class BlogIndex(object):

    def __init__(self, request):
        self.page = request.GET.get('page', '1')
        from paginate import Page
        self.paged_articles = Page(
            ARTICLES,
            page=self.page,
            items_per_page=8,
        )

    def response(self):
        from webob import Response
        return Response(env.get_template('index.html')
                        .render(articles=self.paged_articles)
                        .encode('utf-8'))
```

```
@wsgify
class BlogIndex(object):
   ...
```

Метод `response` должен возвращать WSGI-приложение. В нашем случае это объект класса `Response` из библиотеки `webob`.

Код 36: BlogIndex.response

```
@wsgify
class BlogIndex(object):

    def __init__(self, request):
        self.page = request.GET.get('page', '1')
        from paginate import Page
        self.paged_articles = Page(
            ARTICLES,
            page=self.page,
            items_per_page=8,
        )

    def response(self):
        from webob import Response
        return Response(env.get_template('index.html')
                        .render(articles=self.paged_articles)
                        .encode('utf-8'))
```

Create

Код 37: views.py класс BlogCreate

```
@wsgify
class BlogCreate(object):

    def __init__(self, request):
        self.request = request

    def response(self):
        from webob import Response
        if self.request.method == 'POST':
            max_id = max([art['id'] for art in ARTICLES])
            ARTICLES.append(
                {'id': max_id+1,
                 'title': self.request.POST['title'],
                 'content': self.request.POST['content']
                 }
            )
            return Response(status=302, location='/')
        return Response(env.get_template('create.html').render(article=None))
```

Код 38: views.py изменения в классе BlogCreate

```
@wsgify
class BlogCreate(object):

    def __init__(self, request):
        self.request = request

    def response(self):
        from webob import Response
        if self.request.method == 'POST':
            max_id = max([art['id'] for art in ARTICLES])
            ARTICLES.append(
                {'id': max_id+1,
                 'title': self.request.POST['title'],
                 'content': self.request.POST['content']
                 }
            )
            return Response(status=302, location='/')
        return Response(env.get_template('create.html').render(article=None))
```

```
@wsgify
class BlogCreate(object):
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    ...
```

BaseArticle

Код 39: views.py класс BaseArticle

```
class BaseArticle(object):

    def __init__(self, request):
        self.request = request
        article_id = self.request.environ['wsgiorg.routing_args'][1]['id']
        (self.index,
         self.article) = next(((i, art) for i, art in enumerate(ARTICLES)
                               if art['id'] == int(article_id)),
                              (None, None))
```

BlogRead

Код 40: views.py пример класса BlogRead без webob

```
class BlogRead(BaseArticle):

    def __iter__(self):
        if not self.article:
            self.start('404 Not Found', [('content-type', 'text/plain')])
            yield b'not found'
            return

        self.start('200 OK', [('Content-Type', 'text/html')])
        yield str.encode(
            env.get_template('read.html').render(article=self.article)
        )
```

Код 41: views.py класс BlogRead

```
@wsgify
class BlogRead(BaseArticle):

    def response(self):
        from webob import Response
        if not self.article:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            return Response(status=404)
        return Response(env.get_template('read.html')
                        .render(article=self.article))
```

BlogUpdate

Код 42: views.py пример класса BlogUpdate без webob

```
class BlogUpdate(BaseArticle):

    def __iter__(self):
        if self.environ['REQUEST_METHOD'].upper() == 'POST':
            from urllib.parse import parse_qs
            values = parse_qs(self.environ['wsgi.input'].read())
            self.article['title'] = values[b'title'].pop().decode()
            self.article['content'] = values[b'content'].pop().decode()
            self.start('302 Found',
                       [('Content-Type', 'text/html'),
                        ('Location', '/')])
            return
        self.start('200 OK', [('Content-Type', 'text/html')])
        yield str.encode(
            env.get_template('create.html').render(article=self.article)
        )
```

Код 43: views.py класс BlogUpdate

```
@wsgify
class BlogUpdate(BaseArticle):

    def response(self):
        from webob import Response
        if self.request.method == 'POST':
            self.article['title'] = self.request.POST['title']
            self.article['content'] = self.request.POST['content']
            return Response(status=302, location='/')
        return Response(env.get_template('create.html')
                        .render(article=self.article))
```

BlogDelete

Код 44: views.py пример класса BlogDelete без webob

```
class BlogDelete(BaseArticle):

    def __iter__(self):
        self.start('302 Found',  # '301 Moved Permanently',
                   [('Content-Type', 'text/html'),
                    ('Location', '/')])
        ARTICLES.pop(self.index)
        yield b''
```

Код 45: views.py класс BlogDelete

```
@wsgify
class BlogDelete(BaseArticle):

    def response(self):
        from webob import Response
        ARTICLES.pop(self.index)
        return Response(status=302, location='/')
```

views.py

```
from models import ARTICLES

from jinja2 import Environment, FileSystemLoader

env = Environment(loader=FileSystemLoader('templates'))


def wsgify(view):
    from webob import Request

    def wrapped(environ, start_response):
        request = Request(environ)
        app = view(request).response()
        return app(environ, start_response)
    return wrapped


class BaseArticle(object):

    def __init__(self, request):
        self.request = request
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        article_id = self.request.environ['wsgiorg.routing_args'][1]['id']
        (self.index,
         self.article) = next(((i, art) for i, art in enumerate(ARTICLES)
                               if art['id'] == int(article_id)),
                              (None, None))


@wsgify
class BlogIndex(object):

    def __init__(self, request):
        self.page = request.GET.get('page', '1')
        from paginate import Page
        self.paged_articles = Page(
            ARTICLES,
            page=self.page,
            items_per_page=8,
        )

    def response(self):
        from webob import Response
        return Response(env.get_template('index.html')
                        .render(articles=self.paged_articles)
                        .encode('utf-8'))


@wsgify
class BlogCreate(object):

    def __init__(self, request):
        self.request = request

    def response(self):
        from webob import Response
        if self.request.method == 'POST':
            max_id = max([art['id'] for art in ARTICLES])
            ARTICLES.append(
                {'id': max_id+1,
                 'title': self.request.POST['title'],
                 'content': self.request.POST['content']
                 }
            )
            return Response(status=302, location='/')
        return Response(env.get_template('create.html').render(article=None))

```

(continues on next page)

(продолжение с предыдущей страницы)

```

@wsgify
class BlogRead(BaseArticle):

    def response(self):
        from webob import Response
        if not self.article:
            return Response(status=404)
        return Response(env.get_template('read.html')
                        .render(article=self.article))


@wsgify
class BlogUpdate(BaseArticle):

    def response(self):
        from webob import Response
        if self.request.method == 'POST':
            self.article['title'] = self.request.POST['title']
            self.article['content'] = self.request.POST['content']
            return Response(status=302, location='/')
        return Response(env.get_template('create.html')
                        .render(article=self.article))


@wsgify
class BlogDelete(BaseArticle):

    def response(self):
        from webob import Response
        ARTICLES.pop(self.index)
        return Response(status=302, location='/')
```

1.4.8 Формы

WTForms

См.также:

- http://wtforms.readthedocs.org/

```
from jinja2 import Template
from wtforms import BooleanField, Form, StringField, validators

```

(continues on next page)

(продолжение с предыдущей страницы)

```
4
5  class RegistrationForm(Form):
6      username = StringField('Username', [validators.Length(min=4, max=25)])
7      email = StringField('Email Address', [validators.Length(min=6, max=35)])
8      rules = BooleanField('I accept the site rules',
9                           [validators.InputRequired()])
10
11 if __name__ == '__main__':
12     form = RegistrationForm(username="root")
13     template = Template("""
14 <form method="POST" action="">
15     {% for field in form.data %}
16         {{ form[field].label }} |
17         {{ form[field] }}
18         <br />
19     {% endfor %}
20     <input type="submit" value="Ok">
21 </form>
22     """)
23     print(template.render(form=form))
```

```
1  <form method="POST" action="">
2
3          <label for="username">Username</label> |
4          <input id="username" name="username" type="text" value="root">
5          <br />
6
7          <label for="rules">I accept the site rules</label> |
8          <input id="rules" name="rules" type="checkbox" value="y">
9          <br />
10
11         <label for="email">Email Address</label> |
12         <input id="email" name="email" type="text" value="">
13         <br />
14
15     <input type="submit" value="Ok">
16 </form>
17
```

Deform

См.также:

- Deform

- https://skillsmatter.com/skillscasts/4886-king-forms

Deform — это *Python* библиотека для генерации форм. *Deform* использует Colander как генератор схемы, Peppercorn для десериализации данных из формы и шаблонизатор Chameleon.

Основные задачи, которые выполняет *Deform*:

- Генерирует форму
- Имеет набор виджетов для форм
- Умеет генерировать AJAX формы
- Использует схемы Colander
- Использует шаблоны Chameleon (Но можно использовать и другие, например Jinja2 или Mako)

Примеры форм http://deformdemo.repoze.org/

Colander

См.также:

- Colander

Colander - десериализует данные полученные как XML, JSON, HTTP POST запрос и проверяет правильность их заполнения по заранее заданной схеме.

- Определяет структуру (схему) формы
- Проверяет содержимое формы

Простая форма

Для создания простой формы нам понадобится:

- Схема Colander
- Объект *Form* из Deform
- WSGI-приложение, которое получает POST параметры из запроса
- Шаблон страницы с формой

Схема Colander

Код 46: 0.simple_form.py - Colander схема

```
import colander
import deform
from jinja2 import Environment, FileSystemLoader

env = Environment(loader=FileSystemLoader('templates'))


class Contact(colander.MappingSchema):
    email = colander.SchemaNode(colander.String(), validator=colander.Email())
    name = colander.SchemaNode(colander.String())
    message = colander.SchemaNode(colander.String(),
                                  widget=deform.widget.TextAreaWidget())


def simple_form(environ, start_response):
    start_response('200 OK', [('Content-Type', 'text/html')])
    from webob import Request
    request = Request(environ)

    form = deform.Form(Contact(), buttons=('submit',))
    template = env.get_template('simple.html')
    if request.POST:
        submitted = request.POST.items()
        try:
            form.validate(submitted)
        except deform.ValidationFailure as e:
            return template.render(form=e.render())
    data = {'email': 'jon.staley@fundingoptions.com',
            'name': 'Jon',
            'message': 'Hello World'}
    return template.render(form=form.render(data))


if __name__ == '__main__':
    from paste.httpserver import serve

    serve(simple_form, host='0.0.0.0', port=8000)
```

Форма deform.Form

Код 47: 0.simple_form.py - Форма от Deform

```python
import colander
import deform
from jinja2 import Environment, FileSystemLoader

env = Environment(loader=FileSystemLoader('templates'))


class Contact(colander.MappingSchema):
    email = colander.SchemaNode(colander.String(), validator=colander.Email())
    name = colander.SchemaNode(colander.String())
    message = colander.SchemaNode(colander.String(),
                                  widget=deform.widget.TextAreaWidget())


def simple_form(environ, start_response):
    start_response('200 OK', [('Content-Type', 'text/html')])
    from webob import Request
    request = Request(environ)

    form = deform.Form(Contact(), buttons=('submit',))
    template = env.get_template('simple.html')
    if request.POST:
        submitted = request.POST.items()
        try:
            form.validate(submitted)
        except deform.ValidationFailure as e:
            return template.render(form=e.render())
    data = {'email': 'jon.staley@fundingoptions.com',
            'name': 'Jon',
            'message': 'Hello World'}
    return template.render(form=form.render(data))


if __name__ == '__main__':
    from paste.httpserver import serve

    serve(simple_form, host='0.0.0.0', port=8000)
```

Шаблон simple.html

Код 48: templates/simple.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <title>simple</title>
    </head>
    <body>
        {{ form }}
    </body>
</html>
```

Добавим стилей:

Код 49: templates/simple.html с CSS стилями.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <title>simple</title>
        <link rel="stylesheet" type="text/css" href="https://cdnjs.cloudflare.com/ajax/libs/materialize/0.96.1/css/materialize.min.css" />
        <script type="text/javascript" src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
        <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/materialize/0.96.1/js/materialize.min.js"></script>
        <style type="text/css">
            .deformFormFieldset {
                border: none;
                padding: 0;
                margin: 0;
            }
            .control-label {
                left: 0 !important;
            }
            .form-group
            {
                position: relative;
                margin: 1em 0 0 0;
            }
            .alert-danger
            {
                top: -1.5em;
                position: relative;
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            }
            p.help-block[id*="error"]
            {
                width: 10em;
                color: red !important;
                font-style: normal;
                padding: 0;
                margin: 0 0 1em 0;
                line-height: 1.5;
                font-family: "Roboto", sans-serif;
                font-weight: normal;
                position: absolute;
                left: -12em;
                text-align: right;
                top: 0.85em;
            }
            .required:after
            {
                content: "*";
                position: relative;
                font-family: "Roboto", sans-serif;
                font-weight: normal;
                color: red;
            }
            .errorMsgLbl
            {
                background: red;
                font-family: "Roboto", sans-serif;
                font-weight: normal;
                padding: 0.5em 1em;
                color: #fff;
                margin: 0 0 1em 0;
            }
            .alert-danger .errorMsg
            {
                display: none;
            }
        </style>
    </head>
    <body>
        <div class="form" style="width: 500px; margin: 0 auto; padding: 50px">
            <h1>Simple Form</h1>
            {{ form }}
        </div>
    </body>
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    <script type="text/javascript">
        $(function() {
            $('#item-deformField1').addClass('input-field').addClass('col');
            $('#item-deformField2').addClass('input-field').addClass('col');
            $('#item-deformField3').addClass('input-field').addClass('col');
            $('#deformField3').addClass('materialize-textarea');
        });
    </script>
</html>
```

Widgets

См.также:

- http://deform.readthedocs.org/en/latest/api.html#module-deform.widget

Таблица 4: Соответствие полей Colander и виджетов Deform

Colander	Deform
colander.String	deform.widget.TextInputWidget
colander.Integer	deform.widget.TextInputWidget
colander.Float	deform.widget.TextInputWidget
colander.Decimal	deform.widget.TextInputWidget
colander.Boolean	deform.widget.CheckboxWidget
colander.Date	deform.widget.DateInputWidget
colander.DateTime	deform.widget.DateTimeInputWidget

Deform и Colander в реальных проектах

Наследование схем

Код 50: 1.inheritance.py - Наследование Colander схем

```
import colander
import deform
from jinja2 import Environment, FileSystemLoader

env = Environment(loader=FileSystemLoader('templates'))


class AddressSchema(colander.MappingSchema):
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    line1 = colander.SchemaNode(colander.String(), title='Address line 1')
    line2 = colander.SchemaNode(colander.String(), title='Address line 2',
                                missing=None)
    line3 = colander.SchemaNode(colander.String(), title='Address line 3',
                                missing=None)
    town = colander.SchemaNode(colander.String(), title='Town')
    postcode = colander.SchemaNode(colander.String(), title='Postcode')


class Business(AddressSchema):
    business_name = colander.SchemaNode(colander.String(),
                                        title='Business Name',
                                        insert_before='line1')


def inheritance_form(environ, start_response):
    start_response('200 OK', [('Content-Type', 'text/html')])
    from webob import Request
    request = Request(environ)

    form = deform.Form(Business(), buttons=('submit',))
    template = env.get_template('simple_with_css.html')
    if request.POST:
        submitted = request.POST.items()
        try:
            form.validate(submitted)
        except deform.ValidationFailure as e:
            return template.render(form=e.render())
    return template.render(form=form.render())


if __name__ == '__main__':
    from paste.httpserver import serve

    serve(inheritance_form, host='0.0.0.0', port=8000)
```

Кастомная валидация

Код 51: 2.custom_validators.py - Кастомная валидация

```
import colander
import deform
from jinja2 import Environment, FileSystemLoader
```

(continues on next page)

(продолжение с предыдущей страницы)

```
4
5  env = Environment(loader=FileSystemLoader('templates'))
6
7
8  def month_validator(node, month):
9      if month.isdigit():
10         int_month = int(month)
11         if not 0 < int_month < 13:
12             raise colander.Invalid(node,
13                                    'Please enter a number between 1 and 12')
14     else:
15         raise colander.Invalid(node, 'Please enter a number')
16
17
18 class AddressSchema(colander.MappingSchema):
19     line1 = colander.SchemaNode(colander.String(), title='Address line 1')
20     line2 = colander.SchemaNode(colander.String(), title='Address line 2',
21                                 missing=None)
22     line3 = colander.SchemaNode(colander.String(), title='Address line 3',
23                                 missing=None)
24     town = colander.SchemaNode(colander.String(), title='Town')
25     postcode = colander.SchemaNode(colander.String(), title='Postcode')
26
27
28 class Business(AddressSchema):
29     business_name = colander.SchemaNode(colander.String(),
30                                         title='Business Name',
31                                         insert_before='line1')
32     start_month = colander.SchemaNode(colander.String(), title='Start month',
33                                       validator=month_validator)
34
35
36 def custom_validator_form(environ, start_response):
37     start_response('200 OK', [('Content-Type', 'text/html')])
38     from webob import Request
39     request = Request(environ)
40
41     form = deform.Form(Business(), buttons=('submit',))
42     template = env.get_template('simple_with_css.html')
43     if request.POST:
44         submitted = request.POST.items()
45         try:
46             form.validate(submitted)
47         except deform.ValidationFailure as e:
48             return template.render(form=e.render()).encode("utf-8")
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    return template.render(form=form.render()).encode("utf-8")


if __name__ == '__main__':
    from paste.httpserver import serve

    serve(custom_validator_form, host='0.0.0.0', port=8000)
```

Отложенная валидация

См.также:

- CSRF

Код 52: 3.defered_validators.py - добавление CSRF токена

```
import colander
import deform
from jinja2 import Environment, FileSystemLoader

env = Environment(loader=FileSystemLoader('templates'))


def get_session(request):
    return request.environ.get('paste.session.factory', lambda: {})()


def get_csrf_token(session):
    if 'csrf' not in session:
        from uuid import uuid4
        session['csrf'] = uuid4().hex
    return session['csrf']


@colander.deferred
def deferred_csrf_default(node, kw):
    request = kw.get('request')
    session = get_session(request)
    csrf_token = get_csrf_token(session)
    return csrf_token


@colander.deferred
```

(continues on next page)

(продолжение с предыдущей страницы)

```
def deferred_csrf_validator(node, kw):
    def validate_csrf_token(node, value):
        request = kw.get('request')
        session = get_session(request)
        csrf_token = get_csrf_token(session)
        if value != csrf_token:
            raise colander.Invalid(node, 'Bad CSRF token')

    return validate_csrf_token


class CSRFSchema(colander.Schema):
    csrf = colander.SchemaNode(colander.String(),
                               default=deferred_csrf_default,
                               validator=deferred_csrf_validator,
                               # widget=deform.widget.HiddenWidget(), )
                               )


class Contact(CSRFSchema):
    email = colander.SchemaNode(colander.String(), validator=colander.Email())
    name = colander.SchemaNode(colander.String())
    message = colander.SchemaNode(colander.String(),
                                  widget=deform.widget.TextAreaWidget())


def custom_validator_form(environ, start_response):
    start_response('200 OK', [('Content-Type', 'text/html')])
    from webob import Request
    request = Request(environ)
    session = get_session(request)
    session['csrf'] = get_csrf_token(session)

    schema = Contact().bind(request=request)
    form = deform.Form(schema, buttons=('submit',))
    template = env.get_template('simple_with_css.html')
    if request.POST:
        submitted = request.POST.items()
        try:
            form.validate(submitted)
        except deform.ValidationFailure as e:
            return template.render(form=e.render()).encode("utf-8")
        session.pop('csrf')
        return template.render(form='OK')
    return template.render(form=form.render()).encode("utf-8")
```

(continues on next page)

(продолжение с предыдущей страницы)

```

if __name__ == '__main__':
    from paste.httpserver import serve
    from paste.session import SessionMiddleware

    app = SessionMiddleware(custom_validator_form)

    serve(app, host='0.0.0.0', port=8000)
```

Код 53: 3.defered_validators.py - отложенная валидация

```
import colander
import deform
from jinja2 import Environment, FileSystemLoader

env = Environment(loader=FileSystemLoader('templates'))


def get_session(request):
    return request.environ.get('paste.session.factory', lambda: {})()


def get_csrf_token(session):
    if 'csrf' not in session:
        from uuid import uuid4
        session['csrf'] = uuid4().hex
    return session['csrf']


@colander.deferred
def deferred_csrf_default(node, kw):
    request = kw.get('request')
    session = get_session(request)
    csrf_token = get_csrf_token(session)
    return csrf_token


@colander.deferred
def deferred_csrf_validator(node, kw):
    def validate_csrf_token(node, value):
        request = kw.get('request')
        session = get_session(request)
        csrf_token = get_csrf_token(session)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        if value != csrf_token:
            raise colander.Invalid(node, 'Bad CSRF token')

    return validate_csrf_token


class CSRFSchema(colander.Schema):
    csrf = colander.SchemaNode(colander.String(),
                               default=deferred_csrf_default,
                               validator=deferred_csrf_validator,
                               # widget=deform.widget.HiddenWidget(), )
                               )


class Contact(CSRFSchema):
    email = colander.SchemaNode(colander.String(), validator=colander.Email())
    name = colander.SchemaNode(colander.String())
    message = colander.SchemaNode(colander.String(),
                                  widget=deform.widget.TextAreaWidget())


def custom_validator_form(environ, start_response):
    start_response('200 OK', [('Content-Type', 'text/html')])
    from webob import Request
    request = Request(environ)
    session = get_session(request)
    session['csrf'] = get_csrf_token(session)

    schema = Contact().bind(request=request)
    form = deform.Form(schema, buttons=('submit',))
    template = env.get_template('simple_with_css.html')
    if request.POST:
        submitted = request.POST.items()
        try:
            form.validate(submitted)
        except deform.ValidationFailure as e:
            return template.render(form=e.render()).encode("utf-8")
        session.pop('csrf')
        return template.render(form='OK')
    return template.render(form=form.render()).encode("utf-8")


if __name__ == '__main__':
    from paste.httpserver import serve
    from paste.session import SessionMiddleware
```

(continues on next page)

(продолжение с предыдущей страницы)

```
78
79      app = SessionMiddleware(custom_validator_form)
80
81      serve(app, host='0.0.0.0', port=8000)
```

Переопределение стандартных шаблонов

Код 54: 4.custom_templates.py - переопределение стандартных шаблонов формы

```
1  import os
2
3  import colander
4  import deform
5  from jinja2 import Environment, FileSystemLoader
6  from pkg_resources import resource_filename
7
8  env = Environment(loader=FileSystemLoader('templates'))
9
10
11 deform_path = os.path.abspath('templates/deform')
12 deform_templates = resource_filename('deform', 'templates')
13 print(deform_templates)
14 print(deform_path)
15 search_path = (deform_path, deform_templates)
16 renderer = deform.ZPTRendererFactory(search_path)
17
18
19 class Contact(colander.MappingSchema):
20     email = colander.SchemaNode(colander.String(), validator=colander.Email())
21     name = colander.SchemaNode(colander.String())
22     message = colander.SchemaNode(colander.String(),
23                                   widget=deform.widget.TextAreaWidget())
24
25
26 def custom_template_form(environ, start_response):
27     start_response('200 OK', [('Content-Type', 'text/html')])
28     from webob import Request
29     request = Request(environ)
30
31     form = deform.Form(Contact(), buttons=('submit',), renderer=renderer)
32     template = env.get_template('simple_with_css.html')
33     if request.POST:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        submitted = request.POST.items()
        try:
            form.validate(submitted)
        except deform.ValidationFailure as e:
            return template.render(form=e.render()).encode("utf-8")
    data = {'email': 'jon.staley@fundingoptions.com',
            'name': 'Jon',
            'message': 'Hello World'}
    return template.render(form=form.render(data)).encode("utf-8")


if __name__ == '__main__':
    from paste.httpserver import serve

    serve(custom_template_form, host='0.0.0.0', port=8000)
```

```
$ tree templates/deform/
templates/deform/
├── form.pt
└── mapping_item.pt

0 directories, 2 files
```

Новые шаблоны на Jinja2

См.также:

- http://docs.pylonsproject.org/projects/deform/en/latest/templates.html#using-an-alternative-templating-system

Код 55: 5.custom_jinja2_templates.py - переопределение стандартных шаблонов формы, на свои Jinja2 шаблоны

```
import os

import colander
import deform
from jinja2 import Environment, FileSystemLoader

env = Environment(loader=FileSystemLoader('templates'))


def jinja2_renderer(template_name, **kw):
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    kw['_'] = str  # Hook for translation string with gettext

    from jinja2 import Template
    deform_jinja_path = os.path.abspath('templates/deform_jinja2')
    jinja2_template = os.path.join(deform_jinja_path,
                                   template_name + '.jinja2')
    template = Template(open(jinja2_template).read())
    return template.render(**kw)


class Contact(colander.MappingSchema):
    email = colander.SchemaNode(colander.String(), validator=colander.Email())
    name = colander.SchemaNode(colander.String())
    message = colander.SchemaNode(colander.String(),
                                  widget=deform.widget.TextAreaWidget())


def custom_template_form(environ, start_response):
    start_response('200 OK', [('Content-Type', 'text/html')])
    from webob import Request
    request = Request(environ)

    form = deform.Form(Contact(), buttons=('submit',),
                       renderer=jinja2_renderer)
    template = env.get_template('simple_with_css.html')
    if request.POST:
        submitted = request.POST.items()
        try:
            form.validate(submitted)
        except deform.ValidationFailure as e:
            return template.render(form=e.render()).encode("utf-8")
    data = {'email': 'jon.staley@fundingoptions.com',
            'name': 'Jon',
            'message': 'Hello World'}
    return template.render(form=form.render(data)).encode("utf-8")


if __name__ == '__main__':
    from paste.httpserver import serve

    serve(custom_template_form, host='0.0.0.0', port=8000)
```

```
$ tree templates/deform_jinja2/
templates/deform_jinja2/
├── autocomplete_input.jinja2
```

(continues on next page)

(продолжение с предыдущей страницы)

```
├── checkbox_choice.jinja2
├── checkbox.jinja2
├── checked_input.jinja2
├── checked_password.jinja2
├── dateinput.jinja2
├── dateparts.jinja2
├── datetimeinput.jinja2
├── file_upload.jinja2
├── form.jinja2
├── hidden.jinja2
├── mapping_item.jinja2
├── mapping.jinja2
├── moneyinput.jinja2
├── password.jinja2
├── radio_choice.jinja2
├── readonly
│   ├── checkbox_choice.jinja2
│   ├── checkbox.jinja2
│   ├── checked_input.jinja2
│   ├── checked_password.jinja2
│   ├── dateparts.jinja2
│   ├── file_upload.jinja2
│   ├── form.jinja2
│   ├── mapping_item.jinja2
│   ├── mapping.jinja2
│   ├── password.jinja2
│   ├── radio_choice.jinja2
│   ├── richtext.jinja2
│   ├── select.jinja2
│   ├── sequence_item.jinja2
│   ├── sequence.jinja2
│   ├── textarea.jinja2
│   └── textinput.jinja2
├── richtext.jinja2
├── select.jinja2
├── sequence_item.jinja2
├── sequence.jinja2
├── textarea.jinja2
└── textinput.jinja2

1 directory, 39 files
```

Код 56: Стандартный шаблон textarea.pt из Deform

```
<textarea tal:define="rows rows|field.widget.rows;
                      cols cols|field.widget.cols;
                      css_class css_class|field.widget.css_class;
                      oid oid|field.oid;
                      name name|field.name;
                      style style|field.widget.style"
          tal:attributes="rows rows;
                          cols cols;
                          class string: form-control ${css_class or ''};
                          style style"
          id="${oid}"
          name="${name}">${cstruct}</textarea>
```

Код 57: templates/deform_jinja2/textarea.jinja2 - Переопределенный нами шаблон textarea на Jinja2

```
<textarea
        style="background: green;color:white;"
        {% if field.widget.rows %}
            rows="{{ field.widget.rows }}"
        {% endif %}
        {% if field.widget.cols %}
            cols="{{ field.widget.cols }}"
        {% endif %}
        {% if field.widget.css_class %}
            class="{{ field.widget.css_class }}"
        {% endif %}
            id="{{ field.oid }}"
            name="{{ field.name }}"
        {% if field.description %}
            placeholder="{{ _(field.description) }}"
        {% endif %}>{{ cstruct }}</textarea>
```

Блог

Код 58: forms.py - Форма для создания статьи

```
import deform
import colander

from common import get_csrf_token, get_session

```

(continues on next page)

(продолжение с предыдущей страницы)

```

@colander.deferred
def deferred_csrf_default(node, kw):
    request = kw.get('request')
    session = get_session(request)
    csrf_token = get_csrf_token(session)
    return csrf_token


@colander.deferred
def deferred_csrf_validator(node, kw):
    def validate_csrf_token(node, value):
        request = kw.get('request')
        session = get_session(request)
        csrf_token = get_csrf_token(session)
        if value != csrf_token:
            raise colander.Invalid(node, 'Bad CSRF token')

    return validate_csrf_token


class CSRFSchema(colander.Schema):
    csrf = colander.SchemaNode(colander.String(),
                               default=deferred_csrf_default,
                               validator=deferred_csrf_validator,
                               widget=deform.widget.HiddenWidget(), )


class CreateArticle(CSRFSchema):
    title = colander.SchemaNode(colander.String())
    content = colander.SchemaNode(
        colander.String(),
        widget=deform.widget.TextAreaWidget(
            css_class="blog-form-field__textarea")
    )
```

Код 59: common.py - Функции get_session и get_csrf_token

```
def get_session(request):
    return request.environ.get('paste.session.factory', lambda: {})()


def get_csrf_token(session):
    if 'csrf' not in session:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        from uuid import uuid4
        session['csrf'] = uuid4().hex
    return session['csrf']
```

Код 60: __init__.py - Добавляем механизм сессии в наше WSGI-приложение

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-
# vim:fenc=utf-8
#
# Copyright © 2015 uralbash <root@uralbash.ru>
#
# Distributed under terms of the MIT license.

"""
Simple blog
"""
from paste.auth.basic import AuthBasicHandler

import selector
from views import BlogCreate, BlogDelete, BlogIndex, BlogRead, BlogUpdate


def authfunc(environ, username, password):
    return username == 'admin' and password == '123'


def make_wsgi_app():
    # BasicAuth applications
    create = AuthBasicHandler(BlogCreate, 'www', authfunc)
    update = AuthBasicHandler(BlogUpdate, 'www', authfunc)
    delete = AuthBasicHandler(BlogDelete, 'www', authfunc)

    # URL dispatching middleware
    dispatch = selector.Selector()
    dispatch.add('/', GET=BlogIndex)
    dispatch.prefix = '/article'
    dispatch.add('/add', GET=create, POST=create)
    dispatch.add('/{id:digits}', GET=BlogRead)
    dispatch.add('/{id:digits}/edit', GET=update, POST=update)
    dispatch.add('/{id:digits}/delete', GET=delete)

    # Static files
    from paste.urlparser import StaticURLParser
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    static_app = StaticURLParser("static/")

    from paste import urlmap
    mapping = urlmap.URLMap()
    mapping['/static'] = static_app

    from paste.cascade import Cascade
    app = Cascade([mapping, dispatch])

    return app

if __name__ == '__main__':
    from paste.httpserver import serve
    from paste.session import SessionMiddleware

    app = make_wsgi_app()
    app = SessionMiddleware(app)
    serve(app, host='0.0.0.0', port=8000)
```

Код 61: views.py - Генерация форм в представлениях при помощи Deform

```
import deform
from jinja2 import Environment, FileSystemLoader
from webob import Request, Response

from common import get_csrf_token, get_session
from models import ARTICLES

env = Environment(loader=FileSystemLoader('templates'))


def wsgify(view):
    def wrapped(environ, start_response):
        request = Request(environ)
        app = view(request).response()
        return app(environ, start_response)
    return wrapped


class BaseArticle(object):

    def __init__(self, request):
        self.request = request
        article_id = self.request.environ['wsgiorg.routing_args'][1]['id']
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        (self.index,
         self.article) = next(((i, art) for i, art in enumerate(ARTICLES)
                               if art['id'] == int(article_id)),
                              (None, None))


class BaseArticleForm(object):

    def get_form(self):
        from forms import CreateArticle
        self.session = get_session(self.request)
        self.session['csrf'] = get_csrf_token(self.session)
        schema = CreateArticle().bind(request=self.request)
        submit = deform.Button(name='submit',
                               css_class='blog-form__button')
        self.form = deform.Form(schema, buttons=(submit,))
        return self.form


@wsgify
class BlogIndex(object):

    def __init__(self, request):
        self.page = request.GET.get('page', '1')
        from paginate import Page
        self.paged_articles = Page(
            ARTICLES,
            page=self.page,
            items_per_page=8,
        )

    def response(self):
        return Response(env.get_template('index.html')
                        .render(articles=self.paged_articles))


@wsgify
class BlogCreate(BaseArticleForm):

    def __init__(self, request):
        self.request = request

    def response(self):
        if self.request.method == 'POST':
            submitted = self.request.POST.items()
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            try:
                self.get_form().validate(submitted)
            except deform.ValidationFailure as e:
                return Response(
                    env.get_template('create.html').render(form=e.render()))
            max_id = max([art['id'] for art in ARTICLES])
            ARTICLES.append(
                {'id': max_id+1,
                 'title': self.request.POST['title'],
                 'content': self.request.POST['content']
                 }
            )
            self.session = get_session(self.request).pop('csrf')
            return Response(status=302, location='/')
        return Response(env.get_template('create.html')
                        .render(form=self.get_form().render()))


@wsgify
class BlogRead(BaseArticle):

    def response(self):
        if not self.article:
            return Response(status=404)
        return Response(env.get_template('read.html')
                        .render(article=self.article))


@wsgify
class BlogUpdate(BaseArticle, BaseArticleForm):

    def response(self):
        if self.request.method == 'POST':
            submitted = self.request.POST.items()
            try:
                self.get_form().validate(submitted)
            except deform.ValidationFailure as e:
                return Response(
                    env.get_template('create.html').render(form=e.render()))
            self.article['title'] = self.request.POST['title']
            self.article['content'] = self.request.POST['content']
            self.session = get_session(self.request).pop('csrf')
            return Response(status=302, location='/')
        return Response(
            env.get_template('create.html')
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            .render(form=self.get_form().render(self.article)))


@wsgify
class BlogDelete(BaseArticle):

    def response(self):
        ARTICLES.pop(self.index)
        return Response(status=302, location='/')
```

Код 62: create.html - форма генерируется автоматически

```
{% extends "base.html" %}

{% block title %}Create{% endblock %}

{% block content %}
    <div class="blog__title">
        <a href="/" class="blog__title-link">Simple Blog</a>
        <span class="blog__title-text">Edit</span>
    </div>
    <form action="" method="POST" class="blog-form">
        {{ form }}
    </form>
{% endblock %}
```

Теперь форма имеет валидацию, защиту от CSRF атак и генерируется автоматически при помощи Deform.

1.4.9 Кэширование

Beaker

См.также:

- Beaker

Beaker — это библиотека предназначенная, для кэширования и создания сессии, как в веб-приложениях, так и в чистых *Python* скриптах. Имеет *WSGI*-middleware для *WSGI*-приложений и декоратор (*Декораторы*) для простых приложений.

Конфигурация

http://beaker.readthedocs.org/en/latest/configuration.html

Сессии

Создание

Код 63: common.py - функция `get_session` создает сессию

```
from beaker.session import Session


def get_session(request={}, **kwargs):
    """A shortcut for creating :class:`Session` instance"""
    options = {}
    options.update(**kwargs)
    return Session(request, **options)
```

Код 64: 0.session.py - сохранение данных в сессии

```
# -*- coding: utf-8 -*-
from pprint import pprint
from common import get_session

if '__main__' in __name__:
    """Test if the data is actually persistent across requests"""
    session = get_session()
    session['Suomi'] = 'Kimi Räikkönen'
    session['Great Britain'] = 'Jenson Button'
    session['Deutchland'] = 'Sebastian Vettel'
    session.save()
    print("Session ID: " + session.id)
    pprint(session)

    print
    print("Check session")
    session2 = get_session(id=session.id)
    assert 'Suomi' in session2
    assert 'Great Britain' in session2
    assert 'Deutchland' in session2

    assert session2['Suomi'] == 'Kimi Räikkönen'
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    assert session2['Great Britain'] == 'Jenson Button'
    assert session2['Deutchland'] == 'Sebastian Vettel'
    print("OK")
    print
    assert session2['Russian'] == 'Alexey Popov'
```

Код 65: Результат выполнения программы 0.session.py

```
Session ID: a628f9a5f15e48f99b06d3a710791499
{'Deutchland': 'Sebastian Vettel',
 'Great Britain': 'Jenson Button',
 'Suomi': 'Kimi Räikkönen',
 '_accessed_time': 1475907168.6420162,
 '_creation_time': 1475907168.6420162}
Check session
OK
Traceback (most recent call last):
  File "0.session.py", line 27, in <module>
    assert session2['Russian'] == 'Alexey Popov'
KeyError: 'Russian'
```

Удаление

Код 66: 1.session.delete.py - удаление сессии

```
# -*- coding: utf-8 -*-
from pprint import pprint
from common import get_session

if '__main__' in __name__:
    """Test if the data is actually persistent across requests"""
    session = get_session()
    session['Suomi'] = 'Kimi Räikkönen'
    session['Great Britain'] = 'Jenson Button'
    session['Deutchland'] = 'Sebastian Vettel'
    session.save()
    print("Session ID: " + session.id)
    pprint(session)

    session.delete()
    print
    print("Delete session")
    print("Session ID: " + session.id)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    pprint(session)

    assert 'Suomi' not in session
    assert 'Great Britain' not in session
    assert 'Deutchland' not in session
    assert 'Russian' not in session
```

Код 67: Результат выполнения программы 1.session.delete.py

```
Session ID: f0f6d2a4e02841e49d97ee5ba7ac3985
{'Deutchland': 'Sebastian Vettel',
 'Great Britain': 'Jenson Button',
 'Suomi': 'Kimi Räikkönen',
 '_accessed_time': 1475907306.4009516,
 '_creation_time': 1475907306.4009516}
Delete session
Session ID: f0f6d2a4e02841e49d97ee5ba7ac3985
{}
```

Откат изменений

Код 68: 2.session.revert.py - откат изменений в сессии

```
# -*- coding: utf-8 -*-
from pprint import pprint
from common import get_session

if '__main__' in __name__:
    """Test if the data is actually persistent across requests"""
    session = get_session()
    session['Suomi'] = 'Kimi Räikkönen'
    session['Great Britain'] = 'Jenson Button'
    session['Deutchland'] = 'Sebastian Vettel'
    session.save()
    print("Session ID: " + session.id)
    pprint(session)

    session2 = get_session(id=session.id)
    del session2['Suomi']
    session2['Great Britain'] = 'Lewis Hamilton'
    session2['Deutchland'] = 'Michael Schumacher'
    session2['España'] = 'Fernando Alonso'
```

(continues on next page)

(продолжение с предыдущей страницы)

```
20
21      print
22      print("Modified session")
23      print("Session ID: " + session2.id)
24      pprint(session2)
25
26      session2.revert()
27
28      print
29      print("Revert session")
30      print("Session ID: " + session2.id)
31      pprint(session2)
```

Код 69: Результат выполнения программы 2.session.revert.py

```
1  Session ID: 2887074c36af4aadad354a79bbdb2cc4
2  {'Deutchland': 'Sebastian Vettel',
3   'Great Britain': 'Jenson Button',
4   'Suomi': 'Kimi Räikkönen',
5   '_accessed_time': 1475907344.1409602,
6   '_creation_time': 1475907344.1409602}
7  Modified session
8  Session ID: 2887074c36af4aadad354a79bbdb2cc4
9  {'Deutchland': 'Michael Schumacher',
10  'España': 'Fernando Alonso',
11  'Great Britain': 'Lewis Hamilton',
12  '_accessed_time': 1475907344.1411998,
13  '_creation_time': 1475907344.1409602}
14 Revert session
15 Session ID: 2887074c36af4aadad354a79bbdb2cc4
16 {'Deutchland': 'Sebastian Vettel',
17  'Great Britain': 'Jenson Button',
18  'Suomi': 'Kimi Räikkönen',
19  '_accessed_time': 1475907344.1411998,
20  '_creation_time': 1475907344.1409602}
```

Хранение в файловой системе

По умолчанию сессии хранятся в оперативной памяти и при завершении программы удаляются. Чтобы сессии хранились постоянно, нужно указать место на диске:

Код 70: 3.session.file.py - хранение сессии в файле

```
# -*- coding: utf-8 -*-
from pprint import pprint
from common import get_session

if '__main__' in __name__:
    """Test if the data is actually persistent across requests"""
    session = get_session(data_dir='./cache', type='file')
    session['Suomi'] = 'Kimi Räikkönen'
    session['Great Britain'] = 'Jenson Button'
    session['Deutchland'] = 'Sebastian Vettel'
    session.save()
    print("Session ID: " + session.id)
    pprint(session)

    session2 = get_session(id=session.id, data_dir='./cache', type='file')

    print
    print("File storage session")
    print("Session ID: " + session2.id)
    pprint(session2)
```

Код 71: Результат выполнения программы 3.session.file.py

```
Session ID: 214e9b8724334492a814e5b0b1a797ff
{'Deutchland': 'Sebastian Vettel',
 'Great Britain': 'Jenson Button',
 'Suomi': 'Kimi Räikkönen',
 '_accessed_time': 1475907695.6439698,
 '_creation_time': 1475907695.6439698}
File storage session
Session ID: 214e9b8724334492a814e5b0b1a797ff
{'Deutchland': 'Sebastian Vettel',
 'Great Britain': 'Jenson Button',
 'Suomi': 'Kimi Räikkönen',
 '_accessed_time': 1475907695.6447632,
 '_creation_time': 1475907695.6439698}
```

```
cache/
├── container_file
│   └── 1
│       └── 18
│           └── 18b9908ab7514d8e8d16ae05e1eb09e0.cache
```

(continues on next page)

(продолжение с предыдущей страницы)

```
└── container_file_lock
    └── c
        └── c6
            └── c6e93db703a3eea0207cc7efca5ddd0cbb201919.lock

6 directories, 2 files
```

Код 72: Сериализованный кэш в файле 18b9908ab7514d8e8d16ae05e1eb09e0.cache

```
Ъ}q\X\\\sessionq{}q}(X\\\_accessed_timeqGAXю%МеќИХ\\\SuomiqX\\\Kimi RГ¤ikkГ¶nenqX
\\\Great BritainqX
\\\Jenson ButtonqX\\\_creation_timeqGAXю%МеќИХ
\\\Deutchlandq        X\\\Sebastian Vettelq
us.
```

Хранение в Memcached

См.также:

- Memcached
- http://beaker.readthedocs.org/en/latest/modules/memcached.html

Код 73: 4.session.memcached.py - Хранение сессий в memcached

```
# -*- coding: utf-8 -*-
from pprint import pprint
from common import get_session

if '__main__' in __name__:
    """Test if the data is actually persistent across requests"""
    session = get_session(
        type='ext:memcached',
        url='memcached:11211',
    )
    session['Suomi'] = 'Kimi Räikkönen'
    session['Great Britain'] = 'Jenson Button'
    session['Deutchland'] = 'Sebastian Vettel'
    session.save()
    print("Session ID: " + session.id)
    pprint(session)

```

(continues on next page)

(продолжение с предыдущей страницы)

```
    session2 = get_session(
        id=session.id,
        type='ext:memcached',
        url='memcached:11211'
    )

    print
    print("Memcached storage session")
    print("Session ID: " + session2.id)
    pprint(session2)
```

Код 74: Результат выполнения программы 4.session.memcached.txt

```
Session ID: 8c549978a6984a648d2dadba44becd23
{'Deutchland': 'Sebastian Vettel',
 'Great Britain': 'Jenson Button',
 'Suomi': 'Kimi R\xc3\xa4ikk\xc3\xb6nen',
 '_accessed_time': 1429279819.517085,
 '_creation_time': 1429279819.517085}

Memcached storage session
Session ID: 8c549978a6984a648d2dadba44becd23
{'Deutchland': 'Sebastian Vettel',
 'Great Britain': 'Jenson Button',
 'Suomi': 'Kimi R\xc3\xa4ikk\xc3\xb6nen',
 '_accessed_time': 1429279819.52295,
 '_creation_time': 1429279819.517085}
```

Хранение в Redis

См.также:

- Redis
- https://github.com/didip/beaker_extensions

WSGI-middleware

Код 75: 5.session.wsgi.py - WSGI-meddleware

```
from beaker.middleware import SessionMiddleware

```

(continues on next page)

ие с предыдущей страницы)

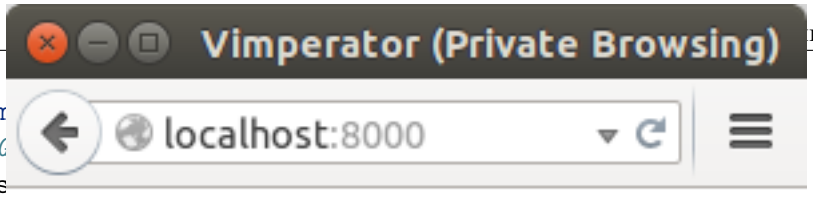


Рис. 45: На страницу заходили 84 раза

```
3
4   def sim
5       # (
6       ses
7
8       # S
9       session['counter'] = session.get('counter', 0) + 1
10      session.save()
11
12      start_response('200 OK', [('Content-type', 'text/plain')])
13      return [
14          str.encode(
15              'Counter value is: {}'.format(session['counter'])
16          )
17      ]
18
19  # Configure the SessionMiddleware
20  session_opts = {
21      'session.type': 'file',
22      'session.data_dir': './cache2/data',
23      'session.lock_dir': './cache2/lock',
24      'session.cookie_expires': True,
25  }
26  wsgi_app = SessionMiddleware(simple_app, session_opts)
27
28
29  if __name__ == '__main__':
30      from paste.httpserver import serve
31
32      serve(wsgi_app, host='0.0.0.0', port=8000)
```

Если хранить кэш на диске, то при значении 13 получим:

```
$ hexdump -v -C cache2/data/container_file/a/a1/a138e686410c40ef9014549d8b339cc9.
↪cache
00000000  80 03 7d 71 00 58 07 00  00 00 73 65 73 73 69 6f  |..}q.X....sessio|
00000010  6e 71 01 7d 71 02 28 58  07 00 00 00 63 6f 75 6e  |nq.}q.(X....coun|
00000020  74 65 72 71 03 4b 0d 58  0e 00 00 00 5f 61 63 63  |terq.K.X...._acc|
00000030  65 73 73 65 64 5f 74 69  6d 65 71 04 47 41 d5 fe  |essed_timeq.GA..|
00000040  24 8e 01 39 c1 58 0e 00  00 00 5f 63 72 65 61 74  |$..9.X...._creat|
00000050  69 6f 6e 5f 74 69 6d 65  71 05 47 41 d5 fe 24 87  |ion_timeq.GA..$.|
00000060  8d 20 eb 75 73 2e                                 |. .us.|
00000066
```

Кэширование

См.также:

- https://github.com/bbangert/beaker/blob/master/tests/test_cache.py

dogpile.cache

См.также:

- https://dogpilecache.readthedocs.org/en/latest/
- https://gist.github.com/sontek/5660624
- http://docs.sqlalchemy.org/en/latest/orm/examples.html?highlight=caching_query#dogpile-caching
- http://techspot.zzzeek.org/2012/04/19/using-beaker-for-caching-why-you-ll-want-to-switch-to-dogpcache/
- http://docs.makotemplates.org/en/latest/caching.html

1.4.10 Базы данных

DB-API 2.0

См.также:

- **PEP 249** - Python DB API
- Р.Сузи «Язык программирования Python»

Несмотря на стандарт *SQL* (ISO/IEC 9075), отдельные СУБД имеют много различий. Чтобы программистам не вникать в реализацию каждой из них, придумали общее API (**PEP 249**) скрывающее эти детали. Любой Python пакет реализующий это API взаимозаменяем.

PEP 249 это **только спецификация**, реализацию которой вам придется выполнить самостоятельно или воспользоваться уже готовой, например для sqlite3.

Также существуют реализации для других СУБД:

- **PostgreSQL** (psycopg2, txpostgres, ...)
- **FireBird** (fdb)
- **MySQL** (mysql-python, PyMySQL, ...)
- **MS SQL** Server (adodbapi, pymssql, mxODBC, pyodbc, ...)

- **Oracle** (cx_Oracle, mxODBC, pyodbc, . . .)
- и другие http://wiki.python.org/moin/DatabaseInterfaces

Большинство из них могут быть установлены стандартным способом:

```
$ pip install psycopg2
$ pip install mysql-python
```

Константы

- **apilevel** - Версия DB-API («1.0» или «2.0»).
- **threadsafety** - Целочисленная константа, описывающая возможности модуля при использовании потоков управления:
- 0 Модуль не поддерживает потоки.
- 1 Потоки могут совместно использовать модуль, но не соединения.
- 2 Потоки могут совместно использовать модуль и соединения.
- 3 Потоки могут совместно использовать модуль, соединения и курсоры. (Под совместным использованием здесь понимается возможность использования упомянутых ресурсов без применения семафоров).
- **paramstyle** - Тип используемых пометок при подстановке параметров. Возможны следующие значения этой константы:
- «format» Форматирование в стиле языка ANSI C (например, «%s», «%i»).
- «pyformat» Использование именованных спецификаторов формата в стиле Python («%(item)s»)
- «qmark» Использование знаков «?» для пометки мест подстановки параметров.
- «numeric» Использование номеров позиций («:1»).
- «named» Использование имен подставляемых параметров («:name»).

Конструктор соединения

Доступ к базе данных осуществляется с помощью объекта-соединения (*connection object*). DB-API-совместимый модуль должен предоставлять функцию-конструктор **connect()** для класса объектов-соединений. Конструктор должен иметь следующие именованные параметры:

- **dsn** - Название источника данных в виде строки
- **user** - Имя пользователя

- **password** - Пароль
- **host** - Адрес хоста, на котором работает СУБД
- **database** - Имя базы данных.

Connection

Объект-соединение, получаемый в результате успешного вызова функции `connect()`, должен иметь следующие методы:

- **.close()** - Закрывает соединение с базой данных.
- **.commit()** - Завершает транзакцию.
- **.rollback()** - Откатывает начатую транзакцию (восстанавливает исходное состояние). Закрытие соединения при незавершенной транзакции автоматически производит откат транзакции.
- **.cursor()** - Возвращает объект-курсор, использующий данное соединение. Если база данных не поддерживает курсоры, модуль сопряжения должен их имитировать.

Cursor

См.также:

- https://ru.wikipedia.org/wiki/\T2A\CYRK\T2A\cyru\T2A\cyrr\T2A\cyrs\T2A\cyro\T2A\cyrr_(\T2A\cyrb\T2A\cyra\T2A\cyrz\T2A\cyrery_\T2A\cyrd\T2A\cyra\T2A\cyrn\T2A\cyrn\T2A\cyrery\T2A\cyrh)

Курсор (*от англ. cursor - CURrrent Set Of Records, текущий набор записей*) служит для работы с результатом запроса. Результатом запроса обычно является одна или несколько прямоугольных таблиц со столбцами-полями и строками-записями. Приложение может читать и обрабатывать полученные таблицы и записи в таблице по одной, поэтому в курсоре хранится информация о текущей таблице и записи. Конкретный курсор в любой момент времени связан с выполнением одной SQL-инструкции.

Настройки

- **arraysize** - Атрибут, равный количеству записей, возвращаемых методом `fetchmany()`. По умолчанию равен 1.
- **setinputsizes(sizes)** - Предопределяет области памяти для параметров, используемых в операциях. Аргумент *sizes* задает последовательность, где каждый элемент соответствует одному входному параметру. Элемент может быть объектом-типом

соответствующего параметра или целым числом, задающим длину строки. Он также может иметь значение None, если о размере входного параметра ничего нельзя сказать заранее или он предполагается очень большим. Метод должен быть вызван до *execute*-методов.

- **setoutputsize(size[, column])** - Устанавливает размер буфера для выходного параметра из столбца с номером *column*. Если *column* не задан, метод устанавливает размер для всех больших выходных параметров. Может использоваться, например, для получения **больших бинарных объектов** (**B** inary **L** arge **O** bject, **BLOB**).

Операции

- **execute(operation[, parameters])** - Исполняет запрос к базе данных или команду СУБД. Параметры (*parameters*) могут быть представлены в принятой в базе данных нотации в соответствии с атрибутом *paramstyle*, описанным выше.
- **executemany(operation, seq_of_parameters)** - Выполняет серию запросов или команд, подставляя параметры в заданный шаблон. Параметр *seq_ of_ parameters* задает последовательность наборов параметров.
- **callproc(procname[, params])** - Вызывает хранимую процедуру *procname* с параметрами из изменчивой последовательности *params*. Хранимая процедура может изменить значения некоторых параметров последовательности. Метод может возвратить результат, доступ к которому осуществляется через *fetch* - методы.

Атрибуты

- **rowcount** - Количество записей, полученных или затронутых в результате выполнения последнего запроса. В случае отсутствия *execute*-запросов или невозможности указать количество записей равен -1.
- **description** - Этот доступный только для чтения атрибут является последовательностью из семиэлементных последовательностей. Каждая из этих последовательностей содержит информацию, описывающую один столбец результата:
  - **name**
  - **type_code**
  - **display_size** (optional)
  - **internal_size** (optional)
  - **precision** (optional)
  - **scale** (optional)

– **null_ok** (optional)

Первые два элемента (имя и тип) обязательны, а вместо остальных (размер для вывода, внутренний размер, точность, масштаб, возможность задания пустого значения) может быть значение *None*. Этот атрибут может быть равным None для операций, не возвращающих значения.

Результат

- **fetchone()** - Возвращает следующую запись (в виде последовательности) из результата запроса или *None* при отсутствии данных.
- **fetchall()** - Возвращает все (или все оставшиеся) записи результата запроса.
- **fetchmany([size])** - Возвращает следующие несколько записей из результатов запроса в виде последовательности последовательностей. Пустая последовательность означает отсутствие данных. Необязательный параметр *size* указывает количество возвращаемых записей (реально возвращаемых записей может быть меньше). По умолчанию size равен атрибуту *arraysize* объекта-курсора.

Типы дынных

DB-API 2.0 предусматривает названия для объектов-типов, используемых для описания полей базы данных:

Объект	Тип
STRING	Строка и символ
BINARY	Бинарный объект
NUMBER	Число
DATETIME	Дата и время
ROWID	Идентификатор записи
None	NULL-значение (отсутствующее значение)

С каждым типом данных (в реальности это - классы) связан конструктор. Совместимый с DB-API модуль должен определять следующие конструкторы:

- **Date** (год, месяц, день) Дата.
- **Time** (час, минута, секунда) Время.
- **Timestamp** (год, месяц, день, час, минута, секунда) Дата-время.
- **DateFromTicks** (secs) Дата в виде числа секунд secs от начала эпохи (1 января 1970 года).
- **TimeFromTicks** (secs) Время, то же.

- **TimestampFromTicks** (secs) Дата-время, то же.
- **Binary** (string) Большой бинарный объект на основании строки string.

Исключения

DB API спецификация требует реализацию классов исключений следующей структуры:

```
StandardError
├──Warning
└──Error
   ├──InterfaceError (a problem with the db api)
   └──DatabaseError (a problem with the database)
      ├──DataError (bad data, values out of range, etc.)
      ├──OperationalError (the db has an issue out of our control)
      ├──IntegrityError
      ├──InternalError
      ├──ProgrammingError (something wrong with the operation)
      └──NotSupportedError (the operation is not supported)
```

SQLite

См.также:

- SQLite
- https://docs.python.org/3.5/library/sqlite3.html
- https://www.sqlite.org/quickstart.html

SQLite - это БД которая хранит базу в одном файле и не требует отдельного процесса для запуска, при этом использует не стандартный вариант языка SQL. Такой подход позволяет встроить `sqlite` прямо в программу, без необходимости установки сервера БД. Пример использования `sqlite` в `C++`:

```
#include <stdio.h>
#include <sqlite3.h>

static int callback(void *NotUsed, int argc, char **argv, char **azColName){
  int i;
  for(i=0; i<argc; i++){
    printf("%s = %s\n", azColName[i], argv[i] ? argv[i] : "NULL");
  }
  printf("\n");
  return 0;
}
```

(continues on next page)

(продолжение с предыдущей страницы)

```
int main(int argc, char **argv){
  sqlite3 *db;
  char *zErrMsg = 0;
  int rc;

  if( argc!=3 ){
    fprintf(stderr, "Usage: %s DATABASE SQL-STATEMENT\n", argv[0]);
    return(1);
  }
  rc = sqlite3_open(argv[1], &db);
  if( rc ){
    fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
    sqlite3_close(db);
    return(1);
  }
  rc = sqlite3_exec(db, argv[2], callback, 0, &zErrMsg);
  if( rc!=SQLITE_OK ){
    fprintf(stderr, "SQL error: %s\n", zErrMsg);
    sqlite3_free(zErrMsg);
  }
  sqlite3_close(db);
  return 0;
}
```

SQLite можно использовать для хранения внутренних данных программы (например FireFox хранит куки в sqlite) или для создания прототипа приложения, а затем портировать код в крупную БД типа Postgres.

Модуль `sqlite3` совместим с DB-API 2.0 спецификацией, описаной в **PEP 249**.

Чтобы использовать этот модуль, вы должны сначала создать объект `sqlite3.Connection` который представляет базу данных.

```
import sqlite3
conn = sqlite3.connect('example.sqlite')
```

Также можно создать БД в ОЗУ при помощи специального имени `:memory:`.

```
import sqlite3
conn = sqlite3.connect(':memory:')
```

После создания объекта `sqlite3.Connection`, можно создать объект `sqlite3.Cursor` и вызвать метод `sqlite3.Cursor.execute()` для выполнения SQL запросов.

```
c = conn.cursor()

# Создание таблицы
c.execute('''CREATE TABLE stocks
             (date text, trans text, symbol text, qty real, price real)''')

# Добавление записи
c.execute("INSERT INTO stocks VALUES ('2006-01-05','BUY','RHAT',100,35.14)")

# Сохранение (commit) изменений
conn.commit()

# Закрытие соединения.
# Если изменения не были сохранены (метод commit), то данные пропадут.
conn.close()
```

Для «экранирования» данных используйте ? заместо %s:

```
# Никогда так не делайте -- не безопасно!
symbol = 'RHAT'
c.execute("SELECT * FROM stocks WHERE symbol = '%s'" % symbol)

# Правильно
t = ('RHAT',)
c.execute('SELECT * FROM stocks WHERE symbol=?', t)
print(c.fetchone())

# Запись сразу нескольких объектов за раз
purchases = [('2006-03-28', 'BUY', 'IBM', 1000, 45.00),
             ('2006-04-05', 'BUY', 'MSFT', 1000, 72.00),
             ('2006-04-06', 'SELL', 'IBM', 500, 53.00),
            ]
c.executemany('INSERT INTO stocks VALUES (?,?,?,?,?)', purchases)
```

Чтение данных:

```
>>> for row in c.execute('SELECT * FROM stocks ORDER BY price'):
        print(row)

('2006-01-05', 'BUY', 'RHAT', 100, 35.14)
('2006-03-28', 'BUY', 'IBM', 1000, 45.0)
('2006-04-06', 'SELL', 'IBM', 500, 53.0)
('2006-04-05', 'BUY', 'MSFT', 1000, 72.0)
```

Postgres

См.также:

- PostgreSQL
- http://initd.org/psycopg/
- http://initd.org/psycopg/docs/usage.html

Psycopg - это самая популярная библиотека для PostgreSQL в языке программирования python. Основные преимущества ее, это реализация DB-API 2.0 спецификации и потокобезопасность. Написана на Си, как обертка над libpq.

```
>>> import psycopg2

# Подключение к существующей базе
>>> conn = psycopg2.connect("dbname=test user=postgres")

# Open a cursor to perform database operations
>>> cur = conn.cursor()

# Выполнение SQL запроса: создает новую базу
>>> cur.execute("CREATE TABLE test (id serial PRIMARY KEY, num integer, data␣
↪varchar);")

# Pass data to fill a query placeholders and let Psycopg perform
# the correct conversion (no more SQL injections!)
>>> cur.execute("INSERT INTO test (num, data) VALUES (%s, %s)",
...      (100, "abc'def"))

# Query the database and obtain data as Python objects
>>> cur.execute("SELECT * FROM test;")
>>> cur.fetchone()
(1, 100, "abc'def")

# Make the changes to the database persistent
>>> conn.commit()

# Close communication with the database
>>> cur.close()
>>> conn.close()
```

SQLAlchemy ORM

См.также:

- SQLAlchemy

- https://ru.wikipedia.org/wiki/ORM
- https://ru.wikipedia.org/wiki/SQLAlchemy
- http://pajhome.org.uk/blog/10_reasons_to_love_sqlalchemy.html

ORM (*англ. object-relational mapping, рус. объектно-реляционное отображение*) — технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных». Существуют как проприетарные, так и свободные реализации этой технологии.

SQLAlchemy — это библиотека на языке Python для работы с реляционными СУБД с применением технологии ORM. Служит для синхронизации объектов Python и записей реляционной базы данных. SQLAlchemy позволяет описывать структуры баз данных и способы взаимодействия с ними на языке Python без использования SQL.

Преимущества использования

Использование SQLAlchemy для автоматической генерации SQL-кода имеет несколько преимуществ по сравнению с ручным написанием SQL:

- **Безопасность**. Параметры запросов экранируются, что делает атаки типа внедрение SQL-кода маловероятными.
- **Производительность**. Повышается вероятность повторного использования запроса к серверу базы данных, что может позволить ему в некоторых случаях применить повторно план выполнения запроса.
- **Переносимость**. SQLAlchemy, при должном подходе, позволяет писать код на Python, совместимый с несколькими back-end СУБД. Несмотря на стандартизацию языка SQL, между базами данных имеются различия в его реализации, абстрагироваться от которых и помогает SQLAlchemy.

Пример

Простейший пример с использованием SQLite в оперативной памяти:

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite:///:memory:')
>>> engine.execute("select 'Hello, World!'").scalar()
u'Hello, World!'
```

Базовые понятия

См.также:

- https://ru.wikibooks.org/wiki/SQLAlchemy
- https://bitbucket.org/zzzeek/pycon2013_student_package/

Соединение (engine)

Создадим две таблицы и добавим сотрудников (`employee`).

Код 76: 2.sqlalchemy/0.simple.example.py

```
from sqlalchemy import create_engine
import os

if os.path.exists("some.db"):
    os.remove("some.db")
e = create_engine("sqlite:///some.db")
e.execute("""
    create table employee (
        emp_id integer primary key,
        emp_name varchar
    )
""")

e.execute("""
    create table employee_of_month (
        emp_id integer primary key,
        emp_name varchar
    )
""")

e.execute("""insert into employee(emp_name) values ('ed')""")
e.execute("""insert into employee(emp_name) values ('jack')""")
e.execute("""insert into employee(emp_name) values ('fred')""")
```

SQL запрос:

```
 create table employee (
     emp_id integer primary key,
     emp_name varchar
 );

 create table employee_of_month (
     emp_id integer primary key,
     emp_name varchar
 );
insert into employee(emp_name) values ('ed');
insert into employee(emp_name) values ('jack');
insert into employee(emp_name) values ('fred');
```

```
$ sqlite3 some.db
sqlite> .tables
employee           employee_of_month
sqlite> SELECT * FROM employee;
1|ed
2|jack
3|fred
sqlite> SELECT * FROM employee_of_month;
sqlite>
```

create_engine

Функция `sqlalchemy.create_engine()` создает новый экземпляр класса `sqlalchemy.engine.Engine` который предоставляет подключение к базе данных.

```
1 from sqlalchemy import create_engine
2 engine = create_engine("sqlite:///some.db")
```

execute

Метод `sqlalchemy.engine.Engine.execute()` выполняет *SQL* запрос в нашем соединении и возвращает объект класса `sqlalchemy.engine.ResultProxy`.

```
1 result = engine.execute(
2             "select emp_id, emp_name from "
3             "employee where emp_id=:emp_id",
4             emp_id=3)
```

В результате выполнится следующий *SQL* запрос:

```
select emp_id, emp_name from employee where emp_id=3;
```

fetchone

Объект класса `sqlalchemy.engine.ResultProxy` реализует некоторые методы из спецификации DB-API 2.0:

- `sqlalchemy.engine.ResultProxy.fetchone()`
- `sqlalchemy.engine.ResultProxy.fetchmany()`
- `sqlalchemy.engine.ResultProxy.fetchall()`

Результат запроса похож на список.

```
row = result.fetchone()
print(row)  # (3, u'fred')
```

Но также выполняет функции словаря.

```
print(row['emp_name'])  # u'fred'
```

fetchall

Объект класса `sqlalchemy.engine.ResultProxy` является итератором, поэтому можно получить список всех строк в цикле.

```
result = engine.execute("select * from employee")
for row in result:
    print(row)
# (1, u'ed')
# (2, u'jack')
# (3, u'fred')
```

Тоже самое делает функция `sqlalchemy.engine.ResultProxy.fetchall()`

```
result = engine.execute("select * from employee")
print(result.fetchall())
# [(1, u'ed'), (2, u'jack'), (3, u'fred')]
```

close

Соединение закроется автоматически после выполнения *SQL* запроса, но можно это сделать и вручную, при помощи метода `sqlalchemy.engine.ResultProxy.close()`

Код 77: Закрытие соединения вручную

```
result.close()
```

Транзакции

`sqlalchemy.engine.Engine.execute()` автоматически подтверждает транзакцию в текущем соединении (выполняет COMMIT)

```
engine.execute("insert into employee_of_month (emp_name) values (:emp_name)",
               emp_name='fred')
```

Мы можем контролировать соединение используя метод `sqlalchemy.engine.Engine.connect()`

```
conn = engine.connect()
result = conn.execute("select * from employee")
result.fetchall()
conn.close()
```

Он также дает возможность управлять транзакциями. Транзакция является объектом класса `sqlalchemy.engine.Transaction` и содержит в себе следующие методы:

- `sqlalchemy.engine.Transaction.close()` - выполняет `rollback`
- `sqlalchemy.engine.Transaction.commit()` - подтверждает транзакцию
- `sqlalchemy.engine.Transaction.rollback()` - отменяет транзакцию

Метод `sqlalchemy.engine.Transaction.commit()` позволяет вам вручную подтвердить транзакцию.

Код 78: Подтверждение транзакции вручную

```
conn = engine.connect()
trans = conn.begin()
conn.execute("insert into employee (emp_name) values (:emp_name)", emp_name="wendy
↪")
conn.execute("update employee_of_month set emp_name = :emp_name", emp_name="wendy")
trans.commit()
conn.close()
```

SQL запрос:

```
BEGIN;
insert into employee (emp_name) values 'wendy';
update employee_of_month set emp_name = 'wendy';
COMMIT;
```

Код 79: Данные успешно записанны в БД

```
print(engine.execute("select * from employee").fetchall())
print(engine.execute("select * from employee_of_month").fetchall())
# [(1, u'ed'), (2, u'jack'), (3, u'fred'), (4, u'wendy')]
# [(1, u'wendy')]
```

Метод `sqlalchemy.engine.Transaction.rollback()` отменяет транзакцию, откатывая данные к начальному состоянию.

Код 80: Отмена транзакции вручную

```
conn = engine.connect()
trans = conn.begin()
conn.execute("insert into employee (emp_name) values (:emp_name)", emp_name="wendy
↪")
conn.execute("update employee_of_month set emp_name = :emp_name", emp_name="wendy")
trans.rollback()
conn.close()
```

SQL запрос:

```
BEGIN;
insert into employee (emp_name) values 'wendy';
update employee_of_month set emp_name = 'wendy';
ROLLBACK;
```

Код 81: Данные не изменились

```
print(engine.execute("select * from employee").fetchall())
print(engine.execute("select * from employee_of_month").fetchall())
# [(1, u'ed'), (2, u'jack'), (3, u'fred')]
# [(1, u'fred')]
```

Контекстный менеджер немного упрощает это процесс:

```
with engine.begin() as conn:
    conn.execute("insert into employee (emp_name) values (:emp_name)", emp_name=
↪"mary")
    conn.execute("update employee_of_month set emp_name = :emp_name", emp_name=
↪"mary")
```

Полный пример

Код 82: 2.sqlalchemy/1.engine.py

```
# slide:: s
from sqlalchemy import create_engine
import os

if os.path.exists("some.db"):
    os.remove("some.db")
e = create_engine("sqlite:///some.db")
e.execute("""
    create table employee (
        emp_id integer primary key,
        emp_name varchar
    )
""")

e.execute("""
    create table employee_of_month (
        emp_id integer primary key,
        emp_name varchar
    )
""")

e.execute("""insert into employee(emp_name) values ('ed')""")
e.execute("""insert into employee(emp_name) values ('jack')""")
e.execute("""insert into employee(emp_name) values ('fred')""")

# # slide::
# ## title:: Engine Basics
# create_engine() builds a *factory* for database connections.

from sqlalchemy import create_engine

engine = create_engine("sqlite:///some.db")

# ## slide:: p
# Engine features an *execute()* method that will run a query on
# a connection for us.

result = engine.execute(
    "select emp_id, emp_name from "
    "employee where emp_id=:emp_id",
    emp_id=3)

# ## slide::
# the result object we get back features methods like fetchone(),
# fetchall()
row = result.fetchone()

# ## slide:: j
# the row looks like a tuple
row

```

Метаданные (metadata)

См.также:

- http://rus-linux.net/MyLDP/BOOKS/Architecture-Open-Source-Applications/Vol-2/sqlalchemy-04.html

Для описания структуры базы данных используют 3 основных класса:

- `sqlalchemy.schema.Table` - таблица
- `sqlalchemy.schema.Column` - поле таблицы
- `sqlalchemy.schema.MetaData` - список таблиц

А также типы полей описанные в модуле `sqlalchemy.types`:

- `sqlalchemy.types.Integer`
- `sqlalchemy.types.String`
- `sqlalchemy.types.Text`
- И другие

```
from sqlalchemy import MetaData
from sqlalchemy import Table, Column
from sqlalchemy import Integer, String

metadata = MetaData()
user_table = Table('user', metadata,
                Column('id', Integer, primary_key=True),
                Column('name', String),
                Column('fullname', String)
             )
```

Создание таблиц таким образом описывают структуру базы данных независимо от объектно-реляционного отображения. Объект `sqlalchemy.schema.Table` представляет имя и другие атрибуты текущей таблицы. Его коллекция объектов `Column` представляет информацию об именах и типах для определенных столбцов таблицы.

Дополнительно в описание схемы базы данных можно включить внешние ключи, индексы, последовательности и т.д.:

- `sqlalchemy.schema.ForeignKey` - внешние ключи
- `sqlalchemy.schema.Index` - индексы
- `sqlalchemy.schema.Sequence` - последовательности

Вся информация о таблицах базы данных складывается в объект класса `sqlalchemy.schema.MetaData`. Получить список таблиц можно при помощи атрибута `sqlalchemy.schema.MetaData.tables`.

Объекты `Table` и `Column` уникальны по сравнению со всеми остальными объектами из пакета для работы со схемами, так как они используют двойное наследование от объектов из пакетов `sqlalchemy.schema` и `sqlalchemy.sql.expression`, работая не только как конструкции уровня обработки схем, но также и как синтаксические единицы языка для создания выражений SQL. Это отношение проиллюстрировано на `sqlalchemy_table_crossover`.

Table

```
>>> user_table
Table('user', MetaData(bind=None),
Column('id', Integer(), table=<user>, primary_key=True, nullable=False),
Column('name', String(), table=<user>),
Column('fullname', String(), table=<user>), schema=None)
```

Имя таблицы

```
>>> user_table.name
'user'
```

Поля таблицы

Поля таблицы хранятся в списке `sqlalchemy.schema.Table.columns` или его более коротком варианте `sqlalchemy.schema.Table.c`.

```
>>> user_table.c
<sqlalchemy.sql.expression.ImmutableColumnCollection object at 0x7fee7d18c450>
>>> print(user_table.c)
['user.id', 'user.name', 'user.fullname']
>>> user_table.c.id
Column('id', Integer(), table=<user>, primary_key=True, nullable=False)
>>> user_table.c.name
Column('name', String(), table=<user>)
>>> user_table.c.fullname
Column('fullname', String(), table=<user>)
```

Сами поля тоже содержат информацию о себе, например в атрибутах `name` и `type`.

```
>>> user_table.c.id
Column('id', Integer(), table=<user>, primary_key=True, nullable=False)
>>> user_table.c.id.name
'id'
>>> user_table.c.id.type
Integer()
>>> user_table.c.name
Column('name', String(), table=<user>)
>>> user_table.c.name.name
```

Скалярные запросы

> Скаляр (от лат. scalaris — ступенчатый) — величина, каждое значение которой может быть выражено одним числом. В математике под «числами» могут подразумеваться элементы произвольного поля, тогда когда в физике имеются в виду действительные или комплексные числа. О функции, принимающей скалярные значения, говорят как о скалярной функции.
>
> —WikiPedia

Скалярный SELECT вернет только одно поле одной строки.

```
>>> address_sel = select([
...                 func.count(address_table.c.id)
...                 ]).\
...                 where(user_table.c.id == address_table.c.user_id)
>>> print(address_sel)
SELECT count(address.id) AS count_1
FROM address, "user"
WHERE "user".id = address.user_id
```

Чтобы его вызвать в подзапросе нужно использовать метод `sqlalchemy.sql.expression.Select.as_scalar()`

```
>>> select_stmt = select([user_table.c.username, address_sel.as_scalar()])
>>> conn.execute(select_stmt).fetchall()

[SQL]: SELECT user.username, (SELECT count(address.id) AS count_1
FROM address
WHERE user.id = address.user_id) AS anon_1
FROM user
[SQL]: ()
[(u'ed', 2), (u'jack', 1), (u'wendy', 1), (u'jack', 0), (u'wendy', 0)]
```

UPDATE

`sqlalchemy.schema.Table.update()`

```
>>> update_stmt = address_table.update().\
...                     values(email_address="jack@msn.com").\
...                     where(address_table.c.email_address == "jack@yahoo.com")
>>> result = conn.execute(update_stmt)

[SQL]: UPDATE address SET email_address=? WHERE address.email_address = ?
[SQL]: ('jack@msn.com', 'jack@yahoo.com')
[SQL]: COMMIT
```

UPDATE запрос значение которого строится из значения полей текущей записи

```
>>> update_stmt = user_table.update().\
...                       values(fullname=user_table.c.username +
...                              " " + user_table.c.fullname)
>>> result = conn.execute(update_stmt)

[SQL]: UPDATE user SET fullname=(user.username || ? || user.fullname)
[SQL]: (' ',)
[SQL]: COMMIT

>>> conn.execute(select([user_table])).fetchall()
[SQL]: SELECT user.id, user.username, user.fullname
FROM user
[SQL]: ()
[(1, u'ed', u'ed Ed Jones'), (2, u'jack', u'jack Jack Burger'), (3, u'wendy', u
↪'wendy Wendy Weathersmith'), (4, u'jack', u'jack Jack Burger'), (5, u'wendy', u
↪'wendy Wendy Weathersmith')]
```

DELETE

sqlalchemy.schema.Table.delete()

```
>>> delete_stmt = address_table.delete().\
...                   where(address_table.c.email_address == "ed@ed.com")
>>> result = conn.execute(delete_stmt)

[SQL]: DELETE FROM address WHERE address.email_address = ?
[SQL]: ('ed@ed.com',)
[SQL]: COMMIT
```

Количество удаленных строк (применимо и для UPDATE).

```
>>> result.rowcount
1
```

Полный пример

Код 84: 2.sqlalchemy/3.sql_expression.py

```
# ## slide::
# ## title:: SQL Expression Language
# We begin with a Table object
```

(continues on next page)

(продолжение с предыдущей страницы)

```
from sqlalchemy import MetaData, Table, Column, String, Integer

metadata = MetaData()
user_table = Table('user', metadata, Column('id', Integer,
                                            primary_key=True),
                   Column('username', String(50)),
                   Column('fullname', String(50)))

# ## slide:: p
# new SQLite database and generate the table.

from sqlalchemy import create_engine
engine = create_engine("sqlite://")
metadata.create_all(engine)

# ## slide::
# as we saw earlier, Table has a collection of Column objects,
# which we can access via table.c.<columnname>

user_table.c.username

# ## slide::
# Column is part of a class known as "ColumnElement",
# which exhibit custom Python expression behavior.

user_table.c.username == 'ed'

# ## slide:: i
# They become SQL when evaluated as a string.
str(user_table.c.username == 'ed')

# ## slide::
# ColumnElements can be further combined to produce more ColumnElements

print((user_table.c.username == 'ed') | (user_table.c.username == 'jack'))

# ## slide::
# OR and AND are available with |, &, or or_() and and_()

from sqlalchemy import and_, or_

print(and_(user_table.c.fullname == 'ed jones',
           or_(user_table.c.username == 'ed',
               user_table.c.username == 'jack')))

```

(continues on next page)

(продолжение с предыдущей страницы)

```
# ## slide::
# comparison operators

print(user_table.c.id > 5)

# ## slide::
# Compare to None produces IS NULL

print(user_table.c.fullname == None)  # noqa

# ## slide::
# "+" might mean "addition"....

print(user_table.c.id + 5)

# ## slide:: i
# ...or might mean "string concatenation"

print(user_table.c.fullname + "some name")

# ## slide::
# an IN

print(user_table.c.username.in_(["wendy", "mary", "ed"]))

# ## slide::
# Expressions produce different strings according to *dialect*
# objects.

expression = user_table.c.username == 'ed'

# ## slide:: i
# MySQL....
from sqlalchemy.dialects import mysql
print(expression.compile(dialect=mysql.dialect()))

# ## slide:: i
# PostgreSQL...
from sqlalchemy.dialects import postgresql
print(expression.compile(dialect=postgresql.dialect()))

# ## slide::
# the Compiled object also converts literal values to "bound"
# parameters.

```

(continues on next page)

(продолжение с предыдущей страницы)

```
compiled = expression.compile()
compiled.params

# ## slide::
# The "bound" parameters are extracted when we execute()

engine.execute(user_table.select().where(user_table.c.username == 'ed'))

# ## slide::
# ## title:: Exercises
# Produce these expressions using "user_table.c.fullname",
# "user_table.c.id", and "user_table.c.username":
#
# 1. user.fullname = 'ed'
#
# 2. user.fullname = 'ed' AND user.id > 5
#
# 3. user.username = 'edward' OR (user.fullname = 'ed' AND user.id > 5)
#

# ## slide:: p
# we can insert data using the insert() construct

insert_stmt = user_table.insert().values(username='ed', fullname='Ed Jones')

conn = engine.connect()
result = conn.execute(insert_stmt)

# ## slide:: i
# executing an insert() gives us the "last inserted id"
result.inserted_primary_key

# ## slide:: p
# insert() and other DML can run multiple parameters at once.

conn.execute(user_table.insert(),
             [{'username': 'jack',
               'fullname': 'Jack Burger'},
              {'username': 'wendy',
               'fullname': 'Wendy Weathersmith'}])

# ## slide:: p
# select() is used to produce any SELECT statement.

from sqlalchemy import select
```

(continues on next page)

(продолжение с предыдущей страницы)

```
select_stmt = select([user_table.c.username, user_table.c.fullname]).\
    where(user_table.c.username == 'ed')
result = conn.execute(select_stmt)
for row in result:
    print(row)

# ## slide:: p
# select all columns from a table

select_stmt = select([user_table])
conn.execute(select_stmt).fetchall()

# ## slide:: p
# specify a WHERE clause

select_stmt = select([user_table]).\
    where(
        or_(
            user_table.c.username == 'ed',
            user_table.c.username == 'wendy'
        )
    )
conn.execute(select_stmt).fetchall()

# ## slide:: p
# specify multiple WHERE, will be joined by AND

select_stmt = select([user_table]).\
    where(user_table.c.username == 'ed').\
    where(user_table.c.fullname == 'ed jones')
conn.execute(select_stmt).fetchall()

# ## slide:: p
# ordering is applied using order_by()

select_stmt = select([user_table]).\
    order_by(user_table.c.username)
print(conn.execute(select_stmt).fetchall())

# ## slide::
# ## title:: Exercises
# 1. use user_table.insert() and "r = conn.execute()" to emit this
# statement:
#
# INSERT INTO user (username, fullname) VALUES ('dilbert', 'Dilbert Jones')
```

(continues on next page)

(продолжение с предыдущей страницы)

```
#
# 2. What is the value of 'user.id' for the above INSERT statement?
#
# 3. Using "select([user_table])", execute this SELECT:
#
# SELECT id, username, fullname FROM user WHERE username = 'wendy' OR
#   username = 'dilbert' ORDER BY fullname
#
#

# ## slide:: p
# ## title:: Joins / Foreign Keys
# We create a new table to illustrate multi-table operations
from sqlalchemy import ForeignKey

address_table = Table("address", metadata, Column('id', Integer,
                                                  primary_key=True),
                      Column('user_id', Integer, ForeignKey('user.id'),
                             nullable=False),
                      Column('email_address', String(100),
                             nullable=False))
metadata.create_all(engine)

# ## slide:: p
# data
conn.execute(address_table.insert(),
             [{"user_id": 1,
               "email_address": "ed@ed.com"},
              {"user_id": 1,
               "email_address": "ed@gmail.com"},
              {"user_id": 2,
               "email_address": "jack@yahoo.com"},
              {"user_id": 3,
               "email_address": "wendy@gmail.com"}, ])

# ## slide::
# two Table objects can be joined using join()
#
# <left>.join(<right>, [<onclause>]).

join_obj = user_table.join(address_table,
                           user_table.c.id == address_table.c.user_id)
print(join_obj)

# ## slide::
```

(continues on next page)

(продолжение с предыдущей страницы)

```
# ForeignKey allows the join() to figure out the ON clause automatically

join_obj = user_table.join(address_table)
print(join_obj)

# ## slide:: pi
# to SELECT from a JOIN, use select_from()

select_stmt = select([user_table, address_table]).select_from(join_obj)
conn.execute(select_stmt).fetchall()

# ## slide::
# the select() object is a "selectable" just like Table.
# it has a .c. attribute also.

select_stmt = select([user_table]).where(user_table.c.username == 'ed')

print(select([select_stmt.c.username]).where(select_stmt.c.username == 'ed'))

# ## slide::
# In SQL, a "subquery" is usually an alias() of a select()

select_alias = select_stmt.alias()
print(select([select_alias.c.username]).where(select_alias.c.username == 'ed'))

# ## slide::
# A subquery against "address" counts addresses per user:

from sqlalchemy import func
address_subq = select([
    address_table.c.user_id,
    func.count(address_table.c.id).label('count')
]).\
    group_by(address_table.c.user_id).\
    alias()
print(address_subq)

# ## slide:: i
# we use join() to link the alias() with another select()

username_plus_count = select([
    user_table.c.username, address_subq.c.count
]).select_from(user_table.join(address_subq)).order_by(user_table.c.username)

# ## slide:: i
```

(continues on next page)

(продолжение с предыдущей страницы)

```
274
275 conn.execute(username_plus_count).fetchall()
276
277 # ## slide::
278 # ## title:: Exercises
279 # Produce this SELECT:
280 #
281 # SELECT fullname, email_address FROM user JOIN address
282 #   ON user.id = address.user_id WHERE username='ed'
283 #   ORDER BY email_address
284 #
285
286 # ## slide::
287 # ## title:: Scalar selects, updates, deletes
288 # a *scalar select* returns exactly one row and one column
289
290 address_sel = select([
291     func.count(address_table.c.id)
292 ]).\
293     where(user_table.c.id == address_table.c.user_id)
294 print(address_sel)
295
296 # ## slide:: ip
297 # scalar selects can be used in column expressions,
298 # specify it using as_scalar()
299
300 select_stmt = select([user_table.c.username, address_sel.as_scalar()])
301 conn.execute(select_stmt).fetchall()
302
303 # ## slide:: p
304 # to round out INSERT and SELECT, this is an UPDATE
305
306 update_stmt = address_table.update().\
307     values(email_address="jack@msn.com").\
308     where(address_table.c.email_address == "jack@yahoo.com")
309
310 result = conn.execute(update_stmt)
311
312 # ## slide:: p
313 # an UPDATE can also use expressions based on other columns
314
315 update_stmt = user_table.update().\
316     values(fullname=user_table.c.username +
317            " " + user_table.c.fullname)
318
```

(continues on next page)

(продолжение с предыдущей страницы)

```
result = conn.execute(update_stmt)

# ## slide:: i
conn.execute(select([user_table])).fetchall()

# ## slide:: p
# and this is a DELETE

delete_stmt = address_table.delete().\
    where(address_table.c.email_address == "ed@ed.com")

result = conn.execute(delete_stmt)

# ## slide:: i
# UPDATE and DELETE have a "rowcount", number of rows matched
# by the WHERE clause.
result.rowcount

# ## slide::
# ## title:: Exercises
# 1. Execute this UPDATE - keep the "result" that's returned
#
#    UPDATE user SET fullname='Ed Jones' where username='ed'
#
# 2. how many rows did the above statement update?
#
# 3. Tricky bonus!  Combine update() along with select().as_scalar()
#    to execute this UPDATE:
#
#    UPDATE user SET fullname=fullname ||
#        (select email_address FROM address WHERE user_id=user.id)
#       WHERE username IN ('jack', 'wendy')
#
# ## slide::
```

ORM (объектно-реляционное отображение)

См.также:

- http://docs.sqlalchemy.org/en/latest/orm/index.html
- http://rus-linux.net/MyLDP/BOOKS/Architecture-Open-Source-Applications/Vol-2/sqlalchemy-06.html

Переключим наше внимание на объектно-реляционное отображение. Первой целью яв-

ляется использование описанной нами системы таблиц метаданных для предоставления возможности переноса функций заданного пользователем класса на коллекцию столбцов в таблице базы данных. Второй целью является предоставление возможности описания отношений между заданными пользователем классами, которые будут основываться на отношениях между таблицами в базе данных.

В SQLAlchemy такая связь называется «отображением», что соответствует широко известному шаблону проектирования с названием «DataMapper», описанному в книге Martin Flower с названием Patterns of Enterprise Application Architecture.

В целом, система объектно-реляционного отображения SQLAlchemy была разработана с применением большого количества приемов, которые описал в своей книге Martin Flower. Она также подверглась значительному влиянию со стороны известной системы реляционного отображения Hibernate для языка программирования Java и продукта SQLObject для языка программирования Python от Ian Bicking.

Классическое представление классов таблиц

См.также:

- http://docs.sqlalchemy.org/en/latest/orm/mapping_styles.html#classical-mappings

Объект класса `sqlalchemy.orm.mapper.Mapper` связывает колонки из схемы таблицы и атрибуты Python класса.

Код 85: 2.sqlalchemy/4.orm.mapper.classic.py

```
# -*- coding: utf-8 -*-
from sqlalchemy import Table, MetaData, Column, Integer, String, ForeignKey
from sqlalchemy.orm import mapper, relationship

metadata = MetaData()

user = Table('user', metadata,
             Column('id', Integer, primary_key=True),
             Column('name', String(50)),
             Column('fullname', String(50)),
             Column('password', String(12))
             )


address = Table('address', metadata,
                Column('id', Integer, primary_key=True),
                Column('user_id', Integer, ForeignKey('user.id')),
                Column('email_address', String(50))
                )
```

(continues on next page)

(продолжение с предыдущей страницы)

```
20
21
22  class User(object):
23      pass
24
25
26  class Address(object):
27      pass
28
29  print(dir(User))
30
31  mapper(
32      User, user,
33      properties={
34          'addresses': relationship(Address, backref='user',
35                                    order_by=address.c.id)
36      })
37
38  print(dir(User))
39
40  mapper(Address, address)
```

```
>>> ['__class__', '__delattr__', '__dict__', '__doc__', '__format__',
'__getattribute__', '__hash__', '__init__', '__module__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__',
'__str__', '__subclasshook__', '__weakref__']

>>> ['__class__', '__delattr__', '__dict__', '__doc__', '__format__',
'__getattribute__', '__hash__', '__init__', '__module__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__',
'__str__', '__subclasshook__', '__weakref__', '_sa_class_manager',
'addresses', 'fullname', 'id', 'name', 'password']
```

Декларативное представление классов таблиц

См.также:

- http://docs.sqlalchemy.org/en/latest/orm/mapping_styles.html#declarative-mapping

Каждый класс, представляющий таблицу в БД, должен наследоваться от базового класса который создается при помощи функции `sqlalchemy.ext.declarative.declarative_base()`.

```
>>> from sqlalchemy.ext.declarative import declarative_base
>>> Base = declarative_base()

>>> from sqlalchemy import Column, Integer, String

>>> class User(Base):
...     __tablename__ = 'user'

...     id = Column(Integer, primary_key=True)
...     name = Column(String)
...     fullname = Column(String)

...     def __repr__(self):
...         return "<User(%r, %r)>" % (
...                 self.name, self.fullname
...             )
```

Схема таблицы

Для каждого класса унаследованного от базового автоматически создается схема таблицы (объект класса `sqlalchemy.schema.Table`) и привязывается к нему через атрибут `__table__`.

```
>>> User.__table__
Table('user', MetaData(bind=None), Column('id', Integer(), table=<user>,
primary_key=True, nullable=False), Column('name', String(), table=<user>),
Column('fullname', String(), table=<user>), schema=None)
```

MetaData

Любой класс таблицы автоматически ассоциируется с объектом `sqlalchemy.schema.Table`, который автоматически добавляется в список `sqlalchemy.schema.MetaData`. Базовый класс `Base`, созданный при помощи функции `sqlalchemy.ext.declarative.declarative_base()`, является более высокоуровневой абстракцией над `sqlalchemy.schema.MetaData`, которая позволяет описывать таблицы декларативным способом. Таким образом все классы-таблицы имеют свою схему, которая хранится в атрибуте `metadata` базового класса `Base`:

```
>>> Base.metadata
MetaData(bind=None)
>>> Base.metadata.tables.items()
[('user', Table('user', MetaData(bind=None), Column('id', Integer(),
```

(continues on next page)

(продолжение с предыдущей страницы)

```
table=<user>, primary_key=True, nullable=False), Column('name', String(),
table=<user>), Column('fullname', String(), table=<user>), schema=None))]
```

Благодаря тому что Base содержит в себе объект sqlalchemy.schema.MetaData, вы можете пользоваться всеми его возможностями.

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite://')
>>> Base.metadata.create_all(engine)

[SQL]: SELECT CAST('test plain returns' AS VARCHAR(60)) AS anon_1
[SQL]: ()
[SQL]: SELECT CAST('test unicode returns' AS VARCHAR(60)) AS anon_1
[SQL]: ()
[SQL]: PRAGMA table_info("user")
[SQL]: ()
[SQL]:
CREATE TABLE user (
    id INTEGER NOT NULL,
    name VARCHAR,
    fullname VARCHAR,
    PRIMARY KEY (id)
)


[SQL]: ()
[SQL]: COMMIT
```

Mapper

Объект класса sqlalchemy.orm.mapper.Mapper связывает колонки из схемы таблицы и атрибуты из класса таблицы унаследованного от Base.

Код 86: 2.sqlalchemy/4.orm.mapper.declarative.py

```
# -*- coding: utf-8 -*-
from sqlalchemy import Column, ForeignKey, Integer, String
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship

Base = declarative_base()


class User(Base):
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    __tablename__ = 'user'

    id = Column(Integer, primary_key=True)
    name = Column(String)
    fullname = Column(String)
    password = Column(String)

    addresses = relationship("Address", backref="user",
                             order_by="Address.id")


class Address(Base):
    __tablename__ = 'address'

    id = Column(Integer, primary_key=True)
    user_id = Column(ForeignKey('user.id'))
    email_address = Column(String)

address1 = Address(email_address="vas@example.com")
address2 = Address(email_address="vas2@example.com")
address3 = Address(email_address="vasya@example.com")

print("Mapper relationship: " + str(User.__mapper__.relationships))
print("Mapper columns: " + str(User.__mapper__.c.items()))
print

user1 = User(name="Вася")
user1.addresses = [address1, address2, address3]
print("User1 columns: " + str(user1.__table__.c.items()))
print
print(address1.user.name)
```

```
Mapper relationship: <sqlalchemy.util._collections.ImmutableProperties object at␣
↪0x7ffeae32da28>
Mapper columns: [('id', Column('id', Integer(), table=<user>,
primary_key=True, nullable=False)), ('name', Column('name', String(),
table=<user>)), ('fullname', Column('fullname', String(), table=<user>)),
('password', Column('password', String(), table=<user>))]

User1 columns: [('id', Column('id', Integer(), table=<user>,
primary_key=True, nullable=False)), ('name', Column('name', String(),
table=<user>)), ('fullname', Column('fullname', String(), table=<user>)),
('password', Column('password', String(), table=<user>))]

Вася
```

Конструктор класса

Декларативно описанный класс таблицы содержит в себе конструктор по умолчанию.

```
>>> ed_user = User(name='ed', fullname='Edward Jones')
```

Можно переопределить конструктор вручную

```
class User(Base):
    __tablename__ = 'user'

    def __init__(self, name, fullname):
       self.name = name
       self.fullname = fullname

    id = Column(Integer, primary_key=True)
    name = Column(String)
    fullname = Column(String)
    password = Column(String)

    addresses = relationship("Address", backref="user",
                             order_by="Address.id")
```

Поле `User.id` является первичным ключом, если его значение не указанно явно или такой `id` не существует в БД, то объект считается новым. После записи объекта в БД, значение поля `id` автоматически присваивается.

```
>>> print(ed_user.name, ed_user.fullname)
('ed', 'Edward Jones')
>>> print(ed_user.id)
None
```

Сессии

См.также:

- http://docs.sqlalchemy.org/en/latest/orm/session_basics.html
- https://ru.wikibooks.org/wiki/SQLAlchemy/Sessions

Сессии являются более абстрактным уровнем над механизмом соединения с СУБД `sqlalchemy.engine.Engine`. Они включают в себя функции хранения состояния объектов таблиц и записи этого состояния, по требованию, в БД.

Примечание: Анологию сессий в SQLAlchemy можно провести с системой контроля версий Git.

- **Ресурсы**
  - `git` управляет файлами.
  - `SQAlchemy` манипулирует объектам таблиц (будущие записи в таблицах).
- **Состояние ресурсов**
  - Область подготовленных файлов (staging area) — это обычный файл, обычно хранящийся в каталоге Git, который содержит информацию о том, какие файлы должны войти в следующий коммит.

    ```
    git add README.txt
    ```

  - В SQLAlchemy это сессия которая хранить в себе объекты для дальнейшей записи в БД.

    ```
    session.add(ed_user)
    ```

- **Запись состояния**
  - Создает рабочую копию файлов, добавленных в `staging area`.

    ```
    git commit
    ```

  - Записывает объекты, добавленные ранее в сессию, в базу данных.

    ```
    session.commit()
    ```

Существуют даже расширения для SQLAlchemy которые позволяют хранить данные в git репозитории вместо СУБД, используя при этом только возможности ORM библиотеки SQLAlchemy, т.к. модуль соединений с БД и конструктор *SQL* выражения для git не нужен (https://github.com/matthias-k/gitdb2).

Сессии создаются при помощи экземпляра класса `sqlalchemy.orm.session.Session`.

```
1 >>> from sqlalchemy.orm import Session
2 >>> session = Session(bind=engine)
```

Для добавления объекта (представляющего таблицу) в сессию, необходимо использовать метод `sqlalchemy.orm.session.Session.add()`.

```
1 >>> session.add(ed_user)
```

Перед выполнением любого запроса из сессии, состояние сессии автоматически переносится в БД. В нашем случае, не сохраненный объект `ed_user` добавляется в БД.

```
>>> our_user = session.query(User).filter_by(name='ed').first()

[SQL]: BEGIN (implicit)
[SQL]: INSERT INTO user (name, fullname) VALUES (?, ?)
[SQL]: ('ed', 'Edward Jones')
[SQL]: SELECT user.id AS user_id, user.name AS user_name, user.fullname AS user_
↪fullname
FROM user
WHERE user.name = ?
 LIMIT ? OFFSET ?
[SQL]: ('ed', 1, 0)

>>> our_user
<User('ed', 'Edward Jones')>
```

Теперь у пользователя `ed_user` появилось значение `id`.

```
>>> our_user.id
1
>>> ed_user.id
1
>>> ed_user == our_user is ed_user
True
```

Добавление нескольких объектов в сессию за раз:

```
>>> session.add_all([
...     User(name='wendy', fullname='Wendy Weathersmith'),
...     User(name='mary', fullname='Mary Contrary'),
...     User(name='fred', fullname='Fred Flinstone')
>>> ])
```

Если объект, находящийся в сессии, поменялся, то он помечается как `dirty`. Все измененные объекты в сессии доступны через атрибут `sqlalchemy.orm.session.Session.dirty`

```
>>> ed_user.fullname = 'Ed Jones'
>>> session.dirty
IdentitySet([<User('ed', 'Ed Jones')>])
```

Новые объекты, попавшие в сессию после ее сохранения или в новую сессию, доступны через атрибут `sqlalchemy.orm.session.Session.new`

```
>>> session.new
IdentitySet([<User('fred', 'Fred Flinstone')>,
             <User('wendy', 'Wendy Weathersmith')>,
             <User('mary', 'Mary Contrary')>])
```

Метод `sqlalchemy.orm.session.Session.commit()` сохраняет состояние сессии в БД и подтверждает *SQL* транзакцию, в рамках которой выполнялись все предыдущие запросы.

```
>>> session.commit()

[SQL]: UPDATE user SET fullname=? WHERE user.id = ?
[SQL]: ('Ed Jones', 1)
[SQL]: INSERT INTO user (name, fullname) VALUES (?, ?)
[SQL]: ('wendy', 'Wendy Weathersmith')
[SQL]: INSERT INTO user (name, fullname) VALUES (?, ?)
[SQL]: ('mary', 'Mary Contrary')
[SQL]: INSERT INTO user (name, fullname) VALUES (?, ?)
[SQL]: ('fred', 'Fred Flinstone')
[SQL]: COMMIT
```

После выполнения `COMMIT` сессия не привязана ни к одной транзакции в СУБД. Любые изменения объектов в сессии создадут новую транзакцию.

```
>>> ed_user.fullname

[SQL]: BEGIN (implicit)
[SQL]: SELECT user.id AS user_id, user.name AS user_name, user.fullname AS user_
↪fullname
FROM user
WHERE user.id = ?
[SQL]: (1,)
u'Ed Jones'
```

Создадим новые изменения объектов и отравим SQL запрос с этими изменениями.

```
>>> ed_user.name = 'Edwardo'
>>> fake_user = User(name='fakeuser', fullname='Invalid')
>>> session.add(fake_user)

>>> ed_user
<User('Edwardo', u'Ed Jones')>

>>> session.query(User).filter(User.name.in_(['Edwardo', 'fakeuser'])).all()

[SQL]: UPDATE user SET name=? WHERE user.id = ?
[SQL]: ('Edwardo', 1)
[SQL]: INSERT INTO user (name, fullname) VALUES (?, ?)
[SQL]: ('fakeuser', 'Invalid')
[SQL]: SELECT user.id AS user_id, user.name AS user_name, user.fullname AS user_
↪fullname
FROM user
```

(continues on next page)

(продолжение с предыдущей страницы)

```
WHERE user.name IN (?, ?)
[SQL]: ('Edwardo', 'fakeuser')
[<User('Edwardo', u'Ed Jones')>, <User('fakeuser', 'Invalid')>]
```

Несмотря на то что *SQL* запросы были выполнены в СУБД, мы все еще находимся в транзакции. Поэтому любые изменения в сессии, даже если они выполнили *SQL* запрос в СУБД, всегда можно отметить при помощи метода `sqlalchemy.orm.session.Session.rollback()`.

```
>>> session.rollback()
[SQL]: ROLLBACK
```

После `ROLLBACK` сессия не привязана ни к одной транзакции в СУБД. Поэтому при изменении объектов в сессии создастся новая транзакция. Причем данные предыдущей сессии не были записаны.

```
>>> ed_user.name

[SQL]: BEGIN (implicit)
[SQL]: SELECT user.id AS user_id, user.name AS user_name, user.fullname AS user_
↪fullname
FROM user
WHERE user.id = ?
[SQL]: (1,)
u'ed'
```

```
>>> fake_user in session
False

>>> session.query(User).filter(User.name.in_(['ed', 'fakeuser'])).all()

[SQL]: SELECT user.id AS user_id, user.name AS user_name, user.fullname AS user_
↪fullname
FROM user
WHERE user.name IN (?, ?)
[SQL]: ('ed', 'fakeuser')
[<User(u'ed', u'Ed Jones')>]
```

SQL запросы через ORM

Операции над атрибутами класса таблицы равносильны операциям над объектом `sqlalchemy.schema.Column`. Поэтому их можно использовать в конструкторе *SQL* запросов. Результатом выполнения *SQL выражения* будет список значений записи в БД.

```
>>> print(User.name == "ed")
"user".name = :name_1

>>> from sqlalchemy import select
>>> sel = select([User.name, User.fullname]).\
...         where(User.name == 'ed').\
...         order_by(User.id)
>>> session.connection().execute(sel).fetchall()

[SQL]: SELECT user.name, user.fullname
FROM user
WHERE user.name = ? ORDER BY user.id
[SQL]: ('ed',)
[(u'ed', u'Ed Jones')]
```

ORM позволяет конструировать запросы при помощи метода `sqlalchemy.orm.session.Session.query()`. Этот метод создает объект класса `sqlalchemy.orm.query.Query`, который является более высокой абстракцией конструктора *SQL выражения* в SQLAlchemy.

ORM, в отличии от стандартного конструктора *SQL выражения*, позволяет создавать запросы более наглядно и возвращать результат в виде объектов которые привязаны к сессии.

```
>>> query = session.query(User).filter(User.name == 'ed').order_by(User.id)
>>> query.all()

[SQL]: SELECT user.id AS user_id, user.name AS user_name, user.fullname AS user_
↪fullname
FROM user
WHERE user.name = ? ORDER BY user.id
[SQL]: ('ed',)
[<User(u'ed', u'Ed Jones')>]
```

Можно также возвращать чистые значения полей, как это делают *SQL выражения*.

```
>>> for name, fullname in session.query(User.name, User.fullname):
...     print(name, fullname)

[SQL]: SELECT user.name AS user_name, user.fullname AS user_fullname
FROM user
[SQL]: ()
(u'ed', u'Ed Jones')
(u'wendy', u'Wendy Weathersmith')
(u'mary', u'Mary Contrary')
(u'fred', u'Fred Flinstone')
```

Или комбинировать значения полей с объектами.

```
>>> for row in session.query(User, User.name):
...     print(row.User, row.name)

[SQL]: SELECT user.id AS user_id, user.name AS user_name, user.fullname AS user_
↪fullname
FROM user
[SQL]: ()
(<User(u'ed', u'Ed Jones')>, u'ed')
(<User(u'wendy', u'Wendy Weathersmith')>, u'wendy')
(<User(u'mary', u'Mary Contrary')>, u'mary')
(<User(u'fred', u'Fred Flinstone')>, u'fred')
```

Ограничения и условия

LIMIT, OFFSET

Выбор конкретной строки запроса делается не средствами языка Python, а на стороне СУБД, за счет конструкции `LIMIT ? OFFSET ?`, что значительно ускоряет выполнение запроса. Для программиста это выглядит прозрачно, как будто он работает с Python списком.

```
>>> u = session.query(User).order_by(User.id)[2]
>>> print(u)

[SQL]: SELECT user.id AS user_id, user.name AS user_name, user.fullname AS user_
↪fullname
FROM user ORDER BY user.id
 LIMIT ? OFFSET ?
[SQL]: (1, 2)
<User(u'mary', u'Mary Contrary')>
```

Аналогично работают и Python срезы.

```
>>> for u in session.query(User).order_by(User.id)[1:3]:
...     print(u)

[SQL]: SELECT user.id AS user_id, user.name AS user_name, user.fullname AS user_
↪fullname
FROM user ORDER BY user.id
 LIMIT ? OFFSET ?
[SQL]: (2, 1)
<User(u'wendy', u'Wendy Weathersmith')>
<User(u'mary', u'Mary Contrary')>
```

WHERE

Условие `WHERE` соответствует методу `sqlalchemy.orm.query.Query.filter_by()`.

```
>>> for name, in session.query(User.name).\
...                 filter_by(fullname='Ed Jones'):
...     print(name)

[SQL]: SELECT user.name AS user_name
FROM user
WHERE user.fullname = ?
[SQL]: ('Ed Jones',)
```

Или более функциональному методу `sqlalchemy.orm.query.Query.filter()`.

```
>>> for name, in session.query(User.name).\
...                 filter(User.fullname == 'Ed Jones'):
...     print(name)

[SQL]: SELECT user.name AS user_name
FROM user
WHERE user.fullname = ?
[SQL]: ('Ed Jones',)
ed
```

```
>>> from sqlalchemy import or_
>>> for name, in session.query(User.name).\
...                 filter(or_(User.fullname == 'Ed Jones', User.id < 5)):
...     print(name)

[SQL]: SELECT user.name AS user_name
FROM user
WHERE user.fullname = ? OR user.id < ?
[SQL]: ('Ed Jones', 5)
ed
wendy
mary
fred
```

Последовательное выполнение методов `sqlalchemy.orm.query.Query.filter()` соединяет условия `WHERE` при помощи оператора `AND`, аналогично конструкции `select().where()`.

```
>>> for user in session.query(User).\
...                         filter(User.name == 'ed').\
...                         filter(User.fullname == 'Ed Jones'):
```

(continues on next page)

(продолжение с предыдущей страницы)

```
...     print(user)
[SQL]: SELECT user.id AS user_id, user.name AS user_name, user.fullname AS user_
↪fullname
FROM user
WHERE user.name = ? AND user.fullname = ?
[SQL]: ('ed', 'Ed Jones')
<User(u'ed', u'Ed Jones')>
```

Выполнение SQL выражений

Сам объект класса `sqlalchemy.orm.query.Query` не выполняет обращений к БД.

```
>>> query = session.query(User).filter_by(fullname='Ed Jones')
>>>
```

all()

Для этого существуют специальные методы этого класса, например `sqlalchemy.orm.query.Query.all()`.

```
>>> query.all()

[SQL]: SELECT user.id AS user_id, user.name AS user_name, user.fullname AS user_
↪fullname
FROM user
WHERE user.fullname = ?
[SQL]: ('Ed Jones',)
[<User(u'ed', u'Ed Jones')>]
```

first()

`sqlalchemy.orm.query.Query.first()` - выполнит запрос и вернет первую строку запроса или `None`.

```
>>> query.first()

[SQL]: SELECT user.id AS user_id, user.name AS user_name, user.fullname AS user_
↪fullname
FROM user
WHERE user.fullname = ?
```

(continues on next page)

(продолжение с предыдущей страницы)

```
 LIMIT ? OFFSET ?
[SQL]: ('Ed Jones', 1, 0)
<User(u'ed', u'Ed Jones')>
```

one()

`sqlalchemy.orm.query.Query.one()` - выполнит запрос, вернет первую строку запроса и проверит что она одна и только одна, иначе вызовет исключение `sqlalchemy.orm.exc.NoResultFound`.

```
>>> query.one()

[SQL]: SELECT user.id AS user_id, user.name AS user_name, user.fullname AS user_
↪fullname
FROM user
WHERE user.fullname = ?
[SQL]: ('Ed Jones',)
<User(u'ed', u'Ed Jones')>

>>>
>>> query = session.query(User).filter_by(fullname='nonexistent')
>>> query.one()

[SQL]: SELECT user.id AS user_id, user.name AS user_name, user.fullname AS user_
↪fullname
FROM user
WHERE user.fullname = ?
[SQL]: ('nonexistent',)
Traceback (most recent call last):
  File "/home/user/.virtualenvs/lectures/local/lib/python2.7/site-packages/
↪sliderepl/core.py", line 291, in run
    exec_(co, environ)
  File "/home/user/.virtualenvs/lectures/local/lib/python2.7/site-packages/
↪sliderepl/compat.py", line 24, in exec_
    exec("""exec _code_ in _globs_, _locs_""")
  File "<string>", line 1, in <module>
  File "<input>", line 1, in <module>
  File "/home/user/.virtualenvs/lectures/local/lib/python2.7/site-packages/
↪sqlalchemy/orm/query.py", line 2478, in one
    raise orm_exc.NoResultFound("No row was found for one()")
NoResultFound: No row was found for one()
```

Если результат запроса вернет больше строк чем одну, это тоже расценивается как ошибка `sqlalchemy.orm.exc.MultipleResultsFound`.

```
>>> query = session.query(User)
>>> query.one()

[SQL]: SELECT user.id AS user_id, user.name AS user_name, user.fullname AS user_
↪fullname
FROM user
[SQL]: ()
Traceback (most recent call last):
  File "/home/uralbash/.virtualenvs/sacrud/local/lib/python2.7/site-packages/
↪sliderepl/core.py", line 291, in run
    exec_(co, environ)
  File "/home/uralbash/.virtualenvs/sacrud/local/lib/python2.7/site-packages/
↪sliderepl/compat.py", line 24, in exec_
    exec("""exec _code_ in _globs_, _locs_""")
  File "<string>", line 1, in <module>
  File "<input>", line 1, in <module>
  File "/home/uralbash/.virtualenvs/sacrud/local/lib/python2.7/site-packages/
↪sqlalchemy/orm/query.py", line 2481, in one
    "Multiple rows were found for one()")
MultipleResultsFound: Multiple rows were found for one()
```

Связи между таблиц

См.также:

- http://docs.sqlalchemy.org/en/latest/orm/relationships.html

Новый класс `Address` имеет связь *Many-To-One* с таблицей `User`. Связь между Python классов осуществляется при помощи функции `sqlalchemy.orm.relationship()`.

```
>>> from sqlalchemy import ForeignKey
>>> from sqlalchemy.orm import relationship

>>> class Address(Base):
...     __tablename__ = 'address'

...     id = Column(Integer, primary_key=True)
...     email_address = Column(String, nullable=False)
...     user_id = Column(Integer, ForeignKey('user.id'))

...     user = relationship("User", backref="addresses")

...     def __repr__(self):
...         return "<Address(%r)>" % self.email_address
```

```
>>> Base.metadata.create_all(engine)

[SQL]: PRAGMA table_info("user")
[SQL]: ()
[SQL]: PRAGMA table_info("address")
[SQL]: ()
[SQL]:
CREATE TABLE address (
    id INTEGER NOT NULL,
    email_address VARCHAR NOT NULL,
    user_id INTEGER,
    PRIMARY KEY (id),
    FOREIGN KEY(user_id) REFERENCES user (id)
)

[SQL]: ()
[SQL]: COMMIT
```

Благодаря параметру `backref`, класс `User` получает обратную ссылку на класс `Adress`.

```
>>> jack = User(name='jack', fullname='Jack Bean')
>>> jack.addresses
[]
```

Добавим пользователю адреса.

```
>>> jack.addresses = [
...                 Address(email_address='jack@gmail.com'),
...                 Address(email_address='j25@yahoo.com'),
...                 Address(email_address='jack@hotmail.com'),
...                 ]
```

`sqlalchemy.orm.backref()` добавляет ссылки друг на друга для каждого объекта.

```
>>> jack
<User('jack', 'Jack Bean')>

>>> jack.addresses[1]
<Address('j25@yahoo.com')>

>>> jack.addresses[1].user
<User('jack', 'Jack Bean')>

>>> jack.addresses[1].user.addresses[1]
<Address('j25@yahoo.com')>

```

(continues on next page)

(продолжение с предыдущей страницы)

```
>>> jack.addresses[1].user.addresses[1].user
<User('jack', 'Jack Bean')>

>>> jack.addresses[1].user.addresses[1].user.addresses[1].user.addresses[1].user
<User('jack', 'Jack Bean')>

>>> jack.addresses[1].user.addresses[1].user.addresses[2].user.addresses[0].user
<User('jack', 'Jack Bean')>
```

Добавление в сессию объекта, который ссылается на другие объекты, автоматически включает их тоже.

```
>>> session.add(jack)
>>> session.new
IdentitySet([<Address('jack@hotmail.com')>,
             <Address('jack@gmail.com')>,
             <User('jack', 'Jack Bean')>,
             <Address('j25@yahoo.com')>])
```

```
>>> session.commit()

[SQL]: INSERT INTO user (name, fullname) VALUES (?, ?)
[SQL]: ('jack', 'Jack Bean')
[SQL]: INSERT INTO address (email_address, user_id) VALUES (?, ?)
[SQL]: ('jack@gmail.com', 5)
[SQL]: INSERT INTO address (email_address, user_id) VALUES (?, ?)
[SQL]: ('j25@yahoo.com', 5)
[SQL]: INSERT INTO address (email_address, user_id) VALUES (?, ?)
[SQL]: ('jack@hotmail.com', 5)
[SQL]: COMMIT
```

После подтверждения транзакции (COMMIT), обращение по ссылке создаст новую транзакцию и считает значения из БД.

```
>>> jack.addresses

[SQL]: BEGIN (implicit)
[SQL]: SELECT user.id AS user_id, user.name AS user_name, user.fullname AS user_
↪fullname
FROM user
WHERE user.id = ?
[SQL]: (5,)
[SQL]: SELECT address.id AS address_id, address.email_address AS address_email_
↪address, address.user_id AS address_user_id
FROM address
```

(continues on next page)

(продолжение с предыдущей страницы)

```
WHERE ? = address.user_id
[SQL]: (5,)
[<Address(u'jack@gmail.com')>, <Address(u'j25@yahoo.com')>, <Address(u
↪'jack@hotmail.com')>]
```

Теперь, считанные объекты находятся в памяти, до тех пор пока мы опять не подтвердим транзакцию (COMMIT) или отменим ее (ROLLBACK).

```
>>> jack.addresses
[<Address(u'jack@gmail.com')>, <Address(u'j25@yahoo.com')>, <Address(u
↪'jack@hotmail.com')>]
```

Привяжем адрес к другому пользователю.

```
>>> fred = session.query(User).filter_by(name='fred').one()
>>> jack.addresses[1].user = fred
>>> fred.addresses
>>> session.commit()

[SQL]: SELECT user.id AS user_id, user.name AS user_name, user.fullname AS user_
↪fullname
FROM user
WHERE user.name = ?
[SQL]: ('fred',)
[SQL]: UPDATE address SET user_id=? WHERE address.id = ?
[SQL]: (4, 2)
[SQL]: SELECT address.id AS address_id, address.email_address AS address_email_
↪address, address.user_id AS address_user_id
FROM address
WHERE ? = address.user_id
[SQL]: (4,)
[<Address(u'j25@yahoo.com')>]
[SQL]: COMMIT
```

Выполнение операции *implicit JOIN*.

```
>>> session.query(User, Address).filter(User.id == Address.user_id).all()

[SQL]: SELECT user.id AS user_id, user.name AS user_name, user.fullname AS
user_fullname, address.id AS address_id, address.email_address AS
address_email_address, address.user_id AS address_user_id FROM user,
address
WHERE user.id = address.user_id
[SQL]: ()
[(<User(u'jack', u'Jack Bean')>, <Address(u'jack@gmail.com')>),
```

(continues on next page)

(продолжение с предыдущей страницы)

```
(<User(u'fred', u'Fred Flinstone')>, <Address(u'j25@yahoo.com')>),
(<User(u'jack', u'Jack Bean')>, <Address(u'jack@hotmail.com')>)]
```

Явный *JOIN*.

```
>>> session.query(User, Address).join(Address, User.id == Address.user_id).all()

[SQL]: SELECT user.id AS user_id, user.name AS user_name, user.fullname AS
user_fullname, address.id AS address_id, address.email_address AS
address_email_address, address.user_id AS address_user_id FROM user JOIN
address ON user.id = address.user_id
[SQL]: ()
[(<User(u'jack', u'Jack Bean')>, <Address(u'jack@gmail.com')>),
(<User(u'fred', u'Fred Flinstone')>, <Address(u'j25@yahoo.com')>),
(<User(u'jack', u'Jack Bean')>, <Address(u'jack@hotmail.com')>)]
```

Более краткий и понятный способ использовать ссылку на таблицу для связи.

```
>>> session.query(User, Address).join(User.addresses).all()

[SQL]: SELECT user.id AS user_id, user.name AS user_name, user.fullname AS
user_fullname, address.id AS address_id, address.email_address AS
address_email_address, address.user_id AS address_user_id FROM user JOIN
address ON user.id = address.user_id
[SQL]: ()
[(<User(u'jack', u'Jack Bean')>, <Address(u'jack@gmail.com')>),
(<User(u'fred', u'Fred Flinstone')>, <Address(u'j25@yahoo.com')>),
(<User(u'jack', u'Jack Bean')>, <Address(u'jack@hotmail.com')>)]
```

В простых случаях можно передавать только класс таблицы.

```
>>> session.query(User, Address).join(Address).all()

[SQL]: SELECT user.id AS user_id, user.name AS user_name, user.fullname AS
user_fullname, address.id AS address_id, address.email_address AS
address_email_address, address.user_id AS address_user_id FROM user JOIN
address ON user.id = address.user_id
[SQL]: ()
[(<User(u'jack', u'Jack Bean')>, <Address(u'jack@gmail.com')>),
(<User(u'fred', u'Fred Flinstone')>, <Address(u'j25@yahoo.com')>),
(<User(u'jack', u'Jack Bean')>, <Address(u'jack@hotmail.com')>)]
```

JOIN с условием WHERE.

```
>>> session.query(User.name).join(User.addresses).\
...     filter(Address.email_address == 'jack@gmail.com').first()
```

(continues on next page)

(продолжение с предыдущей страницы)

```
3
4  [SQL]: SELECT user.name AS user_name
5  FROM user JOIN address ON user.id = address.user_id
6  WHERE address.email_address = ?
7   LIMIT ? OFFSET ?
8  [SQL]: ('jack@gmail.com', 1, 0)
9  (u'jack',)
```

Явный вызов конструкции SELECT FROM JOIN используя метод `sqlalchemy.orm.query.Query.select_from()`.

```
1  >>> session.query(User, Address).select_from(Address).join(Address.user).all()
2
3  [SQL]: SELECT user.id AS user_id, user.name AS user_name, user.fullname AS
4  user_fullname, address.id AS address_id, address.email_address AS
5  address_email_address, address.user_id AS address_user_id FROM address
6  JOIN user ON user.id = address.user_id
7  [SQL]: ()
8  [(<User(u'jack', u'Jack Bean')>, <Address(u'jack@gmail.com')>),
9  (<User(u'fred', u'Fred Flinstone')>, <Address(u'j25@yahoo.com')>),
10 (<User(u'jack', u'Jack Bean')>, <Address(u'jack@hotmail.com')>)]
```

Запросы ссылающиеся на одну сущность более чем один раз, нуждаются в алиасах. Алиасы задаются при помощи функции `sqlalchemy.orm.aliased()`.

```
1  >>> from sqlalchemy.orm import aliased
2  >>> a1, a2 = aliased(Address), aliased(Address)
3  >>> session.query(User).\
4  ...         join(a1).\
5  ...         join(a2).\
6  ...         filter(a1.email_address == 'jack@gmail.com').\
7  ...         filter(a2.email_address == 'jack@hotmail.com').\
8  ...         all()
9
10 [SQL]: SELECT user.id AS user_id, user.name AS user_name, user.fullname AS user_
   ↪fullname
11 FROM user JOIN address AS address_1 ON user.id = address_1.user_id JOIN
12 address AS address_2 ON user.id = address_2.user_id WHERE
13 address_1.email_address = ? AND address_2.email_address = ?
14 [SQL]: ('jack@gmail.com', 'jack@hotmail.com')
15 [<User(u'jack', u'Jack Bean')>]
```

Подзапросы автоматически использую алиасы.

```
1  >>> from sqlalchemy import func
```

(continues on next page)

(продолжение с предыдущей страницы)

```
>>> subq = session.query(
...                 func.count(Address.id).label('count'),
...                 User.id.label('user_id')
...                 ).\
...                 join(Address.user).\
...                 group_by(User.id).\
...                 subquery()
>>> session.query(User.name, func.coalesce(subq.c.count, 0)).\
...             outerjoin(subq, User.id == subq.c.user_id).all()

[SQL]: SELECT user.name AS user_name, coalesce(anon_1.count, ?) AS coalesce_1
FROM user LEFT OUTER JOIN (SELECT count(address.id) AS count, user.id AS user_id
FROM address JOIN user ON user.id = address.user_id GROUP BY user.id) AS anon_1
ON user.id = anon_1.user_id [SQL]: (0,)
[(u'ed', 0), (u'wendy', 0), (u'mary', 0), (u'fred', 1), (u'jack', 2)]
```

При каждом обращении к ссылкам объекта в цикле, вызывается новый запрос:

```
>>> for user in session.query(User):
...     print(user, user.addresses)

[SQL]: SELECT user.id AS user_id, user.name AS user_name, user.fullname AS
user_fullname FROM user
[SQL]: ()
[SQL]: SELECT address.id AS address_id, address.email_address AS
address_email_address, address.user_id AS address_user_id FROM address
WHERE ? = address.user_id
[SQL]: (1,)
(<User(u'ed', u'Ed Jones')>, [])
[SQL]: SELECT address.id AS address_id, address.email_address AS
address_email_address, address.user_id AS address_user_id FROM address
WHERE ? = address.user_id
[SQL]: (2,)
(<User(u'wendy', u'Wendy Weathersmith')>, [])
[SQL]: SELECT address.id AS address_id, address.email_address AS
address_email_address, address.user_id AS address_user_id FROM address
WHERE ? = address.user_id
[SQL]: (3,)
(<User(u'mary', u'Mary Contrary')>, [])
[SQL]: SELECT address.id AS address_id, address.email_address AS
address_email_address, address.user_id AS address_user_id FROM address
WHERE ? = address.user_id
[SQL]: (4,)
(<User(u'fred', u'Fred Flinstone')>, [<Address(u'j25@yahoo.com')>])
(<User(u'jack', u'Jack Bean')>, [<Address(u'jack@gmail.com')>,
<Address(u'jack@hotmail.com')>])
```

Чтобы этого избежать нужно использовать опцию предварительной загрузки sqlalchemy.orm.subqueryload().

```
>>> session.rollback()  # so we can see the load happen again.
>>> from sqlalchemy.orm import subqueryload
>>> for user in session.query(User).options(subqueryload(User.addresses)):
...     print(user, user.addresses)

[SQL]: ROLLBACK
[SQL]: BEGIN (implicit)
[SQL]: SELECT user.id AS user_id, user.name AS user_name, user.fullname AS user_
↪fullname
FROM user
[SQL]: ()
[SQL]: SELECT address.id AS address_id, address.email_address AS
address_email_address, address.user_id AS address_user_id, anon_1.user_id
AS anon_1_user_id FROM (SELECT user.id AS user_id
FROM user) AS anon_1 JOIN address ON anon_1.user_id = address.user_id ORDER BY␣
↪anon_1.user_id
[SQL]: ()
(<User(u'ed', u'Ed Jones')>, [])
(<User(u'wendy', u'Wendy Weathersmith')>, [])
(<User(u'mary', u'Mary Contrary')>, [])
(<User(u'fred', u'Fred Flinstone')>, [<Address(u'j25@yahoo.com')>])
(<User(u'jack', u'Jack Bean')>, [<Address(u'jack@gmail.com')>, <Address(u
↪'jack@hotmail.com')>])
```

Или sqlalchemy.orm.joinedload() чтобы уместить все в один запрос.

```
>>> session.rollback()
>>> from sqlalchemy.orm import joinedload
>>> for user in session.query(User).options(joinedload(User.addresses)):
...     print(user, user.addresses)

[SQL]: ROLLBACK
[SQL]: BEGIN (implicit)
[SQL]: SELECT user.id AS user_id, user.name AS user_name, user.fullname AS
user_fullname, address_1.id AS address_1_id, address_1.email_address AS
address_1_email_address, address_1.user_id AS address_1_user_id FROM user
LEFT OUTER JOIN address AS address_1 ON user.id = address_1.user_id
[SQL]: ()
(<User(u'ed', u'Ed Jones')>, [])
(<User(u'wendy', u'Wendy Weathersmith')>, [])
(<User(u'mary', u'Mary Contrary')>, [])
(<User(u'fred', u'Fred Flinstone')>, [<Address(u'j25@yahoo.com')>])
(<User(u'jack', u'Jack Bean')>, [<Address(u'jack@gmail.com')>, <Address(u
↪'jack@hotmail.com')>])
```

Удаление адреса из списка пользователя `User.addresses`, поменяет значение поля FOREIGN KEY на NULL, но не удалит саму запись.

```
>>> jack = session.query(User).filter_by(name='jack').one()
>>> del jack.addresses[0]
>>> session.commit()

[SQL]: SELECT user.id AS user_id, user.name AS user_name, user.fullname AS user_
↪fullname
FROM user
WHERE user.name = ?
[SQL]: ('jack',)
[SQL]: UPDATE address SET user_id=? WHERE address.id = ?
[SQL]: (None, 1)
[SQL]: COMMIT
```

Мы можем настроить связи между таблицами на каскадное удаление.

```
>>> User.addresses.property.cascade = "all, delete, delete-orphan"

>>> fred = session.query(User).filter_by(name='fred').one()
>>> del fred.addresses[0]
>>> session.commit()

[SQL]: BEGIN (implicit)
[SQL]: SELECT user.id AS user_id, user.name AS user_name, user.fullname AS user_
↪fullname
FROM user
WHERE user.name = ?
[SQL]: ('fred',)
[SQL]: SELECT address.id AS address_id, address.email_address AS
address_email_address, address.user_id AS address_user_id FROM address
WHERE ? = address.user_id
[SQL]: (4,)
[SQL]: DELETE FROM address WHERE address.id = ?
[SQL]: (2,)
[SQL]: COMMIT
```

`delete-orphan` означает что дети не могут существовать без родителей. Поэтому при удалении родителя вся связанные с ним записи тоже удалятся.

```
>>> session.delete(jack)
>>> session.commit()

[SQL]: BEGIN (implicit)
[SQL]: SELECT user.id AS user_id, user.name AS user_name, user.fullname AS user_
↪fullname
```

(continues on next page)

(продолжение с предыдущей страницы)

```
FROM user
WHERE user.id = ?
[SQL]: (5,)
[SQL]: SELECT address.id AS address_id, address.email_address AS address_email_
↪address, address.user_id AS address_user_id
FROM address
WHERE ? = address.user_id
[SQL]: (5,)
[SQL]: DELETE FROM address WHERE address.id = ?
[SQL]: (3,)
[SQL]: DELETE FROM user WHERE user.id = ?
[SQL]: (5,)
[SQL]: COMMIT
```

Полный пример

Код 87: 2.sqlalchemy/4.orm.py

```python
# ## slide::
# ## title:: Object Relational Mapping
# The *declarative* system is normally used to configure
# object relational mappings.

from sqlalchemy.ext.declarative import declarative_base
Base = declarative_base()

# ## slide::
# a basic mapping.  __repr__() is optional.

from sqlalchemy import Column, Integer, String


class User(Base):
    __tablename__ = 'user'

    id = Column(Integer, primary_key=True)
    name = Column(String)
    fullname = Column(String)

    def __repr__(self):
        return "<User(%r, %r)>" % (self.name, self.fullname)

# ## slide::
```

(continues on next page)

(продолжение с предыдущей страницы)

```
# the User class now has a Table object associated with it.

User.__table__

# ## slide::
# The Mapper object mediates the relationship between User
# and the "user" Table object.

User.__mapper__

# ## slide::
# User has a default constructor, accepting field names
# as arguments.

ed_user = User(name='ed', fullname='Edward Jones')

# ## slide::
# The "id" field is the primary key, which starts as None
# if we didn't set it explicitly.

print(ed_user.name, ed_user.fullname)
print(ed_user.id)

# ## slide:: p
# The MetaData object is here too, available from the Base.

from sqlalchemy import create_engine
engine = create_engine('sqlite://')
Base.metadata.create_all(engine)

# ## slide::
# To persist and load User objects from the database, we
# use a Session object.

from sqlalchemy.orm import Session
session = Session(bind=engine)

# ## slide::
# new objects are placed into the Session using add().
session.add(ed_user)

# ## slide:: pi
# the Session will *flush* *pending* objects
# to the database before each Query.

```

(continues on next page)

(продолжение с предыдущей страницы)

```
our_user = session.query(User).filter_by(name='ed').first()
our_user

# ## slide::
# the User object we've inserted now has a value for ".id"
print(ed_user.id)

# ## slide::
# the Session maintains a *unique* object per identity.
# so "ed_user" and "our_user" are the *same* object

ed_user is our_user

# ## slide::
# Add more objects to be pending for flush.

session.add_all([User(name='wendy',
                      fullname='Wendy Weathersmith'),
                 User(name='mary',
                      fullname='Mary Contrary'),
                 User(name='fred',
                      fullname='Fred Flinstone')])

# ## slide::
# modify "ed_user" - the object is now marked as *dirty*.

ed_user.fullname = 'Ed Jones'

# ## slide::
# the Session can tell us which objects are dirty...

session.dirty

# ## slide::
# and can also tell us which objects are pending...

session.new

# ## slide:: p i
# The whole transaction is committed.  Commit always triggers
# a final flush of remaining changes.

session.commit()

# ## slide:: p
```

(continues on next page)

(продолжение с предыдущей страницы)

```
# After a commit, theres no transaction.  The Session
# *invalidates* all data, so that accessing them will automatically
# start a *new* transaction and re-load from the database.

ed_user.fullname

# ## slide::
# Make another "dirty" change, and another "pending" change,
# that we might change our minds about.

ed_user.name = 'Edwardo'
fake_user = User(name='fakeuser', fullname='Invalid')
session.add(fake_user)

# ## slide:: p
# run a query, our changes are flushed; results come back.

session.query(User).filter(User.name.in_(['Edwardo', 'fakeuser'])).all()

# ## slide::
# But we're inside of a transaction.  Roll it back.
session.rollback()

# ## slide:: p
# ed_user's name is back to normal
ed_user.name

# ## slide::
# "fake_user" has been evicted from the session.
fake_user in session

# ## slide:: p
# and the data is gone from the database too.

session.query(User).filter(User.name.in_(['ed', 'fakeuser'])).all()

# ## slide::
# ## title:: Exercises - Basic Mapping
#
# 1. Create a class/mapping for this table, call the class Network
#
# CREATE TABLE network (
#      network_id INTEGER PRIMARY KEY,
#      name VARCHAR(100) NOT NULL,
# )
```

(continues on next page)

(продолжение с предыдущей страницы)

```
#
# 2. emit Base.metadata.create_all(engine) to create the table
#
# 3. commit a few Network objects to the database:
#
# Network(name='net1'), Network(name='net2')
#
#

# ## slide::
# ## title:: ORM Querying
# The attributes on our mapped class act like Column objects, and
# produce SQL expressions.

print(User.name == "ed")

# ## slide:: p
# These SQL expressions are compatible with the select() object
# we introduced earlier.

from sqlalchemy import select

sel = select([User.name, User.fullname]).\
    where(User.name == 'ed').\
    order_by(User.id)

session.connection().execute(sel).fetchall()

# ## slide:: p
# but when using the ORM, the Query() object provides a lot more functionality,
# here selecting the User *entity*.

query = session.query(User).filter(User.name == 'ed').order_by(User.id)

query.all()

# ## slide:: p
# Query can also return individual columns

for name, fullname in session.query(User.name, User.fullname):
    print(name, fullname)

# ## slide:: p
# and can mix entities / columns together.

```

(continues on next page)

(продолжение с предыдущей страницы)

```
for row in session.query(User, User.name):
    print(row.User, row.name)

# ## slide:: p
# Array indexes will OFFSET to that index and LIMIT by one...

u = session.query(User).order_by(User.id)[2]
print(u)

# ## slide:: pi
# and array slices work too.

for u in session.query(User).order_by(User.id)[1:3]:
    print(u)

# ## slide:: p
# the WHERE clause is either by filter_by(), which is convenient

for name, in session.query(User.name).\
        filter_by(fullname='Ed Jones'):
    print(name)

# ## slide:: p
# or filter(), which is more flexible

for name, in session.query(User.name).\
        filter(User.fullname == 'Ed Jones'):
    print(name)

# ## slide:: p
# conjunctions can be passed to filter() as well

from sqlalchemy import or_

for name, in session.query(User.name).\
        filter(or_(User.fullname == 'Ed Jones', User.id < 5)):
    print(name)

# ## slide::
# multiple filter() calls join by AND just like select().where()

for user in session.query(User).\
        filter(User.name == 'ed').\
        filter(User.fullname == 'Ed Jones'):
    print(user)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
251
252 # ## slide::
253 # Query has some variety for returning results
254
255 query = session.query(User).filter_by(fullname='Ed Jones')
256
257 # ## slide:: p
258 # all() returns a list
259
260 query.all()
261
262 # ## slide:: p
263 # first() returns the first row, or None
264
265 query.first()
266
267 # ## slide:: p
268 # one() returns the first row and verifies that there's one and only one
269
270 query.one()
271
272 # ## slide:: p
273 # if there's not one(), you get an error
274
275 query = session.query(User).filter_by(fullname='nonexistent')
276 query.one()
277
278 # ## slide:: p
279 # if there's more than one(), you get an error
280
281 query = session.query(User)
282 query.one()
283
284 # ## slide::
285 # ## title:: Exercises - ORM Querying
286 # 1. Produce a Query object representing the list of "fullname" values for
287 #    all User objects in alphabetical order.
288 #
289 # 2. call .all() on the query to make sure it works!
290 #
291 # 3. build a second Query object from the first that also selects
292 #    only User rows with the name "mary" or "ed".
293 #
294 # 4. return only the second row of the Query from #3.
295
```

(continues on next page)

(продолжение с предыдущей страницы)

```
296 # ## slide::
297 # ## title:: Joins and relationships
298 # A new class called Address, with a *many-to-one* relationship to User.
299
300 from sqlalchemy import ForeignKey
301 from sqlalchemy.orm import relationship
302
303
304 class Address(Base):
305     __tablename__ = 'address'
306
307     id = Column(Integer, primary_key=True)
308     email_address = Column(String, nullable=False)
309     user_id = Column(Integer, ForeignKey('user.id'))
310
311     user = relationship("User", backref="addresses")
312
313     def __repr__(self):
314         return "<Address(%r)>" % self.email_address
315
316 # ## slide:: p
317 # create the new table.
318
319 Base.metadata.create_all(engine)
320
321 # ## slide::
322 # a new User object also gains an empty "addresses" collection now.
323
324 jack = User(name='jack', fullname='Jack Bean')
325 jack.addresses
326
327 # ## slide::
328 # populate this collection with new Address objects.
329
330 jack.addresses = [Address(email_address='jack@gmail.com'),
331                   Address(email_address='j25@yahoo.com'),
332                   Address(email_address='jack@hotmail.com'), ]
333
334 # ## slide::
335 # the "backref" sets up Address.user for each User.address.
336
337 jack.addresses[1]
338 jack.addresses[1].user
339
340 # ## slide::
```

(continues on next page)

(продолжение с предыдущей страницы)

```
# adding User->jack will *cascade* each Address into the Session as well.

session.add(jack)
session.new

# ## slide:: p
# commit.
session.commit()

# ## slide:: p
# After expiration, jack.addresses emits a *lazy load* when first
# accessed.
jack.addresses

# ## slide:: i
# the collection stays in memory until the transaction ends.
jack.addresses

# ## slide:: p
# collections and references are updated by manipulating objects,
# not primary / foreign key values.

fred = session.query(User).filter_by(name='fred').one()
jack.addresses[1].user = fred

fred.addresses

session.commit()

# ## slide:: p
# Query can select from multiple tables at once.
# Below is an *implicit join*.

session.query(User, Address).filter(User.id == Address.user_id).all()

# ## slide:: p
# join() is used to create an explicit JOIN.

session.query(User, Address).join(Address, User.id == Address.user_id).all()

# ## slide:: p
# The most succinct and accurate way to join() is to use the
# the relationship()-bound attribute to specify ON.

session.query(User, Address).join(User.addresses).all()
```

(continues on next page)

(продолжение с предыдущей страницы)

```
386
387 # ## slide:: p
388 # join() will also figure out very simple joins just using entities.
389
390 session.query(User, Address).join(Address).all()
391
392 # ## slide:: p
393 # Either User or Address may be referred to anywhere in the query.
394
395 session.query(User.name).join(User.addresses).\
396     filter(Address.email_address == 'jack@gmail.com').first()
397
398 # ## slide:: p
399 # we can specify an explicit FROM using select_from().
400
401 session.query(User, Address).select_from(Address).join(Address.user).all()
402
403 # ## slide:: p
404 # A query that refers to the same entity more than once in the FROM
405 # clause requires *aliasing*.
406
407 from sqlalchemy.orm import aliased
408
409 a1, a2 = aliased(Address), aliased(Address)
410 session.query(User).\
411     join(a1).\
412     join(a2).\
413     filter(a1.email_address == 'jack@gmail.com').\
414     filter(a2.email_address == 'jack@hotmail.com').\
415     all()
416
417 # ## slide:: p
418 # We can also join with subqueries.  subquery() returns
419 # an "alias" construct for us to use.
420
421 from sqlalchemy import func
422
423 subq = session.query(
424     func.count(Address.id).label('count'),
425     User.id.label('user_id')
426 ).\
427     join(Address.user).\
428     group_by(User.id).\
429     subquery()
430
```

(continues on next page)

(продолжение с предыдущей страницы)

```
session.query(User.name, func.coalesce(subq.c.count, 0)).\
    outerjoin(subq, User.id == subq.c.user_id).all()

# ## slide::
# ## title:: Exercises
# 1. Run this SQL JOIN:
#
#    SELECT user.name, address.email_address FROM user
#    JOIN address ON user.id=address.user_id WHERE
#    address.email_address='j25@yahoo.com'
#
# 2. Tricky Bonus!  Select all pairs of distinct user names.
#    Hint: "... ON user_alias1.name < user_alias2.name"
#

# ## slide:: p
# ## title:: Eager Loading
# the "N plus one" problem refers to the many SELECT statements
# emitted when loading collections against a parent result

for user in session.query(User):
    print(user, user.addresses)

# ## slide:: p
# *eager loading* solves this problem by loading *all* collections
# at once.

session.rollback()  # so we can see the load happen again.

from sqlalchemy.orm import subqueryload

for user in session.query(User).options(subqueryload(User.addresses)):
    print(user, user.addresses)

# ## slide:: p
# joinedload() uses a LEFT OUTER JOIN to load parent + child in one query.

session.rollback()

from sqlalchemy.orm import joinedload

for user in session.query(User).options(joinedload(User.addresses)):
    print(user, user.addresses)

# ## slide:: p
```

(continues on next page)

(продолжение с предыдущей страницы)

```
# eager loading *does not* change the *result* of the Query.
# only how related collections are loaded.

for address in session.query(Address).\
        join(Address.user).\
        filter(User.name == 'jack').\
        options(joinedload(Address.user)):
    print(address, address.user)

# ## slide:: p
# to join() *and* joinedload() at the same time without using two
# JOIN clauses, use contains_eager()

from sqlalchemy.orm import contains_eager

for address in session.query(Address).\
        join(Address.user).\
        filter(User.name == 'jack').\
        options(contains_eager(Address.user)):
    print(address, address.user)

# ## slide:: p
# ## title:: Delete Cascades
# removing an Address sets its foreign key to NULL.
# We'd prefer it gets deleted.

jack = session.query(User).filter_by(name='jack').one()

del jack.addresses[0]
session.commit()

# ## slide::
# This can be configured on relationship() using
# "delete-orphan" cascade on the User->Address
# relationship.

User.addresses.property.cascade = "all, delete, delete-orphan"

# ## slide:: p
# Removing an Address from a User will now delete it.

fred = session.query(User).filter_by(name='fred').one()

del fred.addresses[0]
session.commit()
```

(continues on next page)

(продолжение с предыдущей страницы)

```
521
522 # ## slide:: p
523 # Deleting the User will also delete all Address objects.
524
525 session.delete(jack)
526 session.commit()
527
528 # ## slide::
529 # ## title:: Exercises - Final Exam !
530 # 1. Create a class called 'Account', with table "account":
531 #
532 #      id = Column(Integer, primary_key=True)
533 #      owner = Column(String(50), nullable=False)
534 #      balance = Column(Numeric, default=0)
535 #
536 # 2. Create a class "Transaction", with table "transaction":
537 #      * Integer primary key
538 #      * numeric "amount" column
539 #      * Integer "account_id" column with ForeignKey('account.id')
540 #
541 # 3. Add a relationship() on Transaction named "account", which refers
542 #    to "Account", and has a backref called "transactions".
543 #
544 # 4. Create a database, create tables, then insert these objects:
545 #
546 #      a1 = Account(owner='Jack Jones', balance=5000)
547 #      a2 = Account(owner='Ed Rendell', balance=10000)
548 #      Transaction(amount=500, account=a1)
549 #      Transaction(amount=4500, account=a1)
550 #      Transaction(amount=6000, account=a2)
551 #      Transaction(amount=4000, account=a2)
552 #
553 # 5. Produce a report that shows:
554 #     * account owner
555 #     * account balance
556 #     * summation of transaction amounts per account (should match balance)
557 #       A column can be summed using func.sum(Transaction.amount)
558 #
559 # from sqlalchemy import Integer, String, Numeric
560
561 # ## slide::
```

Применение и аналоги

SQLAlchemy находит применение в веб-фреймворках TurboGears, Pylons, Pyramid, Zope, Flask. Например, известный социальный новостной сайт Reddit построен с использованием SQLAlchemy. Список организаций, использующих SQLAlchemy, можно найти на сайте проекта.

Пагинация

См.также:

- https://github.com/Pylons/paginate_sqlalchemy

`paginate_sqlalchemy` выполняет то же, что и библиотека https://github.com/Pylons/paginate, но гораздо быстрее для SQLAlchemy.

Код 88: 3.pagination/example.py

```
from sqlalchemy import Column, Integer, String, create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

Base = declarative_base()


class Person(Base):
    __tablename__ = 'person'

    id = Column(Integer, primary_key=True)
    name = Column(String(250), nullable=False)

    def __repr__(self):
        return "<{}>".format(self.name)

engine = create_engine('sqlite:///sqlalchemy_example.db')

Base.metadata.drop_all(engine)
Base.metadata.create_all(engine)

DBSession = sessionmaker(bind=engine)
session = DBSession()

for i in range(100):
    new_person = Person(name='new person #%s' % i)
    session.add(new_person)
session.commit()
```

(continues on next page)

(продолжение с предыдущей страницы)

```
29
30 query = session.query(Person)
31 print(query.count())  # 100
32 print
33
34 from paginate_sqlalchemy import SqlalchemyOrmPage
35
36 page = SqlalchemyOrmPage(query, page=5, items_per_page=8)
37 print(page)
38 print(page.items)
39 print(page.items[6].name)
40 print(page.page_count)
```

Результат выполнения

```
100

Page:
Collection type:        <class 'sqlalchemy.orm.query.Query'>
Current page:           5
First item:             33
Last item:              40
First page:             1
Last page:              13
Previous page:          4
Next page:              6
Items per page:         8
Total number of items:  100
Number of pages:        13

[<new person #32>, <new person #33>, <new person #34>, <new person #35>, <new␣
↪person #36>, <new person #37>, <new person #38>, <new person #39>]
new person #38
13
```

Формы

См.также:

- http://docs.formalchemy.org/
- http://colanderalchemy.readthedocs.org
- http://colanderalchemy.readthedocs.org/en/latest/deform.html

Код 89: 4.form/example.py

```
from sqlalchemy import Column, Integer, String
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()


class Person(Base):
    __tablename__ = 'person'

    id = Column(Integer, primary_key=True)
    name = Column(String(250), nullable=False)

    def __repr__(self):
        return "<{}>".format(self.name)

from colanderalchemy import SQLAlchemySchemaNode
person = SQLAlchemySchemaNode(Person)

from deform import Form
form = Form(person, buttons=('submit',))
print(form.render())
```

Код 90: 4.form/example.txt

```
<form
  id="deform"
  method="POST"
  enctype="multipart/form-data"
  accept-charset="utf-8" class="deform"
  >

  <fieldset class="deformFormFieldset">



    <input type="hidden" name="_charset_" />
    <input type="hidden" name="__formid__" value="deform"/>




    <ul>
<li class="field item-id "
    title=""
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    id="item-deformField1">

  <!-- mapping_item -->

  <label
         class="desc"
         title=""
         for="deformField1"
         >Id
  </label>


    <input type="text" name="id" value=""
           id="deformField1"/>









  <!-- /mapping_item -->

</li>
</ul>

    <ul>
<li class="field item-name "
    title=""
    id="item-deformField2">

  <!-- mapping_item -->

  <label
         class="desc"
         title=""
         for="deformField2"
         >Name<span class="req"
                        id="req-deformField2">*</span>
  </label>


    <input type="text" name="name" value=""
           id="deformField2"/>


```

(continues on next page)

(продолжение с предыдущей страницы)

```
66
67
68
69
70    <!-- /mapping_item -->
71
72  </li>
73  </ul>
74
75
76      <ul>
77
78        <li class="buttons">
79
80            <button
81                id="deformsubmit"
82                name="submit"
83                type="submit"
84                class="btnText submit "
85                value="submit">
86              <span>Submit</span>
87            </button>
88
89        </li>
90
91      </ul>
92
93    </fieldset>
94
95
96
97  </form>
```

Блог

Код 91: models.py - данные хранятся в БД SQLite

```
1  # -*- coding: utf-8 -*-
2  from jinja2.utils import generate_lorem_ipsum
3  from sqlalchemy import Column, Integer, String, Text, create_engine
4  from sqlalchemy.ext.declarative import declarative_base
5  from sqlalchemy.orm import sessionmaker
6
7  Base = declarative_base()
```

(continues on next page)

(продолжение с предыдущей страницы)

```

class Articles(Base):
    __tablename__ = 'articles'

    id = Column(Integer, primary_key=True)
    title = Column(String(250), nullable=False)
    content = Column(Text, nullable=False)

    def __repr__(self):
        return "<{}>".format(self.name)


engine = create_engine('sqlite:///foo.db')

Base.metadata.drop_all(engine)
Base.metadata.create_all(engine)

Session = sessionmaker(bind=engine)
dbsession = Session()

for id, article in enumerate(range(100), start=1):
    title = generate_lorem_ipsum(
        n=1,          # Одно предложение
        html=False,   # В виде обычного текста
        min=2,        # Минимум 2 слова
        max=5         # Максимум 5
    )
    content = generate_lorem_ipsum()
    article = Articles(**{'id': id, 'title': title, 'content': content})
    dbsession.add(article)
dbsession.commit()
dbsession.close()
```

Код 92: views.py - SQLAlchemy

```
# -*- coding: utf-8 -*-
import deform
from jinja2 import Environment, FileSystemLoader
from webob import Request, Response

from common import get_csrf_token, get_session
from models import Session, Articles


```

(continues on next page)

(продолжение с предыдущей страницы)

```
env = Environment(loader=FileSystemLoader('templates'))


def wsgify(view):
    def wrapped(environ, start_response):
        request = Request(environ)
        app = view(request).response()
        return app(environ, start_response)
    return wrapped


class BaseArticle(object):

    def __init__(self, request):
        self.request = request
        article_id = self.request.environ['wsgiorg.routing_args'][1]['id']
        dbsession = Session()
        self.article = dbsession.query(Articles).filter_by(id=article_id).one()
        dbsession.close()


class BaseArticleForm(object):

    def get_form(self):
        from forms import CreateArticle
        self.session = get_session(self.request)
        self.session['csrf'] = get_csrf_token(self.session)
        schema = CreateArticle().bind(request=self.request)
        submit = deform.Button(name='submit',
                               css_class='blog-form__button')
        self.form = deform.Form(schema, buttons=(submit,))
        return self.form


@wsgify
class BlogIndex(object):

    def __init__(self, request):
        self.page = request.GET.get('page', '1')
        from paginate import Page
        dbsession = Session()
        articles = dbsession.query(Articles).all()
        self.paged_articles = Page(
            articles,
            page=self.page,
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            items_per_page=8,
        )
        dbsession.close()

    def response(self):
        return Response(env.get_template('index.html')
                        .render(articles=self.paged_articles))


@wsgify
class BlogCreate(BaseArticleForm):

    def __init__(self, request):
        self.request = request

    def response(self):
        if self.request.method == 'POST':
            submitted = self.request.POST.items()
            try:
                self.get_form().validate(submitted)
            except deform.ValidationFailure as e:
                return Response(
                    env.get_template('create.html').render(form=e.render()))
            article = Articles(**{'title': self.request.POST['title'],
                                  'content': self.request.POST['content']
                                  })
            dbsession = Session()
            dbsession.add(article)
            dbsession.commit()
            dbsession.close()
            self.session = get_session(self.request).pop('csrf')
            return Response(status=302, location='/')
        return Response(env.get_template('create.html')
                        .render(form=self.get_form().render()))


@wsgify
class BlogRead(BaseArticle):

    def response(self):
        if not self.article:
            return Response(status=404)
        return Response(env.get_template('read.html')
                        .render(article=self.article))

```

(continues on next page)

(продолжение с предыдущей страницы)

```
100
101 @wsgify
102 class BlogUpdate(BaseArticle, BaseArticleForm):
103
104     def response(self):
105         if self.request.method == 'POST':
106             submitted = self.request.POST.items()
107             try:
108                 self.get_form().validate(submitted)
109             except deform.ValidationFailure as e:
110                 return Response(
111                     env.get_template('create.html').render(form=e.render()))
112             self.article.title = self.request.POST['title']
113             self.article.content = self.request.POST['content']
114             dbsession = Session()
115             dbsession.add(self.article)
116             dbsession.commit()
117             dbsession.close()
118             self.session = get_session(self.request).pop('csrf')
119             return Response(status=302, location='/')
120         return Response(
121             env.get_template('create.html')
122             .render(form=self.get_form().render(
123                 self.article.__dict__)))
124
125
126 @wsgify
127 class BlogDelete(BaseArticle):
128
129     def response(self):
130         dbsession = Session()
131         dbsession.delete(self.article)
132         dbsession.commit()
133         dbsession.close()
134         return Response(status=302, location='/')
```

1.5 Фреймворк Pyramid

См.также:

- Pyramid
- http://trypyramid.com

- https://ru.wikipedia.org/wiki/Pyramid_(\T2A\cyrp\T2A\cyrr\T2A\cyro\T2A\cyrg\T2A\cyrr\T2A\cyra\T2A\cyrm\T2A\cyrm\T2A\cyrn\T2A\cyrery\T2A\cyrishrt_\T2A\cyrk\T2A\cyra\T2A\cyrr\T2A\cyrk\T2A\cyra\T2A\cyrs)

1.5.1 Введение

На создание Pyramid оказали влияние такие фреймворки, как Zope, Pylons и Django. Код Pyramid разрабатывался в проекте repoze.bfg, а название поменялось в результате слияния проектов BFG и Pylons, по решению встречи разработчиков в отеле Luxor (который имеет форму пирамиды, откуда и пошло название фреймворка) в Лас-Вегасе, в 2010 году.

См.также:

- https://trypyramid.com
- awesome-pyramid
- **EN** -
- **RU** -

Пирамида — самый молодой фреймворк для синхронного Веба среди популярных *Python* фреймворков. Разработчики из сообщества Pylons не стали развивать тупиковую ветвь каркасных фреймворков с жестко заданной архитектурой, к которым относятся Pylons и например Ruby On Rails, поняли ошибки монолитных тяжелых фреймворков типа Zope или Django и создали минималистичный, очень гибкий, но, в то же время, легко расширяемый инструмент, сконцентрировав свои усилия на основных задачах фреймворка, как: обработка маршрутов, простой и расширяемый конфиг, система событий и *middleware* (tweens), простая система авторизации построенная на *ACL*, возможность задания маршрутов динамически в виде бинарного дерева и привязки их к ресурсам. Всеми остальными задачами занимаются сторонние библиотеки. По требованию программиста можно выбрать любой `ORM` для работы с БД, любой шаблонизатор, придумать любую схему авторизации и прочее.

Такой подход не ограничивает программиста в архитектуре проекта, в выборе инструментов и не заставляет для решения узких задач тянуть множество ненужных зависимостей и функционала.

Установка

См.также:

- http://docs.pylonsproject.org/projects/pyramid/en/latest/narr/install.html
- http://docs.pylonsproject.org/projects/pyramid/en/1.6-branch/narr/firstapp.html

- http://docs.pylonsproject.org/projects/pyramid/en/latest/quick_tour.html

Обычно, достаточно выполнить:

```
pip install pyramid
```

Hello World

Код 93: helloworld.py - Pyramid приложение в одном файле

```
1  from wsgiref.simple_server import make_server
2  from pyramid.config import Configurator
3  from pyramid.response import Response
4
5
6  def hello(request):
7      return Response('Hello world!')
8
9  if __name__ == '__main__':
10     config = Configurator()
11     config.add_route('hello_world', '/')
12     config.add_view(hello, route_name='hello_world')
13     app = config.make_wsgi_app()
14     server = make_server('0.0.0.0', 8000, app)
15     server.serve_forever()
```

Сохраните код в файл `helloworld.py` и запустите его (`python helloworld.py`). Теперь приложение доступно на 8000 порту. По адресу http://localhost:8000/ отобразится «Hello World».

Так просто, в одном файле, запустить Веб приложение не получится ни в *Django*, ни в *Zope*, это можно сравнить разве что с *WSGI* приложением или минималистичным фреймворком Bottle, который сильно уступает пирамиде по возможностям масштабирования.

Импорты

Код 94: helloworld.py - импортированные модули

```
from wsgiref.simple_server import make_server
from pyramid.config import Configurator
from pyramid.response import Response
```

Pyramid приложение начинается с конфига, который создается при помощи класса `Configurator` из модуля `pyramid.config`. В дальнейшем, экземпляр класса `Configurator` используется для настройки.

Как и многие другие Веб-фреймворки на Python, Pyramid использует *WSGI* протокол для связи приложения и веб-сервера. В этом примере выбран Веб-сервер `wsgiref` для удобства, т.к. он встроен в Python.

`pyramid.response.Response` копия класса `Response` из библиотеки webob. Используется для формирования HTTP ответа.

View

Код 95: helloworld.py - функция hello

```
def hello(request):
    return Response('Hello world!')
```

Такой тип представления в Pyramid называется *view callable*, принимает в качестве аргумента объект класса `pyramid.request.Request` (который наследуется от `webob.request.BaseRequest`) и возвращает объект HTTP ответа `pyramid.response.Response`.

Конфигурация

Код 96: Создаем конфигуратор приложения

```
config = Configurator()
```

```
1 config.add_route('hello_world', '/')
2 config.add_view(hello, route_name='hello_world')
```

В первой строчке вызывается метод конфигуратора `pyramid.config.Configurator.add_route()`, который регистрирует новый маршрут (*route*) с названием «hello_world» и привязывает его к корневому пути сайта «/».

Вторая строка регистрирует функцию `hello(request)` как *view callable* и привязывает ее к маршруту «hello_world». Теперь при обращении по URL адресу http://localhost:8000/ будет запускаться функция `hello` с переданным ей объектом запроса `request`.

WSGI приложение

```
1 app = config.make_wsgi_app()
```

Метод `pyramid.config.Configurator.make_wsgi_app()` формирует *WSGI* приложение из информации, которая хранится в конфигураторе. В дальнейшем, благодаря спецификации *WSGI* (**PEP 333**), можно запустить это приложение на любом совместимом Веб-сервере.

WSGI сервер

```
1 server = make_server('0.0.0.0', 8000, app)
2 server.serve_forever()
```

WSGI-сервер `wsgiref` принимает первым параметром адрес „0.0.0.0“ (доступен извне, в отличие от „127.0.0.1“ по умолчанию), вторым — порт „8000“, третий параметр — это WSGI-приложение, в пирамиде конечное приложение является объектом класса `pyramid.router.Router` (*Router*).

Функция `serve_forever` запускает WSGI приложение.

Резюме

Мы написали очень простое Веб-приложение используя Pyramid фреймворк и настроив его императивно, это означает что настройки были прописаны напрямую в объект конфигуратора (класс `pyramid.config.Configurator`).

1.5.2 Конфигурация

Как и обычный список настроек, конфигуратор инициализирует начальные значения (либо из файла «*.ini», либо через параметры класса). Отличительной особенностью конфигуратора является то, что по мере выполнения программы настройки могут меняться или добавляться.

В Pyramid существует 2 способа настройки приложений Императивный и Декларативный.

Императивный способ конфигурации

Императивный способ конфигурации означает что команды языка Python будут выполнены одна за другой, последовательно. Ниже пример простого приложения на Пирамиде сконфигурированного императивно.

Примечание: Пример будет доступен по адресу http://localhost:8080/hello/

```
from wsgiref.simple_server import make_server
from pyramid.config import Configurator
from pyramid.response import Response

def hello_world(request):
    return Response('Hello world!')

if __name__ == '__main__':
    config = Configurator()

    config.add_route('myHelloRoute', '/hello/')
    config.add_view(hello_world, route_name='myHelloRoute')

    # Создаем и запускаем WSGI приложение
    app = config.make_wsgi_app()
    server = make_server('0.0.0.0', 8080, app)
    server.serve_forever()
```

Декларативный способ конфигурации

Иногда бывает сложно выполнить все настройки императивно в одном месте, т.к. приложение обычно состоит из множества файлов. В таком случае, вам придется постоянно перескакивать между файлами, чтобы посмотреть настройки для блока кода из другого файла. Чтобы этого избежать, фреймворк Pyramid позволяет настраивать приложение декларативных способом (*configuration decoration*), т.е. добавлять настройки как можно ближе к целевому коду, как показано в примере ниже:

Код 97: my-pyramid-app/views.py

```
from pyramid.response import Response
from pyramid.view import view_config

@view_config(route_name='myHelloRoute')
def hello_world(request):
    return Response('Hello')
```

Сам по себе декоратор `pyramid.view.view_config` не произведет ни какого эффекта. Чтобы приложение нашло и применило эти настройки нужно выполнить метод `pyramid.config.Configurator.scan()` (*scan*). После выполнения этот метод проходит по всем нижележащим файлам от текущей директории, ищет декларативное описание настроек и применяет их к проекту.

Код 98: my-pyramid-app/__init__.py

```
from wsgiref.simple_server import make_server
from pyramid.config import Configurator

if __name__ == '__main__':
    config = Configurator()

    config.add_route('myHelloRoute', '/hello/')
    config.scan()

    # Создаем и запускаем WSGI приложение
    app = config.make_wsgi_app()
    server = make_server('0.0.0.0', 8080, app)
    server.serve_forever()
```

В примере выше декоратор `view_config` делает то же что метод `pyramid.config.Configurator.add_view()` но более наглядно:

```
config.add_view(hello_world, route_name='myHelloRoute')
```

Можно этот пример записать в одном файле:

```
from wsgiref.simple_server import make_server
from pyramid.config import Configurator
from pyramid.response import Response
from pyramid.view import view_config

@view_config(route_name='myHelloRoute')
def hello_world(request):
    return Response('Hello world!')

if __name__ == '__main__':
    config = Configurator()

    config.add_route('myHelloRoute', '/hello/')
    config.scan()

    # Создаем и запускаем WSGI приложение
    app = config.make_wsgi_app()
    server = make_server('0.0.0.0', 8080, app)
    server.serve_forever()
```

Резюме

Вы можете выбрать любой способ конфигурации, который вам понравится. Оба способа полностью эквивалентны и часто используются совместно, решая разные задачи конфигурации приложения более удобным способом.

1.5.3 Структура приложения

Хотя не составляет большой трудности написать Pyramid-приложение (проект) с нуля, Pyramid имеет инструменты для инициализации кода нового приложения по выбранному шаблону, или, в терминологии Pyramid, каркасной структуре (scaffolds). Например, в поставке имеются каркасные структуры для проектов, использующих ZODB или SQLAlchemy.

Проект — это каталог, содержащий по крайней мере один пакет на Python.

Типичная структура каталога для небольшого проекта:

```
MyProject/
|-- CHANGES.txt
|-- development.ini
|-- MANIFEST.in
|-- myproject
|   |-- __init__.py
|   |-- static
|   |   |-- favicon.ico
|   |   |-- logo.png
|   |   `-- pylons.css
|   |-- templates
|   |   `-- mytemplate.pt
|   |-- tests.py
|   `-- views.py
|-- production.ini
|-- README.txt
|-- setup.cfg
`-- setup.py
```

Приведённую структуру, как следует из документации, не следует сильно изменять, так как это может помешать другим разработчикам быстро ориентироваться в коде проекта. Тем не менее, растущий проект может потребовать некоторых изменений. Например, виды, модели (если они используются) и тесты можно, разбив на модули, перенести соответственно в подкаталоги views, models и tests (не забыв снабдить их файлом `__init__.py`).

Следует отметить, что Pyramid может работать с любым WSGI-сервером. Проекты, созданные по готовым каркасным структурам, используют сервер Waitress.

Стандартные шаблоны проектов

Список официальных шаблонов найти по адресу https://github.com/Pylons?q=cookiecutter.

`starter` URL маршруты *URL dispatch*, без БД.

`zodb` URL маршрутизация *traversal* и БД *ZODB*.

`alchemy` URL маршрутизация *URL dispatch* и БД *SQLite* с использованием *SQLAlchemy*.

Некоторые пакеты могут дополнять этот список, например Cornice (https://github.com/Cornices/cookiecutter-cornice).

Cookiecutter

См.также:

- https://cookiecutter.readthedocs.io/en/latest/readme.html
- https://cookiecutter.readthedocs.io/en/latest/readme.html#python-pyramid

Cookiecutter - утилита позволяющая создавать проекты из шаблонов.

Установка *Linux*:

```
$ sudo apt install cookiecutter
```

Установка через *Python*:

```
$ pip install cookiecutter --user
```

Установка через Nix:

```
$ nix-env -i cookiecutter
```

Создание проекта

```
$ cookiecutter gh:Pylons/pyramid-cookiecutter-starter
project_name [Pyramid Scaffold]: myproject
repo_name [myproject2]: myproject
Select template_language:
1 - jinja2
2 - chameleon
3 - mako
Choose from 1, 2, 3 [1]: 3
```

(continues on next page)

(продолжение с предыдущей страницы)

```
================================================================================
Documentation: https://docs.pylonsproject.org/projects/pyramid/en/latest/
Tutorials:     https://docs.pylonsproject.org/projects/pyramid_tutorials/en/latest/
Twitter:       https://twitter.com/PylonsProject
Mailing List:  https://groups.google.com/forum/#!forum/pylons-discuss
Welcome to Pyramid.  Sorry for the convenience.
================================================================================

Change directory into your newly created project.
    cd myproject

Create a Python virtual environment.
    python3 -m venv env

Upgrade packaging tools.
    env/bin/pip install --upgrade pip setuptools

Install the project in editable mode with its testing requirements.
    env/bin/pip install -e ".[testing]"

Run your project's tests.
    env/bin/pytest

Run your project.
    env/bin/pserve development.ini
```

```
myproject/
├── CHANGES.txt
├── development.ini
├── MANIFEST.in
├── myproject
│   ├── __init__.py
│   ├── static
│   │   ├── pyramid-16x16.png
│   │   ├── pyramid.png
│   │   └── theme.css
│   ├── templates
│   │   ├── layout.mako
│   │   └── mytemplate.mako
│   ├── tests.py
│   └── views.py
├── production.ini
├── pytest.ini
├── README.txt
```

(continues on next page)

(продолжение с предыдущей страницы)

```
└── setup.py

3 directories, 15 files
```

Установка

```
$ cd myproject
$ python setup.py develop
```

Запуск

Часть настоек проекта, которые часто меняются, находится в файле `development.ini`.

```
$ pserve development.ini
Starting server in PID 16601.
serving on http://0.0.0.0:6543
```

Ниже показан пример настроек сервера. Сервер Waitress запустит `MyProject.main` по адресу `127.0.0.1` и порту `6543`.

Код 99: Пример настроек сервера из `development.ini`

```
[server:main]
use = egg:waitress#main
host = 127.0.0.1
port = 6543
```

Для автоматического перезапуска сервера после изменения файлов нужно указать флаг `--reload`.

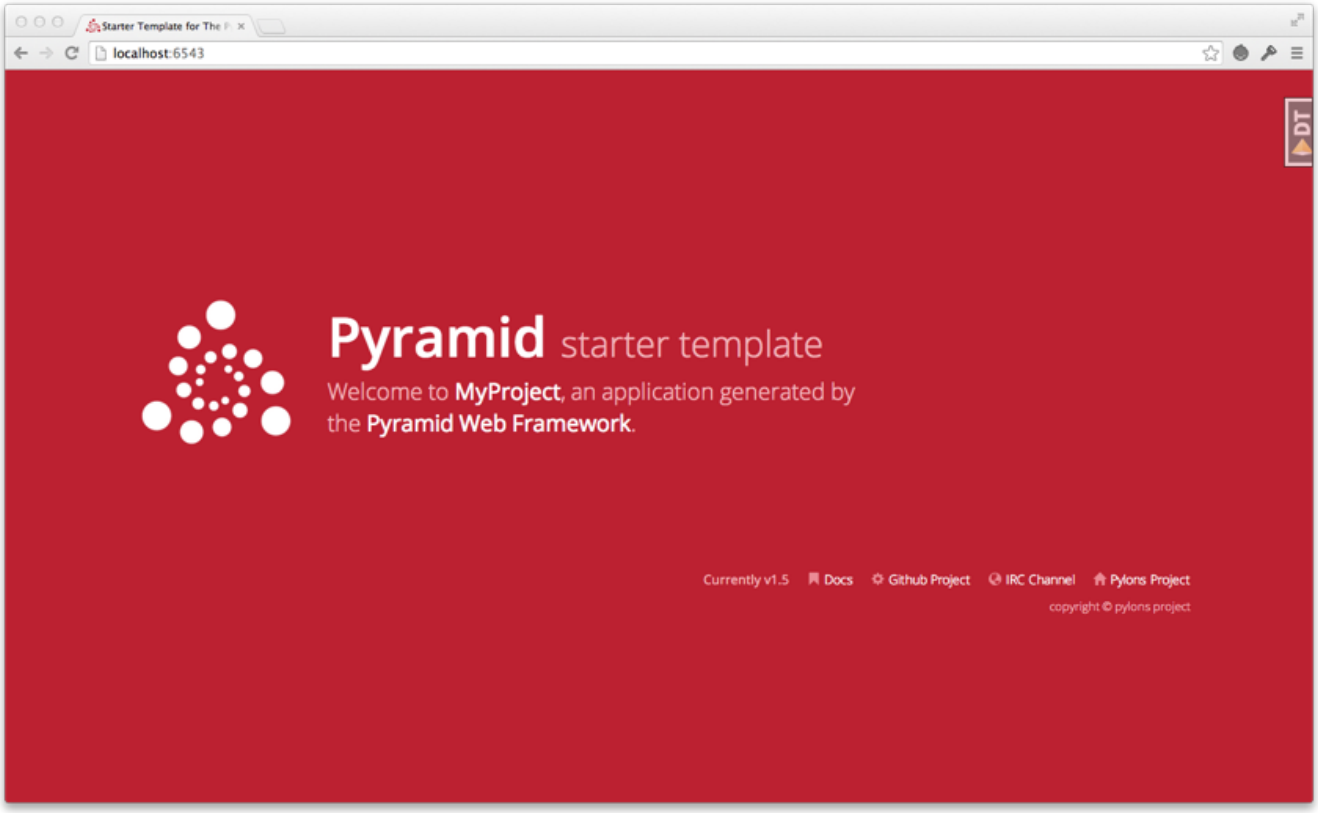
```
$ pserve development.ini --reload
Starting subprocess with file monitor
Starting server in PID 16601.
serving on http://0.0.0.0:6543
```

Теперь, после изменения какого-либо из файлов `.py` или `.ini`, сервер перезапустится автоматически.

```
development.ini changed; reloading...
-------------------- Restarting --------------------
Starting server in PID 16602.
serving on http://0.0.0.0:6543
```

Просмотр

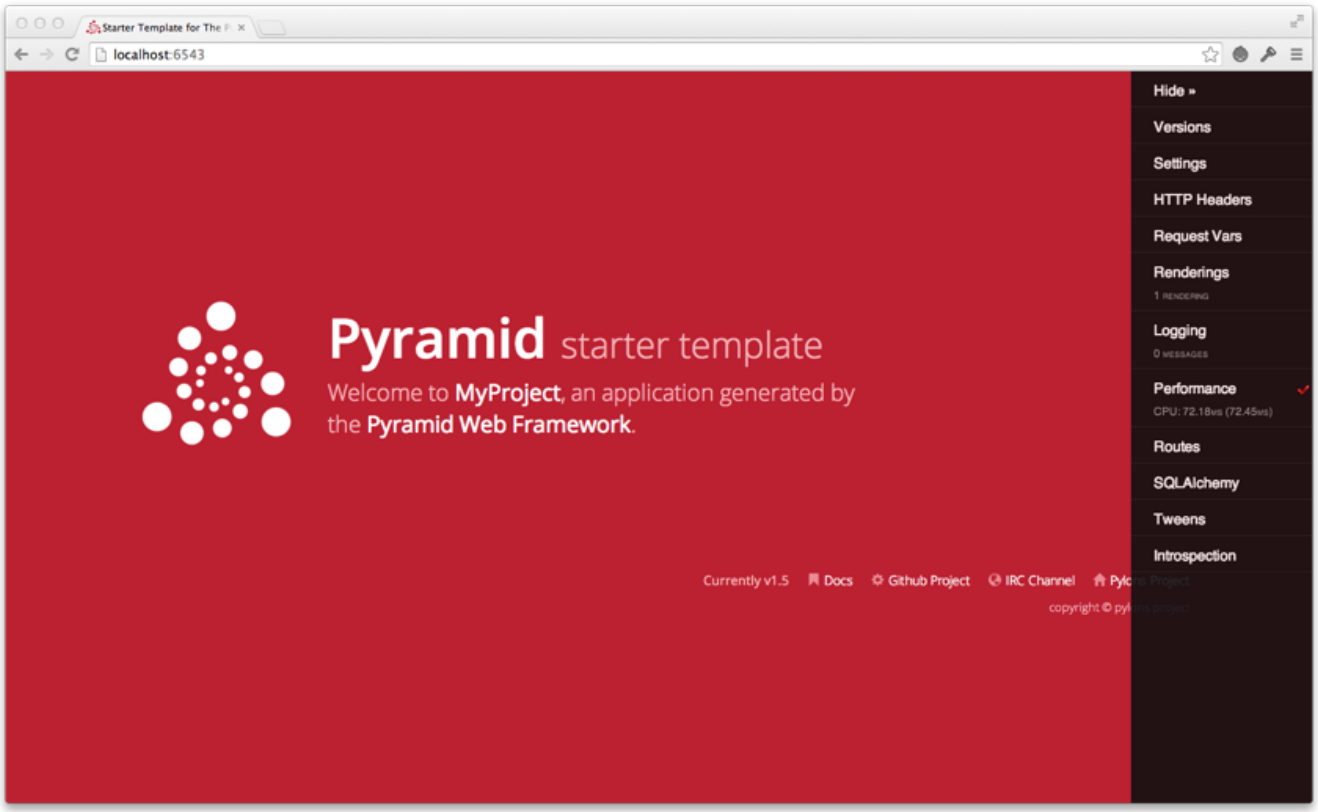
После запуска приложения через `pserve`, можно открыть страницу http://localhost:6543/ в браузере.



Debug Toolbar

См.также:

- http://docs.pylonsproject.org/projects/pyramid_debugtoolbar/en/latest/




```
[app:main]
pyramid.includes =
    pyramid_debugtoolbar
```

1.5.4 Настройки

См.также:

- http://docs.pylonsproject.org/docs/pyramid/en/latest/narr/environment.html

```
from pyramid.config import Configurator

if __name__ == '__main__':
    settings = {
       'debug_all': True,
       'default_locale_name': 'ru',
    }
    config = Configurator(settings=settings)
```

При помощи настоек можно поменять поведение проекта. Некоторые настройки используются самой пирамидой, другие нужны для вашего личного использования. Настройки можно задавать в виде Python словаря или переменных окружения.

Таблица 6: Переменные окружения для настройки Pyramid проекта

Переменная окружения	Словарь Python	Назначение
PYRAMID_RELOAD_ASSETS	pyramid.reload_assets или reload_assets	Не кэширует статику если «true»
PYRAMID_DEBUG_AUTHORIZATION	pyramid.debug_authorization или debug_authorization	Выводит информацию об ошибках и успехах авторизации в stderr если «true»
PYRAMID_DEBUG_NOTFOUND	pyramid.debug_notfound или debug_notfound	Выводит отладочтную информацию связанную с исключением NotFound в stderr если «true»
PYRAMID_DEBUG_ROUTEMATCH	pyramid.debug_routematch или debug_routematch	Показывает отладочную информацию о маршрутах если «true»
PYRAMID_PREVENT_HTTP_CACHE	pyramid.prevent_http_cache или prevent_http_cache	Не использует закешированные запросы если «true»
PYRAMID_DEBUG_ALL	pyramid.debug_all или debug_all	Активирует все настройки начинающиеся с «debug. . . »
PYRAMID_DEFAULT_LOCALE_NAME	pyramid.default_locale_name или default_locale_name	локаль по умолчанию, например `pyramid.default_locale_name = ru`

Includes

Приложение на Pyramid можно дополнять при помощи сторонних модулей или собственных функций. Делается это при помощи метода конфигуратора `pyramid.config.Configurator.include()` или настройки `pyramid.includes`.

Расширение приложения собственными средствами

Метод `include()` вызывает функцию `moreconfiguration` и передает ей в качестве параметра объект конфигуратора. Тем самым как бы являясь продолжением этого конфига. Такой способ называется императивное расширение приложения.

```
def moreconfiguration(config):
    config.add_route('goodbye', '/goodbye')
    config.add_view(goodbye, route_name='goodbye')


config = Configurator()
```

(continues on next page)

(продолжение с предыдущей страницы)

```
config.add_route('home', '/')
config.add_view(hello_world, route_name='home')
config.include(moreconfiguration)
app = config.make_wsgi_app()
```

Расширение через модули

Функции можно добавлять и из других модулей приложения.

```
# myapp.mysubmodule.views.py

def my_view(request):
    from pyramid.response import Response
    return Response('OK')

def includeme(config):
    config.add_view(my_view)
```

Теперь в `include` указывается путь до нашей функции с расширением конфига. Причем его можно указать в виде строки.

```
from pyramid.config import Configurator

def main(global_config, **settings):
    config = Configurator()
    config.include('myapp.mysubmodule.views.includeme')
```

Конфигуратор по умолчанию ищет функцию `includeme` поэтому ее можно опустить.

```
from pyramid.config import Configurator

def main(global_config, **settings):
    config = Configurator()
    config.include('myapp.mysubmodule.views')
```

Расширение через ini файл

В ini файл настроек приложения, помимо самих настроек, так же можно включить расширения при помощи параметра `pyramid.includes`.

```
[app:main]
pyramid.includes = pyramid_debugtoolbar
                   pyramid_tm
```

Этот код эквивалентен следующему:

```
from pyramid.config import Configurator

def main(global_config, **settings):
    config = Configurator(settings=settings)
    # ...
    config.include('pyramid_debugtoolbar')
    config.include('pyramid_tm')
    # ...
```

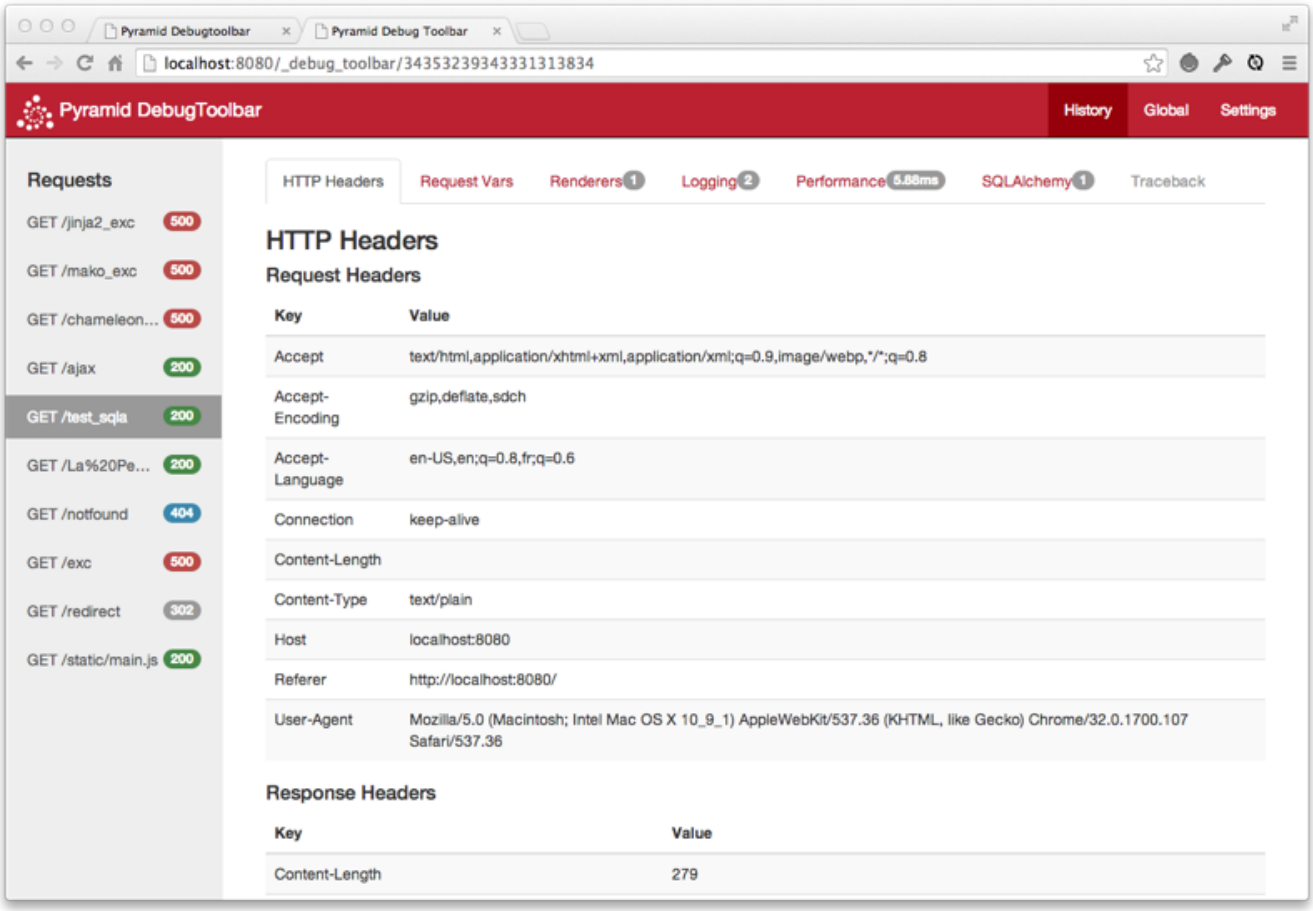
Сторонние модули

Для пирамиды существует огромное множество сторонних модулей которые расширяю функционал вашего приложения. Вот список некоторых их них https://github.com/uralbash/awesome-pyramid.

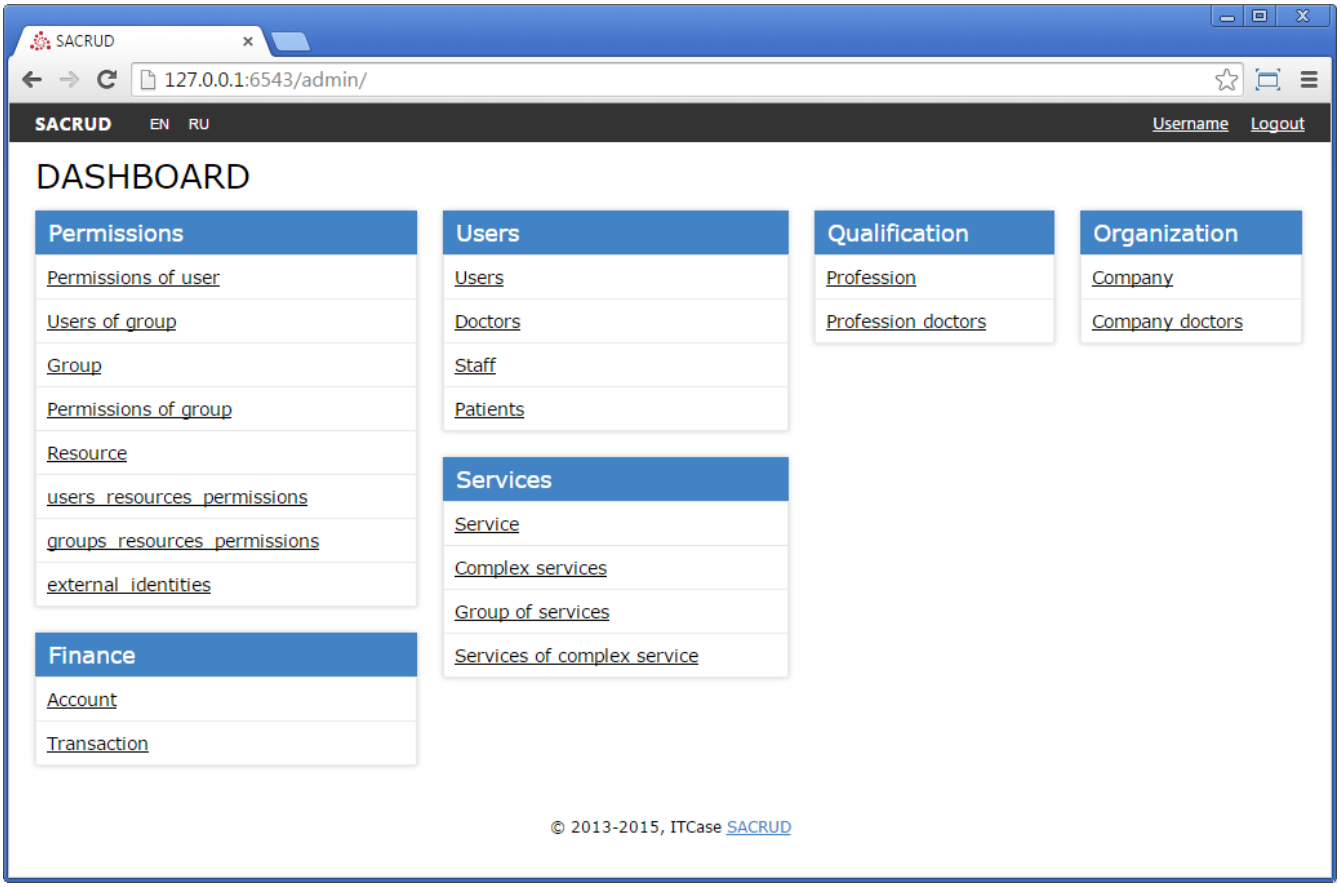
Примечание: Сторонние модули устанавливаются как обычные python пакеты в ваше окружение. Например:

```
pip install pyramid_debugtoolbar pyramid_jinja2 pyramid_sacrud
```

Например добавляя `config.include('pyramid_debugtoolbar')` мы получаем отладочную консоль с Веб интерфейсом.



Или админку `config.include('pyramid_sacrud')`.



`config.include('pyramid_jinja2')` добавляет Jinja2 рендерер для ваших view.

```
@view_config(renderer='templates/mytemplate.jinja2')
def my_view(request):
    return {'foo': 1, 'bar': 2}
```

И так далее...

В виде Python словаря

Настройки в конфигуратор предаются в виде обычного Python словаря.

```
from pyramid.config import Configurator

if __name__ == '__main__':
    settings = {
       'default_locale_name': 'ru',
       'pyramid.includes': 'pyramid_debugtoolbar pyramid_tm',
    }
    config = Configurator(settings=settings)
    print(config.registry.settings)
```

Модули pyramid_debugtoolbar и pyramid_tm автоматически добавляют свои настройки при включении их в конфиг.

```
{'debug_authorization': False,
 'debug_notfound': False,
 'debug_routematch': False,
 'debug_templates': False,
 'debugtoolbar.button_style': '',
 'debugtoolbar.debug_notfound': False,
 'debugtoolbar.debug_routematch': False,
 'debugtoolbar.enabled': True,
 'debugtoolbar.exclude_prefixes': [],
 'debugtoolbar.extra_global_panels': [],
 'debugtoolbar.extra_panels': [],
 'debugtoolbar.global_panels': [<class 'pyramid_debugtoolbar.panels.introspection.
↪IntrospectionDebugPanel'>,
                                <class 'pyramid_debugtoolbar.panels.routes.
↪RoutesDebugPanel'>,
                                <class 'pyramid_debugtoolbar.panels.settings.
↪SettingsDebugPanel'>,
                                <class 'pyramid_debugtoolbar.panels.tweens.
↪TweensDebugPanel'>,
                                <class 'pyramid_debugtoolbar.panels.versions.
↪VersionDebugPanel'>],
 'debugtoolbar.hosts': ['127.0.0.1', '::1'],
 'debugtoolbar.includes': (),
 'debugtoolbar.intercept_exc': 'debug',
 'debugtoolbar.intercept_redirects': False,
 'debugtoolbar.max_request_history': 100,
 'debugtoolbar.max_visible_requests': 10,
 'debugtoolbar.panels': [<class 'pyramid_debugtoolbar.panels.headers.
↪HeaderDebugPanel'>,
                         <class 'pyramid_debugtoolbar.panels.logger.LoggingPanel'>,
                         <class 'pyramid_debugtoolbar.panels.performance.
↪PerformanceDebugPanel'>,
                         <class 'pyramid_debugtoolbar.panels.renderings.
↪RenderingsDebugPanel'>,
                         <class 'pyramid_debugtoolbar.panels.request_vars.
↪RequestVarsDebugPanel'>,
                         <class 'pyramid_debugtoolbar.panels.sqla.SQLADebugPanel'>,
                         <class 'pyramid_debugtoolbar.panels.traceback.
↪TracebackPanel'>],
 'debugtoolbar.prevent_http_cache': False,
 'debugtoolbar.reload_assets': False,
 'debugtoolbar.reload_resources': False,
 'debugtoolbar.reload_templates': False,
 'default_locale_name': 'ru',
```

(continues on next page)

(продолжение с предыдущей страницы)

```
 'prevent_cachebust': False,
 'prevent_http_cache': False,
 'pyramid.debug_authorization': False,
 'pyramid.debug_notfound': False,
 'pyramid.debug_routematch': False,
 'pyramid.debug_templates': False,
 'pyramid.default_locale_name': 'ru',
 'pyramid.includes': 'pyramid_debugtoolbar pyramid_tm',
 'pyramid.prevent_cachebust': False,
 'pyramid.prevent_http_cache': False,
 'pyramid.reload_assets': False,
 'pyramid.reload_resources': False,
 'pyramid.reload_templates': False,
 'reload_assets': False,
 'reload_resources': False,
 'reload_templates': False}
```

В ini файле

См.также:

- http://pyramid.readthedocs.org/en/master/quick_tutorial/ini.html
- http://pyramid.readthedocs.org/en/master/narr/paste.html#paste-chapter

Настройки можно хранить в ini файле, включая туда не только настройки пирамиды, но и настройки веб сервера. Т.е. мы указывает веб серверу как запускать наше приложение на Pyramid.

В разделе `[app:main]` указываются настройки Pyramid приложения.

```
###
# app configuration
# http://docs.pylonsproject.org/projects/pyramid/en/latest/narr/environment.html
###

[app:main]
use = egg:MyProject

pyramid.reload_templates = true
pyramid.debug_authorization = false
pyramid.debug_notfound = false
pyramid.debug_routematch = false
pyramid.default_locale_name = en
pyramid.includes =
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    pyramid_debugtoolbar

# By default, the toolbar only appears for clients from IP addresses
# '127.0.0.1' and '::1'.
# debugtoolbar.hosts = 127.0.0.1 ::1
```

В разделе `[server:main]` указываются настройки веб сервера совместимые с Paste Deployment.

```
###
# wsgi server configuration
###

[server:main]
use = egg:waitress#main
host = 0.0.0.0
port = 6543
```

Запуск приложения.

```
pserver development.ini
```

Утилита `pserve` читает конфиг, вызывает функцию `MyProject.main` с настройками из этого конфига, получает WSGI приложение от этой функции и запускает его, в нашем случае, при помощи веб сервера waitress.

```
from pyramid.config import Configurator


def main(global_config, **settings):
    """ This function returns a Pyramid WSGI application.
    """
    config = Configurator(settings=settings)
    ...
    return app
```

Резюме

Pyramid предоставляет очень мощную систему настройки вашего приложения, а так же простой и гибкий метод расширения функционала при помощи `include()`.

- Настройки которые практически ни когда не меняются или формируются динамически удобно создавать в коде приложения.
- Настройки которые вводятся вручную и могут меняться лучше хранить в ini фай-

ле, например это может быть строка подключения к БД, нет смысла её прописывать в коде, т.к. наверняка на реальном сервере она будет отличаться. Вы же не будете править код на сервере? Лучше создать отдельный файл настроек под сервер.

1.5.5 Базы данных (Models)

Сам фреймворк Pyramid не имеет встроенных возможностей работы с базами данных, в отличии от таких фреймворков как Django (Django ORM) и Ruby on Rails (Active Record). Хорошим выбором для реляционных БД будет ORM SQLAlchemy.

SQLAlchemy

Организация БД в пирамиде не зависит от фреймворка, поэтому можно использовать любую структуру, которая вам удобна. Ниже я приведу один из вариантов, более подробно про SQLAlchemy можно прочитать в разделе *SQLAlchemy ORM*.

Вынесем модели и то что касается соединения с БД в отдельный файл `models.py`.

```
# models.py
from sqlalchemy import Column, Integer, Text, create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

engine = create_engine('sqlite:///foo.db')
Session = sessionmaker()
Base = declarative_base(bind=engine)


class User(Base):
    __tablename__ = 'users'
    id = Column(Integer, primary_key=True)
    name = Column(Text)

    def __repr__(self):
        return self.name
```

В представлениях мы просто создаем объект `sqlalchemy.orm.session.Session` и работаем с объектами, как описано в документации SQLAlchemy. При этом в каждом представлении нам необходимо создавать новую SQLAlchemy сессию, а если были изменения подтверждать их при помощи метода `sqlalchemy.orm.session.Session.commit()`.

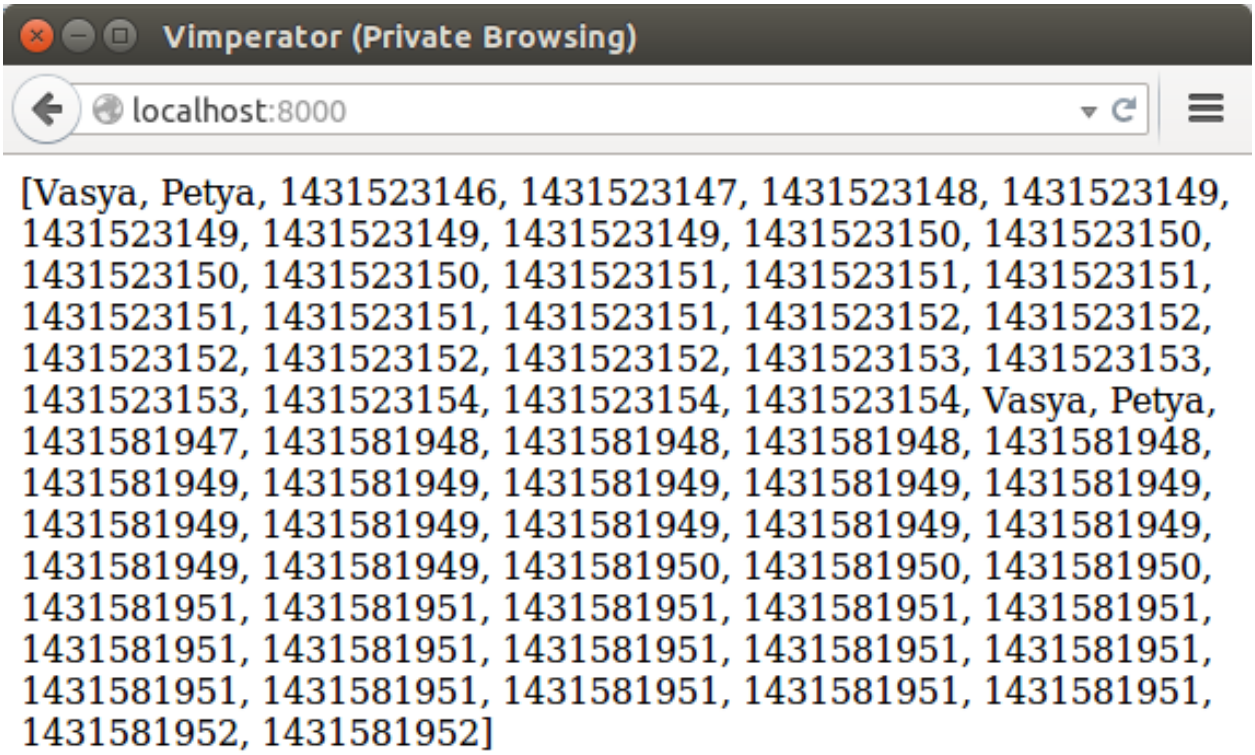
```
# __init__.py
from wsgiref.simple_server import make_server
```

(continues on next page)

(продолжение с предыдущей страницы)

```
from pyramid.config import Configurator
from pyramid.response import Response
from models import User, Session, Base, engine


def hello(request):
    DBSession = Session(bind=engine)
    result = DBSession.query(User).all()
    import time
    timestamp = int(time.time())
    new_user = User(name=str(timestamp))
    DBSession.add(new_user)
    DBSession.commit()
    return Response(str(result))

if __name__ == '__main__':
    Base.metadata.create_all()
    DBSession = Session(bind=engine)
    DBSession.add(User(name='Vasya'))
    DBSession.add(User(name='Petya'))
    DBSession.commit()

    config = Configurator()
    config.add_route('hello_world', '/')
    config.add_view(hello, route_name='hello_world')
    app = config.make_wsgi_app()
    server = make_server('0.0.0.0', 8000, app)
    server.serve_forever()
```

Данный пример при каждом обновлении делает новую запись в БД и отдает их браузеру.



ZopeTransactionExtension

См.также:

- https://pypi.python.org/pypi/zope.sqlalchemy
- https://metaclassical.com/what-the-zope-transaction-manager-means-to-me-and-you/

transaction

См.также:

- http://zodb.readthedocs.org/en/latest/transactions.html

`ZopeTransactionExtension` это расширение для SQLAlchemy, которое привязывает сессии к универсальному менеджеру транзакций transaction.

Добавим его в наш пример:

```
# models.py
from sqlalchemy import Column, Integer, Text, create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
from zope.sqlalchemy import ZopeTransactionExtension
```

(continues on next page)

(продолжение с предыдущей страницы)

```
engine = create_engine('sqlite:///foo.db')
Session = sessionmaker(bind=engine,
                       extension=ZopeTransactionExtension())
Base = declarative_base(bind=engine)


class User(Base):
    __tablename__ = 'users'
    id = Column(Integer, primary_key=True)
    name = Column(Text)

    def __repr__(self):
        return self.name
```

Теперь вместо `DBSession.commit`, нужно использовать `transaction.commit()`.

```
# __init__.py
from wsgiref.simple_server import make_server
from pyramid.config import Configurator
from pyramid.response import Response
from models import User, Session, Base, engine

import transaction


def hello(request):
    DBSession = Session(bind=engine)
    result = str(DBSession.query(User).all())
    import time
    timestamp = int(time.time())
    new_user = User(name=str(timestamp))
    DBSession.add(new_user)
    transaction.commit()
    return Response(result)

if __name__ == '__main__':
    Base.metadata.create_all()
    DBSession = Session(bind=engine)
    DBSession.add(User(name='Vasya'))
    DBSession.add(User(name='Petya'))
    transaction.commit()

    config = Configurator()
    config.add_route('hello_world', '/')
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    config.add_view(hello, route_name='hello_world')
    app = config.make_wsgi_app()
    server = make_server('0.0.0.0', 8000, app)
    server.serve_forever()
```

transaction.abort

Теперь мы используем общий, глобальный менеджер транзакций, который работает не только с SQLAlchemy но и со всеми модулями которые его поддерживают. Ниже пример сессии в которой одновременно участвуют SQLAlchemy и pyramid_mailer.

Внимание: Пример ниже работает с версией `repoze.sendmail==4.1`. Установить ее можно припомощи команды:

```
pip install repoze.sendmail==4.1
```

Если возникает ошибка `raise ValueError("TPC in progress")`, смотри https://github.com/repoze/repoze.sendmail/issues/31

```
# __init__.py
from wsgiref.simple_server import make_server
from pyramid.config import Configurator
from pyramid.response import Response
from models import User, Session, Base, engine

import transaction

from pyramid_mailer.message import Message

message = Message(subject="hello world",
                  sender="example@yandex.ru",
                  recipients=["me@uralbash.ru"],
                  body="hello, uralbash")


def hello(request):
    DBSession = Session(bind=engine)
    result = str(DBSession.query(User).all())
    import time
    timestamp = int(time.time())
    new_user = User(name=str(timestamp),
                    id=100500)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    DBSession.add(new_user)

    from pyramid_mailer import get_mailer
    mailer = get_mailer(request)
    mailer.send(message)
    try:
        transaction.commit()
    except Exception as e:
        transaction.abort()
        return Response(str(e))

    return Response(result)

if __name__ == '__main__':
    Base.metadata.create_all()
    DBSession = Session(bind=engine)
    DBSession.add(User(name='Vasya'))
    DBSession.add(User(name='Petya'))
    transaction.commit()

    settings = {'mail.host': 'smtp.yandex.ru',
                'mail.port': '465',
                'mail.ssl': True,
                'pyramid_mailer.prefix': 'mail.',
                'mail.username': 'example@yandex.ru',
                'mail.password': 'example password'}
    config = Configurator(settings=settings)
    config.include('pyramid_mailer')
    config.add_route('hello_world', '/')
    config.add_view(hello, route_name='hello_world')
    app = config.make_wsgi_app()
    server = make_server('0.0.0.0', 8000, app)
    server.serve_forever()
```

В этом примере вьюха `hello` записывает нового пользователя с `id=100500` в БД и отправляет письмо на адрес `me@uralbash.ru`. При первом обновлении страницы пользователь добавится в БД и отправится письмо. При последующих обновлениях произойдет ошибка т.к. пользователь с таким `id` уже существует, при этом `transaction.abort()` откатит изменения как в сессии SQLAlchemy, так и в сессии pyramid_mailer, поэтому письмо не отправится.

pyramid_tm

pyramid_tm автоматически подтверждает транзакцию в каждом запросе. Т.е. если мы

забыли написать `transaction.commit()`, то он все равно вызовется, при этом мы также можем вызывать его явно.

```
# __init__.py
from wsgiref.simple_server import make_server
from pyramid.config import Configurator
from pyramid.response import Response
from models import User, Session, Base, engine


def hello(request):
    DBSession = Session(bind=engine)
    result = str(DBSession.query(User).all())
    import time
    timestamp = int(time.time())
    new_user = User(name=str(timestamp))
    DBSession.add(new_user)
    return Response(result)

if __name__ == '__main__':
    Base.metadata.create_all()
    DBSession = Session(bind=engine)
    DBSession.add(User(name='Vasya'))
    DBSession.add(User(name='Petya'))

    config = Configurator()
    config.include('pyramid_tm')
    config.add_route('hello_world', '/')
    config.add_view(hello, route_name='hello_world')
    app = config.make_wsgi_app()
    server = make_server('0.0.0.0', 8000, app)
    server.serve_forever()
```

pyramid_sqlalchemy

pyramid_sqlalchemy создает объект базового класса `Base` и сессии `Session` автоматически. Мы просто указываем строку подключения к БД в настройках и включаем модуль `pyramid_sqlalchemy` в проект.

```
# __init__.py
from wsgiref.simple_server import make_server
from pyramid.config import Configurator
from pyramid.response import Response
from models import User
from pyramid_sqlalchemy import BaseObject, Session
```

(continues on next page)

(продолжение с предыдущей страницы)

```
import transaction


def hello(request):
    result = str(Session.query(User).all())
    import time
    timestamp = int(time.time())
    new_user = User(name=str(timestamp))
    Session.add(new_user)
    return Response(result)

if __name__ == '__main__':
    settings = {'sqlalchemy.url': 'sqlite:///:memory:'}
    config = Configurator(settings=settings)
    config.include('pyramid_tm')
    config.include('pyramid_sqlalchemy')

    BaseObject.metadata.create_all()
    Session.add(User(name='Vasya'))
    Session.add(User(name='Petya'))
    transaction.commit()

    config.add_route('hello_world', '/')
    config.add_view(hello, route_name='hello_world')
    app = config.make_wsgi_app()
    server = make_server('0.0.0.0', 8000, app)
    server.serve_forever()
```

Файл с моделями теперь выглядит значительно проще.

```
# models.py
from sqlalchemy import Column, Integer, Text
from pyramid_sqlalchemy import BaseObject


class User(BaseObject):
    __tablename__ = 'users'
    id = Column(Integer, primary_key=True)
    name = Column(Text)

    def __repr__(self):
        return self.name
```

Резюме

Несмотря на то, что фреймворк Pyramid не предоставляет инструментов для работы с базами дынных, есть большое количество сторонних модулей и расширений (написанных специально для Pyramid) которые реализуют этот функционал.

Для быстрого старта существует шаблон проекта `alchemy`, по которому можно быстро начать использовать пирамиду вместе с SQLAlchemy.

```
$ pcreate --scaffold alchemy sqla_demo
$ cd sqla_demo
$ python setup.py develop
```

Дополнительную информацию можно найти в Pyramid Cookbook.

1.5.6 Диспетчеризация URL

См.также:

- http://docs.pylonsproject.org/projects/pyramid/en/1.6-branch/narr/urldispatch.html
- http://pyramid-cookbook.readthedocs.org/en/latest/routing/index.html
- Для тех кто в Djang'е (http://blog.delaguardia.com.mx/pyramid-view-configuration-let-me-count-the-ways.html)

Каждый поступающий на сервер приложений Pyramid запрос (**request**) должен найти вид (**view**), который и будет его обрабатывать.

В Pyramid имеется два базовых подхода к поиску нужного вида для обрабатываемого запроса: на основе сопоставления (**matching**), как в большинстве подобных фреймворков, и обхода (**traversal**), как в Zope. Кроме того, в одном приложении можно с успехом сочетать оба подхода.

Pattern Matching

Простейший пример с заданием маршрута (заимствован из документации):

```
# Здесь config - экземпляр pyramid.config.Configurator
config.add_route('idea', 'site/{id}')
config.add_view('mypackage.views.site_view', route_name='idea')
```

Traversal

См.также:

- http://docs.pylonsproject.org/projects/pyramid/en/latest/narr/traversal.html

Использование обхода лучше проиллюстрировать на небольшом примере:

```
from wsgiref.simple_server import make_server
from pyramid.config import Configurator
from pyramid.response import Response

# Класс некоторого ресурса
class Resource(dict):
    pass

# Дерево ресурсов (жёстко закодированное) в фабрике корня
def get_root(request):
    return Resource({'a': Resource({'b': Resource({'c': Resource()})})})

# Вид-для-вызова, который умеет показывать ресурс Resource (в context)
def hello_world_of_resources(context, request):
    output = "Ресурс и его дети: %s" % context
    return Response(output)

if __name__ == '__main__':
    config = Configurator(root_factory=get_root)
    config.add_view(hello_world_of_resources, context=Resource)
    app = config.make_wsgi_app()
    server = make_server('0.0.0.0', 8080, app)
    server.serve_forever()
```

В этом примере иерархия для обхода жестко задана в методе `get_root` с помощью вложенных словарей, тогда как реальные приложения должны сами определять необходимый доступ по ключам (метод `__getitem__` помогает организовать такой доступ). В коде также присутствует корневая фабрика, с которой собственно и начинается обход узлов (node) дерева ресурсов. Вид-для-вызова (*view callable*) представлен функцией `hello_world_of_resources`. Говоря несколько упрощённо, на основе URL запроса в результате обхода иерархии Pyramid находит ресурс и применяет к нему «наилучший» вид-для-вызова (в нашем примере — он единственный).

Обход словаря

Предупреждение: В примерах используется Python3

Метод обхода дерева (traversal), позволяет находить ресурсы во вложенных структурах. Такой механизм хорошо применим для некоторых практических задач, например список всех URL маршрутов сайта является деревом, в котором каждый конечный URL

это отдельная ветка дерева. Поэтому всю структуру сайта можно поместить в словарь.

К примеру, известно, что смерь сказочного персонажа «кащея» находится в яйце, которое в свою очередь в утке, которая в зайце и т.д. В сумме получается вложенная структура, которую можно описать так:

`остров -> дуб -> сундук -> заяц -> утка -> яйцо -> игла -> СмертьКощея`

Мы можем такую, плоскую, вложенную структуру легко представить в виде URL:

`http://localhost:8080/mytraversal/остров/дуб/сундук/заяц/утка/яйцо/игла`

А так-как любой URL является веткой дерева, то несложно описать это в Python:

```
{
     'остров': {
         'дуб': {
             'сундук': {
                 'заяц': {
                     'утка': {
                         'яйцо': {
                             'игла': СмертьКощея()
                         }
                     }
                 }
             }
         }
     }
 }
```

`СмертьКощея()` - это объект класса `СмертьКощея`, который может выглядеть к примеру так:

```
class СмертьКощея(object):

    def __json__(self, request):

        return {
            'имя': 'кощей',
            'статус': request.context == self and 'мертв' or 'жив ещё',
        }
```

В принципе, этого достаточно чтобы наш сайт-убийца «кощея» заработал. Осталось лишь прописать пути и добавить представление (view).

View будет просто возвращать объект, например, если мы ввели:

```
URL: 'остров'
объект: {'дуб': {
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            'сундук': {
                'заяц': {
                    'утка': {
                        'яйцо': {
                            'игла': СмертьКощея()
                        }
                    }
                }
            }
        }}
```

```
URL: 'остров/дуб'
объект: {'сундук': {
            'заяц': {
                'утка': {
                    'яйцо': {
                        'игла': СмертьКощея()
                    }
                }
            }
        }}
```

```
URL: 'остров/дуб/сундук'
объект: {'заяц': {
            'утка': {
                'яйцо': {
                    'игла': СмертьКощея()
                }
            }
        }}
```

```
URL: 'остров/дуб/сундук/заяц'
объект: {'утка': {
            'яйцо': {
                'игла': СмертьКощея()
            }
        }}
```

```
URL: 'остров/дуб/сундук/заяц/утка'
объект: {'яйцо': {
            'игла': СмертьКощея()
        }}
```

```
URL: 'остров/дуб/сундук/заяц/утка/яйцо'
объект: {'игла': СмертьКощея()}
```

```
URL: 'остров/дуб/сундук/заяц/утка/яйцо/игла'
объект: СмертьКощея()
```

Такие функции-представления (View) должны принимать 2 параметра, где первый параметр будет являться объектом, обычно именуемым `context`, а второй параметр `request`:

```
def traverse_koshey(context, request):
    return context  # Наш объект
```

Роуты создаются почти так же как в `pattern matching`, за исключением того, что структура путей передается в виде «фабрики», которая возвращает словарь или ему подобный (dict-like) объект. Путь указывается в виде статической и динамической части, например `/fixedpath/*traverse`:

```
config.add_route('koshey', '/mytraversal/*traverse', factory=my_factory)
```

Фабрика, которая возвращает структуру сайта:

```
def my_factory(request):
    return {
        'остров': {
            'дуб': {
                'сундук': {
                    'заяц': {
                        'утка': {
                            'яйцо': {
                                'игла': СмертьКощея()
                            }
                        }
                    }
                }
            }
        }
    }
```

View добавляется стандартно:

```
config.add_view(traverse_koshey, route_name='koshey', renderer='json')
```

Все готово, можно перемещаться по объектам:

- http://localhost:8080/mytraversal/

- http://localhost:8080/mytraversal/\T2A\cyro\T2A\cyrs\T2A\cyrt\T2A\cyrr\T2A\cyro\T2A\cyrv
- http://localhost:8080/mytraversal/\T2A\cyro\T2A\cyrs\T2A\cyrt\T2A\cyrr\T2A\cyro\T2A\cyrv/\T2A\cyrd\T2A\cyru\T2A\cyrb
- http://localhost:8080/mytraversal/\T2A\cyro\T2A\cyrs\T2A\cyrt\T2A\cyrr\T2A\cyro\T2A\cyrv/\T2A\cyrd\T2A\cyru\T2A\cyrb/\T2A\cyrs\T2A\cyru\T2A\cyrn\T2A\cyrd\T2A\cyru\T2A\cyrk
- http://localhost:8080/mytraversal/\T2A\cyro\T2A\cyrs\T2A\cyrt\T2A\cyrr\T2A\cyro\T2A\cyrv/\T2A\cyrd\T2A\cyru\T2A\cyrb/\T2A\cyrs\T2A\cyru\T2A\cyrn\T2A\cyrd\T2A\cyru\T2A\cyrk/\T2A\cyrz\T2A\cyra\T2A\cyrya\T2A\cyrc
- http://localhost:8080/mytraversal/\T2A\cyro\T2A\cyrs\T2A\cyrt\T2A\cyrr\T2A\cyro\T2A\cyrv/\T2A\cyrd\T2A\cyru\T2A\cyrb/\T2A\cyrs\T2A\cyru\T2A\cyrn\T2A\cyrd\T2A\cyru\T2A\cyrk/\T2A\cyrz\T2A\cyra\T2A\cyrya\T2A\cyrc/\T2A\cyru\T2A\cyrt\T2A\cyrk\T2A\cyra
- http://localhost:8080/mytraversal/\T2A\cyro\T2A\cyrs\T2A\cyrt\T2A\cyrr\T2A\cyro\T2A\cyrv/\T2A\cyrd\T2A\cyru\T2A\cyrb/\T2A\cyrs\T2A\cyru\T2A\cyrn\T2A\cyrd\T2A\cyru\T2A\cyrk/\T2A\cyrz\T2A\cyra\T2A\cyrya\T2A\cyrc/\T2A\cyru\T2A\cyrt\T2A\cyrk\T2A\cyra/\T2A\cyrya\T2A\cyrishrt\T2A\cyrc\T2A\cyro
- http://localhost:8080/mytraversal/\T2A\cyro\T2A\cyrs\T2A\cyrt\T2A\cyrr\T2A\cyro\T2A\cyrv/\T2A\cyrd\T2A\cyru\T2A\cyrb/\T2A\cyrs\T2A\cyru\T2A\cyrn\T2A\cyrd\T2A\cyru\T2A\cyrk/\T2A\cyrz\T2A\cyra\T2A\cyrya\T2A\cyrc/\T2A\cyru\T2A\cyrt\T2A\cyrk\T2A\cyra/\T2A\cyrya\T2A\cyrishrt\T2A\cyrc\T2A\cyro/\T2A\cyri\T2A\cyrg\T2A\cyrl\T2A\cyra

Полный пример:

```
from wsgiref.simple_server import make_server

from pyramid.config import Configurator


def traverse_koshey(context, request):
    return context


class СмертьКощея(object):

    def __json__(self, request):

        return {
            'имя': 'кощей',
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            'статус': request.context == self and 'мертв' or 'жив ещё',
        }


def my_factory(request):
    return {
        'остров': {
            'дуб': {
                'сундук': {
                    'заяц': {
                        'утка': {
                            'яйцо': {
                                'игла': СмертьКощея()
                            }
                        }
                    }
                }
            }
        }
    }


if __name__ == '__main__':
    config = Configurator()

    # Traversal routing
    config.add_route('koshey', '/mytraversal/*traverse', factory=my_factory)
    config.add_view(traverse_koshey, route_name='koshey', renderer='json')

    # Make app and serve
    app = config.make_wsgi_app()
    server = make_server('0.0.0.0', 8080, app)
    server.serve_forever()
```

Есть один нюанс, json renderer, по умолчанию, все не латинские символы отображает как UTF коды \uxxxx, поэтому мы увидим следующий вывод:

```
{"\u043e\u0441\u0442\u0440\u043e\u0432": {"\u0434\u0443\u0431": {
↪"\u0441\u0443\u043d\u0434\u0443\u043a": {"\u0437\u0430\u044f\u0446": {
↪"\u0443\u0442\u043a\u0430": {"\u044f\u0439\u0446\u043e": {
↪"\u0438\u0433\u043b\u0430": {"\u0441\u0442\u0430\u0442\u0443\u0441":
↪"\u0436\u0438\u0432 \u0435\u0449\u0451", "\u0438\u043c\u044f":
↪"\u043a\u043e\u0449\u0435\u0439"}}}}}}}}
```

Но можно изменить его поведение следующим образом:

```
from pyramid.renderers import JSON
...
config.add_renderer('myjson', JSON(indent=4, ensure_ascii=False))
config.add_view(traverse_koshey, route_name='koshey', renderer='myjson')
```

Результат:

http://localhost:8080/mytraversal/

```
{
    "остров": {
        "дуб": {
            "сундук": {
                "заяц": {
                    "утка": {
                        "яйцо": {
                            "игла": {
                                "имя": "кощей",
                                "статус": "жив ещё"
                            }
                        }
                    }
                }
            }
        }
    }
}
```

http://localhost:8080/mytraversal/\T2A\cyro\T2A\cyrs\T2A\cyrt\T2A\cyrr\T2A\cyro\T2A\cyrv/\T2A\cyrd\T2A\cyru\T2A\cyrb/\T2A\cyrs\T2A\cyru\T2A\cyrn\T2A\cyrd\T2A\cyru\T2A\cyrk/\T2A\cyrz\T2A\cyra\T2A\cyrya\T2A\cyrc/\T2A\cyru\T2A\cyrt\T2A\cyrk\T2A\cyra/\T2A\cyrya\T2A\cyrishrt\T2A\cyrc\T2A\cyro/\T2A\cyri\T2A\cyrg\T2A\cyrl\T2A\cyra

```
{
    "имя": "кощей",
    "статус": "мертв"
}
```

Полный код:

```
1  from wsgiref.simple_server import make_server
2
3  from pyramid.config import Configurator
4  from pyramid.renderers import JSON
5
```

(continues on next page)

(продолжение с предыдущей страницы)

```
6
7  def traverse_koshey(context, request):
8      return context
9
10
11 class СмертьКощея(object):
12
13     def __json__(self, request):
14
15         return {
16             'имя': 'кощей',
17             'статус': request.context == self and 'мертв' or 'жив ещё',
18         }
19
20
21 def my_factory(request):
22     return {
23         'остров': {
24             'дуб': {
25                 'сундук': {
26                     'заяц': {
27                         'утка': {
28                             'яйцо': {
29                                 'игла': СмертьКощея()
30                             }
31                         }
32                     }
33                 }
34             }
35         }
36     }
37
38
39 if __name__ == '__main__':
40     config = Configurator()
41
42     # ensure_ascii JSON renderer
43     config.add_renderer('myjson', JSON(indent=4, ensure_ascii=False))
44
45     # Traversal routing
46     config.add_route('koshey', '/mytraversal/*traverse', factory=my_factory)
47     config.add_view(traverse_koshey, route_name='koshey', renderer='myjson')
48
49     # Make app and serve
50     app = config.make_wsgi_app()
```

(continues on next page)

(продолжение с предыдущей страницы)

```
51      server = make_server('0.0.0.0', 8080, app)
52      server.serve_forever()
```

Привязка View к ресурсам

В Пирамиде объект (context) который передается во вью, именуют еще как «ресурс». Есть возможность жестко привязать View к типу ресурса. Например, наше представление `traverse_koshey` должно вызываться, только когда пришел объект класса `СмертьКощея`:

```
config.add_view(traverse_koshey, route_name='koshey_context',
                renderer='myjson',
                context=СмертьКощея)
```

Параметр `context` указывает на то, что это View принадлежит ТОЛЬКО объектам класса `СмертьКащея`.

Все пути, кроме полного (который возвращает нужный объект), вернут 404 код ответа. Полный путь http://localhost:8080/mytraversal/\T2A\cyro\T2A\cyrs\T2A\cyrt\T2A\cyrr\T2A\cyro\T2A\cyrv/\T2A\cyrd\T2A\cyru\T2A\cyrb/\T2A\cyrs\T2A\cyru\T2A\cyrn\T2A\cyrd\T2A\cyru\T2A\cyrk/\T2A\cyrz\T2A\cyra\T2A\cyrya\T2A\cyrc/\T2A\cyru\T2A\cyrt\T2A\cyrk\T2A\cyra/\T2A\cyrya\T2A\cyrishrt\T2A\cyrc\T2A\cyro/\T2A\cyri\T2A\cyrg\T2A\cyrl\T2A\cyra.

Добавим в нашу структуру еще ресурсов:

```
def my_factory(request):
    return {
        'превед': Человек(),
        'остров': {
            'ясень': {
                'что то здесь': 'не так!'
            },
            'дуб': {
                'сундук': {
                    'заяц': {
                        'утка': {
                            'яйцо': {
                                'игла': СмертьКощея()
                            }
                        }
                    }
                }
            }
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        }
    }
```

Здесь `Человек()` это новый тип ресурса, который имеет метод `__getitem__` как у словаря и при обращении по ключу возвращает другой ресурс:

```
class Человек(object):

    name = 'Человек'

    def __getitem__(self, name):
        return Имя(name)


class Имя(object):

    __parent__ = Человек()

    def __init__(self, name):
        self.name = name
```

Например мы обращаемся по URL http://localhost:8080/mytraversal/\T2A\cyrp\T2A\cyrr\T2A\cyre\T2A\cyrv\T2A\cyre\T2A\cyrd/\T2A\CYRP\T2A\cyri\T2A\cyrr\T2A\cyra\T2A\cyrm\T2A\cyri\T2A\cyrd. `превед` вернет ресурс `Человек`, а `Пирамид` вызовет метод `__getitem__`, который вернет ресурс `Имя('Пирамид')`. Таким образом мы можем строить дерево динамически при помощи dict-like объектов.

Для ресурса `Имя` мы можем создать отдельное представление и жестко привязать его к этому типу.

```
def traverse_hello(context, request):
    """
    http://localhost:8080/mytraversal/превед/Пирамид
    """
    return Response('Превед ' + context.__parent__.name + ' ' + context.name)

...

config.add_view(traverse_hello, route_name='koshey_context',
                renderer='text',
                context=Имя)
```

Результат вывода по адресу http://localhost:8080/mytraversal/\T2A\cyrp\T2A\cyrr\T2A\cyre\T2A\cyrv\T2A\cyre\T2A\cyrd/\T2A\CYRP\T2A\cyri\T2A\cyrr\T2A\cyra\T2A\cyrm\T2A\cyri\T2A\cyrd, будет обычный текст (`Content-Type: plain/text`):

```
Превед Человек Пирамид
```

Полный пример:

```
from wsgiref.simple_server import make_server

from pyramid.config import Configurator
from pyramid.response import Response
from pyramid.renderers import JSON


def traverse_koshey(context, request):
    """
    http://localhost:8080/mytraversal/остров/дуб/сундук/заяц/утка/яйцо/игла
    """
    return context


def traverse_hello(context, request):
    """
    http://localhost:8080/mytraversal/превед/Пирамид
    """
    return Response('Превед ' + context.__parent__.name + ' ' + context.name)


class Человек(object):

    name = 'Человек'

    def __getitem__(self, name):
        return Имя(name)

    def __json__(self, request):
        return {'name': self.name}


class Имя(object):

    __parent__ = Человек()

    def __init__(self, name):
        self.name = name

    def __json__(self, request):
        return {'name': self.name}

```

(continues on next page)

(продолжение с предыдущей страницы)

```
43
44  class СмертьКощея(object):
45
46      def __json__(self, request):
47
48          return {
49              'имя': 'кощей',
50              'статус': request.context == self and 'мертв' or 'жив ещё',
51          }
52
53
54  def my_factory(request):
55      return {
56          'превед': Человек(),
57          'остров': {
58              'ясень': {
59                  'что то здесь': 'не так!'
60              },
61              'дуб': {
62                  'сундук': {
63                      'заяц': {
64                          'утка': {
65                              'яйцо': {
66                                  'игла': СмертьКощея()
67                              }
68                          }
69                      }
70                  }
71              }
72          }
73      }
74
75
76  if __name__ == '__main__':
77      config = Configurator()
78
79      # ensure_ascii JSON renderer
80      config.add_renderer('myjson', JSON(indent=4, ensure_ascii=False))
81
82      # Traversal routing
83      config.add_route('koshey', '/mytraversal/*traverse', factory=my_factory)
84      config.add_view(traverse_koshey, route_name='koshey', renderer='myjson')
85
86      # Traversal routing with context constraint
87      config.add_route('koshey_context', '/mytraversal_context/*traverse',
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                      factory=my_factory)
    config.add_view(traverse_koshey, route_name='koshey_context',
                    renderer='myjson',
                    context=СмертьКощея)
    config.add_view(traverse_hello, route_name='koshey_context',
                    renderer='text',
                    context=Имя)

    # Make app and serve
    app = config.make_wsgi_app()
    server = make_server('0.0.0.0', 8080, app)
    server.serve_forever()
```

Комбинация обоих методов

Фреймворк Pyramid позволяет использовать оба способа URL маршрутизации одновременно.

Добавим к примеру с «кащеем» hello world с использованием pattern matching:

```
def hello_world(request):
    return Response('Hello %(name)s!' % request.matchdict)

...

# Pattern matching routes
config.add_route('hello', '/hello/{name}')
config.add_view(hello_world, route_name='hello')
```

Полный пример:

Код 100: Комбинированный способ маршрутизации traversal и pattern matching

```
from wsgiref.simple_server import make_server

from pyramid.config import Configurator
from pyramid.response import Response
from pyramid.renderers import JSON


def hello_world(request):
    return Response('Hello %(name)s!' % request.matchdict)

```

(continues on next page)

(продолжение с предыдущей страницы)

```
11
12  def traverse_koshey(context, request):
13      """
14      http://localhost:8080/mytraversal/остров/дуб/сундук/заяц/утка/яйцо/игла
15      """
16      return context
17
18
19  def traverse_hello(context, request):
20      """
21      http://localhost:8080/mytraversal/первед/Пирамид
22      """
23      return Response('Превед ' + context.__parent__.name + ' ' + context.name)
24
25
26  class Человек(object):
27
28      name = 'Человек'
29
30      def __getitem__(self, name):
31          return Имя(name)
32
33      def __json__(self, request):
34          return {'name': self.name}
35
36
37  class Имя(object):
38
39      __parent__ = Человек()
40
41      def __init__(self, name):
42          self.name = name
43
44      def __json__(self, request):
45          return {'name': self.name}
46
47
48  class СмертьКощея(object):
49
50      def __json__(self, request):
51
52          return {
53              'имя': 'кощей',
54              'статус': request.context == self and 'мертв' or 'жив ещё',
55          }
```

(continues on next page)

(продолжение с предыдущей страницы)

```
56
57
58  def my_factory(request):
59      return {
60          'превед': Человек(),
61          'остров': {
62              'ясень': {
63                  'что то здесь': 'не так!'
64              },
65              'дуб': {
66                  'сундук': {
67                      'заяц': {
68                          'утка': {
69                              'яйцо': {
70                                  'игла': СмертьКощея()
71                              }
72                          }
73                      }
74                  }
75              }
76          }
77      }
78
79
80  if __name__ == '__main__':
81      config = Configurator()
82
83      # Pattern matching routes
84      config.add_route('hello', '/hello/{name}')
85      config.add_view(hello_world, route_name='hello')
86
87      # ensure_ascii JSON renderer
88      config.add_renderer('myjson', JSON(indent=4, ensure_ascii=False))
89
90      # Traversal routing
91      config.add_route('koshey', '/mytraversal/*traverse', factory=my_factory)
92      config.add_view(traverse_koshey, route_name='koshey', renderer='myjson')
93
94      # Traversal routing with context constraint
95      config.add_route('koshey_context', '/mytraversal_context/*traverse',
96                       factory=my_factory)
97      config.add_view(traverse_koshey, route_name='koshey_context',
98                      renderer='myjson',
99                      context=СмертьКощея)
100     config.add_view(traverse_hello, route_name='koshey_context',
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                    renderer='text',
                    context=Имя)

    # Make app and serve
    app = config.make_wsgi_app()
    server = make_server('0.0.0.0', 8080, app)
    server.serve_forever()
```

1.5.7 REST API

См.также:

- https://ru.wikipedia.org/wiki/REST
- Для тех кто в Djang'е (http://blog.delaguardia.com.mx/pyramid-view-configuration-let-me-count-the-ways.html)
- http://docs.pylonsproject.org/projects/pyramid_cookbook/en/latest/testing/testing_post_curl.html

`REST API` подразумевает под собой простые правила:

- Каждый URL является ресурсом
- При обращении к ресурсу методом `GET` возвращается описание этого ресурса
- Метод `POST` добавляет новый ресурс
- Метод `PUT` изменяет ресурс
- Метод `DELETE` удаляет ресурс

Эти правила предоставляют простой `CRUD` интерфейс для других приложений, взаимодействие с которым происходит через протокол `HTTP`.

Соответствие `CRUD` операций и `HTTP` методов:

- **CREATE** - `POST`
- **READ** - `GET`
- **UPDATE** - `PUT`
- **DELETE** - `DELETE`

`REST API` интерфейс очень удобен для межпрограммного взаимодействия, например мобильное приложение может выступать в роли клиента, который манипулирует данными посредством `REST`.

Pattern matching

```
from wsgiref.simple_server import make_server

from pyramid.view import view_config, view_defaults
from pyramid.config import Configurator


@view_defaults(
    route_name='rest_people',
    renderer='json'
)
class RESTViewPeople(object):
    def __init__(self, request):
        self.request = request

    @view_config(request_method='GET')
    def get(self):
        return {
            'id': self.request.matchdict['id'],
            'method': self.request.method,
            'get': dict(self.request.GET)
        }

    @view_config(request_method='POST')
    def post(self):
        return {
            'id': self.request.matchdict['id'],
            'method': self.request.method,
            'post': dict(self.request.POST)
        }

    @view_config(request_method='DELETE')
    def delete(self):
        return {'status': 'success'}


if __name__ == '__main__':
    config = Configurator()
    config.add_route('rest_people', '/api/v1/people/{id:\d+}')
    config.add_view(RESTViewPeople, route_name='rest_people')
    config.scan('.')

    # make wsgi app
    app = config.make_wsgi_app()
    server = make_server('0.0.0.0', 8080, app)
    server.serve_forever()
```

Пример выше добавляет View с тремя методами, каждый из которых вызывается при соответствующем `GET`, `POST`, `DELETE` запросе. Ресурсом здесь является конкретный человек, получить которого можно по URL http://localhost:8080/api/v1/people/123

Результатом запроса будет:

```
{"get": {}, "id": "123", "method": "GET"}
```

Для отправки `POST` запроса воспользуемся консольной утилитой curl:

```
$ curl -X POST -d 'param1=value1&param2=value2' http://localhost:8080/api/v1/
↪people/1
```

Результат запроса:

```
{"id": "1", "post": {"param1": "value1", "param2": "value2"}, "method": "POST"}
```

`DELETE` запрос выполняется по аналогии:

```
$ curl -X DELETE http://localhost:8080/api/v1/people/1
```

Результат запроса:

```
{"status": "success"}
```

Traversal

См.также:

Метод URL диспетчеризации *Traversal*

В предыдущем примере показан только один ресурс - конкретный человек и в принципе все выглядит неплохо, пока не появится другой смежный ресурс, например список всех людей по адресу http://localhost:8080/api/v1/people

В этом случае, придется добавлять новый путь (rout), привязывать его к представлению (View) и самое неприятное менять само представление, или еще хуже писать новое. Таким образом с увеличением ресурсов, сложность REST API растет не пропорционально и в какой то момент код становится не читаемым из-за больших размеров и постоянно меняющейся логики во View.

Выход из данной ситуации - отделить ресурсы от представлений, тем самым вынести часть логики и сделать представления более универсальными.

Ресурсы

Ресурсы могут выглядеть так:

Код 101: Список всех людей

```
class PeopleResource(object):

    def __getitem__(self, people_id):
        if str(people_id).isdigit():
            return PersonResource(people_id)

    def __json__(self, request):
        return {
            'params': request.matchdict,
            'method': request.method,
        }
```

`PeopleResource` представляет список всех людей и будет доступен по адресу http://localhost:8080/api/v1/people. `PeopleResource` имеет метод `__getitem__`, что делает его похожим на словарь. При обращении к объекту ресурса как к словарю, он вызовет эту функцию и передаст ключ в параметр `people_id`, например:

```
foo = PeopleResource()
bar = foo[123]  # Вернет объект PersonResource(123)
```

Метод `__json__` определяет каким образом преобразовывать ресурс в json.

`PersonResource` представляет конкретного человека и будет доступен по адресу http://localhost:8080/api/v1/people/\protect\T2A\textbraceleftid\protect\T2A\textbraceright. Здесь отличительной особенностью является то, что метод `__json__` наследует часть словаря из класса `PeopleResource`, при помощи конструкции `super`:

Код 102: Конкретный человек

```
class PersonResource(PeopleResource):

    def __init__(self, people_id):
        self.id = people_id

    def __json__(self, request):
        return {
            'id': self.id,
            **super().__json__(request)
        }
```

View

Перепишем View таким образом, чтобы она возвращала только ресурс, а так-как ресурс уже содержит в себе информацию как отдавать json, то это представление будет универсальным как для `PeopleResource`, так и для `PersonResource` и возможно подойдет другим ресурсам которые мы будем писать в будущем.

Код 103: Представление (View) для traversal ресурсов

```
@view_defaults(
    route_name='rest_api',
    renderer='json',
    context=PeopleResource
)
class RESTViewPeople(object):
    def __init__(self, context, request):
        self.context = context
        self.request = request

    @view_config(request_method='GET')
    def get(self):
        return self.context

    @view_config(request_method='POST')
    def post(self):
        return self.context

    @view_config(request_method='DELETE')
    def delete(self):
        return {'status': 'success'}
```

Рендерер `json` по умолчанию ищет метод `__json__` и если он есть то возвращает его

результат вызова.

Route

Путь, в нашем случае, будет один, так-как вся структура вынесена в ресурсы (метод `__getitem__`).

```
config.add_route('rest_api', '/api/v1/*traverse', factory=rest_factory)
```

Полный пример

```
from wsgiref.simple_server import make_server

from pyramid.view import view_config, view_defaults
from pyramid.config import Configurator


class PeopleResource(object):

    def __getitem__(self, people_id):
        if str(people_id).isdigit():
            return PersonResource(people_id)

    def __json__(self, request):
        return {
            'params': request.matchdict,
            'method': request.method,
        }


class PersonResource(PeopleResource):

    def __init__(self, people_id):
        self.id = people_id

    def __json__(self, request):
        return {
            'id': self.id,
            **super().__json__(request)
        }


class AnimalsResource(object):
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    pass


@view_defaults(
    route_name='rest_api',
    renderer='json',
    context=PeopleResource
)
class RESTViewPeople(object):
    def __init__(self, context, request):
        self.context = context
        self.request = request

    @view_config(request_method='GET')
    def get(self):
        return self.context

    @view_config(request_method='POST')
    def post(self):
        return self.context

    @view_config(request_method='DELETE')
    def delete(self):
        return {'status': 'success'}


def rest_factory(request):
    return {
        'people': PeopleResource(),
        'animals': AnimalsResource(),
    }


if __name__ == '__main__':
    config = Configurator()
    config.add_route('rest_api', '/api/v1/*traverse', factory=rest_factory)
    config.add_view(RESTViewPeople, route_name='rest_api')
    config.scan('.')

    # make wsgi app
    app = config.make_wsgi_app()
    server = make_server('0.0.0.0', 8080, app)
    server.serve_forever()
```

1.5.8 Представления (Views)

См.также:

- http://docs.pylonsproject.org/projects/pyramid/en/latest/narr/viewconfig.html
- http://pyramid-cookbook.readthedocs.org/en/latest/pylons/views.html
- http://blog.delaguardia.com.mx/pyramid-view-configuration-let-me-count-the-ways.html

Представления (views) создаются в виде функций или методов и могут находится в любом месте проекта. В качестве аргумента функция принимает объект `request`, а возвращает объект `response`:

```
from pyramid.response import Response

def my_view(request):
    return Response("Hello, world!")
```

В классе, который содержит представления-методы, объект `request` передается в конструктор:

```
class MyHandler(object):
    def __init__(self, request):
        self.request = request

    def my_view(self):
        return Response("Hello, classy world!")
```

Конфигурация

route_name

Имя для привязки к роуту. Нужно если используется URL диспетчеризация.

renderer

Имя обработчика (string, json) или шаблон (index.jinja2, index.pt, index.mako).

permission

Строка и именем права доступа, которое текущий пользователь должен иметь чтобы вызвать это представление.

request_method

“GET”, “POST”, “PUT”, “DELETE’, “HEAD”.

request_param

Проверяет наличие параметров в запросе, например «foo» означает что в запросе должен быть параметр с именем «foo». «foo=1» означает что этот параметр должен быть равен 1.

match_param

Тоже что и `request_param` но проверяет все параметры, в том числе которые пришли от URL диспетчеризации.

Декларативный способ

Декларативный способ задания представлений осуществляется при помощи декораторов pyramid.view.view_config и `pyramid.view.view_defaults`.

```
from pyramid.view import view_config

class Handler(object):
    def __init__(self, request):
        self.request = request

class Main(Handler):

    @view_config(route_name="home", renderer="index.mako")
    def index(self):
        return {"project": "Akhet Demo"}
```

Функция или метод может быть привязана к нескольким представлениям.

```
class Main(Handler):

    @view_config(route_name="home", renderer="index.mako")
    @view_config(route_name="home_json", renderer="json")
    def index(self):
        return {"project": "Akhet Demo"}
```

Пример REST

```
from pyramid.view import view_defaults
from pyramid.view import view_config
from pyramid.response import Response

@view_defaults(route_name='rest')
class RESTView(object):
    def __init__(self, request):
        self.request = request
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    @view_config(request_method='GET')
    def get(self):
        return Response('get')

    @view_config(request_method='POST')
    def post(self):
        return Response('post')

    @view_config(request_method='DELETE')
    def delete(self):
        return Response('delete')
```

Императивный способ

```
from pyramid.config import not_

...

   config.add_view(Main.index, route_name="home", request_method=not_('POST'))
```

Пример REST.

```
from pyramid.response import Response
from pyramid.config import Configurator

class RESTView(object):
    def __init__(self, request):
        self.request = request

    def get(self):
        return Response('get')

    def post(self):
        return Response('post')

    def delete(self):
        return Response('delete')

def main(global_config, **settings):
    config = Configurator()
    config.add_route('rest', '/rest')
    config.add_view(RESTView, route_name='rest', attr='get', request_method='GET')
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    config.add_view(RESTView, route_name='rest', attr='post', request_method='POST
↪')
    config.add_view(RESTView, route_name='rest', attr='delete', request_method=
↪'DELETE')
    return config.make_wsgi_app()
```

Совмещенный способ

```
from pyramid.view import view_defaults
from pyramid.response import Response
from pyramid.config import Configurator

@view_defaults(route_name='rest')
class RESTView(object):
    def __init__(self, request):
        self.request = request

    def get(self):
        return Response('get')

    def post(self):
        return Response('post')

    def delete(self):
        return Response('delete')

def main(global_config, **settings):
    config = Configurator()
    config.add_route('rest', '/rest')
    config.add_view(RESTView, attr='get', request_method='GET')
    config.add_view(RESTView, attr='post', request_method='POST')
    config.add_view(RESTView, attr='delete', request_method='DELETE')
    return config.make_wsgi_app()
```

1.5.9 Шаблоны (Templates)

См.также:

- http://docs.pylonsproject.org/projects/pyramid/en/latest/narr/templates.html
- http://docs.pylonsproject.org/projects/pyramid/en/latest/narr/renderers.html

В пирамиде нет встроенного шаблонизатора. Представления (*view callable*) всегда отдают объект `response`. Этот объект может формироваться напрямую, на-

пример `Response("Hello, world!")`. При помощи встроенных обработчиков (string, json, jsonp), самописных или сторонних. Или через специальные функции, например `pyramid.renderers.render_to_response()`.

Дополнительные обработчики могут поставляться сторонними модулями:

```
config.include('pyramid_chameleon') # Chameleon - template engine
config.include('pyramid_jinja2')    # Jinja2 -template engine
config.include('pyramid_mako')      # Mako -template engine
```

Использование напрямую

Обработка напрямую происходит при помощи функции `pyramid.renderers.render_to_response()`.

Примечание: Предварительно нужно добавить расширение, которое знает как обрабатывать шаблоны Chameleon.

```
config.include('pyramid_chameleon')
```

```
from pyramid.renderers import render_to_response

def sample_view(request):
    return render_to_response('mypackage:templates/foo.pt',
                              {'foo':1, 'bar':2},
                              request=request)
```

Функция `pyramid.renderers.render()` вернет только текст, а не объект `response`.

```
from pyramid.renderers import render
from pyramid.response import Response

def sample_view(request):
    result = render('mypackage:templates/foo.pt',
                    {'foo':1, 'bar':2},
                    request=request)
    response = Response(result)
    return response
```

Такой подход позволяет, например, использовать возможности самого шаблонизатора напрямую.

```
from mako.template import Template
from pyramid.response import Response
```

(continues on next page)

(продолжение с предыдущей страницы)

```
def make_view(request):
    template = Template(filename='/templates/template.mak')
    result = template.render(name=request.params['name'])
    response = Response(result)
    return response
```

Использование через обработчики (renderer)

См.также:

- http://docs.pylonsproject.org/projects/pyramid/en/latest/narr/renderers.html

Альтернативный способ функции `render_to_response()`, это привязывать к представлению свой обработчик. При этом представление возвращает только словарь, который в последующем будет обработан этим рендерером.

```
from pyramid.view import view_config

@view_config(renderer='mypackage:templates/foo.jinja2')
def my_view(request):
    return {'foo':1, 'bar':2}
```

Этот код идентичен:

```
from pyramid.renderers import render
from pyramid.response import Response

def sample_view(request):
    result = render('mypackage:templates/foo.jinja2',
                    {'foo':1, 'bar':2},
                    request=request)
    response = Response(result)
    return response
```

pyramid_jinja2

См.также:

- http://docs.pylonsproject.org/projects/pyramid/en/latest/quick_tutorial/jinja2.html
- http://docs.pylonsproject.org/projects/pyramid-jinja2/en/latest/

Установка

```
pip install pyramid_jinja2
```

Настройка

Добавляется стандартными средствами:

```
config.Configurator()
config.include('pyramid_jinja2')
```

или

```
pyramid.includes=
    pyramid_jinja2
```

Использование

```
@view_config(renderer='mypackage:templates/mytemplate.jinja2')
def my_view(request):
    return {'foo': 1, 'bar': 2}
```

По умолчанию `pyramid_jinja2` ищет директорию с шаблонами относительно вашего проекта, поэтому можно опустить название проекта.

```
@view_config(renderer='templates/mytemplate.jinja2')
def my_view(request):
    return {'foo': 1, 'bar': 2}
```

Код 104: templates/mytemplate.jinja2

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Hello World!</title>
</head>
<body>
   <h1>{{ foo }}</h1>
   <h1>{{ bar }}</h1>
</body>
</html>
```

Резюме

Фреймворк Pyramid не ограничивает вас в использовании какого-либо определенного шаблонизатора. Вы можете выбрать любой который вам понравится или пользоваться несколькими, при этом можно написать собственные обработчики запросов, даже не привязанные к движкам шаблонов и даже написать свой собственный шаблонизатор с расширением для пирамиды, как например Tonnikala.

1.5.10 Сессии

См.также:

- http://docs.pylonsproject.org/projects/pyramid/en/latest/narr/sessions.html
- http://docs.pylonsproject.org/projects/pyramid/en/latest/quick_tutorial/sessions.html
- http://docs.pylonsproject.org/projects/pyramid/en/latest/api/session.html

Фреймворк Pyramid имеет модуль `pyramid.session`, который содержит в себе несколько методов организации сессий.

Встроенный механизм сессий

Для того, чтобы использовать сессии, необходимо задать *session factory* во время конфигурации.

Очень простой, небезопасный способ создания сессии реализуется при помощи функции `pyramid.session.UnencryptedCookieSessionFactoryConfig()`. Он использует куки для хранения информации сеанса. Эта реализация имеет следующие ограничения:

- значения куков не шифруется, поэтому их может просмотреть любой, кто имеет доступ к трафику или к браузеру.
- Максимальное число байт для сессии 4000. Это подходит только для очень небольших наборов данных.

Функция `pyramid.session.SignedCookieSessionFactory()` шифрует данные, поэтому их тяжело подделать.

Добавление сессий в конфиг происходит следующим образом:

```
from pyramid.session import SignedCookieSessionFactory
my_session_factory = SignedCookieSessionFactory('itsaseekreet')

from pyramid.config import Configurator
config = Configurator()
config.set_session_factory(my_session_factory)
```

или через атрибут конструктора:

```
from pyramid.session import SignedCookieSessionFactory
my_session_factory = SignedCookieSessionFactory('itsaseekreet')

from pyramid.config import Configurator
config = Configurator(session_factory=my_session_factory)
```

Использование сессий

После добавления сессий в конфиг, вы можете получить доступ к объектам сессии из любого запроса. Например:

```
from pyramid.response import Response

def myview(request):
    session = request.session

    if 'counter' in session:
        session['counter'] += 1
    else:
        session['counter'] = 1

    return Response(session['counter'])
```

Альтернативные механизмы сессий

См.также:

- https://github.com/uralbash/awesome-pyramid#caching
- pyramid_redis_sessions - предоставляет механизм сессий который использует хранилище Redis.
- pyramid_beaker - использует в качестве бэкенда систему сессий Beaker.

В самом простом случае достаточно включить модуль в проект:

```
config = Configurator()
config.include('pyramid_beaker')
```

Теперь можно использовать сессии:

```
from pyramid.response import Response

def myview(request):
```

(continues on next page)

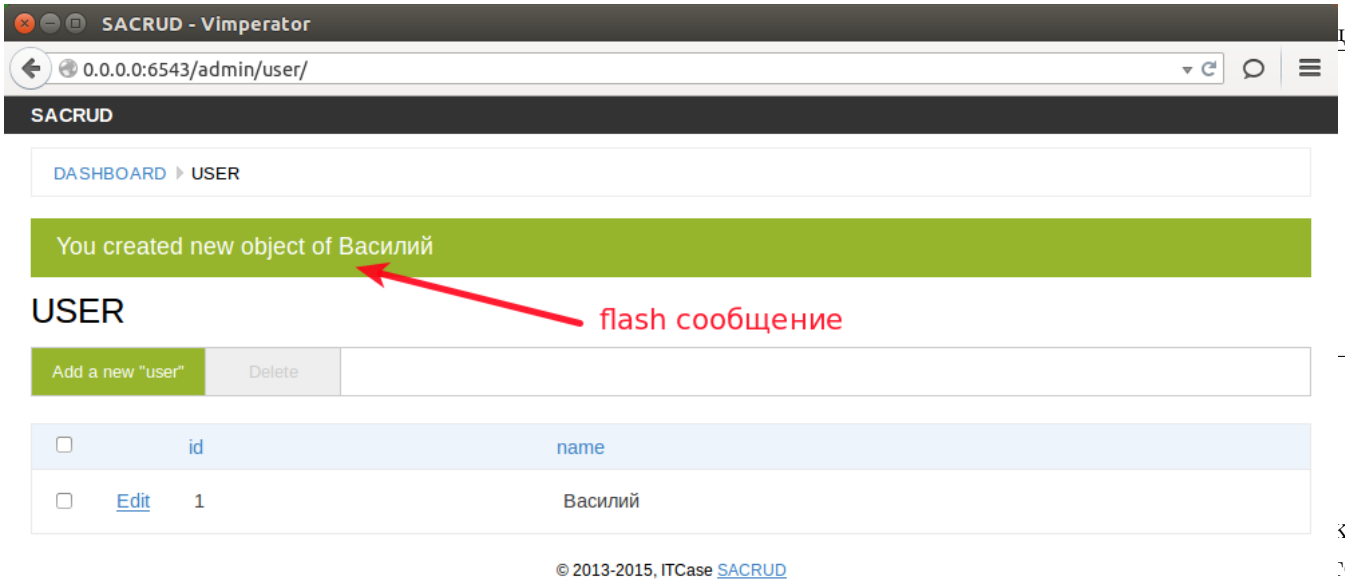


Рис. 49: Пример всплывающего сообщения после добавления записи в админке pyramid_sacrud

Чтобы добавить сообщение, достаточно выполнить метод сессии `flash()`

```
request.session.flash('Congratulations "rm -rf /" successful')
```

Чтобы извлечь сообщение, нужно вызвать метод сессии `pop_flash()`.

```
>>> request.session.flash('info message')
>>> request.session.pop_flash()
['info message']
>>> request.session.pop_flash()
[]
```

Для получения очереди, не извлекая сообщения из нее, нужно использовать метод `peek_flash()`.

```
>>> request.session.flash('info message')
>>> request.session.peek_flash()
['info message']
>>> request.session.peek_flash()
['info message']
>>> request.session.pop_flash()
['info message']
>>> request.session.peek_flash()
[]
```

Например, всплывающие сообщения используются в модуле pyramid_sacrud. Это простой CRUD веб-интерфейс который выводит сообщения после какой-либо операции.

Cross-Site Request Forgery (CSRF)

Для получения токена используется метод `session.get_csrf_token()`.

```
token = request.session.get_csrf_token()
```

Для создание нового токена:

```
token = request.session.new_csrf_token()
```

Пример добавления CSRF токена из текущей сессии в форму:

```
<form method="post" action="/myview">
  <input type="hidden" name="csrf_token" value="{{ request.session.get_csrf_
↪token() }}">
  <input type="submit" value="Delete Everything">
</form>
```

Проверка токена:

```
from pyramid.session import check_csrf_token

def myview(request):
    # Require CSRF Token
    check_csrf_token(request)

    # ...
```

или

```
@view_config(request_method='POST', check_csrf=True, ...)
def myview(request):
    ...
```

Резюме

1.5.11 Админка

См.также:

- https://ru.wikipedia.org/wiki/CRUD
- http://pyramid-sacrud.readthedocs.org/en/latest/

Фреймворк Pyramid не имеет CRUD веб-интерфейса или встроенной админки, как у фреймворков Django и web2py. Но за счет стороннего модуля pyramid_sacrud этот функционал можно добавить.

```
from .models import (Model1, Model2, Model3,)
# add sacrud and project models
config.include('pyramid_sacrud')
settings = config.registry.settings
settings['pyramid_sacrud.models'] = (('Group1', [Model1, Model2]),
                                     ('Group2', [Model3]))
```

Установка

См.также:

- http://pyramid-sacrud.readthedocs.org/en/latest/pages/install.html

```
pip install pyramid_sacrud
```

Использование

См.также:

- https://github.com/sacrud/pyramid_sacrud/tree/master/example

pyramid_sacrud предоставляет *CRUD* интерфейс для моделей SQLAlchemy. Создадим 3 простых таблицы (Car, Manufacturer, User) для примера:

```
from sqlalchemy import Column, ForeignKey, Integer, String
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import backref, relationship

Base = declarative_base()


class User(Base):
    __tablename__ = 'user'
    id = Column(Integer, primary_key=True)
    name = Column(String(30))

    def __repr__(self):
        return self.name


class Manufacturer(Base):
    __tablename__ = 'manufacturers'
    id = Column(Integer, primary_key=True)
    name = Column(String(30))


class Car(Base):
    __tablename__ = 'cars'
    id = Column(Integer, primary_key=True)
    name = Column(String(30))
    manufacturer_id = Column(Integer, ForeignKey('manufacturers.id'))
    manufacturer = relationship('Manufacturer',
                                backref=backref('cars', lazy='dynamic'))
```

Далее создадим Pyramid приложение и добавляем настройки БД.

```
from wsgiref.simple_server import make_server

from pyramid.config import Configurator

# ...

def database_settings(config):
    from sqlalchemy import create_engine
    config.registry.settings['sqlalchemy.url'] = db_url = "sqlite:///example.db"
    engine = create_engine(db_url)
    Base.metadata.bind = engine
    Base.metadata.create_all()

if __name__ == '__main__':
    config = Configurator()
    config.include(database_settings)
    app = config.make_wsgi_app()
    server = make_server('0.0.0.0', 6543, app)
    server.serve_forever()
```

Теперь опишем настройки нашего CRUD интерфейса:

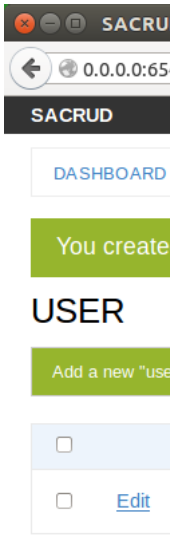
```
1 def sacrud_settings(config):
2     config.include('pyramid_sacrud', route_prefix='admin')
3     config.registry.settings['pyramid_sacrud.models'] = (
4         ('Vehicle', [Manufacturer, Car]),
5         ('Group2', [User])
6     )
```

`route_prefix='admin'` означает что интерфейс будет доступен по адресу http://localhost:6543/admin/ (по умолчанию http://localhost:6543/sacrud/).

В настройках (settings) параметр `pyramid_sacrud.models` отвечает за список моделей которые будут отображаться в интерфейсе. В нашем случае это 3 модели, поделенные на 2 группы (Vehicle и Group2).

Осталось включить эти настройки в проект:

```
# ...

if __name__ == '__main__':
    from pyramid.session import SignedCookieSessionFactory
    my_session_factory = SignedCookieSessionFactory('itsaseekreet')
    config = Configurator(session_factory=my_session_factory)
    config.include(database_settings)
    config.include(sacrud_settings)
    app = config.make_wsgi_app()
    server = make_server('0.0.0.0', 6543, app)
    server.serve_forever()
```



И запустить:

```
python __init__.py
```

По адресу http://localhost:6543/admin/ будет доступна наша админка!

Чтобы после *CRUD* операций появлялись всплывающие сообщение необходимо добавить в проект поддержку сессий.

```
# ...

if __name__ == '__main__':
    from pyramid.session import SignedCookieSessionFactory
    my_session_factory = SignedCookieSessionFactory('itsaseekreet')
    config = Configurator(session_factory=my_session_factory)

    # ...
```

Полный исходный код:

```
from wsgiref.simple_server import make_server

from pyramid.config import Configurator
from sqlalchemy import Column, ForeignKey, Integer, String
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import backref, relationship

Base = declarative_base()


class User(Base):
    __tablename__ = 'user'
    id = Column(Integer, primary_key=True)
    name = Column(String(30))
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    def __repr__(self):
        return self.name


class Manufacturer(Base):
    __tablename__ = 'manufacturers'
    id = Column(Integer, primary_key=True)
    name = Column(String(30))


class Car(Base):
    __tablename__ = 'cars'
    id = Column(Integer, primary_key=True)
    name = Column(String(30))
    manufacturer_id = Column(Integer, ForeignKey('manufacturers.id'))
    manufacturer = relationship('Manufacturer',
                                backref=backref('cars', lazy='dynamic'))


def sacrud_settings(config):
    config.include('pyramid_sacrud', route_prefix='admin')
    config.registry.settings['pyramid_sacrud.models'] = (
        ('Vehicle', [Manufacturer, Car]),
        ('Group2', [User])
    )


def database_settings(config):
    from sqlalchemy import create_engine
    config.registry.settings['sqlalchemy.url'] = db_url =\
        "sqlite:///example.db"
    engine = create_engine(db_url)
    Base.metadata.bind = engine
    Base.metadata.create_all()


if __name__ == '__main__':
    from pyramid.session import SignedCookieSessionFactory
    my_session_factory = SignedCookieSessionFactory('itsaseekreet')
    config = Configurator(session_factory=my_session_factory)
    config.include(database_settings)
    config.include(sacrud_settings)
    app = config.make_wsgi_app()
    server = make_server('0.0.0.0', 6543, app)
    server.serve_forever()
```

Резюме

CRUD интерфейс вряд ли может использоваться как основной инструмент клиента, но он может помочь визуализировать данные при разработке и выполнять простые операции связанные с администрированием приложения.

pyramid_sacrud довольно простой способ добавить в ваше приложение веб CRUD интерфейс, больше информации о настройке можно найти по адресу http://pyramid-sacrud.readthedocs.org/en/latest/pages/configuration.html, также pyramid_sacrud полностью совместим с настройками ColanderAlchemy.

Особенностью pyramid_sacrud является то что он не накладывает ограничений на структуру БД, а наоборот отталкивается от уже существующей. Ниже приведен пример как подключить его к БД не зная ее структуры:

См.также:

- https://gist.github.com/uralbash/019c0629e1448c9d4e71

```
"""
Funny application demonstrates the capabilities of SQLAlchemy and Pyramid.
It is something between phpMyAdmin and django.contrib.admin. SQLAlchemy with
Pyramid mapped on existing Django generated database but not vice versa.

Requirements
------------

pip install pyramid, sqlalchemy
pip install git+https://github.com/sacrud/pyramid_sacrud.git@develop

Demonstration
-------------

python SQLAlchemyMyAdmin.py

goto http://localhost:8080/sacrud/
"""
from wsgiref.simple_server import make_server

from pyramid.config import Configurator
from sqlalchemy import engine_from_config, MetaData
from sqlalchemy.ext.automap import automap_base
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import scoped_session, sessionmaker
from zope.sqlalchemy import ZopeTransactionExtension

DBSession = scoped_session(sessionmaker(extension=ZopeTransactionExtension()))
Base = declarative_base()
```

(continues on next page)

(продолжение с предыдущей страницы)

```
def get_metadata(engine):
    # produce our own MetaData object
    metadata = MetaData()
    metadata.reflect(engine)
    # we can then produce a set of mappings from this MetaData.
    Base = automap_base(metadata=metadata)
    # calling prepare() just sets up mapped classes and relationships.
    Base.prepare()
    return metadata


def quick_mapper(table):
    class GenericMapper(Base):
        __table__ = table
        __tablename__ = table.name
    return GenericMapper


def get_app():
    config = Configurator()
    settings = config.registry.settings
    settings['sqlalchemy.url'] = "postgresql://login:password@localhost/your_
↪database_name"

    # Database
    engine = engine_from_config(settings)
    DBSession.configure(bind=engine)
    metadata = get_metadata(engine)
    tables = [quick_mapper(table) for table in metadata.sorted_tables]

    # SACRUD
    settings['pyramid_sacrud.models'] = (
        ('', tables),
    )
    config.include('pyramid_sacrud')

    return config.make_wsgi_app()

if __name__ == '__main__':
    app = get_app()
    server = make_server('0.0.0.0', 8080, app)
    server.serve_forever()
```

1.5.12 Безопасность

См.также:

- http://docs.pylonsproject.org/projects/pyramid/en/latest/narr/security.html
- http://pyramid-cookbook.readthedocs.org/en/latest/pylons/auth.html
- http://michael.merickel.org/projects/pyramid_auth_demo/

Видео:

- https://skillsmatter.com/skillscasts/4536-back-to-school-and-talks

Аутентификация vs Авторизация

См.также:

- Аутентификация в Интернете
- Авторизация

В пирамиде система безопасности поделена на 2 части. Первая это аутентификация, которая производит идентификацию пользователя, его проверку (например что он есть в БД и он не заблокирован) и определяет какими правами он наделен. Второе это авторизация, система которая проверяет имеет ли этот пользователь доступ к запрошенному ресурсу.

Кто ты?

Примечание: Фреймворк Repoze.bfg имеет расширение repoze.who, которое отвечает за идентификацию и аутентификацию пользователя.

Who? т.е. *Кто?* ты.

Для авторизации используется расширение repoze.what, которое проверяет какие ресурсы тебе доступны.

What? т.е. *Что?* доступно тебе.

Несмотря на то, что фреймворк Pyramid это по сути переименованный repoze.bfg, в нем есть собственный механизм авторизации и аутентификации из коробки.

Определение текущего пользователя при поступлении HTTP запроса, это задача аутентификации (*authentication policy*). Производится она в 3 этапа:

1. Идентифицируем пользователя путем проверки токенов/заголовков/итд в HTTP запросе. (см. `pyramid.request.Request.unauthenticated_userid`)

Например: ищем `auth_token` в куках запроса, проверяем что токен правильно подписан, и возвращаем `id` пользователя.

2. Подтверждаем статус идентифицированного пользователя. (`authenticated_userid`)

   Например: проверяем что `id` этого пользователя все еще в базе данных и пользователь еще активен. Пользователя могли удалить из БД, но при этом в куках браузера хранится валидный токен `auth_token`.

3. Ищем группы (*principal*) которые принадлежат пользователю и добавляем их в список. (`effective_principals`)

   Например: берем из БД группы пользователя и добавляем в список. Для текущего идентифицированного пользователя это может быть: «vasya», «user_admin», «editor».

Что тебе дозволенно?

Каждый ресурс пирамиды может быть защищен правами доступа (*permission*). Задача авторизации определение того, какие пользователи имеют доступ к ресурсам.

После аутентификации создается список групп пользователя (*principal*). Политика авторизации (*authorization policy*) запрещает или разрешает доступ к ресурсу на основании этого списка групп, сверяя его с правами ресурса.

Добавление авторизации в проект

См.также:

- `pyramid.authorization`
- `pyramid.authentication`

В пирамиде по умолчанию авторизация отключена. Все представления (`views`) полностью доступны анонимным пользователям. Для того чтобы их защитить нужно добавить в настройки политику безопасности.

Для включения политики авторизации используется метод конфигуратора `pyramid.config.Configurator.set_authorization_policy()`. Для аутентификации `pyramid.config.Configurator.set_authentication_policy()` соответственно. Так-как авторизация не может существовать без аутентификации, необходимо указывать обе политики в проекте.

```
from pyramid.config import Configurator
from pyramid.authentication import AuthTktAuthenticationPolicy
from pyramid.authorization import ACLAuthorizationPolicy
```

(continues on next page)

(продолжение с предыдущей страницы)

```
authn_policy = AuthTktAuthenticationPolicy('seekrit', hashalg='sha512')
authz_policy = ACLAuthorizationPolicy()

config = Configurator()
config.set_authentication_policy(authn_policy)
config.set_authorization_policy(authz_policy)
```

Здесь pyramid.authentication.AuthTktAuthenticationPolicy это механизм аутентификации пользователя, который ищет его из «auth ticket» cookie. pyramid.authorization.ACLAuthorizationPolicy механизм авторизации по аксесс листам (*ACL*).

Права доступа для *View*

Императивно:

```
config.add_view('mypackage.views.blog_entry_add_view',
                name='add_entry.html',
                permission='add')
```

Декларативно:

```
from pyramid.view import view_config
from resources import Blog

@view_config(name='add_entry.html', permission='add')
def blog_entry_add_view(request):
    """ Add blog entry code goes here """
    # ...
```

Права доступа по умолчанию

Если ресурсу не присвоены права доступа, то используются права по умолчанию. В пирамиде права по умолчанию (pyramid.security.NO_PERMISSION_REQUIRED) подразумевают что ресурсы доступны всем, даже анонимным пользователям.

Это поведение возможно изменить при помощи метода pyramid.config.Configurator.set_default_permission().

```
config.set_default_permission('my_default_permission')
```

Аксесс листы (ACL)

См.также:

- https://ru.wikipedia.org/wiki/ACL

Access Control List или *ACL* — список контроля доступа, который определяет, кто или что может получать доступ к конкретному объекту, и какие именно операции разрешено или запрещено этому субъекту проводить над объектом.

В пирамиде аксесс лист это список содержащий записи, определяющие права индивидуального пользователя или группы на ресурсы проекта. Элементы ACL также еще называют Access Control Entry или *ACE*.

Например:

```
from pyramid.security import Allow, Deny
from pyramid.security import Everyone

__acl__ = [
    (Deny, 'vasya', 'move'),
    (Deny, 'group:blacklist', ('add', 'delete', 'edit')),

    (Allow, Everyone, 'view'),
    (Allow, 'group:editors', ('add', 'edit')),
    (Allow, 'group:editors', 'move'),
    (Allow, 'group:deleter', 'delete'),
]
```

`__acl__` из примера выше, это список контроля доступа (*ACL*).

`(Allow, Everyone, 'delete')` это *ACE*, т.е. запись в *ACL*.

1. Первый элемент в списке *ACE* это действие, т.е. «что делать?» разрешить или запретить. Действия представляются константами `pyramid.security.Allow` и `pyramid.security.Deny`.
2. Второй элемент списка это группы к которым принадлежит пользователь (*principal*).
3. Последний элемент это права или список прав.

Также существую специальные группы (*principal*):

- `pyramid.security.Everyone` - для всех.
- `pyramid.security.Authenticated` - для аутентифицированнных пользователей.

Если мы захотим запретить все, кроме тех *ACE* которые в списке, мы можем написать это так:

```
from pyramid.security import Allow
from pyramid.security import ALL_PERMISSIONS

__acl__ = [(Allow, 'fred', 'view'),
           (Deny, Everyone, ALL_PERMISSIONS)]
```

или воспользоваться встроенным в пирамиду *ACE*:

```
from pyramid.security import Allow
from pyramid.security import DENY_ALL

__acl__ = [(Allow, 'fred', 'view'),
           DENY_ALL]
```

ACL для ресурса

ACL для роутов

```
from wsgiref.simple_server import make_server
from pyramid.config import Configurator
from pyramid.response import Response
from pyramid.authentication import AuthTktAuthenticationPolicy
from pyramid.authorization import ACLAuthorizationPolicy

from pyramid.security import Allow
from pyramid.security import Everyone


class HelloFactory(object):
    def __init__(self, request):
        self.__acl__ = [
            (Allow, Everyone, 'view'),
            (Allow, 'group:editors', 'add'),
            (Allow, 'group:editors', 'edit'),
        ]


def hello_world(request):
    return Response('Hello %(name)s!' % request.matchdict)

if __name__ == '__main__':
    authn_policy = AuthTktAuthenticationPolicy('seekrit', hashalg='sha512')
    authz_policy = ACLAuthorizationPolicy()

```

(continues on next page)

(продолжение с предыдущей страницы)

```
    config = Configurator()
    config.set_authentication_policy(authn_policy)
    config.set_authorization_policy(authz_policy)

    config.add_route('hello', '/hello/{name}',
                     factory=HelloFactory)
    config.add_view(hello_world,
                    route_name='hello',
                    permission='view')

    app = config.make_wsgi_app()
    server = make_server('0.0.0.0', 8080, app)
    server.serve_forever()
```

Глобальный ACL

См.также:

- http://docs.pylonsproject.org/projects/pyramid/en/latest/tutorials/wiki2/authorization.html

```
from wsgiref.simple_server import make_server
from pyramid.config import Configurator
from pyramid.response import Response
from pyramid.authentication import AuthTktAuthenticationPolicy
from pyramid.authorization import ACLAuthorizationPolicy

from pyramid.security import Allow
from pyramid.security import Everyone


class HelloFactory(object):
    def __init__(self, request):
        self.__acl__ = [
            (Allow, Everyone, 'view'),
            (Allow, 'group:editors', 'add'),
            (Allow, 'group:editors', 'edit'),
        ]


def hello_world(request):
    return Response('Hello %(name)s!' % request.matchdict)

if __name__ == '__main__':
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    authn_policy = AuthTktAuthenticationPolicy('seekrit', hashalg='sha512')
    authz_policy = ACLAuthorizationPolicy()

    config = Configurator(root_factory=HelloFactory)
    config.set_authentication_policy(authn_policy)
    config.set_authorization_policy(authz_policy)

    config.add_route('hello', '/hello/{name}')
    config.add_view(hello_world,
                    route_name='hello',
                    permission='view')

    app = config.make_wsgi_app()
    server = make_server('0.0.0.0', 8080, app)
    server.serve_forever()
```

Логин & Логаут

```
from wsgiref.simple_server import make_server

from pyramid.authentication import AuthTktAuthenticationPolicy
from pyramid.authorization import ACLAuthorizationPolicy
from pyramid.config import Configurator
from pyramid.httpexceptions import HTTPFound
from pyramid.response import Response
from pyramid.security import Allow, forget, remember


class HelloFactory(object):
    def __init__(self, request):
        self.__acl__ = [
            (Allow, 'vasya', 'view'),
            (Allow, 'group:editors', 'add'),
            (Allow, 'group:editors', 'edit'),
        ]


def hello_world(request):
    return Response('Hello %(name)s!' % request.matchdict)


def login(request):
    headers = remember(request, 'vasya')
    return HTTPFound(location=request.route_url('hello', name='vasya'),
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                      headers=headers)


def logout(request):
    headers = forget(request)
    return HTTPFound(location=request.route_url('hello', name='log out!!!'),
                      headers=headers)


if __name__ == '__main__':
    authn_policy = AuthTktAuthenticationPolicy('seekrit', hashalg='sha512')
    authz_policy = ACLAuthorizationPolicy()

    config = Configurator(root_factory=HelloFactory)
    config.set_authentication_policy(authn_policy)
    config.set_authorization_policy(authz_policy)

    config.add_route('hello', '/hello/{name}')
    config.add_view(hello_world,
                    route_name='hello',
                    permission='view')

    # login form
    config.add_route('login', '/login')
    config.add_route('logout', '/logout')
    config.add_view(login, route_name='login')
    config.add_view(logout, route_name='logout')

    app = config.make_wsgi_app()
    server = make_server('0.0.0.0', 8080, app)
    server.serve_forever()
```

Basic Auth

См.также:

- https://gist.github.com/inklesspen/48cf6f3c7baa21df7839

```
from __future__ import absolute_import

from waitress import serve
from pyramid.config import Configurator
from pyramid.response import Response
from paste.httpheaders import WWW_AUTHENTICATE, AUTHORIZATION
```

(continues on next page)

(продолжение с предыдущей страницы)

```
from pyramid.security import Authenticated, Allow, Everyone
from pyramid.authorization import ACLAuthorizationPolicy


class Root(object):
    __acl__ = [
        (Allow, Authenticated, 'view'),
    ]

    def __init__(self, request):
        self.request = request


def checkauth(username, password):
    return username == 'pyramid' and password == 'aliens'


class BasicAuthenticationPolicy(object):
    def authenticated_userid(self, request):
        authorization = AUTHORIZATION(request.environ)
        if not authorization:
            return None
        (authmeth, auth) = authorization.split(' ', 1)
        if 'basic' != authmeth.lower():
            return None
        auth = auth.strip().decode('base64')
        username, password = auth.split(':', 1)
        if not checkauth(username, password):
            return None
        return username

    def effective_principals(self, request):
        ep = [Everyone]
        username = self.authenticated_userid(request)
        if username is not None:
            ep.append(Authenticated)
            ep.append(username)
            ep.append('g:admin')
        return ep

    def unauthenticated_userid(self, request):
        authorization = AUTHORIZATION(request.environ)
        if not authorization:
            return None
        (authmeth, auth) = authorization.split(' ', 1)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        if 'basic' != authmeth.lower():
            return None
        auth = auth.strip().decode('base64')
        username, password = auth.split(':', 1)
        return username

    def remember(self, request, principal, **kw):
        return []

    def forget(self, request):
        return []


def forbidden_view(request):
    head = WWW_AUTHENTICATE.tuples('Basic realm="%s"' % 'fnord')
    return Response('Not Authorized', status='401 Not Authorized', headers=head)


def hello_world(request):
    return Response('Hello {!r}!'.format(request.effective_principals))

if __name__ == '__main__':
    config = Configurator(root_factory=Root)
    config.add_route('hello', '/hello')
    config.add_view(hello_world, route_name='hello', permission='view')
    config.set_authentication_policy(BasicAuthenticationPolicy())
    config.set_authorization_policy(ACLAuthorizationPolicy())
    config.add_forbidden_view(forbidden_view)
    app = config.make_wsgi_app()
    serve(app, host='0.0.0.0', port=8080)
```

1.5.13 Блог

См.также:

- http://pyramid-blogr.readthedocs.org/en/latest/

Структура проекта

Создадим структуру будущего блога.

```
$ pcreate -t alchemy pyramid_blogr
```

```
$ cd pyramid_blogr
$ tree
.
├── CHANGES.txt
├── development.ini  <- файл с настройками проекта
├── MANIFEST.in
├── production.ini
├── pyramid_blogr
│   ├── __init__.py  <- точка входа нашего приложения, функция main.
│   │                   Создает конфиг и возвращает WSGI-приложение.
│   ├── models.py    <- описание схемы БД при помощи ORM SQLAlchemy
│   ├── scripts
│   │   ├── initializedb.py <- скрипт инициализации проекта
│   │   └── __init__.py
│   ├── static/      <- статические файлы (картинки, стили, javascript, ...)
│   ├── templates/   <- шаблоны
│   ├── tests.py
│   └── views.py     <- вьюхи (бизнес-логика приложения)
├── README.txt
└── setup.py
```

Базы данных

В скаффолде `alchemy`, который мы использовали для создания блога, уже существуют минимальные настройки для работы с БД.

Подключение к БД прописано в файле `development.ini`.

```
[app:main]
use = egg:pyramid_blogr

pyramid.reload_templates = true
pyramid.debug_authorization = false
pyramid.debug_notfound = false
pyramid.debug_routematch = false
pyramid.default_locale_name = en
pyramid.includes =
    pyramid_debugtoolbar
    pyramid_tm

sqlalchemy.url = sqlite:///%(here)s/pyramid_blogr.sqlite
```

Объект сессии создается в файле `pyramid_blogr/models.py`. Там же находится базовый класс для моделей.

```
from sqlalchemy import (
    Column,
    Index,
    Integer,
    Text,
    )

from sqlalchemy.ext.declarative import declarative_base

from sqlalchemy.orm import (
    scoped_session,
    sessionmaker,
    )

from zope.sqlalchemy import ZopeTransactionExtension

DBSession = scoped_session(sessionmaker(extension=ZopeTransactionExtension()))
Base = declarative_base()


class MyModel(Base):
    __tablename__ = 'models'
    id = Column(Integer, primary_key=True)
    name = Column(Text)
    value = Column(Integer)

Index('my_index', MyModel.name, unique=True, mysql_length=255)
```

В главном файле проекта `pyramid_blogr/__init__.py` находится функция `main`, которая вызывается при запуске команды `pserve development.ini`. Причем, настройки из файла `development.ini` передаются в эту функцию через атрибут `settings` (`def main(global_config, **settings):`).

`pserve` знает что нужно запустить функцию `main`, потому что это указанно в самом файле настроек `development.ini`.

```
###
# wsgi server configuration
###

[server:main]
use = egg:waitress#main
host = 0.0.0.0
port = 6543
```

Подключение к БД берется из настроек при помощи функции `sqlalchemy.engine_from_config()`. Далее объекту сессии и базовому классу указывается строка

подключения.

```
from pyramid.config import Configurator
from sqlalchemy import engine_from_config

from .models import (
    DBSession,
    Base,
    )


def main(global_config, **settings):
    """ This function returns a Pyramid WSGI application.
    """
    engine = engine_from_config(settings, 'sqlalchemy.')
    DBSession.configure(bind=engine)
    Base.metadata.bind = engine
    config = Configurator(settings=settings)
    config.include('pyramid_chameleon')
    config.add_static_view('static', 'static', cache_max_age=3600)
    config.add_route('home', '/')
    config.scan()
    return config.make_wsgi_app()
```

pyramid_sqlalchemy

См.также:

- http://pyramid-sqlalchemy.readthedocs.org/en/latest/

pyramid_sqlalchemy - расширение для Pyramid которое делает многие настройки БД за вас.

Установка:

```
$ pip install pyramid_sqlalchemy
```

Файл `__init__.py` стал значительно проще.

```
from pyramid.config import Configurator


def main(global_config, **settings):
    """ This function returns a Pyramid WSGI application.
    """
    config = Configurator(settings=settings)
    config.include('pyramid_sqlalchemy')
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    config.include('pyramid_chameleon')
    config.add_static_view('static', 'static', cache_max_age=3600)
    config.add_route('home', '/')
    config.scan()
    return config.make_wsgi_app()
```

Базовый класс и сессия импортируются прямо из библиотеки.

- `pyramid_sqlalchemy.BaseObject`
- `pyramid_sqlalchemy.Session`

Поэтому можно удалить `Base` и `DBSession` из файла `models.py`.

```
from sqlalchemy import (
    Column,
    Index,
    Integer,
    Text,
    )

from pyramid_sqlalchemy import BaseObject


class MyModel(BaseObject):
    __tablename__ = 'models'
    id = Column(Integer, primary_key=True)
    name = Column(Text)
    value = Column(Integer)

Index('my_index', MyModel.name, unique=True, mysql_length=255)
```

Сессии работаю аналогично. Пример `views.py`.

```
from pyramid.response import Response
from pyramid.view import view_config

from sqlalchemy.exc import DBAPIError

from pyramid_sqlalchemy import Session as DBSession
from .models import MyModel


@view_config(route_name='home', renderer='templates/mytemplate.pt')
def my_view(request):
    try:
        one = DBSession.query(MyModel).filter(MyModel.name == 'one').first()
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    except DBAPIError:
        return Response(conn_err_msg, content_type='text/plain', status_int=500)
    return {'one': one, 'project': 'pyramid_blogr'}


conn_err_msg = """\
Pyramid is having a problem using your SQL database.  The problem
might be caused by one of the following things:

A.  You may need to run the "initialize_pyramid_blogr_db" script
    to initialize your database tables.  Check your virtual
    environment's "bin" directory for this script and try to run it.

B.  Your database server may not be running.  Check that the
    database server referred to by the "sqlalchemy.url" setting in
    your "development.ini" file is running.

After you fix the problem, please restart the Pyramid application to
try it again.
"""
```

Таблицы блога

В файле `models.py` заменим `MyModel` на таблицы блога:

- User - для авторизации
- Article - статьи

```
import datetime

from pyramid_sqlalchemy import BaseObject
from sqlalchemy import Column, DateTime, Integer, Unicode, UnicodeText


class User(BaseObject):
    __tablename__ = 'users'
    id = Column(Integer, primary_key=True)
    name = Column(Unicode(255), unique=True, nullable=False)
    password = Column(Unicode(255), nullable=False)
    last_logged = Column(DateTime, default=datetime.datetime.utcnow)


class Article(BaseObject):
    __tablename__ = 'articles'
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    id = Column(Integer, primary_key=True)
    title = Column(Unicode(255), unique=True, nullable=False)
    content = Column(UnicodeText, default=u'')
    created = Column(DateTime, default=datetime.datetime.utcnow)
    edited = Column(DateTime, default=datetime.datetime.utcnow)
```

Инициализация

В скаффорлде существует файл инициализации проекта `pyramid_blogr/scripts/initializedb.py`. Его можно выполнить как команду окружения:

```
$ initialize_pyramid_blogr_db development.ini
```

В окружение эта команда попадает после установки (`python setup.py develop`) пакета, т.к. прописана в настройках `setup.py`.

```
# ...
setup(name='pyramid_blogr',
      version='0.0',
      description='pyramid_blogr',
      long_description=README + '\n\n' + CHANGES,
      classifiers=[
          "Programming Language :: Python",
          "Framework :: Pyramid",
          "Topic :: Internet :: WWW/HTTP",
          "Topic :: Internet :: WWW/HTTP :: WSGI :: Application",
      ],
      author='',
      author_email='',
      url='',
      keywords='web wsgi bfg pylons pyramid',
      packages=find_packages(),
      include_package_data=True,
      zip_safe=False,
      test_suite='pyramid_blogr',
      install_requires=requires,
      entry_points="""\
      [paste.app_factory]
      main = pyramid_blogr:main
      [console_scripts]
      initialize_pyramid_blogr_db = pyramid_blogr.scripts.initializedb:main
      """,
      )
```

Добавим в этот скрипт инициализации, создание новых таблиц, добавление пользователя «admin» и статей.

```
# -*- coding: utf-8 -*-
import os
import sys

import transaction
from pyramid.paster import get_appsettings, setup_logging
from pyramid.scripts.common import parse_vars
from pyramid_sqlalchemy import BaseObject as Base
from pyramid_sqlalchemy import Session as DBSession
from sqlalchemy import engine_from_config

from ..models import Article, User


def usage(argv):
    cmd = os.path.basename(argv[0])
    print('usage: %s <config_uri> [var=value]\n'
          '(example: "%s development.ini")' % (cmd, cmd))
    sys.exit(1)


def main(argv=sys.argv):
    if len(argv) < 2:
        usage(argv)
    config_uri = argv[1]
    options = parse_vars(argv[2:])
    setup_logging(config_uri)
    settings = get_appsettings(config_uri, options=options)
    engine = engine_from_config(settings, 'sqlalchemy.')
    DBSession.configure(bind=engine)

    Base.metadata.drop_all(engine)
    Base.metadata.create_all(engine)
    with transaction.manager:
        model = User(name=u'admin', password=u'admin')
        DBSession.add(model)
        from jinja2.utils import generate_lorem_ipsum
        for id, article in enumerate(range(100), start=1):
            title = generate_lorem_ipsum(
                n=1,          # Одно предложение
                html=False,   # В виде обычного текста
                min=2,        # Минимум 2 слова
                max=5         # Максимум 5
            )
```

(continues on next page)

(продолжение с предыдущей страницы)

```
45              content = generate_lorem_ipsum()
46              article = Article(**{'title': title, 'content': content})
47              DBSession.add(article)
```

Теперь при выполнении этого скрипта, наша БД будет пересоздаваться.

```
$ initialize_pyramid_blogr_db development.ini

CREATE TABLE articles (
        id INTEGER NOT NULL,
        title VARCHAR(255) NOT NULL,
        content TEXT,
        created DATETIME,
        edited DATETIME,
        PRIMARY KEY (id),
        UNIQUE (title)
)


2015-05-05 12:49:59,749 INFO  [sqlalchemy.engine.base.Engine][MainThread] ()
2015-05-05 12:49:59,755 INFO  [sqlalchemy.engine.base.Engine][MainThread] COMMIT
2015-05-05 12:49:59,755 INFO  [sqlalchemy.engine.base.Engine][MainThread]
CREATE TABLE users (
        id INTEGER NOT NULL,
        name VARCHAR(255) NOT NULL,
        password VARCHAR(255) NOT NULL,
        last_logged DATETIME,
        PRIMARY KEY (id),
        UNIQUE (name)
)


2015-05-05 12:49:59,755 INFO  [sqlalchemy.engine.base.Engine][MainThread] ()
2015-05-05 12:49:59,761 INFO  [sqlalchemy.engine.base.Engine][MainThread] COMMIT
2015-05-05 12:49:59,764 INFO  [sqlalchemy.engine.base.Engine][MainThread] BEGIN␣
↪(implicit)
2015-05-05 12:49:59,766 INFO  [sqlalchemy.engine.base.Engine][MainThread] INSERT␣
↪INTO users (name, password, last_logged) VALUES (?, ?, ?)
2015-05-05 12:49:59,767 INFO  [sqlalchemy.engine.base.Engine][MainThread] (u'admin
↪', u'admin', '2015-05-05 12:49:59.766198')
2015-05-05 12:49:59,769 INFO  [sqlalchemy.engine.base.Engine][MainThread] COMMIT
```

URL маршруты

Таблица 7: URL маршруты для блога

URL	Назначение
/	Главная страница со списком статей
/static/jquery.js	Статические файлы
/sign/in	Вход под своей учетной записью
/sign/out	Выход
/add	Добавление новой статьи
/article/13	Просмотр статьи с id=13
/article/13/edit	Редактирование статьи с id=13
/article/13/delete	Удаление статьи с id=13

Добавим пути в кофигуратор в файле `__init__.py`.

```
from pyramid.config import Configurator


def main(global_config, **settings):
    """ This function returns a Pyramid WSGI application.
    """
    config = Configurator(settings=settings)
    config.include('pyramid_sqlalchemy')
    config.include('pyramid_chameleon')

    config.add_static_view('static', 'static', cache_max_age=3600)
    config.add_route('blog', '/')
    config.add_route('blog_article', '/article/{id:\d+}')
    config.add_route('blog_action', '/article/{id:\d+}/{action}')
    config.add_route('auth', '/sign/{action}')

    config.scan()
    return config.make_wsgi_app()
```

Views

Создадим представления для нашего блога. Пока в виде «заглушек».

```
from pyramid.view import view_config


@view_config(route_name='blog',
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            renderer='blog/index.jinja2')
def index_page(request):
    return {}


@view_config(route_name='blog_article', renderer='blog/read.jinja2')
def blog_view(request):
    return {}


@view_config(route_name='blog_action', match_param='action=create',
            renderer='blog/edit.jinja2')
def blog_create(request):
    return {}


@view_config(route_name='blog_action', match_param='action=edit',
            renderer='blog/edit.jinja2')
def blog_update(request):
    return {}


@view_config(route_name='auth', match_param='action=in', renderer='string',
            request_method='POST')
@view_config(route_name='auth', match_param='action=out', renderer='string')
def sign_in_out(request):
    return {}
```

Главная страница

views.py

```
@view_config(route_name='blog',
            renderer='blog/index.jinja2')
def index_page(request):
    page = int(request.params.get('page', 1))
    paginator = Article.get_paginator(request, page)
    return {'paginator': paginator}
```

models.py Article

```
@classmethod
def get_paginator(cls, request, page=1):
    query = Session.query(Article).order_by(desc(Article.created))
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    query_params = request.GET.mixed()

    def url_maker(link_page):
        query_params['page'] = link_page
        return request.current_route_url(_query=query_params)
    return SqlalchemyOrmPage(query, page, items_per_page=5,
                             url_maker=url_maker)
```

Просмотр статей

views.py

```
@view_config(route_name='blog_article', renderer='blog/read.jinja2')
def blog_view(request):
    id = int(request.matchdict.get('id', -1))
    article = Article.by_id(id)
    if not article:
        return HTTPNotFound()
    return {'article': article}
```

models.py Article

```
@classmethod
def by_id(cls, id):
    return Session.query(Article).filter(Article.id == id).first()
```

Создание и редактирование

views.py

```
@view_config(route_name='blog_create',
             renderer='blog/edit.jinja2')
@view_config(route_name='blog_action', match_param='action=edit',
             renderer='blog/edit.jinja2')
def blog_create(request):
    form = get_form(request)
    if request.method == 'POST':
        try:
            values = form.validate(request.POST.items())
        except deform.ValidationFailure as e:
            return {'form': e.render(),
                    'action': request.matchdict.get('action')}
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        if request.matchdict['action'] == 'edit':
            article = Session.query(Article)\
                .filter_by(id=request.matchdict['id']).one()
            article.title = request.POST['title']
            article.content = request.POST['content']
        else:
            article = Article(**values)
        Session.add(article)
        return HTTPFound(location=request.route_url('blog'))
    values = {}
    if request.matchdict['action'] == 'edit':
        values = Session.query(Article)\
            .filter_by(id=request.matchdict['id']).one().__dict__
    return {'form': form.render(values),
            'action': request.matchdict.get('action')}
```

Полный код

```
import deform
from pyramid.httpexceptions import HTTPFound, HTTPNotFound
from pyramid.view import view_config
from pyramid_sqlalchemy import Session

from .forms import get_form
from .models import Article


@view_config(route_name='blog',
             renderer='blog/index.jinja2')
def index_page(request):
    page = int(request.params.get('page', 1))
    paginator = Article.get_paginator(request, page)
    return {'paginator': paginator}


@view_config(route_name='blog_article', renderer='blog/read.jinja2')
def blog_view(request):
    id = int(request.matchdict.get('id', -1))
    article = Article.by_id(id)
    if not article:
        return HTTPNotFound()
    return {'article': article}
```

(continues on next page)

(продолжение с предыдущей страницы)

```
@view_config(route_name='blog_create',
             renderer='blog/edit.jinja2')
@view_config(route_name='blog_action', match_param='action=edit',
             renderer='blog/edit.jinja2')
def blog_create(request):
    form = get_form(request)
    if request.method == 'POST':
        try:
            values = form.validate(request.POST.items())
        except deform.ValidationFailure as e:
            return {'form': e.render(),
                    'action': request.matchdict.get('action')}
        if request.matchdict.get('action', '') == 'edit':
            article = Session.query(Article)\
                .filter_by(id=request.matchdict['id']).one()
            article.title = request.POST['title']
            article.content = request.POST['content']
        else:
            article = Article(**values)
        Session.add(article)
        return HTTPFound(location=request.route_url('blog'))
    values = {}
    if request.matchdict.get('action', '') == 'edit':
        values = Session.query(Article)\
            .filter_by(id=request.matchdict['id']).one().__dict__
    return {'form': form.render(values),
            'action': request.matchdict.get('action')}


@view_config(route_name='blog_action', match_param='action=delete')
def blog_delete(request):
    article = Session.query(Article)\
        .filter_by(id=request.matchdict['id']).one()
    Session.delete(article)
    return HTTPFound(location=request.route_url('blog'))


@view_config(route_name='auth', match_param='action=in', renderer='string',
             request_method='POST')
@view_config(route_name='auth', match_param='action=out', renderer='string')
def sign_in_out(request):
    return {}
```

1.5.14 WSGI приложения

```
from wsgiref.simple_server import make_server
from pyramid.config import Configurator
from pyramid.response import Response


def hello(request):
    return Response('Hello world! See you blog <a href="blog/">there</a>!')


from pyramid.wsgi import wsgiapp


@wsgiapp
def hello_world(environ, start_response):
    body = 'Hello world'
    start_response('200 OK', [('Content-Type', 'text/plain'),
                              ('Content-Length', str(len(body)))])
    return [body]

if __name__ == '__main__':
    config = Configurator()
    config.add_route('hello_world', '/')
    config.add_route('hello_world_wsgi', '/hello_wsgi')
    config.add_view(hello, route_name='hello_world')
    config.add_view(hello_world, route_name='hello_world_wsgi')

    from my_wsgi_blog import make_wsgi_app
    blog_app = make_wsgi_app()
    from paste import urlmap
    mapping = urlmap.URLMap()
    mapping['/blog'] = blog_app

    from paste.cascade import Cascade
    pyramid_app = config.make_wsgi_app()
    app = Cascade([mapping, pyramid_app])

    server = make_server('0.0.0.0', 8000, app)
    server.serve_forever()
```

1.5.15 Glossary Pyramid

ACE An *access control entry*. An access control entry is one element in an *ACL*. An access control entry is a three-tuple that describes three things: an *action* (one of either

`Allow` or `Deny`), a *principal* (a string describing a user or group), and a *permission*. For example the ACE, `(Allow, 'bob', 'read')` is a member of an ACL that indicates that the principal `bob` is allowed the permission `read` against the resource the ACL is attached to.

ACL An *access control list*. An ACL is a sequence of *ACE* tuples. An ACL is attached to a resource instance. An example of an ACL is `[ (Allow, 'bob', 'read'), (Deny, 'fred', 'write')]`. If an ACL is attached to a resource instance, and that resource is findable via the context resource, it will be consulted any active security policy to determine wither a particular request can be fulfilled given the *authentication* information in the request.

action Represents a pending configuration statement generated by a call to a *configuration directive*. The set of pending configuration actions are processed when `pyramid.config.Configurator.commit()` is called.

add-on A Python *distribution* that uses Pyramid's extensibility to plug into a Pyramid application and provide extra, configurable services.

Agendaless Consulting A consulting organization formed by Paul Everitt, Tres Seaver, and Chris McDonough.

См.также:

See also Agendaless Consulting.

Akhet Akhet is a Pyramid library and demo application with a Pylons-like feel. It's most known for its former application scaffold, which helped users transition from Pylons and those preferring a more Pylons-like API. The scaffold has been retired but the demo plays a similar role.

application registry A registry of configuration information consulted by Pyramid while servicing an application. An application registry maps resource types to views, as well as housing other application-specific component registrations. Every Pyramid application has one (and only one) application registry.

asset Any file contained within a Python *package* which is *not* a Python source code file.

asset descriptor An instance representing an *asset specification* provided by the `pyramid.path.AssetResolver.resolve()` method. It supports the methods and attributes documented in `pyramid.interfaces.IAssetDescriptor`.

asset specification A colon-delimited identifier for an *asset*. The colon separates a Python *package* name from a package subpath. For example, the asset specification `my.package:static/baz.css` identifies the file named `baz.css` in the `static` subdirectory of the `my.package` Python *package*. See Understanding Asset Specifications for more info.

authentication The act of determining that the credentials a user presents during a particular request are «good». Authentication in Pyramid is performed via an *authentication policy*.

authentication policy An authentication policy in Pyramid terms is a bit of code which has an API which determines the current *principal* (or principals) associated with a request.

authorization The act of determining whether a user can perform a specific action. In pyramid terms, this means determining whether, for a given resource, any *principal* (or principals) associated with the request have the requisite *permission* to allow the request to continue. Authorization in Pyramid is performed via its *authorization policy*.

authorization policy An authorization policy in Pyramid terms is a bit of code which has an API which determines whether or not the principals associated with the request can perform an action associated with a permission, based on the information found on the *context* resource.

Babel A collection of tools for internationalizing Python applications. Pyramid does not depend on Babel to operate, but if Babel is installed, additional locale functionality becomes available to your application.

Chameleon chameleon is an attribute language template compiler which supports the *ZPT* templating specification. It is written and maintained by Malthe Borch. It has several extensions, such as the ability to use bracketed (Mako-style) `${name}` syntax. It is also much faster than the reference implementation of ZPT. Pyramid offers Chameleon templating out of the box in ZPT and text flavors.

configuration declaration An individual method call made to a *configuration directive*, such as registering a *view configuration* (via the `add_view()` method of the configurator) or *route configuration* (via the `add_route()` method of the configurator). A set of configuration declarations is also implied by the *configuration decoration* detected by a *scan* of code in a package.

configuration decoration Metadata implying one or more *configuration declaration* invocations. Often set by configuration Python *decorator* attributes, such as `pyramid.view.view_config`, aka `@view_config`.

configuration directive A method of the *Configurator* which causes a configuration action to occur. The method `pyramid.config.Configurator.add_view()` is a configuration directive, and application developers can add their own directives as necessary (see Adding Methods to the Configurator via add_directive).

configurator An object used to do *configuration declaration* within an application. The most common configurator is an instance of the `pyramid.config.Configurator` class.

conflict resolution Pyramid attempts to resolve ambiguous configuration statements made by application developers via automatic conflict resolution. Automatic conflict resolution is described in Automatic Conflict Resolution. If Pyramid cannot resolve ambiguous configuration statements, it is possible to manually resolve them as described in Manually Resolving Conflicts.

console script A script written to the `bin` (on UNIX, or `Scripts` on Windows) directory of a Python installation or *virtualenv* as the result of running `setup.py install` or

`setup.py develop`.

context A resource in the resource tree that is found during *traversal* or *URL dispatch* based on URL data; if it's found via traversal, it's usually a *resource* object that is part of a resource tree; if it's found via *URL dispatch*, it's an object manufactured on behalf of the route's «factory». A context resource becomes the subject of a *view*, and often has security information attached to it. See the Traversal chapter and the URL Dispatch chapter for more information about how a URL is resolved to a context resource.

CPython The C implementation of the Python language. This is the reference implementation that most people refer to as simply «Python»; *Jython*, Google's App Engine, and PyPy are examples of non-C based Python implementations.

declarative configuration The configuration mode in which you use the combination of *configuration decoration* and a *scan* to configure your Pyramid application.

decorator A wrapper around a Python function or class which accepts the function or class as its first argument and which returns an arbitrary object. Pyramid provides several decorators, used for configuration and return value modification purposes.

См.также:

See also PEP 318.

Default Locale Name The *locale name* used by an application when no explicit locale name is set. See Localization-Related Deployment Settings.

default permission A *permission* which is registered as the default for an entire application. When a default permission is in effect, every *view configuration* registered with the system will be effectively amended with a `permission` argument that will require that the executing user possess the default permission in order to successfully execute the associated *view callable*.

См.также:

See also Setting a Default Permission.

default root factory If an application does not register a *root factory* at Pyramid configuration time, a *default* root factory is used to created the default root object. Use of the default root object is useful in application which use *URL dispatch* for all URL-to-view code mappings, and does not (knowingly) use traversal otherwise.

Default view The default view of a *resource* is the view invoked when the *view name* is the empty string (`''`). This is the case when *traversal* exhausts the path elements in the PATH_INFO of a request before it returns a *context* resource.

Deployment settings Deployment settings are settings passed to the *Configurator* as a `settings` argument. These are later accessible via a `request.registry.settings` dictionary in views or as `config.registry.settings` in configuration code. Deployment settings can be used as global application values.

discriminator The unique identifier of an *action*.

distribute Distribute is a fork of *setuptools* which runs on both Python 2 and Python 3.

distribution (Setuptools/distutils terminology). A file representing an installable library or application. Distributions are usually files that have the suffix of `.egg`, `.tar.gz`, or `.zip`. Distributions are the target of Setuptools-related commands such as `easy_install`.

distutils The standard system for packaging and distributing Python packages. See http://docs.python.org/distutils/index.html for more information. *setuptools* is actually an *extension* of the Distutils.

Django A full-featured Python web framework.

domain model Persistent data related to your application. For example, data stored in a relational database. In some applications, the *resource tree* acts as the domain model.

dotted Python name A reference to a Python object by name using a string, in the form `path.to.modulename:attributename`. Often used in Pyramid and setuptools configurations. A variant is used in dotted names within configurator method arguments that name objects (such as the «add_view» method's «view» and «context» attributes): the colon (`:`) is not used; in its place is a dot.

entry point A *setuptools* indirection, defined within a setuptools *distribution* setup.py. It is usually a name which refers to a function somewhere in a package which is held by the distribution.

event An object broadcast to zero or more *subscriber* callables during normal Pyramid system operations during the lifetime of an application. Application code can subscribe to these events by using the subscriber functionality described in Using Events.

exception response A *response* that is generated as the result of a raised exception being caught by an *exception view*.

Exception view An exception view is a *view callable* which may be invoked by Pyramid when an exception is raised during request processing. See Custom Exception Views for more information.

finished callback A user-defined callback executed by the *router* unconditionally at the very end of request processing . See Using Finished Callbacks.

Forbidden view An *exception view* invoked by Pyramid when the developer explicitly raises a `pyramid.httpexceptions.HTTPForbidden` exception from within *view* code or *root factory* code, or when the *view configuration* and *authorization policy* found for a request disallows a particular view invocation. Pyramid provides a default implementation of a forbidden view; it can be overridden. See Changing the Forbidden View.

Genshi An XML templating language by Christopher Lenz.

Gettext The GNU gettext library, used by the Pyramid translation machinery.

Google App Engine Google App Engine (aka «GAE») is a Python application hosting service offered by Google. Pyramid runs on GAE.

Green Unicorn Aka `gunicorn`, a fast *WSGI* server that runs on UNIX under Python 2.6+ or Python 3.1+. See http://gunicorn.org/ for detailed information.

Grok A web framework based on Zope 3.

HTTP Exception The set of exception classes defined in `pyramid.httpexceptions`. These can be used to generate responses with various status codes when raised or returned from a *view callable*.

См.также:

See also HTTP Exceptions.

imperative configuration The configuration mode in which you use Python to call methods on a *Configurator* in order to add each *configuration declaration* required by your application.

interface A Zope interface object. In Pyramid, an interface may be attached to a *resource* object or a *request* object in order to identify that the object is «of a type». Interfaces are used internally by Pyramid to perform view lookups and other policy lookups. The ability to make use of an interface is exposed to an application programmers during *view configuration* via the `context` argument, the `request_type` argument and the `containment` argument. Interfaces are also exposed to application developers when they make use of the *event* system. Fundamentally, Pyramid programmers can think of an interface as something that they can attach to an object that stamps it with a «type» unrelated to its underlying Python type. Interfaces can also be used to describe the behavior of an object (its methods and attributes), but unless they choose to, Pyramid programmers do not need to understand or use this feature of interfaces.

Internationalization The act of creating software with a user interface that can potentially be displayed in more than one language or cultural context. Often shortened to «i18n» (because the word «internationalization» is I, 18 letters, then N).

См.также:

See also *Localization*.

introspectable An object which implements the attributes and methods described in `pyramid.interfaces.IIntrospectable`. Introspectables are used by the *introspector* to display configuration information about a running Pyramid application. An introspectable is associated with a *action* by virtue of the `pyramid.config.Configurator.action()` method.

introspector An object with the methods described by `pyramid.interfaces.IIntrospector` that is available in both configuration code (for registration) and at runtime (for querying) that allows a developer to introspect configuration statements and relationships between those statements.

Jinja2 A text templating language by Armin Ronacher.

jQuery A popular Javascript library.

JSON JavaScript Object Notation is a data serialization format.

Jython A Python implementation written for the Java Virtual Machine.

lineage An ordered sequence of objects based on a «*location* -aware» resource. The lineage of any given *resource* is composed of itself, its parent, its parent's parent, and so on. The order of the sequence is resource-first, then the parent of the resource, then its parent's parent, and so on. The parent of a resource in a lineage is available as its `__parent__` attribute.

Lingua A package by Wichert Akkerman which provides the `pot-create` command to extract translateable messages from Python sources and Chameleon ZPT template files.

Locale Name A string like `en`, `en_US`, `de`, or `de_AT` which uniquely identifies a particular locale.

Locale Negotiator An object supplying a policy determining which *locale name* best represents a given *request*. It is used by the `pyramid.i18n.get_locale_name()`, and `pyramid.i18n.negotiate_locale_name()` functions, and indirectly by `pyramid.i18n.get_localizer()`. The `pyramid.i18n.default_locale_negotiator()` function is an example of a locale negotiator.

Localization The process of displaying the user interface of an internationalized application in a particular language or cultural context. Often shortened to «l10» (because the word «localization» is L, 10 letters, then N).

См.также:

See also *Internationalization*.

Localizer An instance of the class `pyramid.i18n.Localizer` which provides translation and pluralization services to an application. It is retrieved via the `pyramid.i18n.get_localizer()` function.

location The path to an object in a *resource tree*. See Location-Aware Resources for more information about how to make a resource object *location-aware*.

Mako Mako is a template language which refines the familiar ideas of componentized layout and inheritance using Python with Python scoping and calling semantics.

matchdict The dictionary attached to the *request* object as `request.matchdict` when a *URL dispatch* route has been matched. Its keys are names as identified within the route pattern; its values are the values matched by each pattern name.

Message Catalog A *gettext* `.mo` file containing translations.

Message Identifier A string used as a translation lookup key during localization. The `msgid` argument to a *translation string* is a message identifier. Message identifiers are also present in a *message catalog*.

METAL Macro Expansion for TAL, a part of *ZPT* which makes it possible to share common look and feel between templates.

middleware *Middleware* is a *WSGI* concept. It is a WSGI component that acts both as a server and an application. Interesting uses for middleware exist, such as caching, content-transport encoding, and other functions. See WSGI.org or PyPI to find middleware for your application.

mod_wsgi mod_wsgi is an Apache module developed by Graham Dumpleton. It allows *WSGI* applications (such as applications developed using Pyramid) to be served using the Apache web server.

module A Python source file; a file on the filesystem that typically ends with the extension `.py` or `.pyc`. Modules often live in a *package*.

multidict An ordered dictionary that can have multiple values for each key. Adds the methods `getall`, `getone`, `mixed`, `add` and `dict_of_lists` to the normal dictionary interface. See Multidict and `pyramid.interfaces.IMultiDict`.

Not Found View An *exception view* invoked by Pyramid when the developer explicitly raises a `pyramid.httpexceptions.HTTPNotFound` exception from within *view* code or *root factory* code, or when the current request doesn't match any *view configuration*. Pyramid provides a default implementation of a Not Found View; it can be overridden. See Changing the Not Found View.

package A directory on disk which contains an `__init__.py` file, making it recognizable to Python as a location which can be `import` -ed. A package exists to contain *module* files.

PasteDeploy PasteDeploy is a library used by Pyramid which makes it possible to configure *WSGI* components together declaratively within an `.ini` file. It was developed by Ian Bicking.

permission A string or unicode object that represents an action being taken against a *context* resource. A permission is associated with a view name and a resource type by the developer. Resources are decorated with security declarations (e.g. an *ACL*), which reference these tokens also. Permissions are used by the active security policy to match the view permission against the resources's statements about which permissions are granted to which principal in a context in order to answer the question «is this user allowed to do this». Examples of permissions: `read`, or `view_blog_entries`.

physical path The path required by a traversal which resolve a *resource* starting from the *physical root*. For example, the physical path of the `abc` subobject of the physical root object is `/abc`. Physical paths can also be specified as tuples where the first element is the empty string (representing the root), and every other element is a Unicode object, e.g. `('', 'abc')`. Physical paths are also sometimes called «traversal paths».

physical root The object returned by the application *root factory*. Unlike the *virtual root* of a request, it is not impacted by Virtual Hosting: it will always be the actual object returned by the root factory, never a subobject.

pipeline The *PasteDeploy* term for a single configuration of a WSGI server, a WSGI application, with a set of *middleware* in-between.

pkg_resources A module which ships with *setuptools* and *distribute* that provides an API for addressing «asset files» within a Python *package*. Asset files are static files, template files, etc; basically anything non-Python-source that lives in a Python package can be considered a asset file.

См.также:

See also PkgResources.

predicate A test which returns `True` or `False`. Two different types of predicates exist in Pyramid: a *view predicate* and a *route predicate*. View predicates are attached to *view configuration* and route predicates are attached to *route configuration*.

predicate factory A callable which is used by a third party during the registration of a route, view, or subscriber predicates to extend the configuration system. See Adding a Third Party View, Route, or Subscriber Predicate for more information.

pregenerator A pregenerator is a function associated by a developer with a *route*. It is called by `route_url()` in order to adjust the set of arguments passed to it by the user for special purposes. It will influence the URL returned by `route_url()`. See `pyramid.interfaces.IRoutePregenerator` for more information.

principal A *principal* is a string or unicode object representing an entity, typically a user or group. Principals are provided by an *authentication policy*. For example, if a user had the *userid «bob»*, and was part of two groups named *«group foo»* and «group bar», the request might have information attached to it that would indicate that Bob was represented by three principals: *«bob»*, *«group foo»* and *«group bar»*.

project (Setuptools/distutils terminology). A directory on disk which contains a `setup.py` file and one or more Python packages. The `setup.py` file contains code that allows the package(s) to be installed, distributed, and tested.

Pylons A lightweight Python web framework and a predecessor of Pyramid.

PyPI The Python Package Index, a collection of software available for Python.

PyPy PyPy is an «alternative implementation of the Python language»: http://pypy.org/

Pyramid Cookbook Additional documentation for Pyramid which presents topical, practical uses of Pyramid: http://docs.pylonsproject.org/projects/pyramid_cookbook/en/latest.

pyramid_debugtoolbar A Pyramid add-on which displays a helpful debug toolbar «on top of» HTML pages rendered by your application, displaying request, routing, and

database information. pyramid_debugtoolbar is configured into the development.ini of all applications which use a Pyramid *scaffold*. For more information, see http://docs.pylonsproject.org/projects/pyramid_debugtoolbar/en/latest/.

pyramid_exclog A package which logs Pyramid application exception (error) information to a standard Python logger. This add-on is most useful when used in production applications, because the logger can be configured to log to a file, to UNIX syslog, to the Windows Event Log, or even to email. See its documentation.

pyramid_handlers An add-on package which allows Pyramid users to create classes that are analogues of Pylons 1 «controllers». See http://docs.pylonsproject.org/projects/pyramid_handlers/dev/ .

pyramid_jinja2 *Jinja2* templating system bindings for Pyramid, documented at http://docs.pylonsproject.org/projects/pyramid_jinja2/dev/ . This package also includes a scaffold named pyramid_jinja2_starter, which creates an application package based on the Jinja2 templating system.

pyramid_redis_sessions A package by Eric Rasmussen which allows you to store Pyramid session data in a Redis database. See https://pypi.python.org/pypi/pyramid_redis_sessions for more information.

pyramid_zcml An add-on package to Pyramid which allows applications to be configured via *ZCML*. It is available on *PyPI*. If you use pyramid_zcml, you can use ZCML as an alternative to *imperative configuration* or *configuration decoration*.

Python The programming language in which Pyramid is written.

renderer A serializer that can be referred to via *view configuration* which converts a non-*Response* return values from a *view* into a string (and ultimately a response). Using a renderer can make writing views that require templating or other serialization less tedious. See Writing View Callables Which Use a Renderer for more information.

renderer factory A factory which creates a *renderer*. See Adding and Changing Renderers for more information.

renderer globals Values injected as names into a renderer by a pyramid.event.BeforeRender event.

Repoze «Repoze» is essentially a «brand» of software developed by Agendaless Consulting and a set of contributors. The term has no special intrinsic meaning. The project's website has more information. The software developed «under the brand» is available in a Subversion repository. Pyramid was originally known as repoze.bfg.

repoze.catalog An indexing and search facility (fielded and full-text) based on zope.index. See the documentation for more information.

repoze.lemonade Zope2 CMF-like data structures and helper facilities for CA-and-ZODB-based applications useful within Pyramid applications.

repoze.who Authentication middleware for *WSGI* applications. It can be used by Pyramid to provide authentication information.

repoze.workflow Barebones workflow for Python apps . It can be used by Pyramid to form a workflow system.

request An object that represents an HTTP request, usually an instance of the `pyramid.request.Request` class. See Request and Response Objects (narrative) and pyramid.request (API documentation) for information about request objects.

request factory An object which, provided a *WSGI* environment as a single positional argument, returns a Pyramid-compatible request.

request type An attribute of a *request* that allows for specialization of view invocation based on arbitrary categorization. The every *request* object that Pyramid generates and manipulates has one or more *interface* objects attached to it. The default interface attached to a request object is `pyramid.interfaces.IRequest`.

resource An object representing a node in the *resource tree* of an application. If *traversal* is used, a resource is an element in the resource tree traversed by the system. When traversal is used, a resource becomes the *context* of a *view*. If *url dispatch* is used, a single resource is generated for each request and is used as the context resource of a view.

Resource Location The act of locating a *context* resource given a *request*. *Traversal* and *URL dispatch* are the resource location subsystems used by Pyramid.

resource tree A nested set of dictionary-like objects, each of which is a *resource*. The act of *traversal* uses the resource tree to find a *context* resource.

response An object returned by a *view callable* that represents response data returned to the requesting user agent. It must implement the `pyramid.interfaces.IResponse` interface. A response object is typically an instance of the `pyramid.response.Response` class or a subclass such as `pyramid.httpexceptions.HTTPFound`. See Request and Response Objects for information about response objects.

response adapter A callable which accepts an arbitrary object and «converts» it to a `pyramid.response.Response` object. See Changing How Pyramid Treats View Responses for more information.

response callback A user-defined callback executed by the *router* at a point after a *response* object is successfully created.

См.также:

See also Using Response Callbacks.

response factory An object which, provided a *request* as a single positional argument, returns a Pyramid-compatible response. See `pyramid.interfaces.IResponseFactory`.

reStructuredText A plain text markup format that is the defacto standard for documenting Python projects. The Pyramid documentation is written in reStructuredText.

root The object at which *traversal* begins when Pyramid searches for a *context* resource (for *URL Dispatch*, the root is *always* the context resource unless the `traverse=` argument is used in route configuration).

root factory The «root factory» of a Pyramid application is called on every request sent to the application. The root factory returns the traversal root of an application. It is conventionally named `get_root`. An application may supply a root factory to Pyramid during the construction of a *Configurator*. If a root factory is not supplied, the application creates a default root object using the *default root factory*.

route A single pattern matched by the *url dispatch* subsystem, which generally resolves to a *root factory* (and then ultimately a *view*).

См.также:

See also *url dispatch*.

route configuration Route configuration is the act of associating request parameters with a particular *route* using pattern matching and *route predicate* statements. See URL Dispatch for more information about route configuration.

route predicate An argument to a *route configuration* which implies a value that evaluates to `True` or `False` for a given *request*. All predicates attached to a *route configuration* must evaluate to `True` for the associated route to «match» the current request. If a route does not match the current request, the next route (in definition order) is attempted.

router The *WSGI* application created when you start a Pyramid application. The router intercepts requests, invokes traversal and/or URL dispatch, calls view functions, and returns responses to the WSGI server on behalf of your Pyramid application.

Routes A system by Ben Bangert which parses URLs and compares them against a number of user defined mappings. The URL pattern matching syntax in Pyramid is inspired by the Routes syntax (which was inspired by Ruby On Rails pattern syntax).

routes mapper An object which compares path information from a request to an ordered set of route patterns. See URL Dispatch.

scaffold A project template that generates some of the major parts of a Pyramid application and helps users to quickly get started writing larger applications. Scaffolds are usually used via the `pcreate` command.

scan The term used by Pyramid to define the process of importing and examining all code in a Python package or module for *configuration decoration*.

session A namespace that is valid for some period of continual activity that can be used to represent a user's interaction with a web application.

session factory A callable, which, when called with a single argument named `request` (a *request* object), returns a *session* object. See Using the Default Session Factory, Using Alternate Session Factories and `pyramid.config.Configurator.set_session_factory()` for more information.

setuptools Setuptools builds on Python's `distutils` to provide easier building, distribution, and installation of libraries and applications. As of this writing, setuptools runs under Python 2, but not under Python 3. You can use *distribute* under Python 3 instead.

SQLAlchemy SQLAlchemy is an object relational mapper used in tutorials within this documentation.

subpath A list of element «left over» after the *router* has performed a successful traversal to a view. The subpath is a sequence of strings, e.g. `['left', 'over', 'names']`. Within Pyramid applications that use URL dispatch rather than traversal, you can use `*subpath` in the route pattern to influence the subpath. See Using *subpath in a Route Pattern for more information.

subscriber A callable which receives an *event*. A callable becomes a subscriber via *imperative configuration* or via *configuration decoration*. See Using Events for more information.

template A file with replaceable parts that is capable of representing some text, XML, or HTML when rendered.

thread local A thread-local variable is one which is essentially a global variable in terms of how it is accessed and treated, however, each thread used by the application may have a different value for this same «global» variable. Pyramid uses a small number of thread local variables, as described in Thread Locals.

См.также:

See also the `stdlib documentation` for more information.

Translation Context A string representing the «context» in which a translation was made within a given *translation domain*. See the gettext documentation, 11.2.5 Using contexts for solving ambiguities for more information.

Translation Directory A translation directory is a *gettext* translation directory. It contains language folders, which themselves contain `LC_MESSAGES` folders, which contain `.mo` files. Each `.mo` file represents a set of translations for a language in a *translation domain*. The name of the `.mo` file (minus the .mo extension) is the translation domain name.

Translation Domain A string representing the «context» in which a translation was made. For example the word «java» might be translated differently if the translation domain is «programming-languages» than would be if the translation domain was «coffee». A translation domain is represented by a collection of `.mo` files within one or more *translation directory* directories.

Translation String An instance of `pyramid.i18n.TranslationString`, which is a class that behaves like a Unicode string, but has several extra attributes such as `domain`, `msgid`, and `mapping` for use during translation. Translation strings are usually created by hand within software, but are sometimes created on the behalf of the system for automatic template translation. For more information, see Internationalization and Localization.

Translator A callable which receives a *translation string* and returns a translated Unicode object for the purposes of internationalization. A *localizer* supplies a translator to a Pyramid application accessible via its `translate` method.

traversal The act of descending «up» a tree of resource objects from a root resource in order to find a *context* resource. The Pyramid *router* performs traversal of resource objects when a *root factory* is specified. See the Traversal chapter for more information. Traversal can be performed *instead* of *URL dispatch* or can be combined *with* URL dispatch. See Combining Traversal and URL Dispatch for more information about combining traversal and URL dispatch (advanced).

tween A bit of code that sits between the Pyramid router's main request handling function and the upstream WSGI component that uses Pyramid as its „app". The word «tween» is a contraction of «between». A tween may be used by Pyramid framework extensions, to provide, for example, Pyramid-specific view timing support, bookkeeping code that examines exceptions before they are returned to the upstream WSGI application, or a variety of other features. Tweens behave a bit like *WSGI middleware* but they have the benefit of running in a context in which they have access to the Pyramid *application registry* as well as the Pyramid rendering machinery. See Registering Tweens.

URL dispatch An alternative to *traversal* as a mechanism for locating a *context* resource for a *view*. When you use a *route* in your Pyramid application via a *route configuration*, you are using URL dispatch. See the URL Dispatch for more information.

userid A *userid* is a string or unicode object used to identify and authenticate a real-world user (or client). A userid is supplied to an *authentication policy* in order to discover the user's *principals*. The default behavior of the authentication policies Pyramid provides is to return the user's userid as a principal, but this is not strictly necessary in custom policies that define their principals differently.

Venusian Venusian is a library which allows framework authors to defer decorator actions. Instead of taking actions when a function (or class) decorator is executed at import time, the action usually taken by the decorator is deferred until a separate «scan» phase. Pyramid relies on Venusian to provide a basis for its *scan* feature.

view Common vernacular for a *view callable*.

view callable A «view callable» is a callable Python object which is associated with a *view configuration*; it returns a *response* object . A view callable accepts a single argument: `request`, which will be an instance of a *request* object. An alternate calling convention allows a view to be defined as a callable which accepts a pair of arguments: `context` and `request`: this calling convention is useful for traversal-based applications in which

a *context* is always very important. A view callable is the primary mechanism by which a developer writes user interface code within Pyramid. See Views for more information about Pyramid view callables.

view configuration View configuration is the act of associating a *view callable* with configuration information. This configuration information helps map a given *request* to a particular view callable and it can influence the response of a view callable. Pyramid views can be configured via *imperative configuration*, or by a special `@view_config` decorator coupled with a *scan*. See View Configuration for more information about view configuration.

View handler A view handler ties together `pyramid.config.Configurator.add_route()` and `pyramid.config.Configurator.add_view()` to make it more convenient to register a collection of views as a single class when using *url dispatch*. View handlers ship as part of the *pyramid_handlers* add-on package.

View Lookup The act of finding and invoking the «best» *view callable*, given a *request* and a *context* resource.

view mapper A view mapper is a class which implements the `pyramid.interfaces.IViewMapperFactory` interface, which performs view argument and return value mapping. This is a plug point for extension builders, not normally used by «civilians».

view name The «URL name» of a view, e.g `index.html`. If a view is configured without a name, its name is considered to be the empty string (which implies the *default view*).

view predicate An argument to a *view configuration* which evaluates to `True` or `False` for a given *request*. All predicates attached to a view configuration must evaluate to true for the associated view to be considered as a possible callable for a given request.

virtual root A resource object representing the «virtual» root of a request; this is typically the *physical root* object unless Virtual Hosting is in use.

virtualenv A term referring both to an isolated Python environment, or the leading tool that allows one to create such environments.

Note: whenever you encounter commands prefixed with `$VENV` (Unix) or `%VENV` (Windows), know that that is the environment variable whose value is the root of the virtual environment in question.

Waitress A *WSGI* server that runs on UNIX and Windows under Python 2.6+ and Python 3.2+. Projects generated via Pyramid scaffolding use Waitress as a WGSI server. See http://docs.pylonsproject.org/projects/waitress/en/latest/ for detailed information.

WebOb WebOb is a WSGI request/response library created by Ian Bicking.

WebTest WebTest is a package which can help you write functional tests for your WSGI application.

WSGI Web Server Gateway Interface. This is a Python standard for connecting web applications to web servers, similar to the concept of Java Servlets. Pyramid requires

that your application be served as a WSGI application.

ZCML Zope Configuration Markup Language, an XML dialect used by Zope and *pyramid_zcml* for configuration tasks.

ZODB Zope Object Database, a persistent Python object store.

Zope The Z Object Publishing Framework, a full-featured Python web framework.

Zope Component Architecture The Zope Component Architecture (aka ZCA) is a system which allows for application pluggability and complex dispatching based on objects which implement an *interface*. Pyramid uses the ZCA «under the hood» to perform view dispatching and other application configuration tasks.

ZPT The Zope Page Template templating language.

1.6 Асинхронный Веб

Предупреждение: Comet:

- https://bitbucket.org/MiCHiLU/my/src/b71dbf5a814b9840a7e3a155e560e81f2d04eac9/python/comet/comet_server.py?at=default
- http://d.hatena.ne.jp/MiCHiLU/20081108/1226147665
- http://www.pubnub.com/blog/websockets-and-long-polling-in-javascript-ruby-and-python/

WS:

- http://dev.enekoalonso.com/2010/05/22/more-websockets-now-with-python/
- http://yz.mit.edu/wp/web-sockets-tutorial-with-simple-python-server/
- http://stackoverflow.com/questions/16996789/creating-a-websocket-client-in-python
- http://popdevelop.com/2010/03/a-minimal-python-websocket-server/
- http://diesel.io/

Concurency:

- https://docs.python.org/3/library/concurrency.html
- http://www.toptal.com/python/beginners-guide-to-concurrency-and-parallelism-in-python
- https://github.com/volker48/python-concurrency
- http://www.slideshare.net/dabeaz/an-introduction-to-python-concurrency
- http://www.youtube.com/watch?v=ys8lW8eQaJQ
- https://lincolnloop.com/blog/concurrency-python-vs-go/

Asyncio:

- http://sahandsaba.com/understanding-asyncio-node-js-python-3-4.html
- https://docs.python.org/3/library/asyncio.html

1.6.1 AJAX

См.также:

- https://ru.wikipedia.org/wiki/AJAX

1.6.2 HTTP Comet

Polling

Long-poll

1.6.3 Асинхронный ввод/вывод

Примечание:

- http://www.curiousefficiency.org/posts/2015/07/asyncio-background-calls.html

---

Gevent

См.также:

- gevent

Asyncio

См.также:

- asyncio

1.6.4 WebSocket

См.также:

- http://tools.ietf.org/html/rfc6455

nodejs

python 3.4

aiohttp

http://www.slideshare.net/andrew_svetlov/aiohttp

aiopyramid

1.6.5 Фреймворки

aiohttp

См.также:

- aiohttp

Pulsar

См.также:

- Pulsar

Tornado

См.также:

- Tornado

1.7 Не браузер и не консоль Веб

1.7.1 Области применения

Работа с периферией:

- персональный компьютер (касса, сканер штрих кодов, ...)
- смартфон (камера, файловая система, навигация, ...)

1.7.2 Преимущества

- интерфейс создает верстальщик, а не программист
- загрузка интерфейса по сети

1.7.3 Технологии

Qt + WebKit

- http://www.bogotobogo.com/Qt/Qt5_WebKit_WebView_WebBrowser_QtCreator_B.php
- http://doc.qt.io/qt-5/qtwebengine-quicknanobrowser-example.html

node.js + WebKit

См.также:

- https://github.com/nwjs/nw.js
- http://electron.atom.io/

Установка:

```
$ npm install nw
```

Hello World

Примечание:

Исходный код примера:

- https://github.com/ustu/lectures.www/tree/master/sourcecode/7.www.gui/js/hello

---

Структура файлов:

```
.
├── index.html
└── package.json

0 directories, 2 files
```

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello World!</title>
  </head>
  <body>
    <h1>Hello World!</h1>
    We are using node.js <script>document.write(process.version)</script>.
  </body>
</html>
```

package.json

```
{
  "name": "Hello",
  "version": "1.0.0",
  "description": "",
  "main": "index.html",
  "window": {
    "toolbar": false,
    "width": 200,
    "height": 180
  }
}
```

Запуск:

```
$ nw
```

C#, обращение к REST API

См.также:

- https://github.com/iitwebdev/CSharpYandexAPI
- https://tech.yandex.ru/translate/doc/dg/concepts/About-docpage/

Пример online-переводчика, который использует класс WebRequest для обращения к API Яндекс.Переводчика.

См.также:

Как делать HTTP запросы на C#

Для удобства в программе используется простой GUI интерфейс.


Vimperator
localhost:8000/cgi-bin/test.py
<type 'exceptions.NameError'>
Python 2.7.8: /usr/bin/python
Thu Mar 5 13:32:36 2015
A problem occurred in a Python script. Here is the sequence of function calls leading up to the error, in the order they occurred.
/home/vagrant/sourcecode/cgi-bin/test.py in ()
23 def func3(arg3):
24 local_var = arg2 / 2
25 return local_var
26
=> 27 func1(1)
func1 = <function func1>
/home/vagrant/sourcecode/cgi-bin/test.py in func1(arg1=1)
13 def func1(arg1):
14 local_var = arg1*2
=> 15 return func2(local_var)
16
17
global func2 = <function func2>, local_var = 2
/home/vagrant/sourcecode/cgi-bin/test.py in func2(arg2=2)
18 def func2(arg2):
19 local_var = arg2 + 2
=> 20 return func3(local_var)
21
22
global func3 = <function func3>, local_var = 4
/home/vagrant/sourcecode/cgi-bin/test.py in func3(arg3=4)
22
23 def func3(arg3):
=> 24 local_var = arg2 / 2
25 return local_var
26
local_var undefined, arg2 undefined
<type 'exceptions.NameError'>: global name 'arg2' is not defined
args = ("global name 'arg2' is not defined",)
message = "global name 'arg2' is not defined"

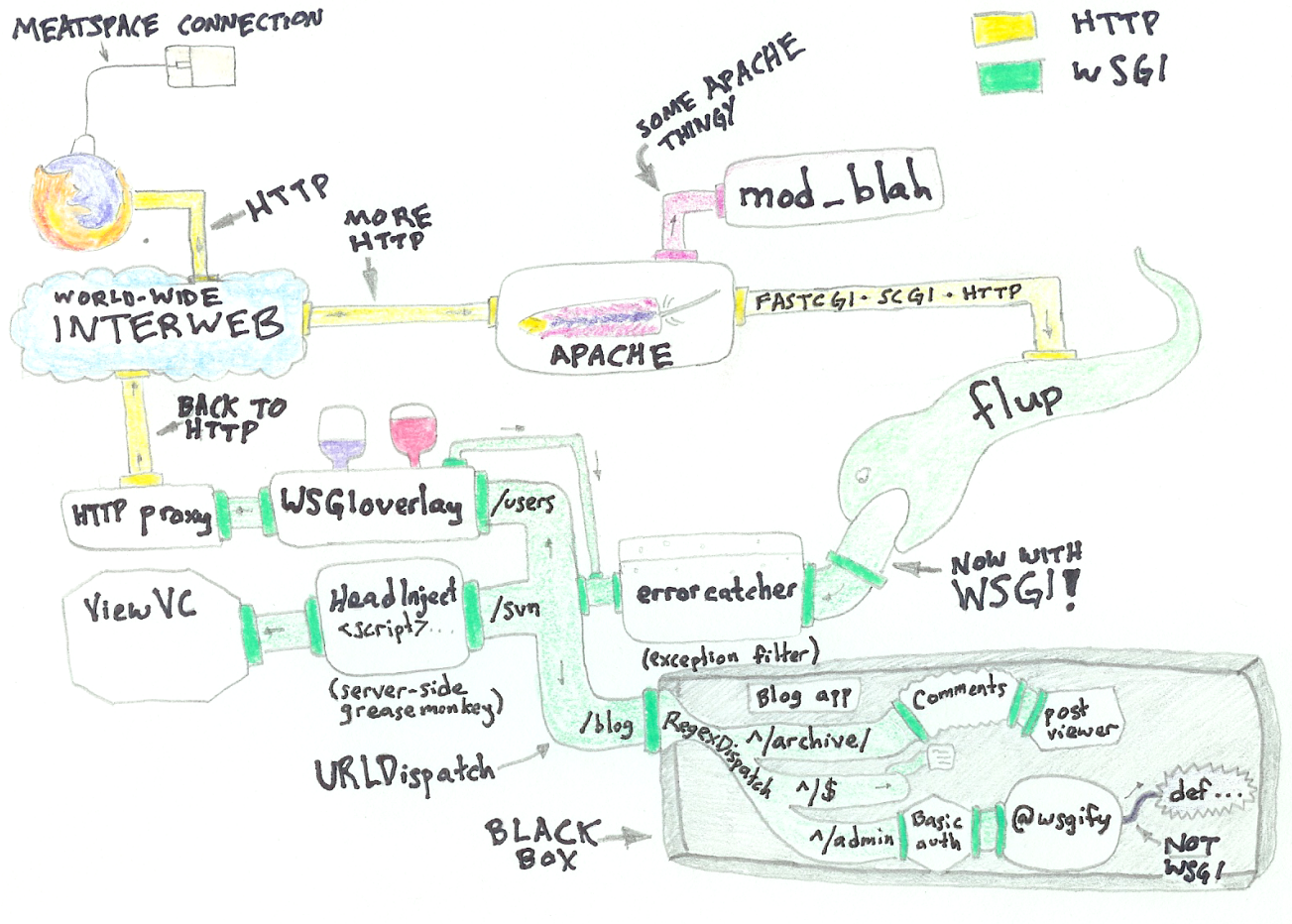


Рис. 2: Пример работы *WSGI* (автор Ян Бикинг)

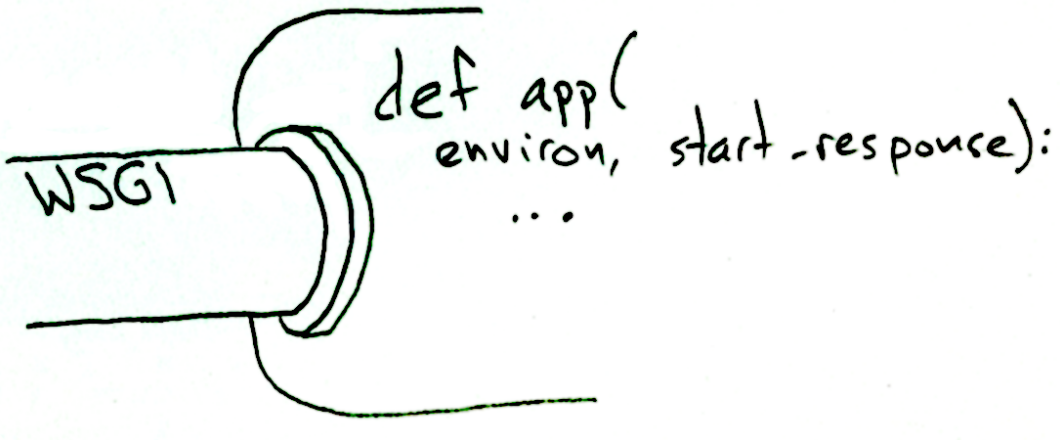


Рис. 3: *WSGI-приложение*

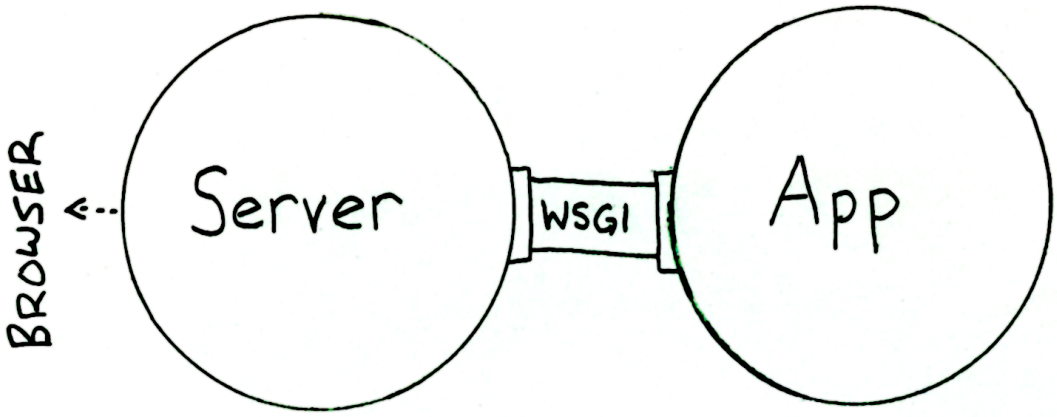


Рис. 4: *WSGI-сервер*

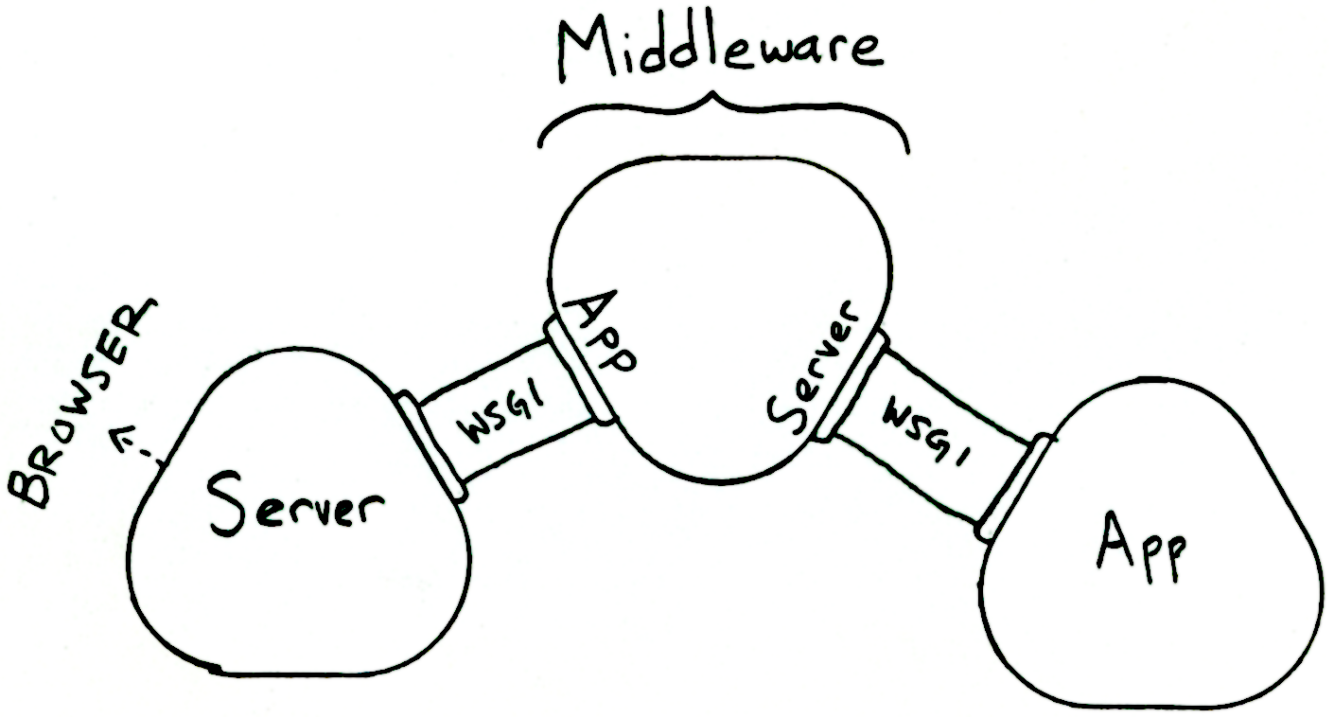
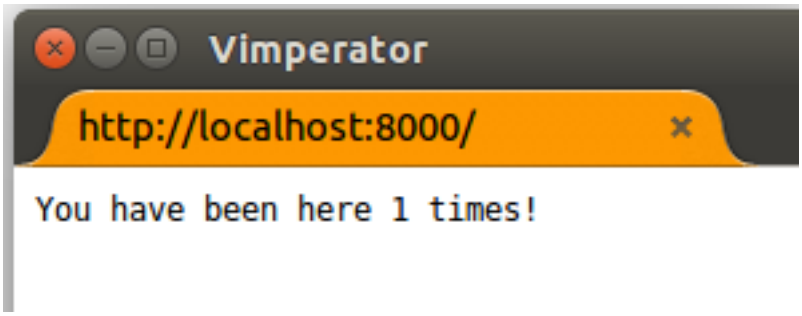
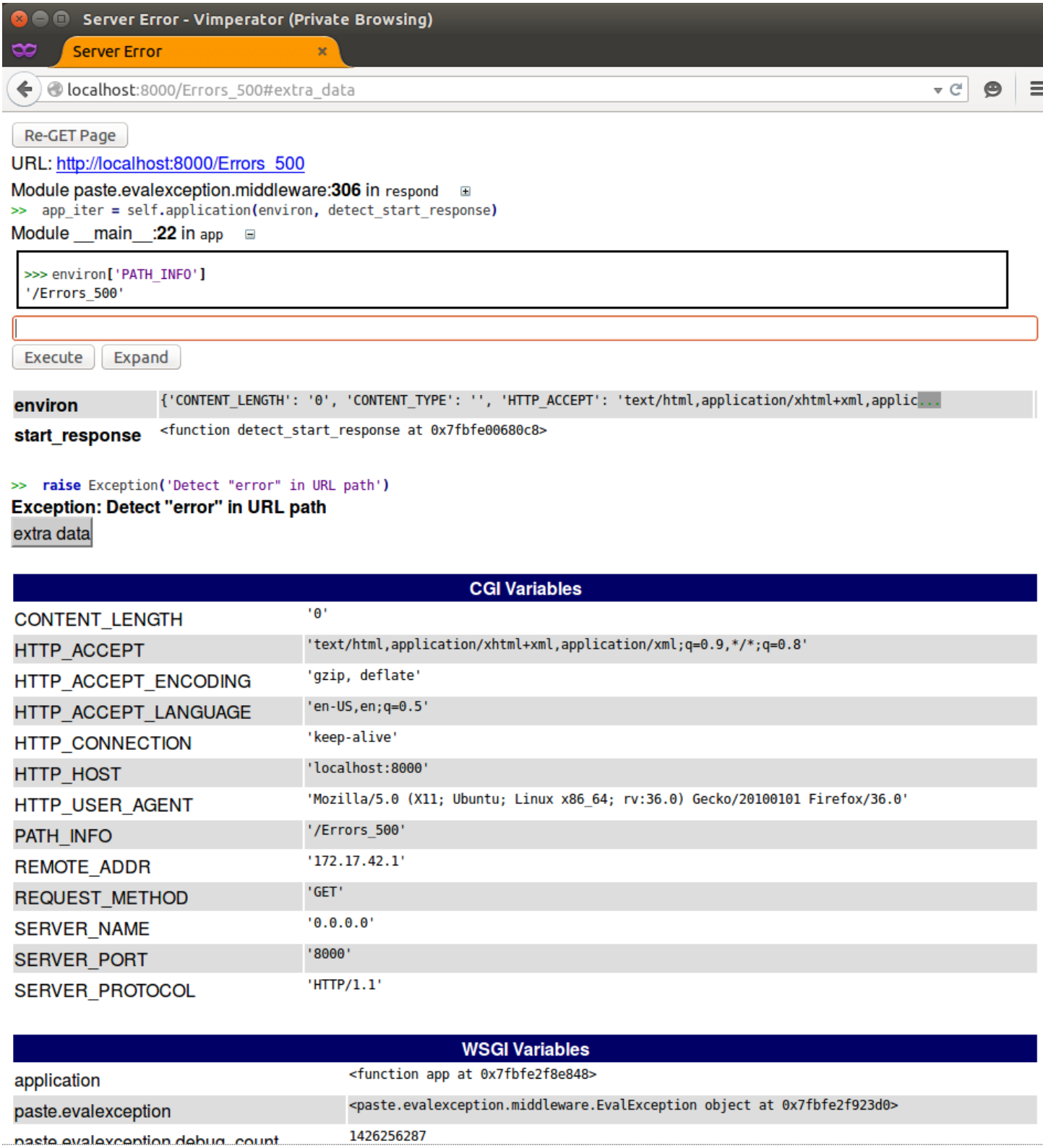
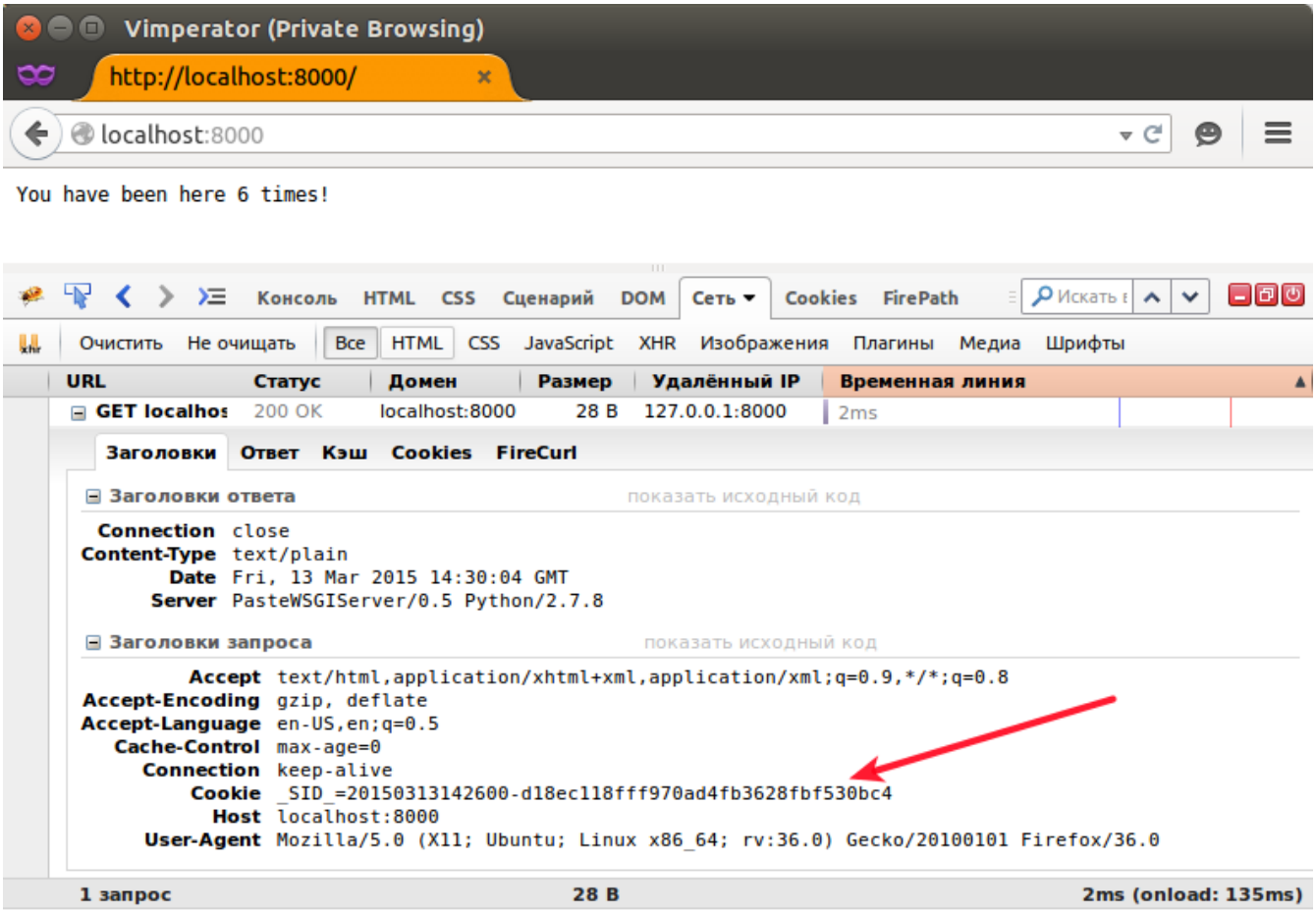
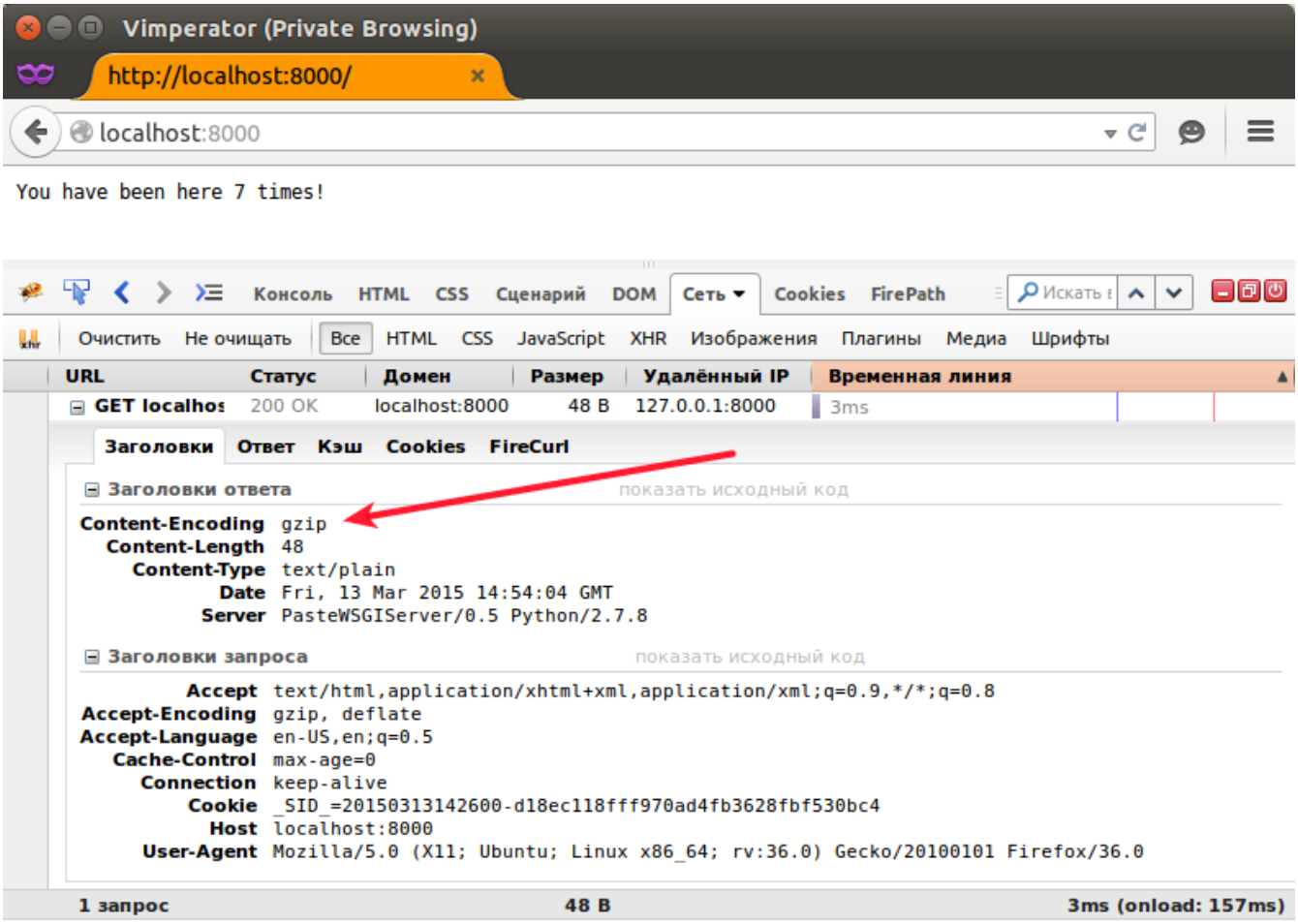


Рис. 5: *WSGI-middleware*




Server Error - Vimperator (Private Browsing)
Server Error
localhost:8000/Errors_500#extra_data
Re-GET Page
URL: http://localhost:8000/Errors_500
Module paste.evalexception.middleware:306 in respond
>> app_iter = self.application(environ, detect_start_response)
Module __main__:22 in app
>>> environ['PATH_INFO']
'/Errors_500'
Execute
Expand
environ
{'CONTENT_LENGTH': '0', 'CONTENT_TYPE': '', 'HTTP_ACCEPT': 'text/html,application/xhtml+xml,applic...
start_response
<function detect_start_response at 0x7fbfe00680c8>
>> raise Exception('Detect "error" in URL path')
Exception: Detect "error" in URL path
extra data
CGI Variables
CONTENT_LENGTH '0'
HTTP_ACCEPT 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8'
HTTP_ACCEPT_ENCODING 'gzip, deflate'
HTTP_ACCEPT_LANGUAGE 'en-US,en;q=0.5'
HTTP_CONNECTION 'keep-alive'
HTTP_HOST 'localhost:8000'
HTTP_USER_AGENT 'Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:36.0) Gecko/20100101 Firefox/36.0'
PATH_INFO '/Errors_500'
REMOTE_ADDR '172.17.42.1'
REQUEST_METHOD 'GET'
SERVER_NAME '0.0.0.0'
SERVER_PORT '8000'
SERVER_PROTOCOL 'HTTP/1.1'
WSGI Variables
application <function app at 0x7fbfe2f8e848>
paste.evalexception <paste.evalexception.middleware.EvalException object at 0x7fbfe2f923d0>
paste.evalexception.debug_count 1426256287


Vimperator (Private Browsing)
http://localhost:8000/
localhost:8000
You have been here 6 times!
Консоль HTML CSS Сценарий DOM Сеть Cookies FirePath
Очистить Не очищать Все HTML CSS JavaScript XHR Изображения Плагины Медиа Шрифты
URL Статус Домен Размер Удалённый IP Временная линия
GET localhos 200 OK localhost:8000 28 B 127.0.0.1:8000 2ms
Заголовки Ответ Кэш Cookies FireCurl
Заголовки ответа показать исходный код
Connection close
Content-Type text/plain
Date Fri, 13 Mar 2015 14:30:04 GMT
Server PasteWSGIServer/0.5 Python/2.7.8
Заголовки запроса показать исходный код
Accept text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding gzip, deflate
Accept-Language en-US,en;q=0.5
Cache-Control max-age=0
Connection keep-alive
Cookie _SID_=20150313142600-d18ec118fff970ad4fb3628fbf530bc4
Host localhost:8000
User-Agent Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:36.0) Gecko/20100101 Firefox/36.0
1 запрос 28 B 2ms (onload: 135ms)
http://localhost:8000/ <[1/1]All


Vimperator (Private Browsing)
http://localhost:8000/
localhost:8000
You have been here 7 times!
Консоль HTML CSS Сценарий DOM Сеть Cookies FirePath
Очистить Не очищать Все HTML CSS JavaScript XHR Изображения Плагины Медиа Шрифты
URL Статус Домен Размер Удалённый IP Временная линия
GET localhos 200 OK localhost:8000 48 B 127.0.0.1:8000 3ms
Заголовки Ответ Кэш Cookies FireCurl
Заголовки ответа показать исходный код
Content-Encoding gzip
Content-Length 48
Content-Type text/plain
Date Fri, 13 Mar 2015 14:54:04 GMT
Server PasteWSGIServer/0.5 Python/2.7.8
Заголовки запроса показать исходный код
Accept text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding gzip, deflate
Accept-Language en-US,en;q=0.5
Cache-Control max-age=0
Connection keep-alive
Cookie _SID_=20150313142600-d18ec118fff970ad4fb3628fbf530bc4
Host localhost:8000
User-Agent Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:36.0) Gecko/20100101 Firefox/36.0
1 запрос 48 B 3ms (onload: 157ms)
http://localhost:8000/ <[1/1]All

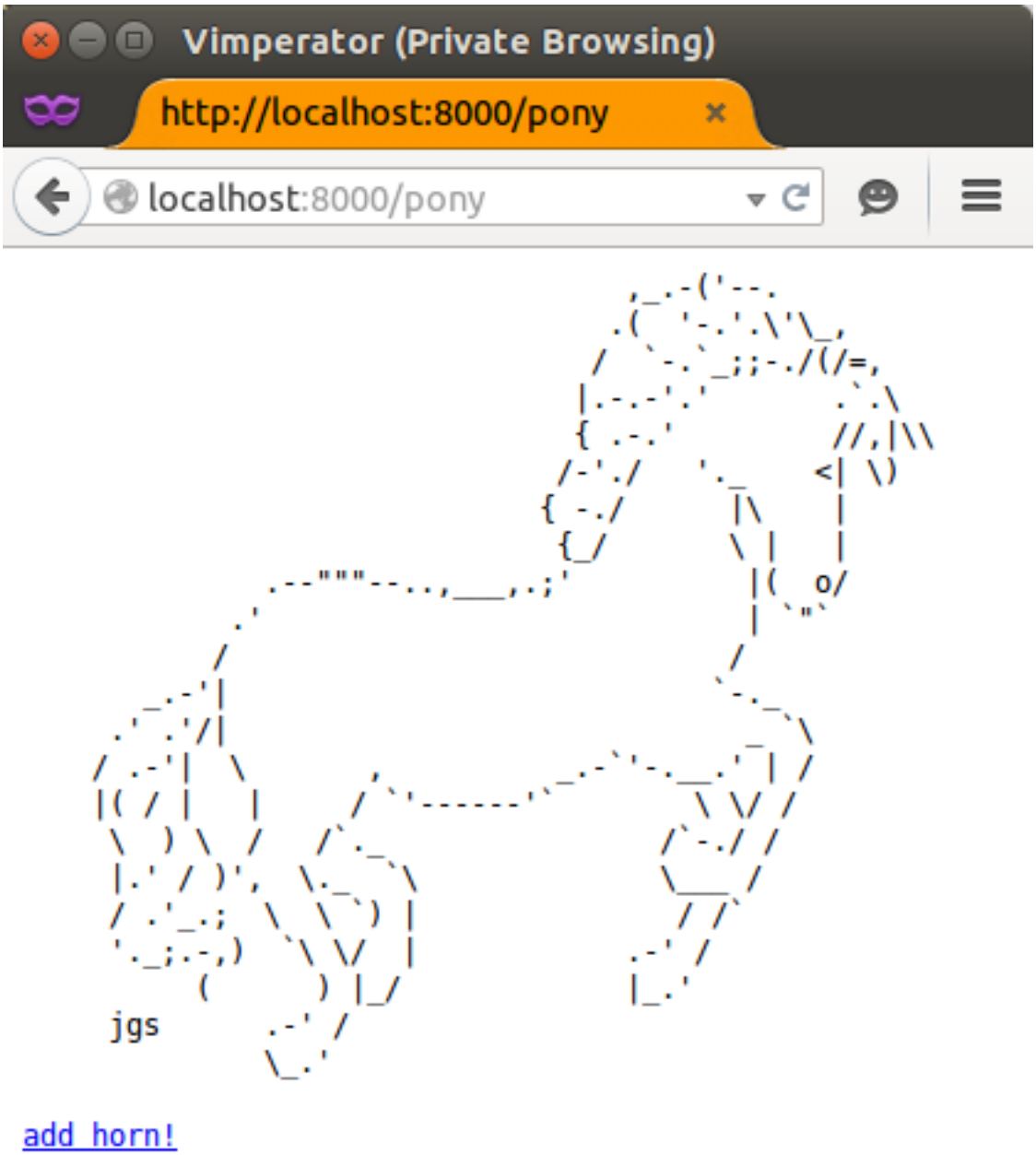


Рис. 6: Схема работы WSGI-приложения *Blog*

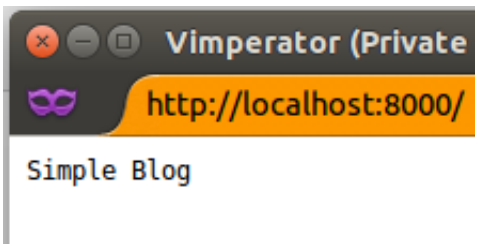


Рис. 7: Главная страница блога

Рис. 8: URLDispatch middleware

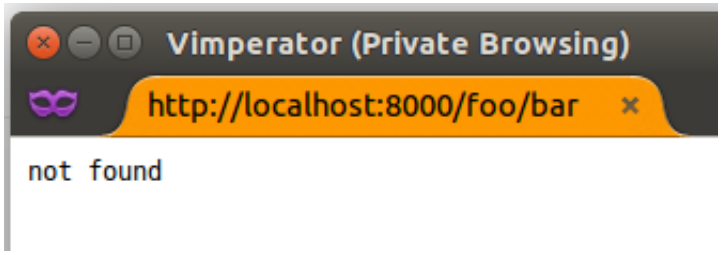


Рис. 9: *404 Not Found*

Рис. 10: Сопоставление путей и *WSGI-приложений*

Рис. 11: URL пути на регулярных выражениях

Рис. 12: Список статей на главной странице

Рис. 13: Страница статьи

Рис. 14: Форма создания новой статьи

Рис. 15: Форма редактирования статьи

Рис. 16: BasicAuth *WSGI-middleware* для авторизации

Рис. 17: Паттерн MVC (Model-View-Controller)

Рис. 18: TADA!! Django invented MTV

Рис. 19: Паттерн MTV (Model-Template-View)

Рис. 20: Паттерн RV (Resources-View)

Index

Welcome Петя to my awesome homepage.

© Copyright 2008 by you.

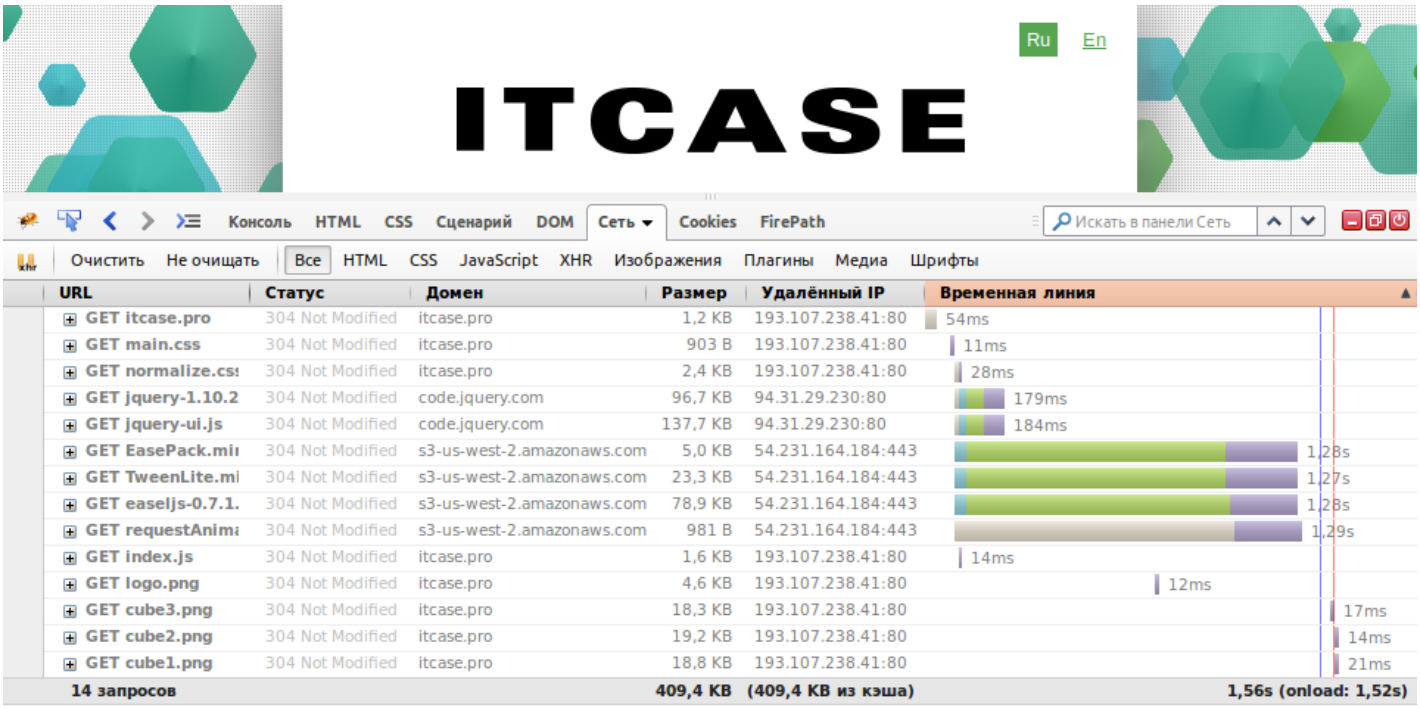


Рис. 22: http://itcase.pro

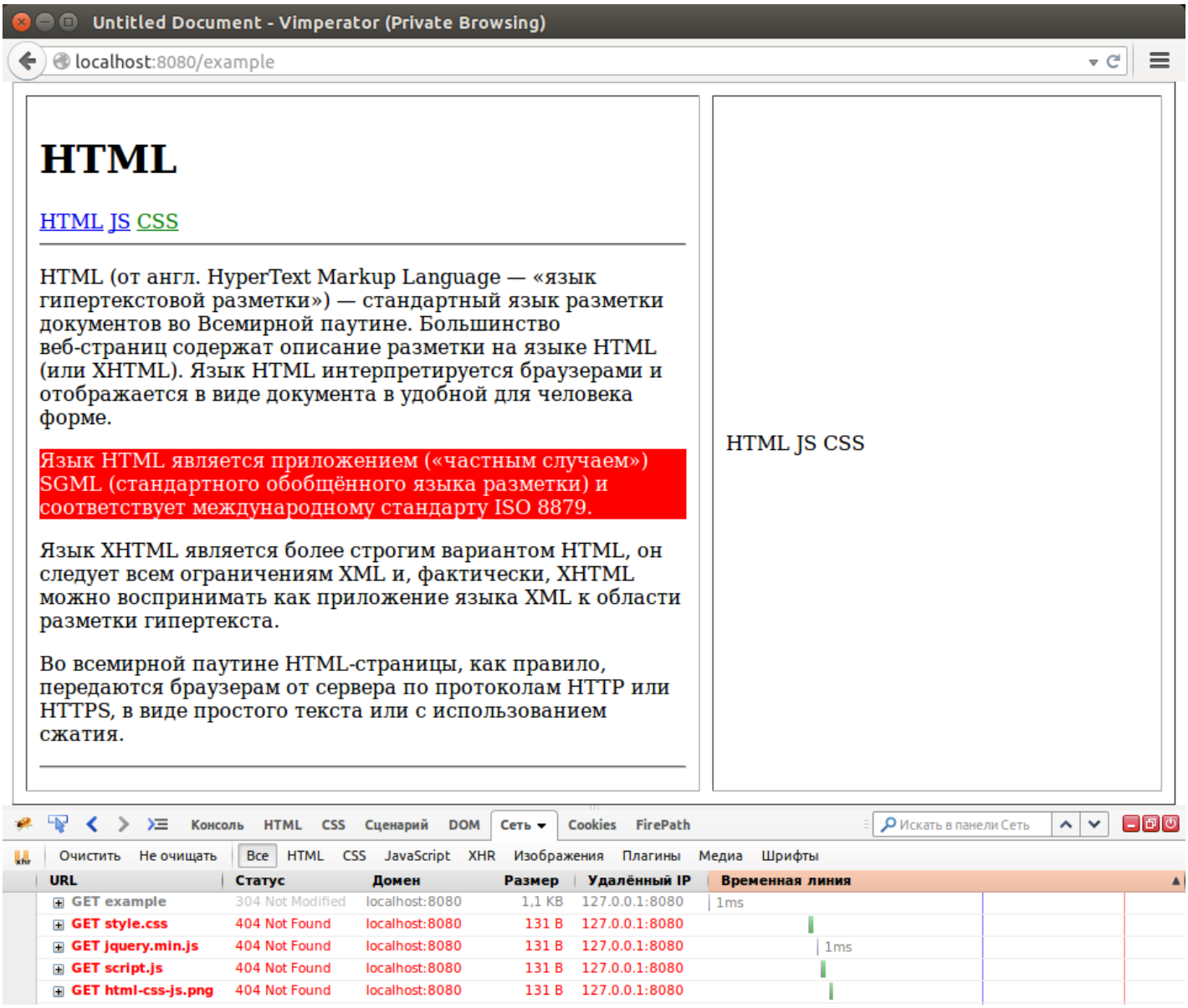


Рис. 23: Пример index2.html без статики

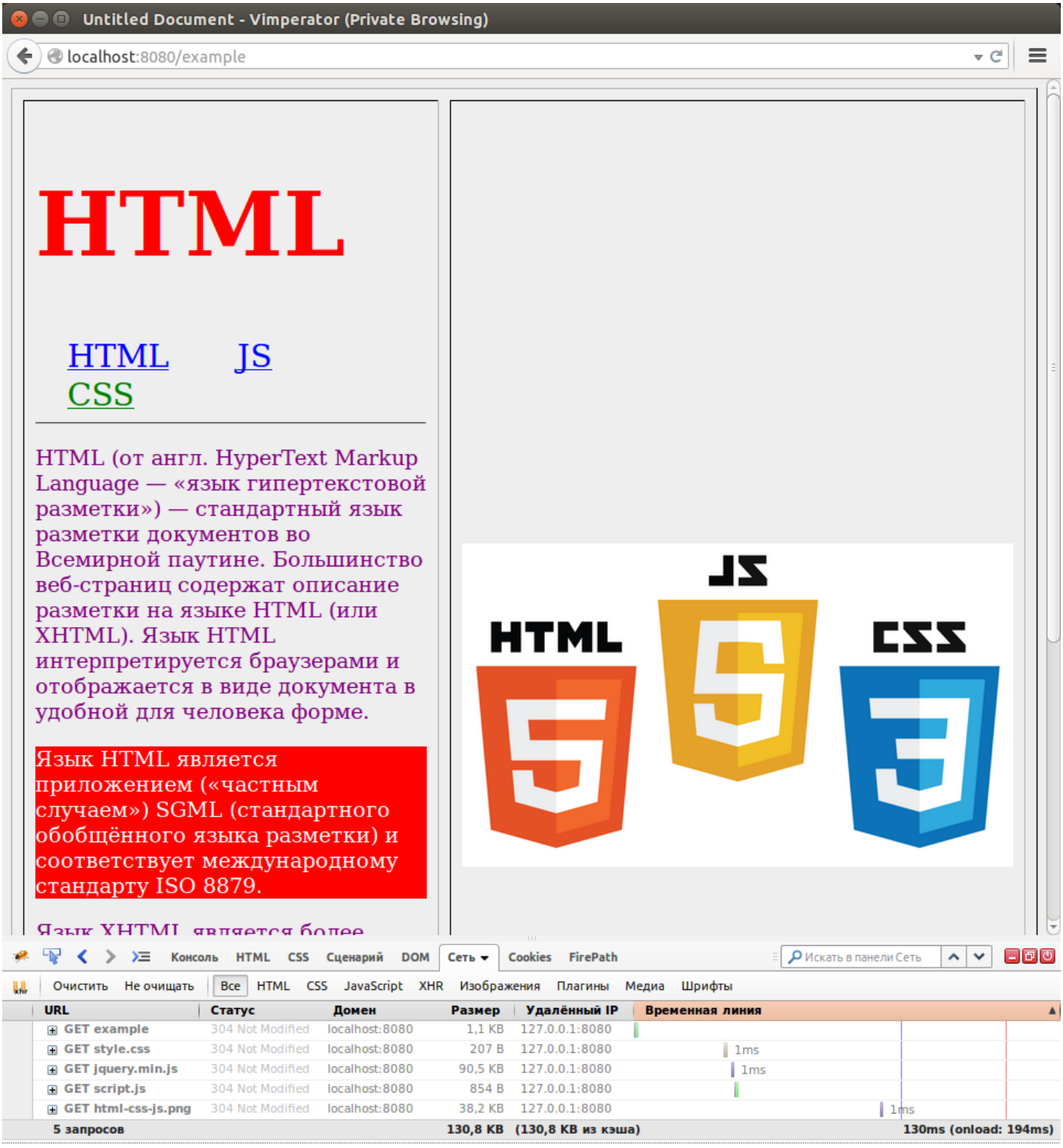


Рис. 24: Пример index2.html со статикой

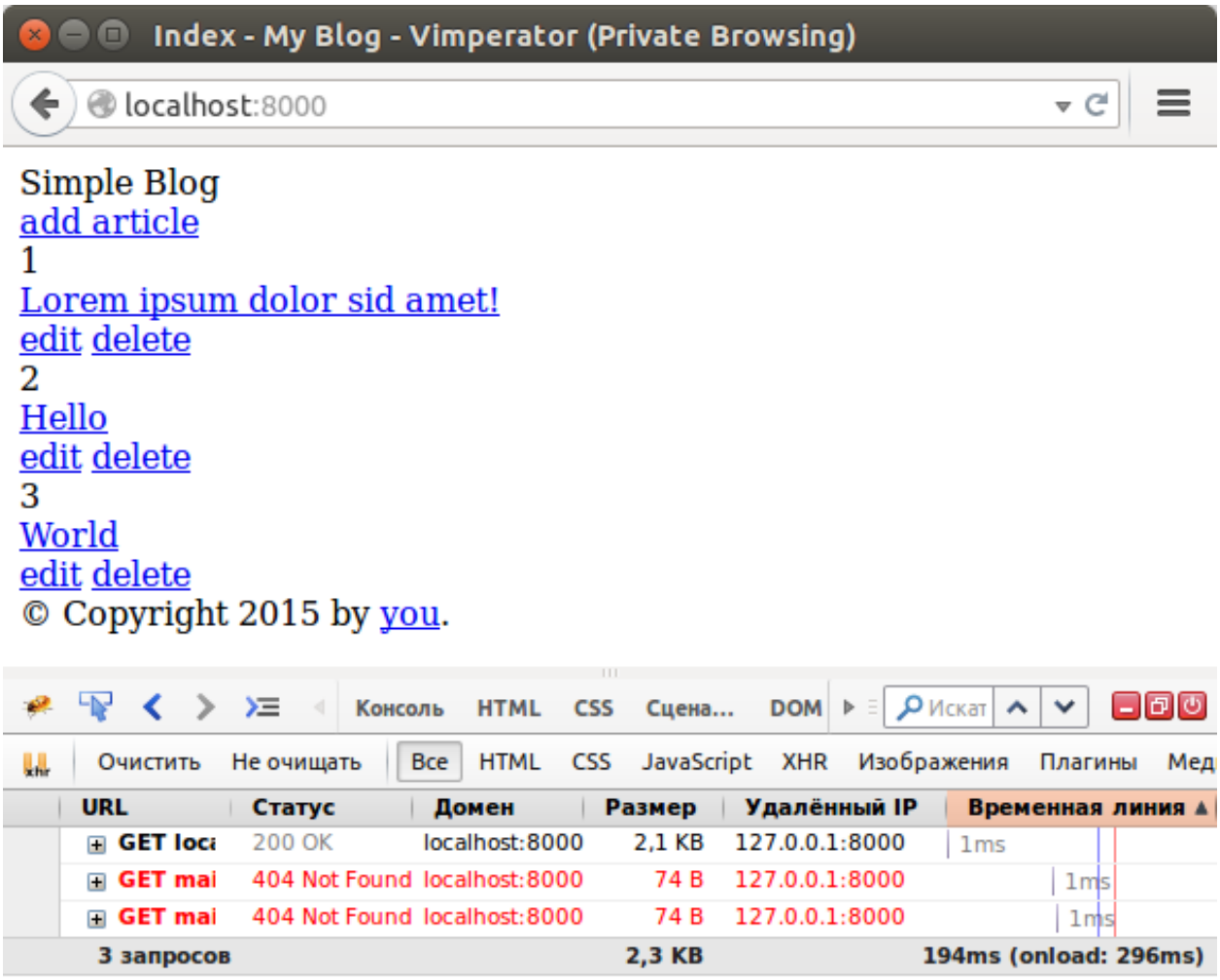


Рис. 25: Новые шаблоны блога без статики

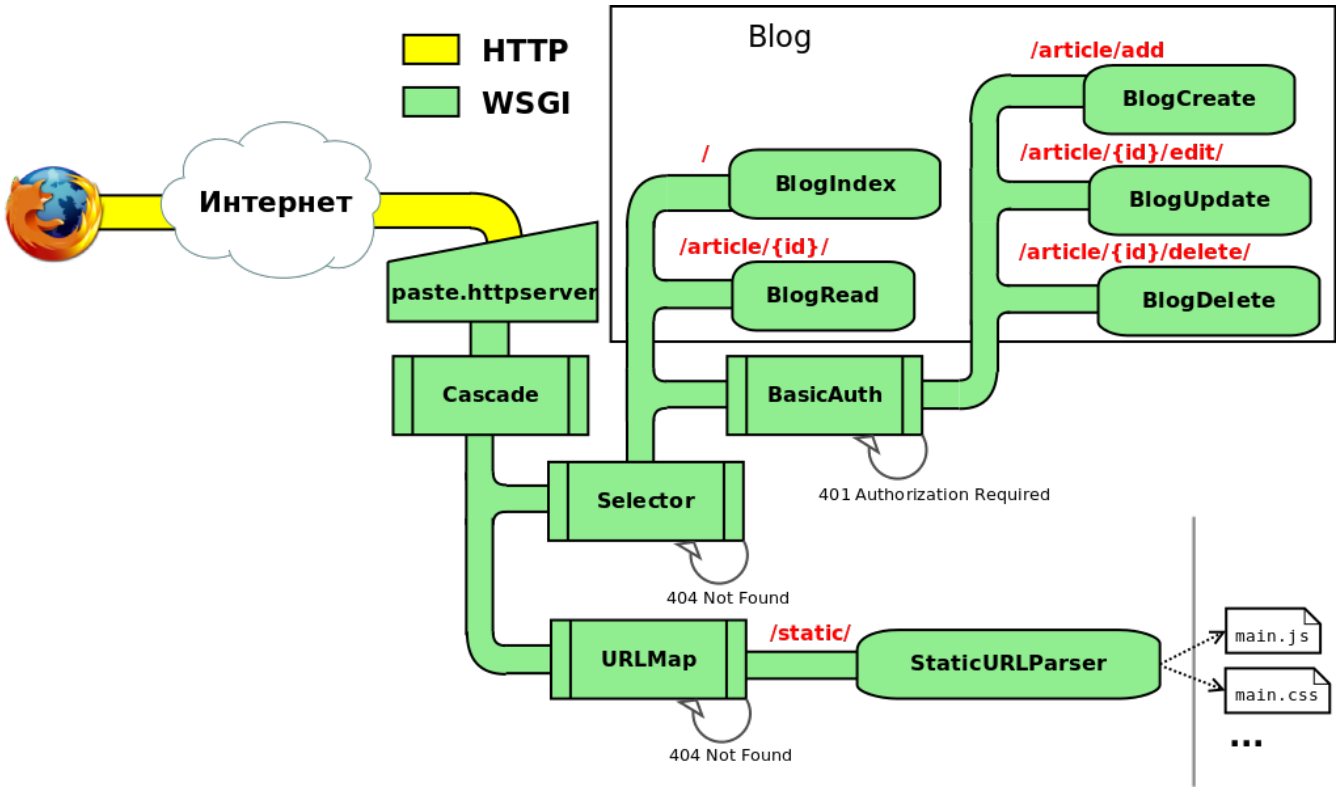


Рис. 26: Структура блога со статикой

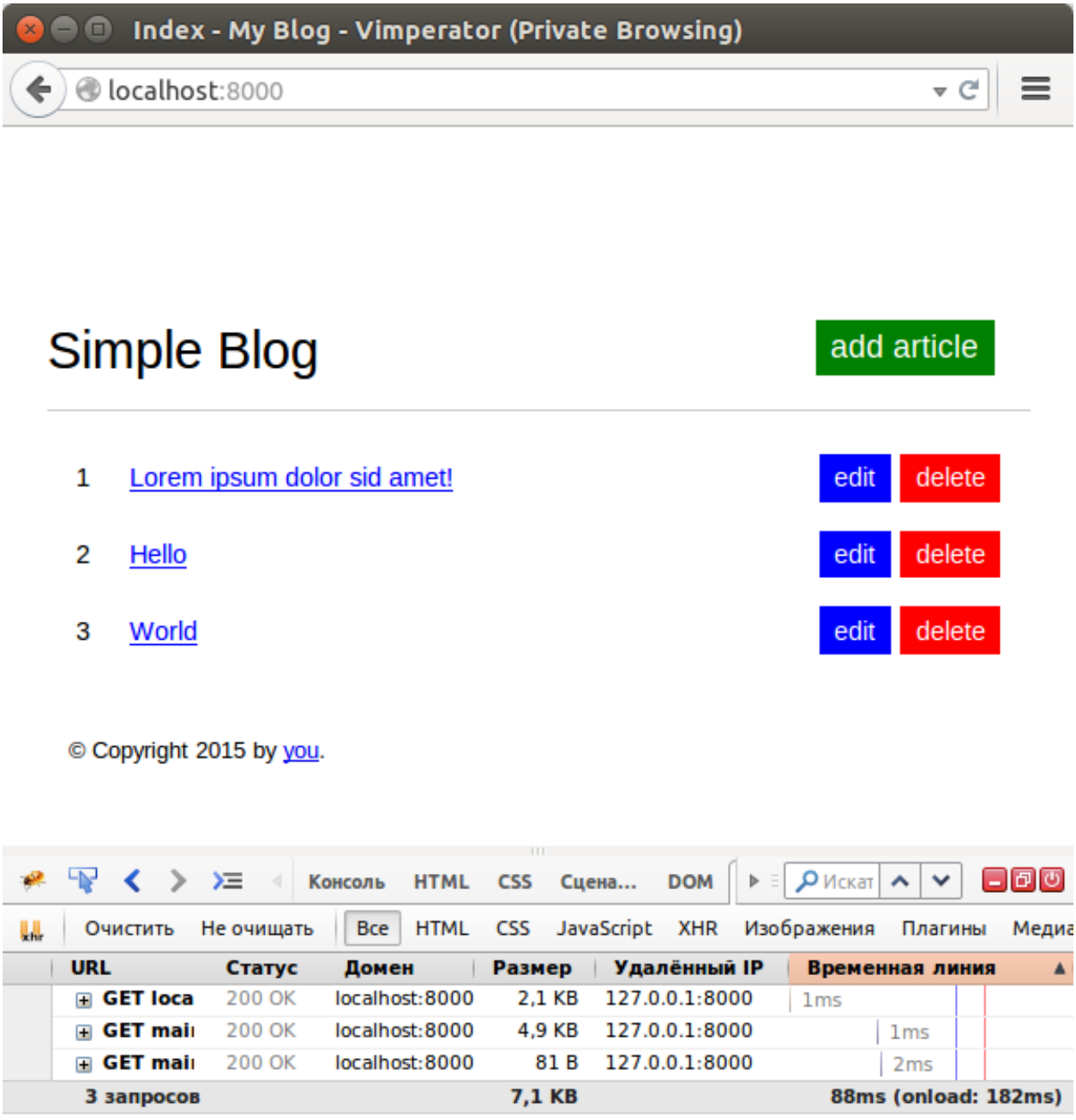


Рис. 27: Новые шаблоны блога со статикой

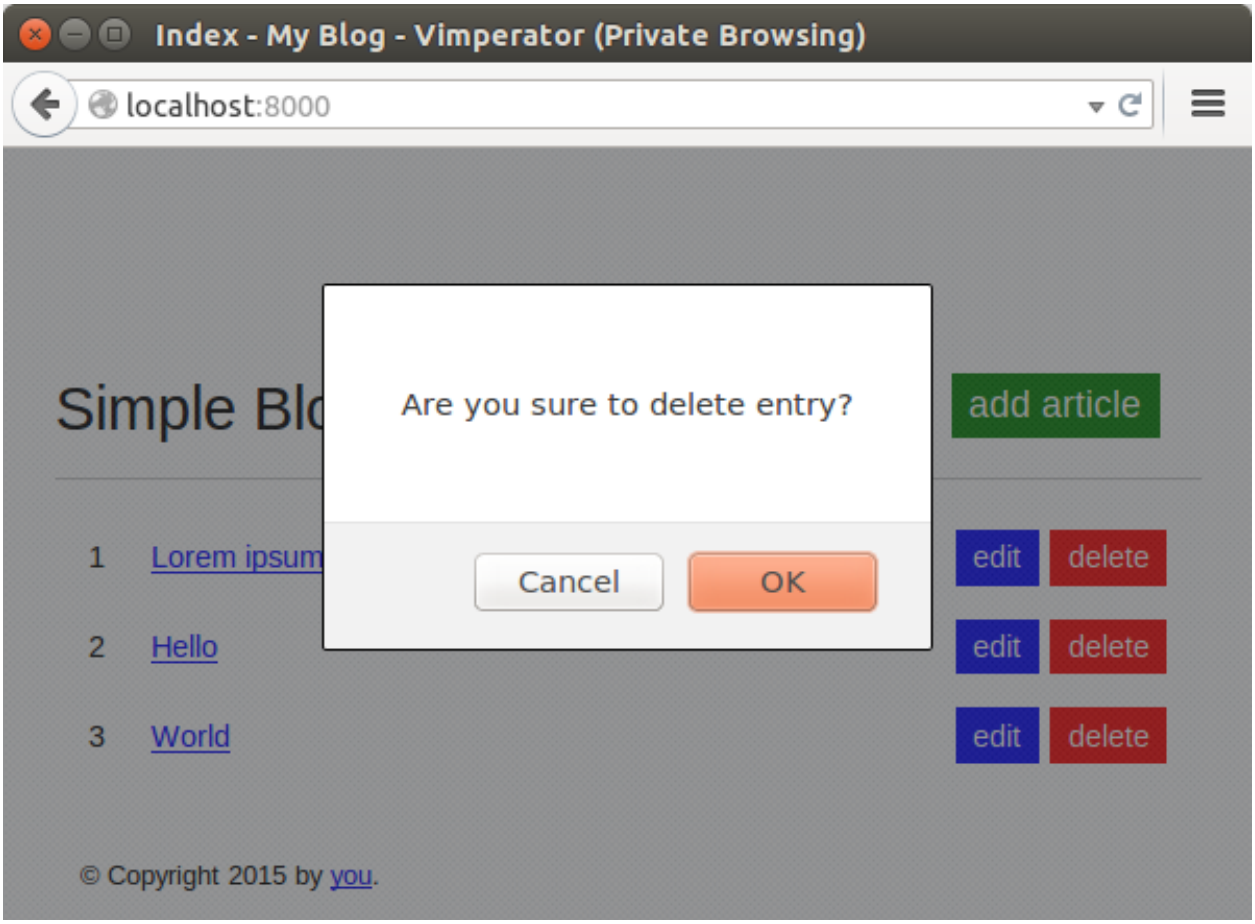


Рис. 28: Окно подтверждения при удалении статьи

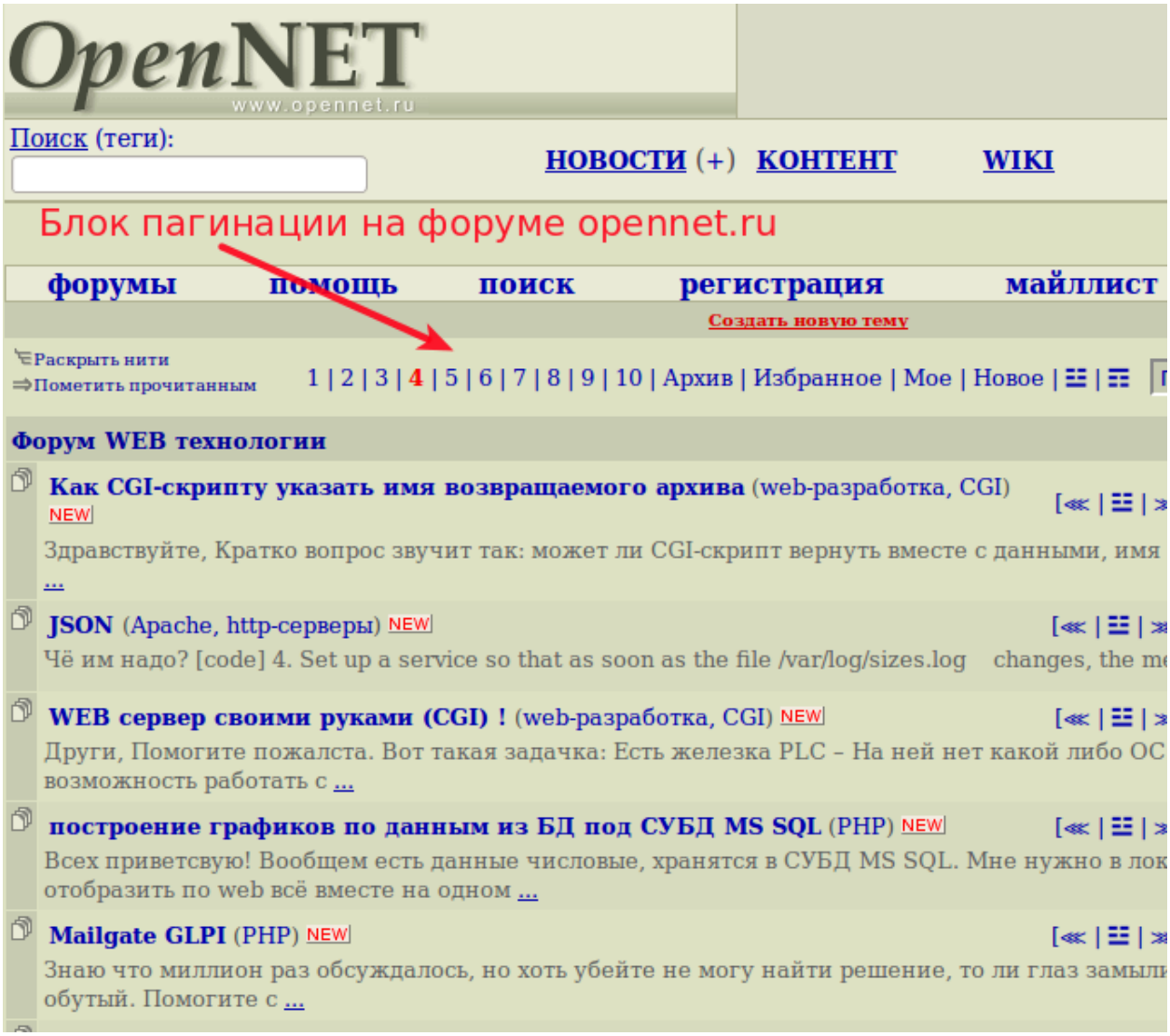


Рис. 29: OpenNET.ru

Рис. 30: Bootstrap4 pagination

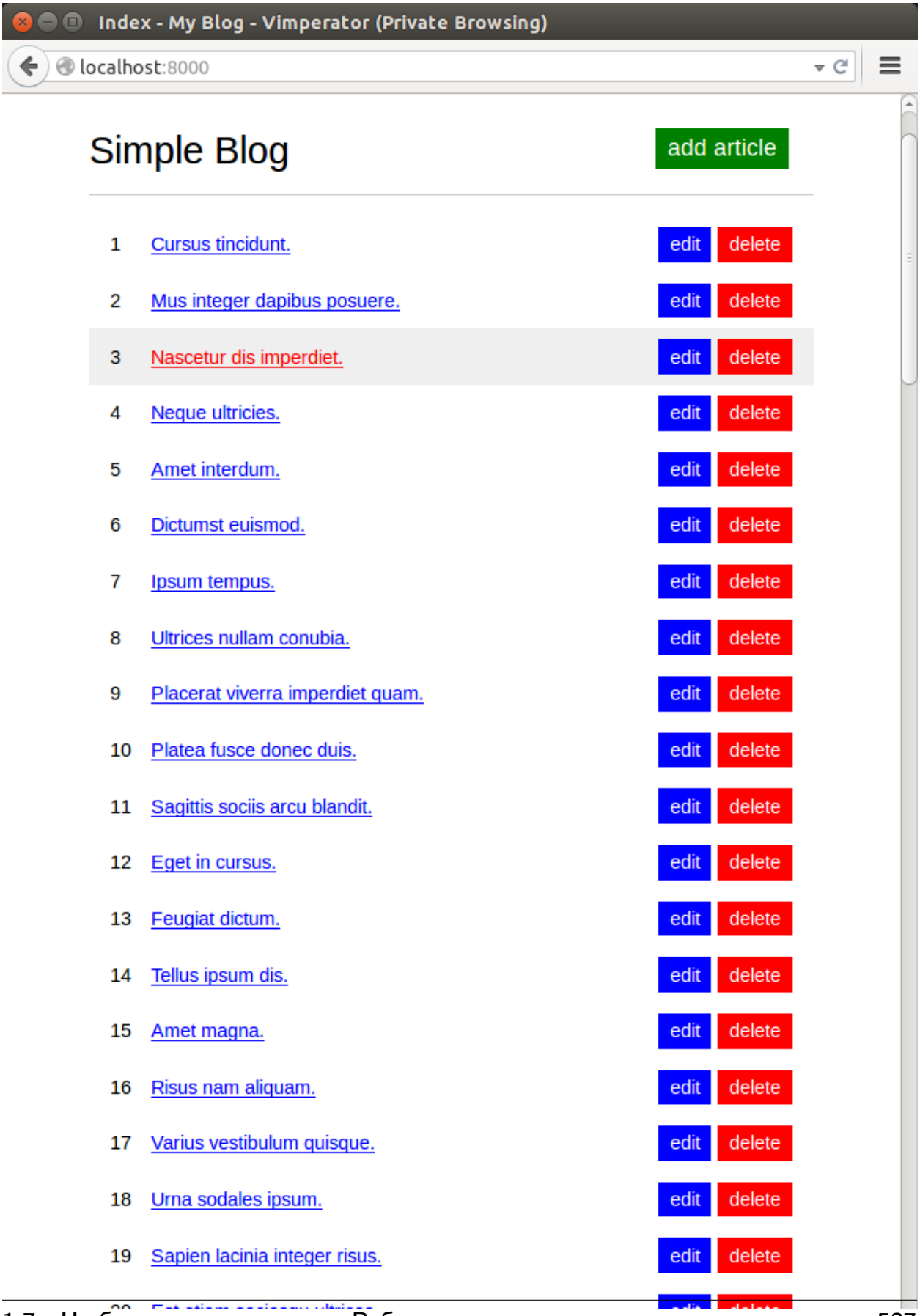


Рис. 31: Много статей не помещаются на экран

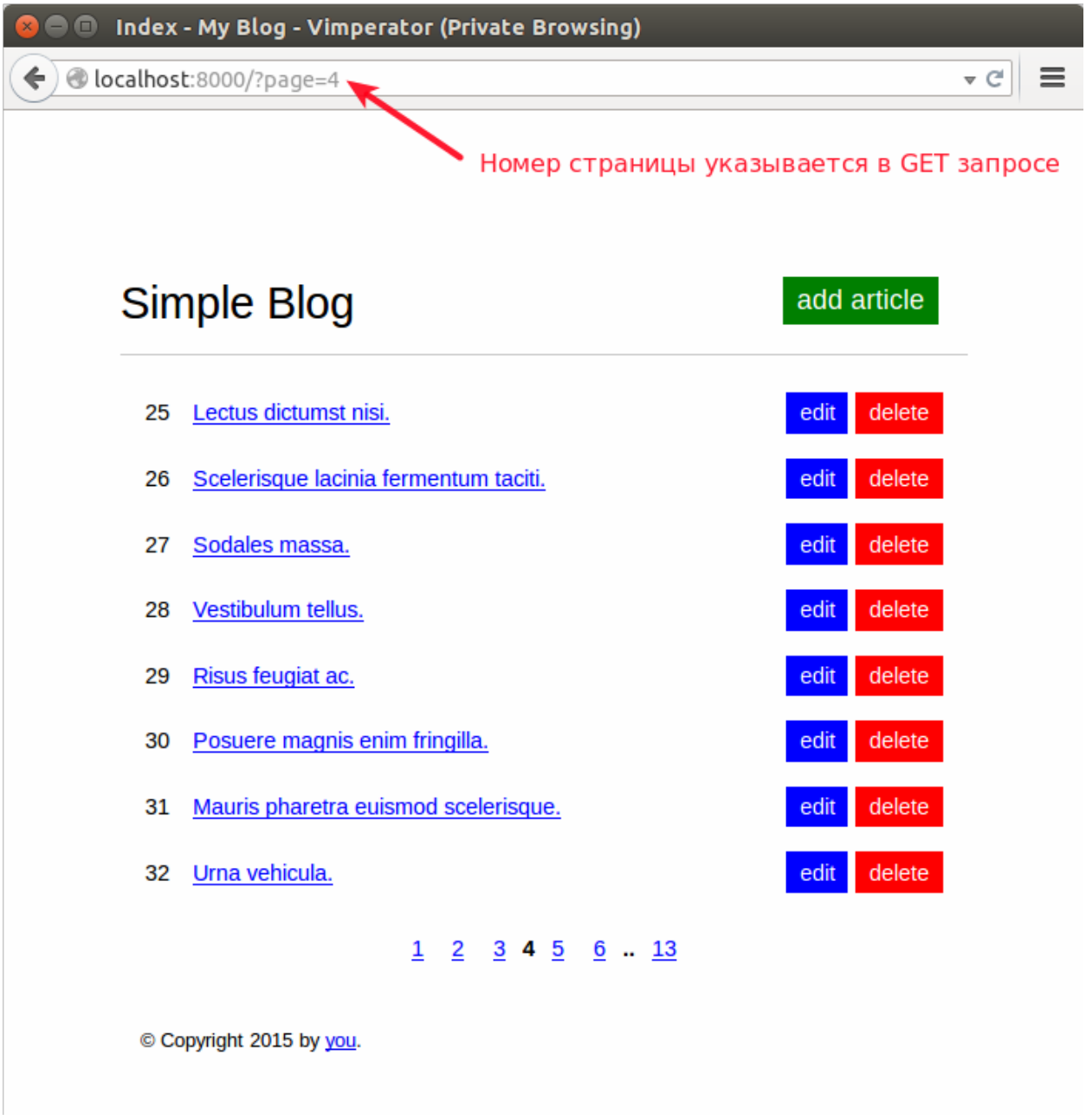


Рис. 32: Блог со страницами

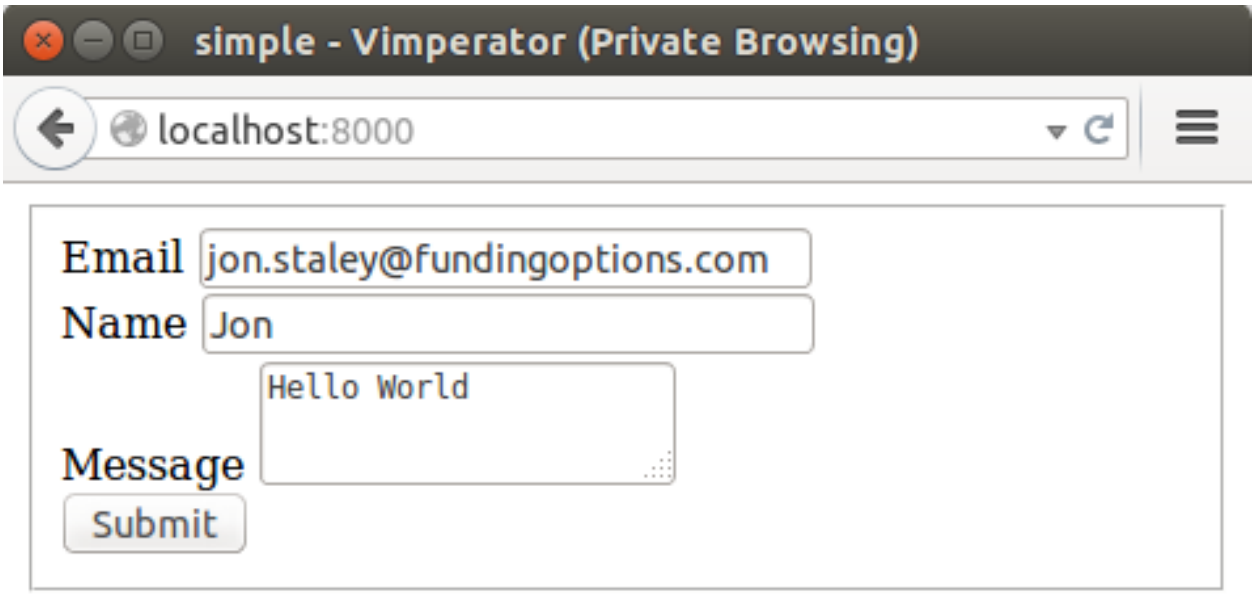


Рис. 33: Сгенерированная форма

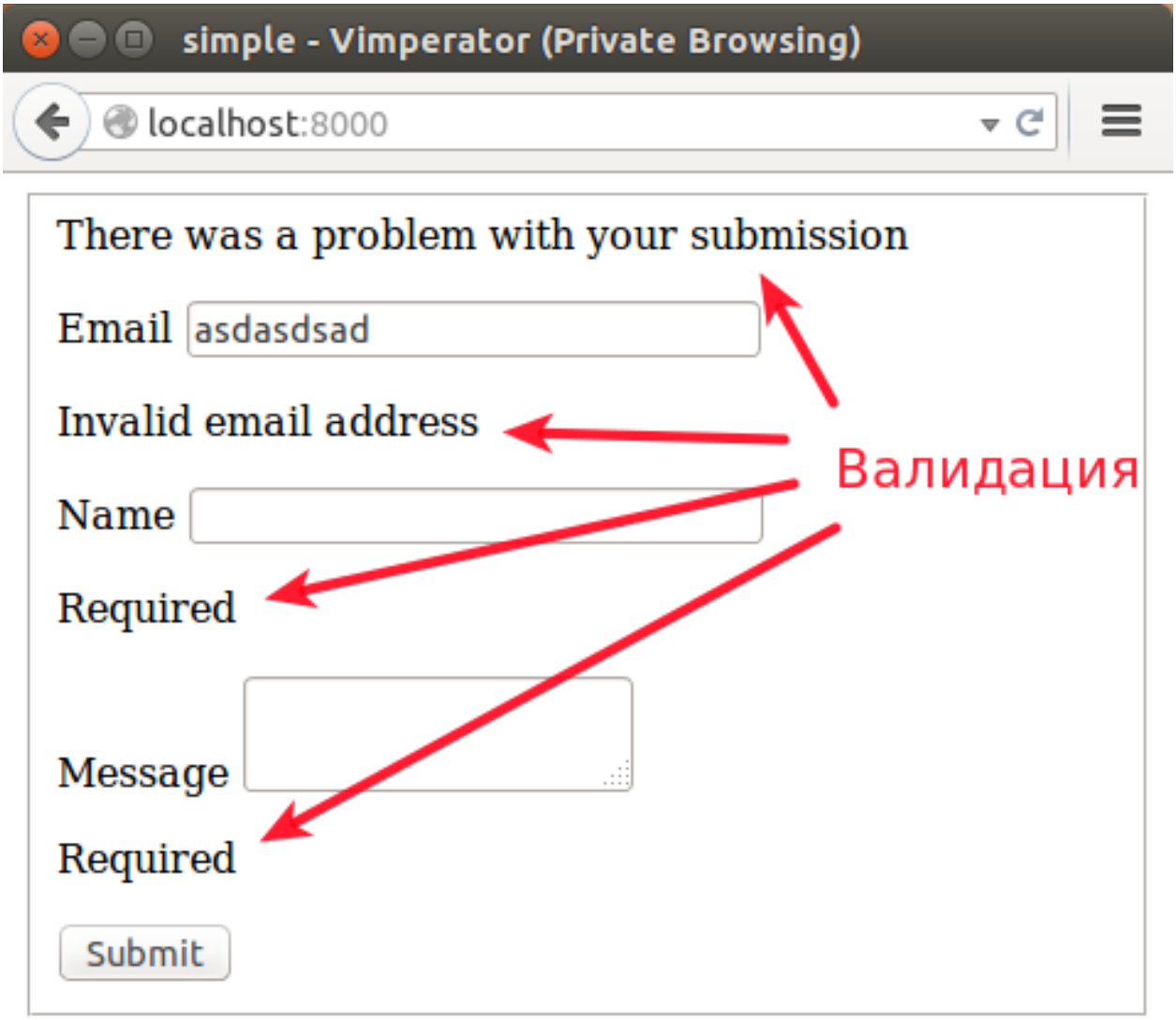


Рис. 34: Валидация формы

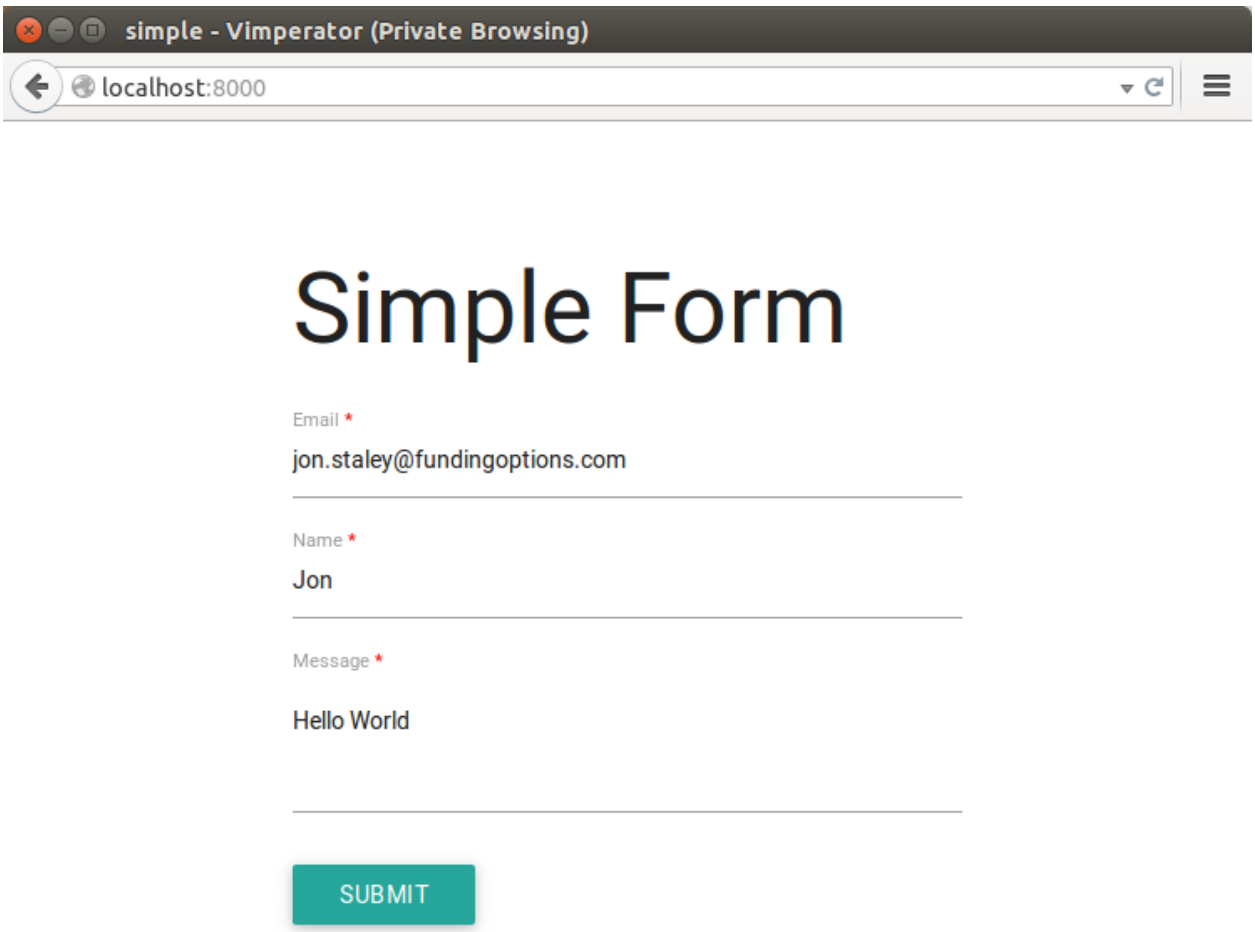


Рис. 35: Сгенерированная форма с применением CSS стилей

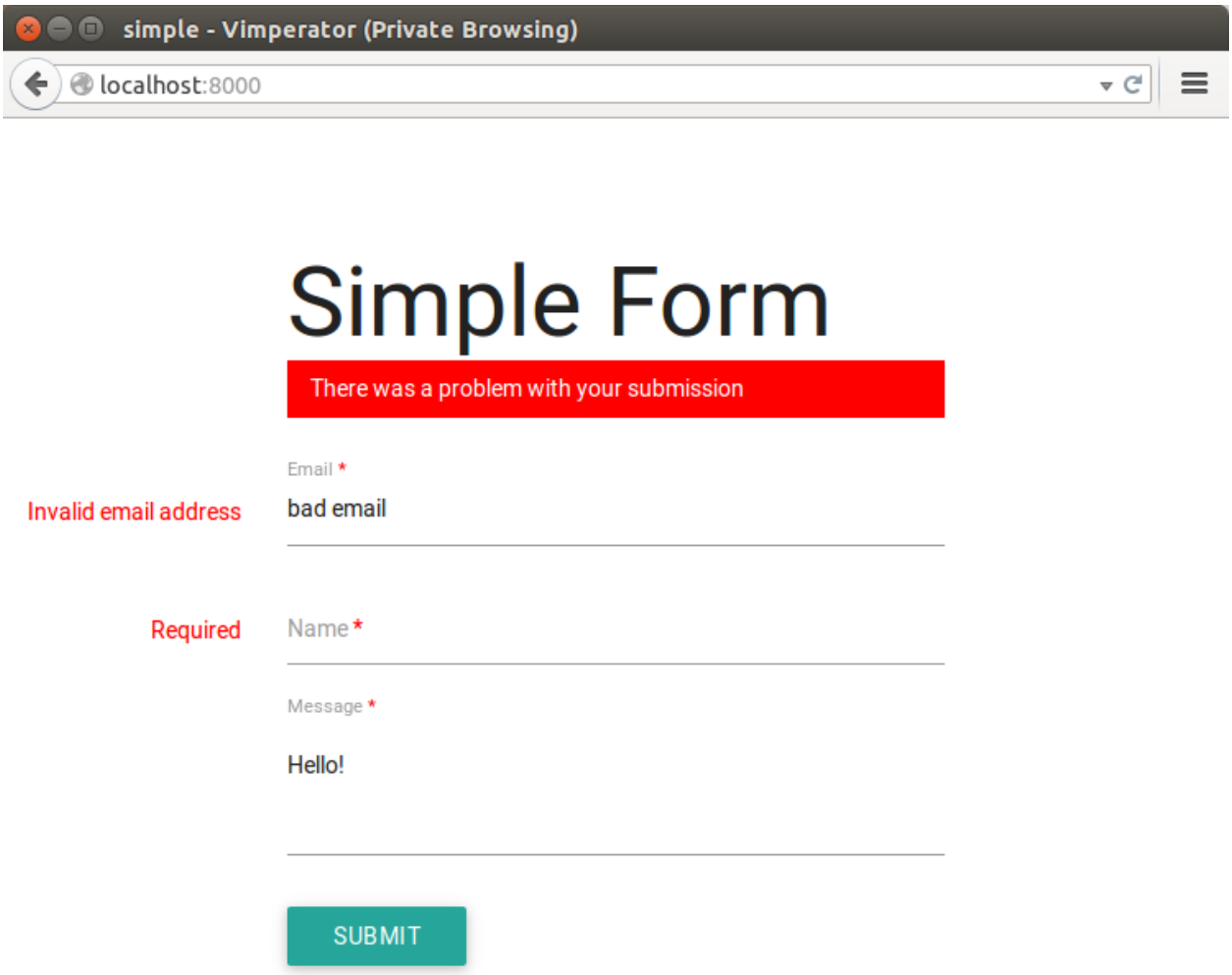


Рис. 36: Валидация формы с применением CSS стилей

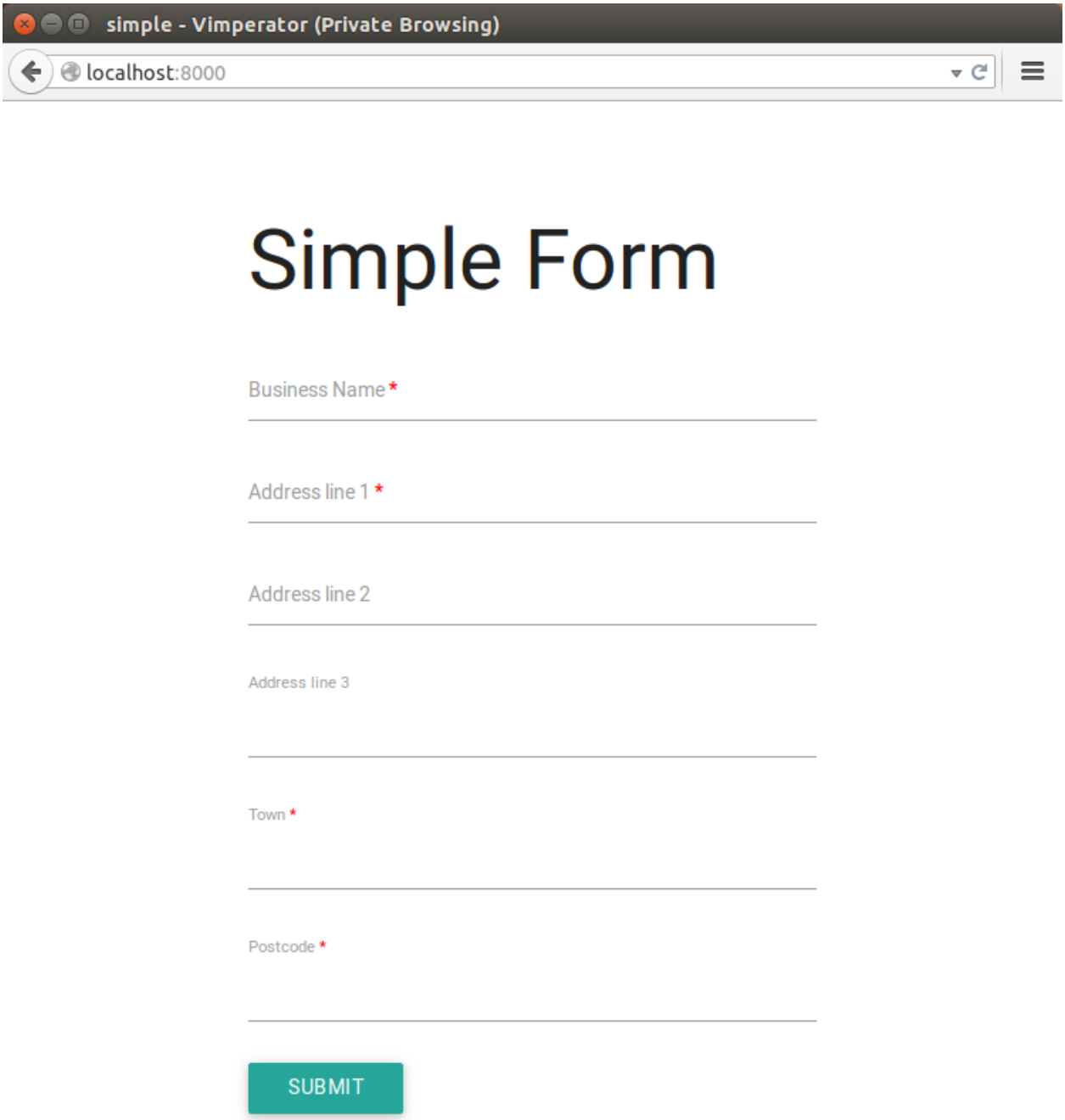


Рис. 37: Наследование Colander схемы

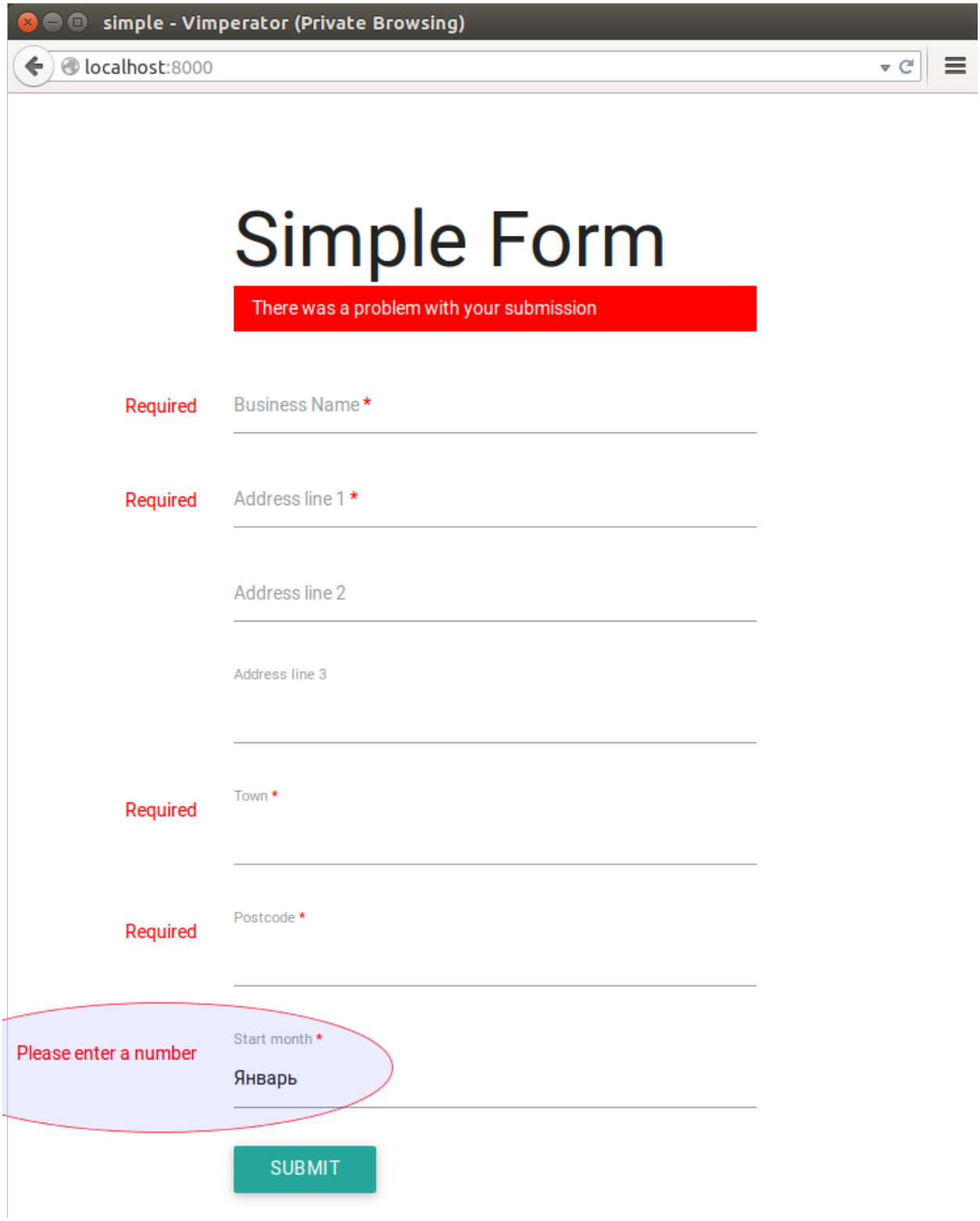


Рис. 38: Кастомная валидация поля

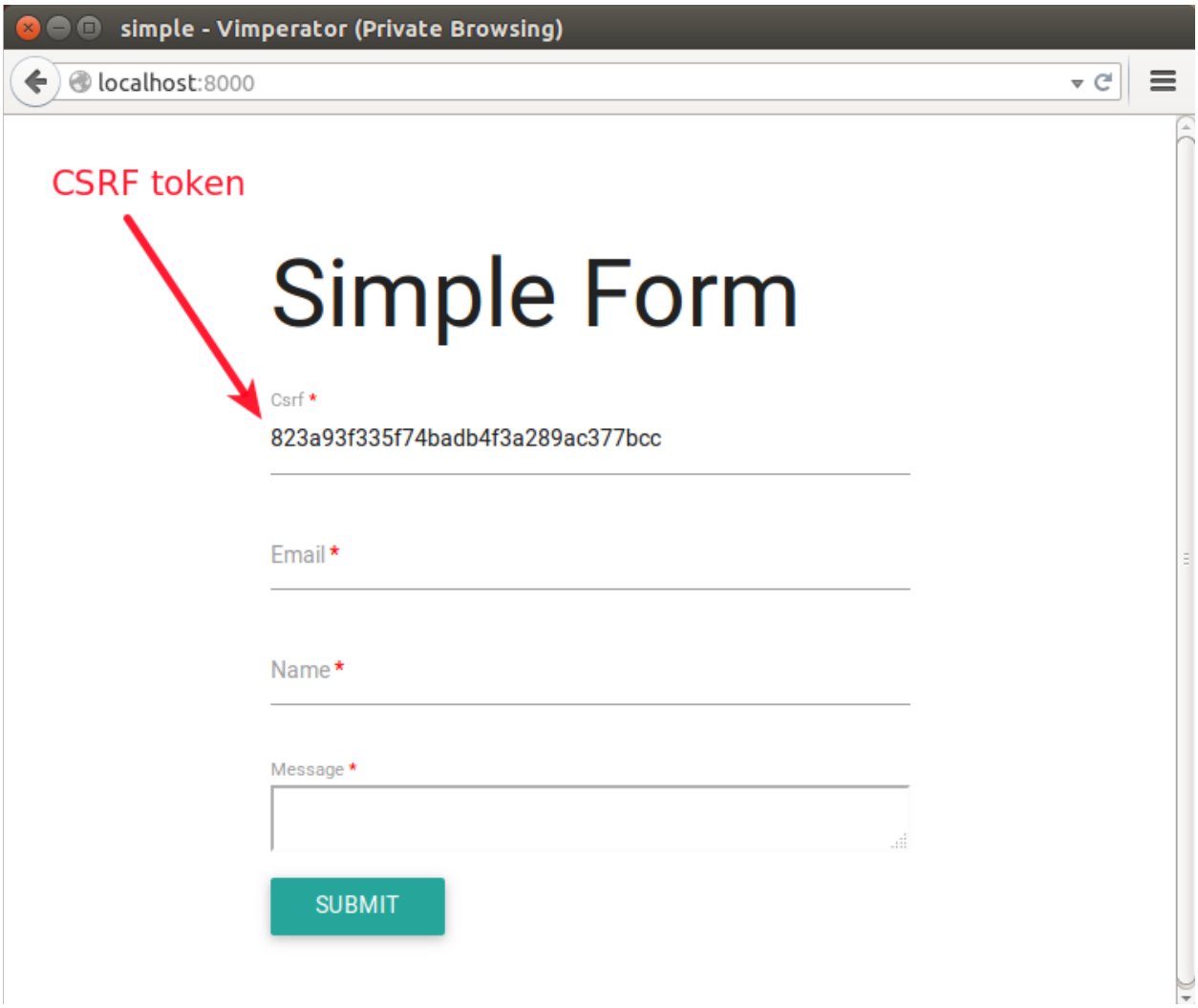


Рис. 39: Ключ CSRF

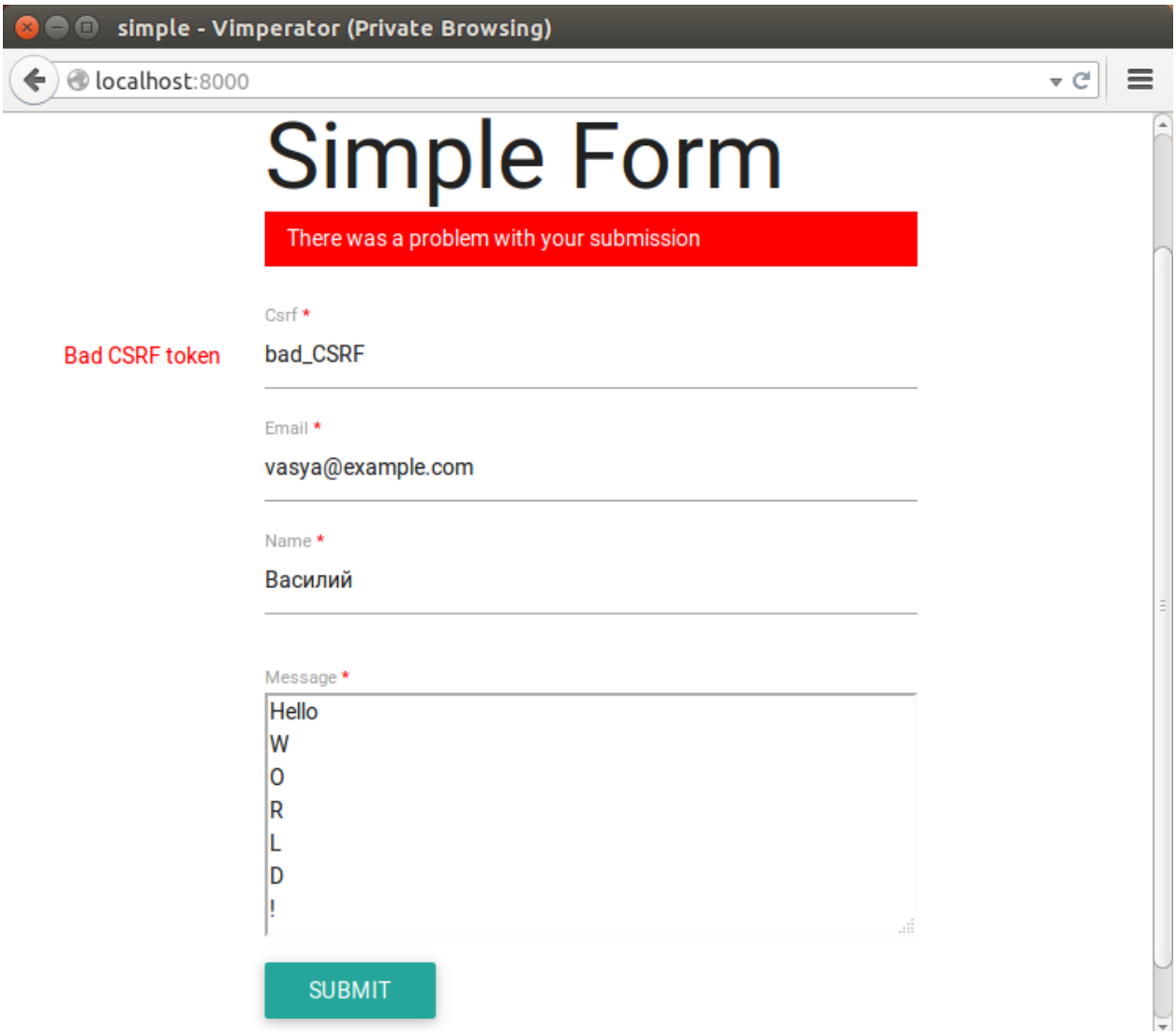


Рис. 40: Ключ CSRF

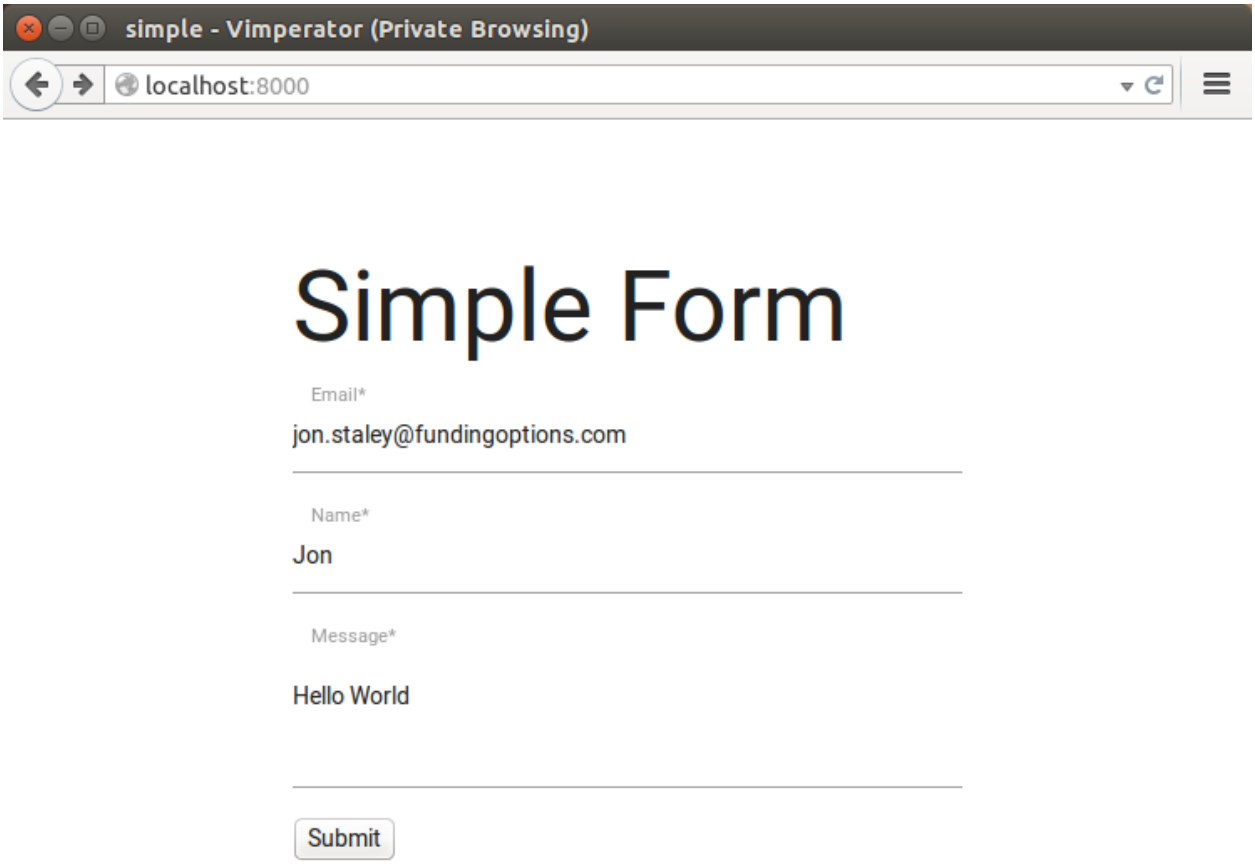


Рис. 41: Переопределенный шаблон form.pt

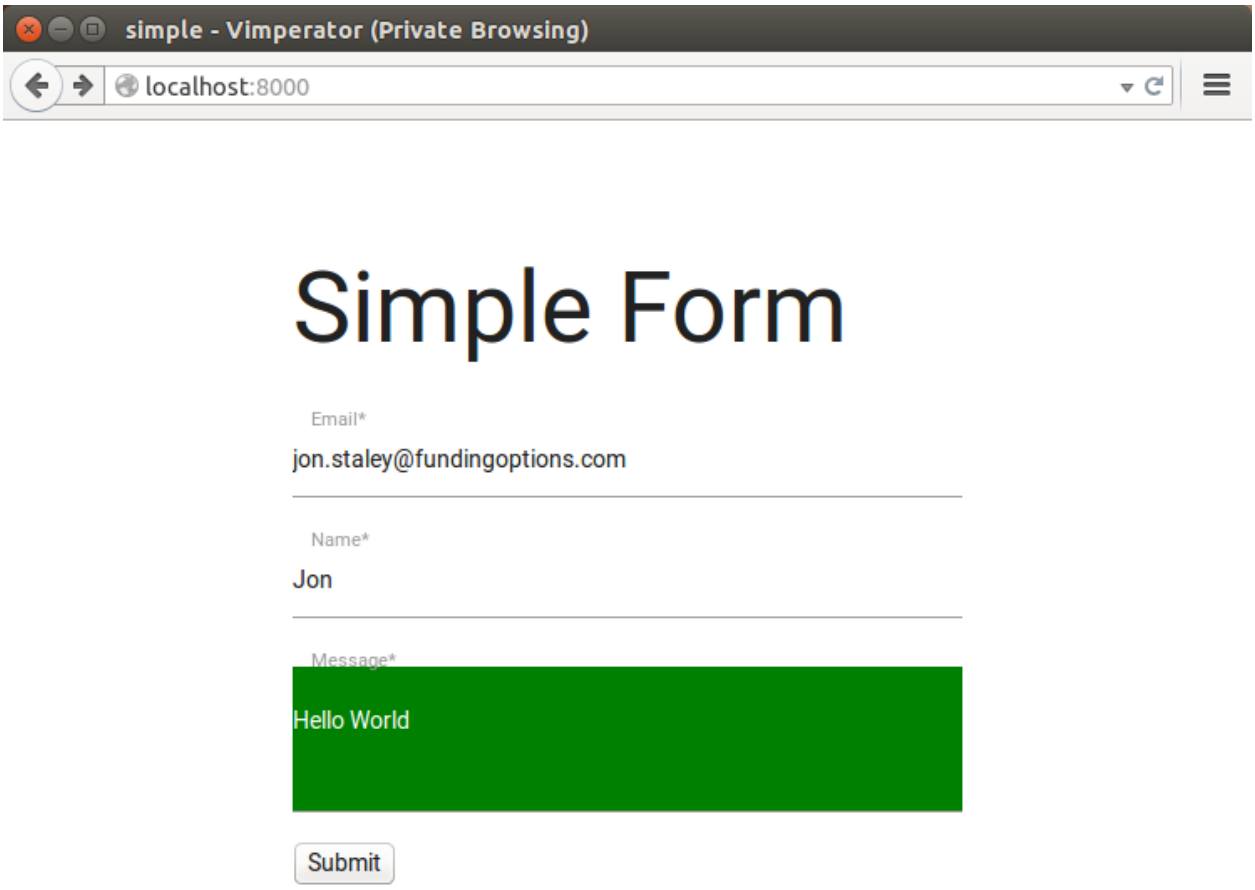


Рис. 42: Переопределенный шаблон textarea.jinja2

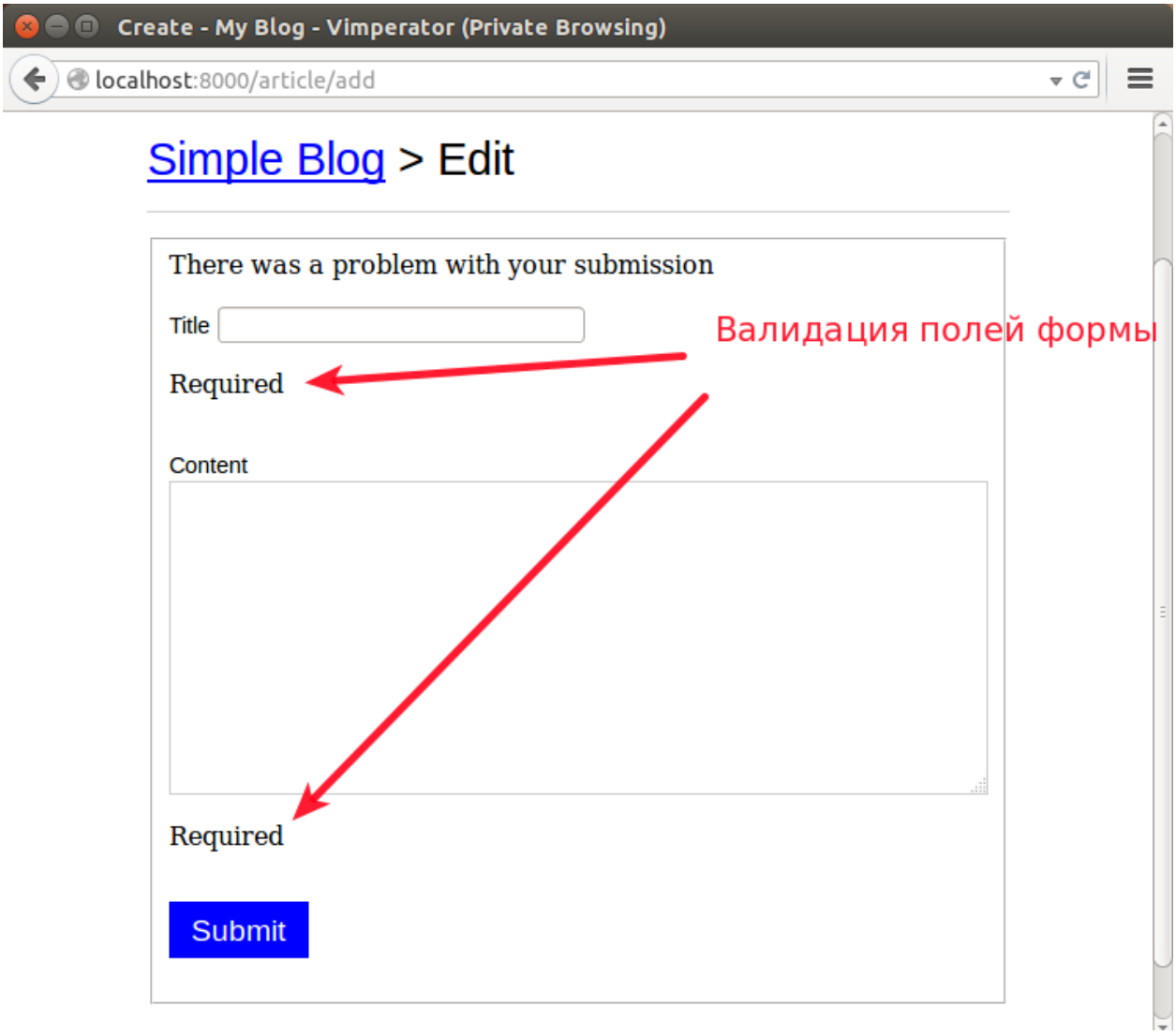


Рис. 43: Валидация формы

Глава 2

Закрепление материала

2.1 Закрепление материала «WSGI»

2.1.1 Цель работы

Получить практические навыки по работе со спецификацией `WSGI`.

2.1.2 Замечания к выполнению

Пример *WSGI middleware*, которое вставляет в тело *HTML*-страниц строки следующим образом:

```
<html>
<head>
   ...
</head>
<body>
   <div class='top'>Middleware TOP</div>

   ...

   <div class='botton'>Middleware BOTTOM</div>
</body>
</html>
```

Пример реализации:

```
from paste.httpserver import serve

TOP = "<div class='top'>Middleware TOP</div>"
BOTTOM = "<div class='botton'>Middleware BOTTOM</div>"


class WsgiTopBottomMiddleware(object):
    '''
    WSGI Middleware, которое добавляет TOP, BOTTOM в HTML документ
    '''

    def __init__(self, app):
        self.app = app

    def __call__(self, environ, start_response):
        response = self.app(environ, start_response).decode()  # bytes to str
        if response.find('<body>') > -1:
            header, body = response.split('<body>')
            data, htmlend = body.split('</body>')
            data = '<body>' + TOP + data + BOTTOM+'</body>'
            yield (header + data + htmlend).encode()  # str to bytes
        else:
            yield (TOP + response + BOTTOM).encode()  # str to bytes


def app(environ, start_response):
    '''
    WSGI приложение, которое отдает HTML документ
    '''
    response_code = '200 OK'
    response_type = ('Content-Type', 'text/HTML')
    start_response(response_code, [response_type])
    return '''
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>HTML Document</title>
   </head>
   <body>
      <p>
         <b>
            Этот текст будет полужирным,
            <i>а этот - ещё и курсивным</i>
         </b>
```

(continues on next page)

(продолжение с предыдущей страницы)

```
      </p>
   </body>
</html>
    '''.encode()  # str to bytes


# Оборачиваем WSGI приложение в middleware
app = WsgiTopBottomMiddleware(app)

# Запускаем сервер
serve(app, host='localhost', port=8000)
```

2.1.3 Задания

Описание заданий находится в разделе Работа с протоколом HTTP через telnet.

Задание 1

- Написать *WSGI* приложение, которое отдает статикой файлы *index.html* и *about.html*.
- Написать *WSGI middleware*, которое будет вставлять в *HTML* документ *JavaScript* и *CSS* файлы из списка типа:

```
includes = [
    'app.js',
    'react.js',
    'leaflet.js',
    'D3.js',
    'moment.js',
    'math.js',
    'main.css',
    'bootstrap.css',
    'normalize.css',
]
```

Следующим образом:

```
<html>
<head>

   ...
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    <link rel="stylesheet" href="/_static/main.css"/>
    <link rel="stylesheet" href="/_static/bootstrap.css"/>
    <link rel="stylesheet" href="/_static/normalize.css"/>
</head>
<body>

    ...

    <script src="/_static/app.js"></script>
    <script src="/_static/react.js"></script>
    <script src="/_static/leaflet.js"></script>
    <script src="/_static/D3.js"></script>
    <script src="/_static/moment.js"></script>
    <script src="/_static/math.js"></script>
</body>
</html>
```

Задание 2, 3, 4

Делать не надо.

2.1.4 Содержание отчета

На каждое задание создать отчет, который должен быть оформлен в виде репозитария на GitHub. В отчете должно быть: исходный код программы, описание последовательности действий, результат выполнения заданий и выводы по работе.

2.2 Закрепление материала «Web»

2.2.1 Цель работы

Изучить возможности шаблонизаторов на языке программирования *Python*. Получить практические навыки по обработке *HTTP* запросов/ответов при помощи библиотеки *WebOb*.

2.2.2 Замечания к выполнению

Пример создания *HTTP* запроса при помощи библиотеки *WebOb*.

Код 1: 9.send_request.py

```
from webob import Request

req = Request.blank('http://en.wikipedia.org/wiki/HTTP')

from pprint import pprint
pprint(req)
print
print(req.get_response())
```

2.2.3 Задания

Задание 1

- Переписать первое задание из *Закрепление материала «WSGI»*, используя любой шаблонизатор на языке программирования *Python* (например Jinja или Mako) для *HTML* файлов.
- Файлы `aboutme.html` и `index.html` должны наследоваться от `base.html`.
- *WSGI* приложение должно выводить результат запроса при помощи библиотеки *WebOb* (`get_response()`).

Задание 2,3,4

- Сформировать HTTP запросы задания 2,3,4 Работа с протоколом HTTP через telnet, при помощи WebOb.

2.2.4 Содержание отчета

На каждое задание создать отчет, который должен быть оформлен в виде репозитария на GitHub или Gist заметок. В отчете должно быть: исходный код программы, описание последовательности действий, результат выполнения заданий и выводы по работе.

2.3 Закрепление материала «SQL»

2.3.1 Цель работы

Научиться работать с БД используя *ORM SQLAlchemy*.

2.3.2 Задания

Задание 1

Выполнить упражнения из презентации https://bitbucket.org/zzzeek/pycon2013_student_package.

2.3.3 Содержание отчета

На каждое задание создать отчет, который должен быть оформлен в виде репозитария на GitHub или Gist заметок. В отчете должно быть: исходный код программы, описание последовательности действий, результат выполнения заданий и выводы по работе.

2.4 Закрепление материала «Pyramid»

2.4.1 Цель работы

Ознакомиться с фреймворком *Pyramid*.

2.4.2 Задания

Задание 1

Переписать *WSGI* приложение первого задания из *Закрепление материала «Web»*, используя фреймворк Pyramid.

Задание 2, 3, 4

Делать не надо.

2.4.3 Содержание отчета

Создать отчет, который должен быть оформлен в виде репозитария на GitHub. В отчете должно быть: исходный код программы, описание последовательности действий, результат выполнения заданий и выводы по работе.

Глава 3

Слайды

3.1 Презентации

3.1.1 HTTP протокол

3.1.2 Анализ трафика

3.1.3 Сокеты

3.1.4 Веб сервер

3.1.5 WSGI

3.1.6 Разработка без фреймворков

3.1.7 Базы данных

DB-API

SQLAlchemy

Глава 4

Справочник

4.1 Python

См.также:

Полезные ссылки http://wiht.co/python-guide

4.1.1 Установка Python

Установка Python в ОС Linux

Сборка из исходников (UNIX)

Скачиваем

Примечание: В оф. документации предлагают скачать ртутью с фирменного сайта:

```
$ hg clone https://hg.python.org/cpython
$ hg update 3.5
```

Скачиваем с гитхаба python/cpython:

```
git clone https://github.com/python/cpython.git
```

Выбираем ветку `3.5` (*cpython* версии 3.5):

```
git checkout 3.5
```

Собираем

Укажем локальную директорию для сборки:

```
./configure --prefix=$HOME/Projects/bin/python3.5
```

Скомпилируем:

```
make && make install
```

Теперь можно запускать:

```
$ $HOME/Projects/bin/python3.5/bin/python3
Python 3.5.0+ (default, Oct 10 2015, 13:35:25)
[GCC 4.9.2] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

```
>>> {*range(4), 4, *(5, 6, 7)}
{0, 1, 2, 3, 4, 5, 6, 7}
>>> import asyncio
>>> async def foo(bar): await asyncio.sleep(42)
```

virtualenv

Укажем виртуальному окружению где находится интерпретатор *cpython*:

```
$ mkvirtualenv --python=$HOME/Projects/bin/python3.5/bin/python3 python35_env
Running virtualenv with interpreter /home/uralbash/Projects/bin/python3.5/bin/
↪python3
Using base prefix '/home/uralbash/Projects/bin/python3.5'
New python executable in aiohttp/bin/python3
Also creating executable in aiohttp/bin/python
Installing setuptools, pip, wheel...done.
```

Linux

Установка интерпретатора CPython

```
$ sudo apt-get install python
```

Пакетный менеджер pip

```
$ sudo apt-get install python-setuptools python-dev build-essential
$ sudo easy_install pip
```

Виртуальное окружение Virtualenv

```
$ sudo pip install virtualenv virtualenvwrapper
$ source /usr/local/bin/virtualenvwrapper.sh
```

Компиляция пакетов

Некоторые Python пакеты написаны с использование языка программирования Си, поэтому при установке они требуют компиляции. Если у вас не установлен компилятор, пакет не будет установлен.

```
$ sudo apt-get install gcc python-dev
```

Установка git

```
$ sudo apt-get intall git
```

Пример

Склонируем репозитарий админки https://github.com/sacrud/pyramid_sacrud.git в директорию `/home/user/Projects`.

```
$ cd /home/user/Projects/
$ git clone https://github.com/sacrud/pyramid_sacrud.git
```

Установим pyramid_sacrud из исходных кодов.

```
$ cd /home/user/Projects/pyramid_sacrud
$ mkvirtualenv pyramid_sacrud
$ python setup.py develop
```

Далее установим пример `pyramid_sacrud/example`

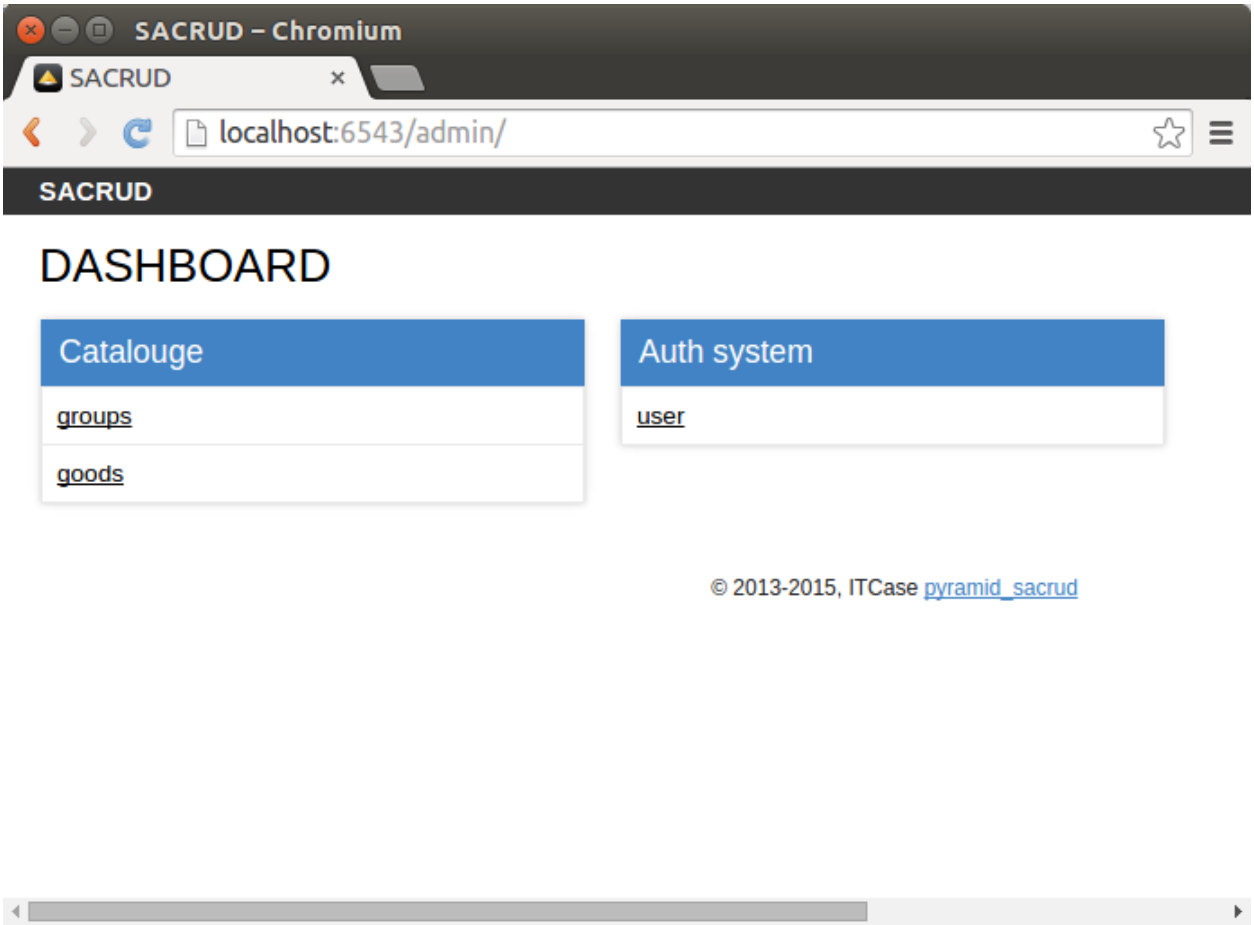
```
$ cd /home/user/Projects/pyramid_sacrud/example
$ workon pyramid_sacrud
$ python setup.py develop
```

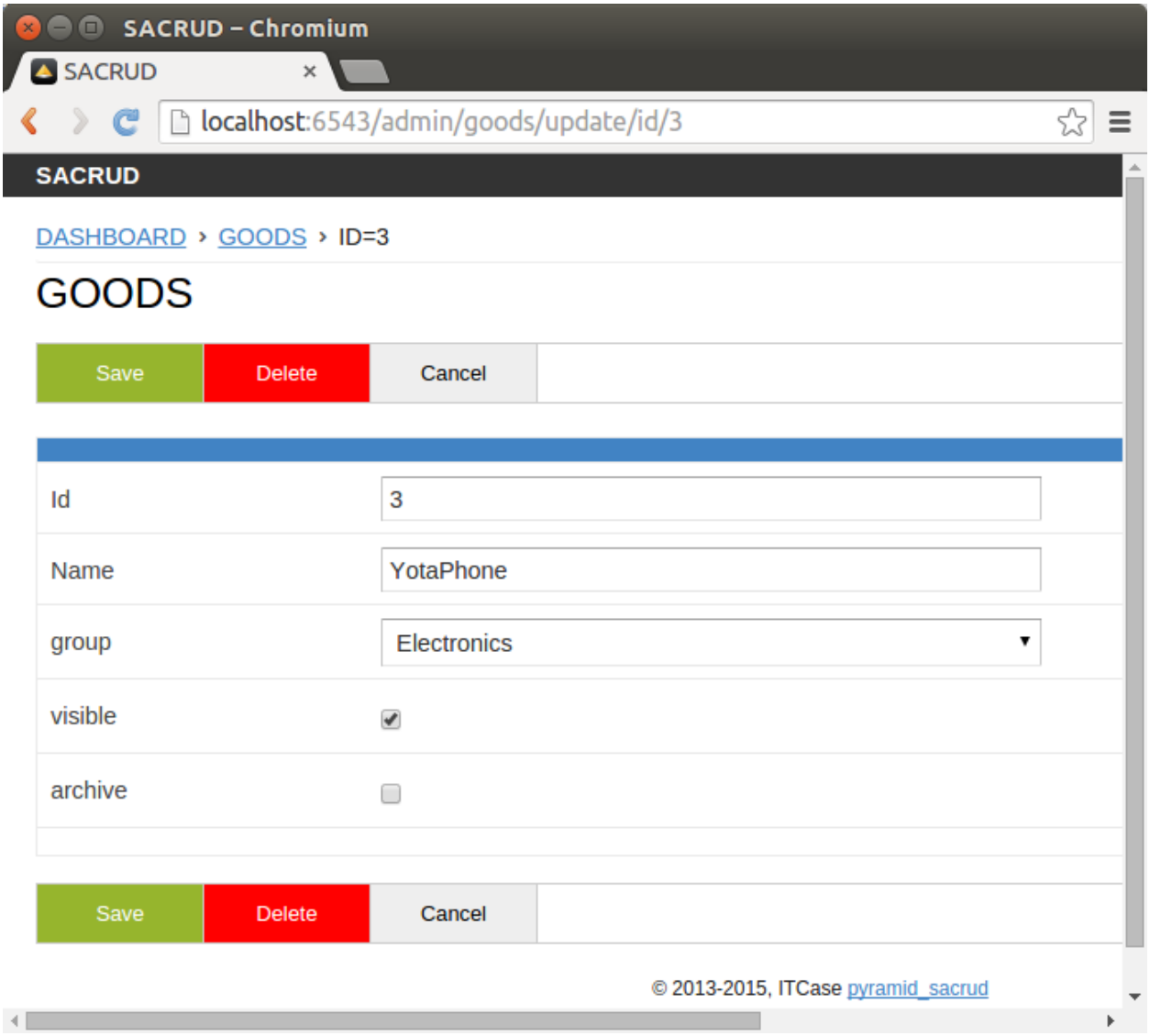
Пакеты устанавливаются в виртуальное окружение с названием `pyramid_sacrud`.

Теперь можно запустить пример:

```
$ cd /home/user/Projects/pyramid_sacrud/example
$ workon pyramid_sacrud
$ pserve development.ini
```

Заходим на http://localhost:6543/admin/





Установка Python в ОС MacOS

Homebrew

Homebrew является очень удобным пакетным менеджером для MacOS. Все дальнейшие манипуляции по установке пакетов будут осуществлены с его использованием (где это возможно, конечно).

Установка

```
$ ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
↪master/install)"
```

Установка интерпретатора CPython

```
$ brew install python
```

Пакетный менеджер pip

При использовании Homebrew для установки python'а pip поставится автоматически.

Виртуальное окружение Virtualenv

```
$ sudo pip install virtualenv virtualenvwrapper
$ source /usr/local/bin/virtualenvwrapper.sh
```

Компиляция пакетов

Некоторые Python пакеты написаны с использование языка программирования Си, поэтому при установке они требуют компиляции. Если у вас не установлен компилятор, пакет не будет установлен.

```
$ brew install gcc
```

Для успешной установки GCC необходимо наличие установленного XCode в системе.

Примечание: Для старых версий MacOS необходимо установить старую же версию XCode с диска, который поставляется вместе с Вашей операционной системой.

Установка git

```
$ brew intall git
```

Пример

Склонируем репозитарий админки https://github.com/sacrud/pyramid_sacrud.git в директорию `/home/user/Projects`.

```
$ cd /home/user/Projects/
$ git clone https://github.com/sacrud/pyramid_sacrud.git
```

Установим pyramid_sacrud из исходных кодов.

```
$ cd /home/user/Projects/pyramid_sacrud
$ mkvirtualenv pyramid_sacrud
$ python setup.py develop
```

Далее установим пример `pyramid_sacrud/example`

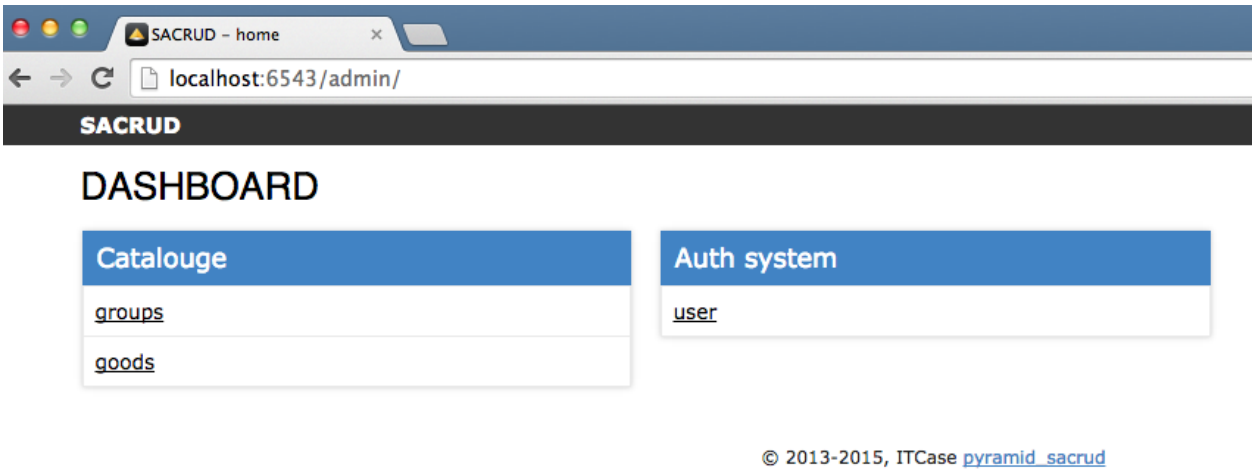
```
$ cd /home/user/Projects/pyramid_sacrud/example
$ workon pyramid_sacrud
$ python setup.py develop
```

Пакеты устанавливаются в виртуальное окружение с названием `pyramid_sacrud`.

Теперь можно запустить пример:

```
$ cd /home/user/Projects/pyramid_sacrud/example
$ workon pyramid_sacrud
$ pserve development.ini
```

Заходим на http://localhost:6543/admin/

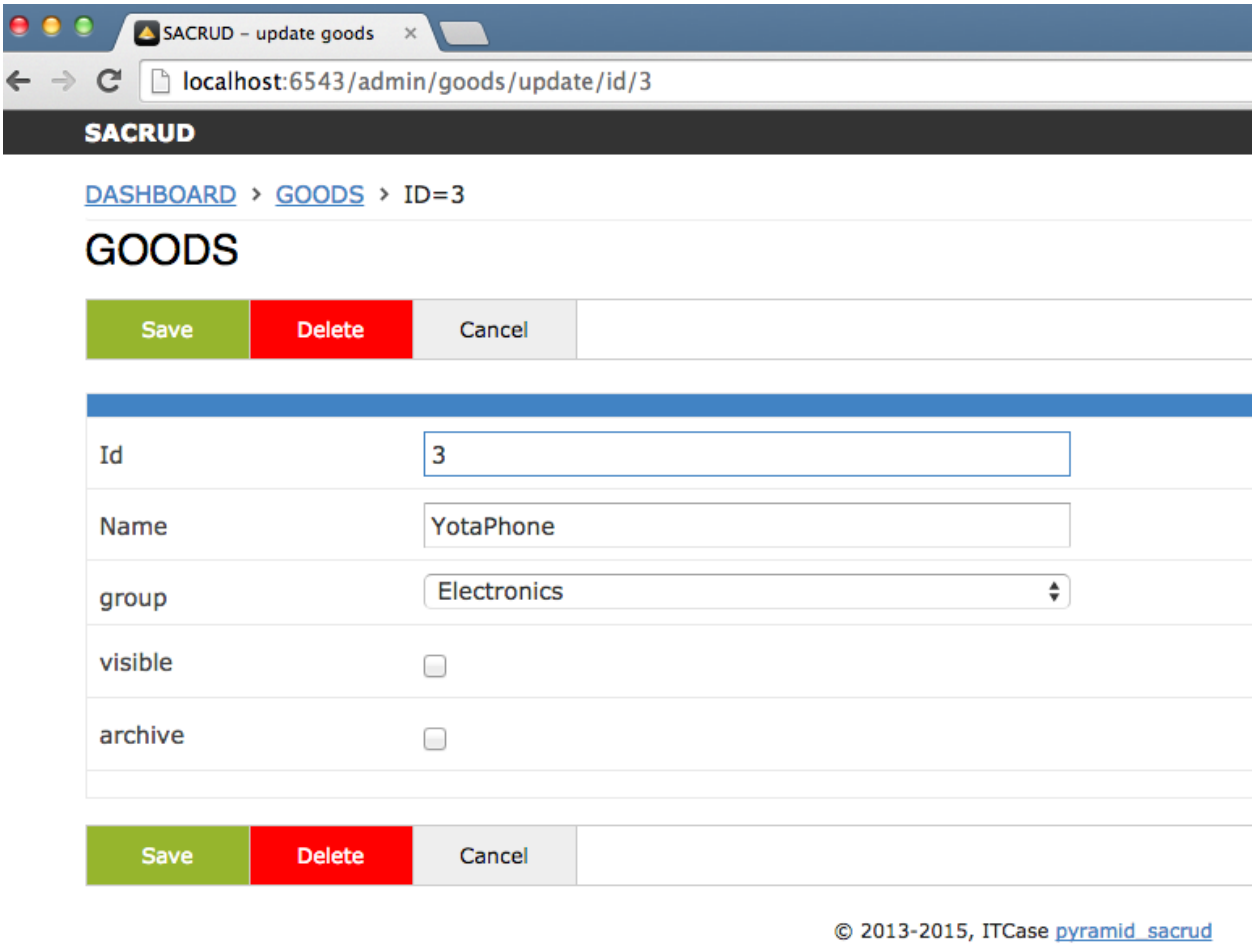


Установка Python в ОС Windows

Установка интерпретатора CPython

Все версии CPython можно найти по адресу https://www.python.org/downloads/

Выберем, например, версию 2.7.10 для 32 битной операционной системы.



Запускаем инсталятор:

По умолчанию Python устанавливается в директорию `C:\Python27\`.

Выбираем опцию «добавить python.exe в окружение».

Теперь интерпретатор Python доступен из консоли.

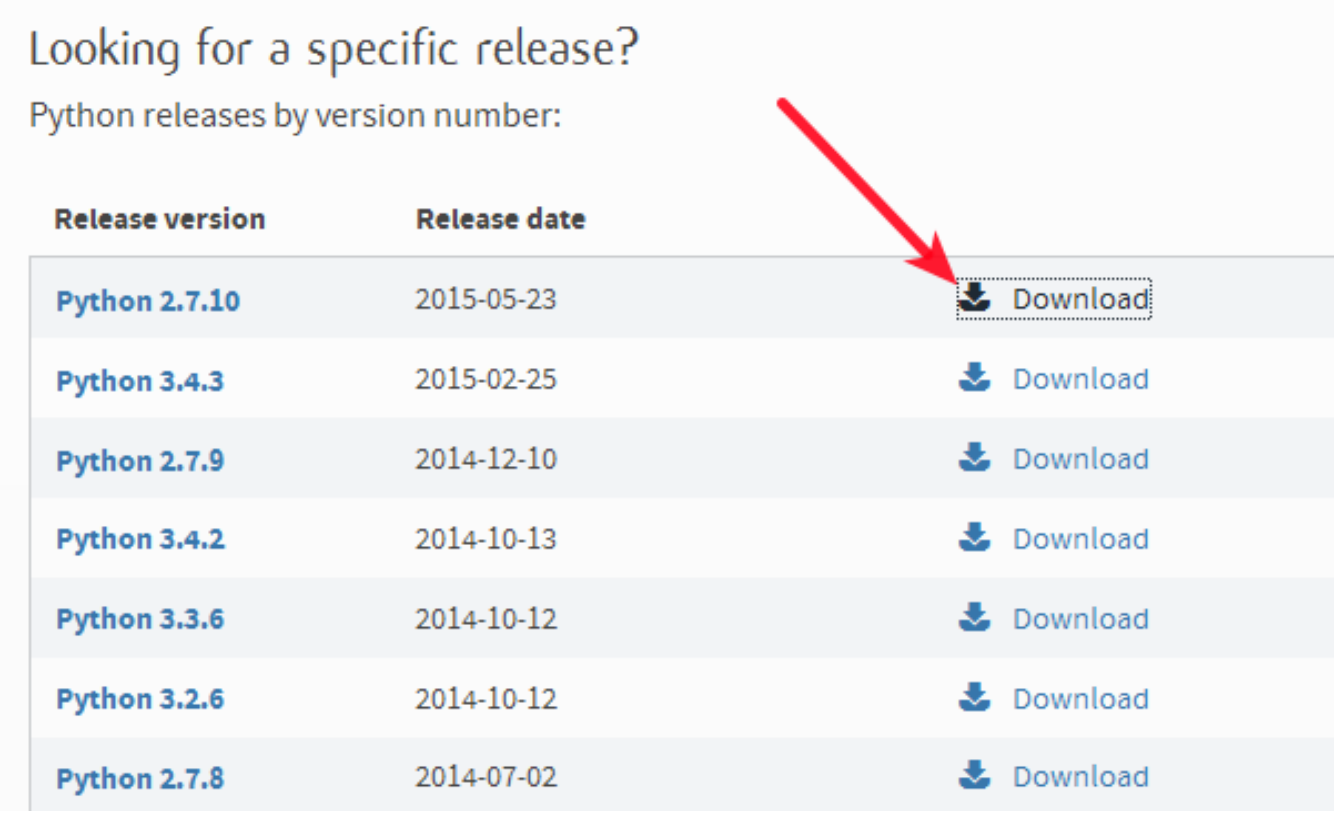
Пример Hello Word!.

Пакетный менеджер pip

После установки CPython в окружении появится утилита `easy_install`. С помощью нее можно установит *pip*, следующим образом:

```
$ easy_install pip
```

Или при помощи скрипта `get-pip.py`. Скрипт можно скачать по прямой ссылке https://raw.github.com/pypa/pip/master/contrib/get-pip.py



Looking for a specific release?

Python releases by version number:

Release version	Release date	
Python 2.7.10	2015-05-23	Download
Python 3.4.3	2015-02-25	Download
Python 2.7.9	2014-12-10	Download
Python 3.4.2	2014-10-13	Download
Python 3.3.6	2014-10-12	Download
Python 3.2.6	2014-10-12	Download
Python 2.7.8	2014-07-02	Download

Запускается скрипт как обычная Python программа.

Теперь можно устанавливать Python пакеты.

Виртуальное окружение Virtualenv

Зададим переменную окружения `WORKON_HOME` которая указывает где будут хранится изолированные окружения.

Теперь можно создавать изолированные окружения для каждого проекта.

Компиляция пакетов

Некоторые Python пакеты написаны с использование языка программирования Си, поэтому при установке они требуют компиляции. Если у вас не установлен компилятор, пакет не будет установлен.

Попробуем установить NumPy без компилятора.

```
$ pip install numpy
```

После установки следующих приложений для Windows:

Microsoft .NET Framework 2.0 с пакетом обновления 2 (SP2)
https://www.microsoft.com/en-us/download/details.aspx?id=1639

Microsoft Visual C++ Compiler for Python 2.7
http://www.microsoft.com/en-us/download/details.aspx?id=44266

Компиляция пройдет успешно:

Установка git

Скачайте и запустите инсталятор по ссылке http://git-scm.com/download/win.

Пример

Склонируем репозитарий админки https://github.com/sacrud/pyramid_sacrud.git в директорию `C:\Projects`.

```
$ git clone https://github.com/sacrud/pyramid_sacrud.git
```

Установим pyramid_sacrud из исходных кодов.

```
$ cd C:\Projects\pyramid_sacrud
$ mkvirtualenv pyramid_sacrud
$ python setup.py develop
```

Далее установим пример `pyramid_sacrud/example`

```
$ cd C:\Projects\pyramid_sacrud\example
$ workon pyramid_sacrud
$ python setup.py develop
```

Пакеты устанавливаются в виртуальное окружение с названием `pyramid_sacrud`.

Установим дополнительные пакеты `six`, `pyramid_jinja2==1.10` и `iso8601`:

```
$ pip install six iso8601 pyramid_jinja2==1.10
```

Теперь можно запустить пример:

```
$ cd C:\Projects\pyramid_sacrud\example
$ workon pyramid_sacrud
$ pserve development.ini
```

Заходим на http://localhost:6543/admin/

Установка Anaconda в Windows

См.также:

- https://www.anaconda.com/
- https://www.anaconda.com/download/
- https://en.wikipedia.org/wiki/Anaconda_(Python_distribution)

Anaconda – свободный open source дистрибутив для языков программирования Python и R с открытым кодом для обработки данных большого объема, построения аналитических прогнозов и научных вычислений. Разработчики дистрибутива имеют цель упростить управление и использование пакетов. Версии пакетов контролируются системой управления пакетами conda. По умолчанию, вместе с Anaconda устанавливается также:

- JupyterLab
- Jupyter Notebook
- Spyder

Пакетный менеджер conda

См.также:

https://conda.io/docs/user-guide/getting-started.html

После установки дистрибутива *Anaconda* в командной строке (*cmd.exe*) должна появится команда пакетного менеджера *conda*.

Проверим версию выполнив команду в терминале:

```
C:\Users\user>conda --version
conda 4.3.30
```

Неплохо было бы обновится до последней версии, делается это командой *update*:

```
C:\Users\user>conda update conda
Fetching package metadata ...............
Solving package specifications: .

Package plan for installation in environment C:\Users\user\Anaconda3:

The following packages will be UPDATED:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    conda:   4.3.30-py36h404fb56_0 --> 4.5.11-py36_0
    pycosat: 0.6.1-py36_1          --> 0.6.3-py36hfa6e2cd_0

Proceed ([y]/n)? y

pycosat-0.6.3- 100% |################################| Time: 0:00:00   1.40 MB/s
conda-4.5.11-p 100% |################################| Time: 0:00:00   5.15 MB/s
```

Anaconda дополнительно устанавливает множество различных python пакетов для того, что бы узнать, что у нас установлено необходимо выполнить команду *list*:

```
C:\Users\user\Project\pyramid_test>conda list
# packages in environment at C:\Users\user\Anaconda3:
#
# Name                    Version                   Build  Channel
_license                  1.1                      py36_1
alabaster                 0.7.9                    py36_0
anaconda                  custom                   py36_0
anaconda-client           1.6.0                    py36_0
anaconda-navigator        1.5.0                    py36_0
anaconda-project          0.4.1                    py36_0
anyqt                     0.0.8                    py36_0
astroid                   1.4.9                    py36_0
astropy                   1.3                 np111py36_0
babel                     2.3.4                    py36_0
backports                 1.0                      py36_0
beautifulsoup4            4.5.3                    py36_0
bitarray                  0.8.1                    py36_1
blas                      1.0                         mkl
blaze                     0.10.1                   py36_0
bokeh                     0.12.4                   py36_0
boto                      2.45.0                   py36_0
bottleneck                1.2.0               np111py36_0
bzip2                     1.0.6                    vc14_3  [vc14]
cffi                      1.9.1                    py36_0
chardet                   2.3.0                    py36_0
chest                     0.2.3                    py36_0
click                     6.7                      py36_0
cloudpickle               0.2.2                    py36_0
clyent                    1.2.2                    py36_0
colorama                  0.3.7                    py36_0
comtypes                  1.1.2                    py36_0
conda                     4.5.11                   py36_0
conda-env                 2.6.0                         0
```

(continues on next page)

(продолжение с предыдущей страницы)

```
configobj                5.0.6                    py36_0
console_shortcut         0.1.1                    py36_1
contextlib2              0.5.4                    py36_0
cryptography             1.7.1                    py36_0
curl                     7.52.1                   vc14_0  [vc14]
...
```

Виртуальное окружение Conda

Conda позволяет создавать виртуальные окружения для изолированной разработки программ. Делается это при помощи команды *create*:

```
C:\Users\user>conda create --name myenv sqlite
Solving environment: done

## Package Plan ##

  environment location: C:\Users\user\Anaconda3\envs\myenv

  added / updated specs:
    - sqlite


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    vc-14.1                    |       h0510ff6_4           6 KB
    sqlite-3.25.2              |       hfa6e2cd_0         897 KB
    vs2015_runtime-14.15.26706 |       h3a45250_0         2.1 MB
    ------------------------------------------------------------
                                           Total:         2.9 MB

The following NEW packages will be INSTALLED:

    sqlite:         3.25.2-hfa6e2cd_0
    vc:             14.1-h0510ff6_4
    vs2015_runtime: 14.15.26706-h3a45250_0

Proceed ([y]/n)? y


Downloading and Extracting Packages
vc-14.1               | 6 KB      | ##################################################
↪############################## | 100%
```

(continues on next page)

(продолжение с предыдущей страницы)

```
sqlite-3.25.2        | 897 KB    | ##############################################
↪############################## | 100%
vs2015_runtime-14.15 | 2.1 MB    | ##############################################
↪############################## | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use:
# > activate myenv
#
# To deactivate an active environment, use:
# > deactivate
#
# * for power-users using bash, you must source
#


C:\Users\user>
```

Активация виртуального окружения осуществляется при помощи команды *activate*:

```
C:\Users\user>activate myenv

(myenv) C:\Users\user>conda list
# packages in environment at C:\Users\user\Anaconda3\envs\myenv:
#
# Name                    Version                   Build  Channel
sqlite                    3.25.2               hfa6e2cd_0
vc                        14.1                 h0510ff6_4
vs2015_runtime            14.15.26706          h3a45250_0
```

После активации, мы как бы находимся внутри изолированного окружения, подтверждением этого является пригласительная надпись в круглых скобках в начале строки с именем нашего окружения *(myenv)*. Теперь если запустить команду *list* (список установленных пакетов) мы получим намного меньший список только того, что установлено в наше новое виртуальное окружение.

Пакетный менеджер pip

Пакетный менеджер *pip* это универсальный инструмент установки пакетов в мире *python*, он устанавливает официальные пакеты из общего хранилища пакетов *PyPi*. Поэтому *pip* незаменимый инструмент для разработки на *Python*. Установим его при помощи команды *install*.

```
(myenv) C:\Users\user>conda install pip
Solving environment: done


==> WARNING: A newer version of conda exists. <==
  current version: 4.4.6
  latest version: 4.5.11

Please update conda by running

    $ conda update -n base conda



## Package Plan ##

  environment location: C:\Users\user\Anaconda3\envs\myenv

  added / updated specs:
    - pip


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    setuptools-40.4.3          |           py37_0         575 KB
    wincertstore-0.2           |           py37_0          13 KB
    pip-10.0.1                 |           py37_0         1.7 MB
    python-3.7.1               |       h33f27b4_3        18.6 MB
    wheel-0.32.2               |           py37_0          51 KB
    sqlite-3.20.1              |   vc14hf772eac_1         796 KB
    certifi-2018.10.15         |           py37_0         138 KB
    ------------------------------------------------------------
                                           Total:        21.9 MB

The following NEW packages will be INSTALLED:

    certifi:      2018.10.15-py37_0
    pip:          10.0.1-py37_0
    python:       3.7.1-h33f27b4_3
    setuptools:   40.4.3-py37_0
    wheel:        0.32.2-py37_0
    wincertstore: 0.2-py37_0

The following packages will be DOWNGRADED:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    sqlite:       3.25.2-hfa6e2cd_0 --> 3.20.1-vc14hf772eac_1

Proceed ([y]/n)? y


Downloading and Extracting Packages
setuptools 40.4.3: ################################################################
↪############################## | 100%
wincertstore 0.2: ################################################################
↪############################## | 100%
pip 10.0.1: ######################################################################
↪############################## | 100%
python 3.7.1: ####################################################################
↪############################## | 100%
wheel 0.32.2: ####################################################################
↪############################## | 100%
sqlite 3.20.1: ###################################################################
↪############################## | 100%
certifi 2018.10.15: ##############################################################
↪############################## | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

Теперь нам доступны все пакеты с *PyPi*, установим фреймворк *Pyramid*:

```
(myenv) C:\Users\user>pip install pyramid
Collecting pyramid
  Downloading https://files.pythonhosted.org/packages/85/c7/
↪0a14873ef7bbb6d30e38678334d5b5faee1ccae2f5a59f093d104a3cc5ee/pyramid-1.9.2-py2.
↪py3-none-any.whl (582kB)
    100% || 583kB 4.0MB/s
Collecting zope.deprecation>=3.5.0 (from pyramid)
  Downloading https://files.pythonhosted.org/packages/ee/33/
↪625098914ec59b3006adf2cdf44a721e9671f4836af9eeb8cbe14e485954/zope.deprecation-4.
↪3.0-py2.py3-none-any.whl
Collecting zope.interface>=3.8.0 (from pyramid)
  Downloading https://files.pythonhosted.org/packages/55/99/
↪f728599ef08137889cacc58c08e3b1affe974fcd029528a822ec7b7efffa/zope.interface-4.6.
↪0-cp37-cp37m-win32.whl (132kB)
    100% || 133kB 2.0MB/s
Collecting plaster-pastedeploy (from pyramid)
  Downloading https://files.pythonhosted.org/packages/d9/e2/
↪de7cd499923dbf6aacc9b243f262817bfea3ffbbd4dcc5847e1aaec817a7/plaster_pastedeploy-
↪0.6-py2.py3-none-any.whl
```

(continues on next page)

(продолжение с предыдущей страницы)

```
Collecting translationstring>=0.4 (from pyramid)
  Downloading https://files.pythonhosted.org/packages/26/e7/
↪9dcf5bcd32b3ad16db542845ad129c06927821ded434ae88f458e6190626/translationstring-1.
↪3-py2.py3-none-any.whl
Requirement already satisfied: setuptools in␣
↪c:\users\user\anaconda3\envs\myenv\lib\site-packages (from pyramid) (40.4.3)
Collecting PasteDeploy>=1.5.0 (from pyramid)
  Downloading https://files.pythonhosted.org/packages/31/28/
↪51201a54aeecbd02eff767d17050b302f6fd98fdfecb4e3f4c9301ba6ef8/PasteDeploy-1.5.2-
↪py2.py3-none-any.whl
Collecting plaster (from pyramid)
  Downloading https://files.pythonhosted.org/packages/61/29/
↪3ac8a5d03b2d9e6b876385066676472ba4acf93677acfc7360b035503d49/plaster-1.0-py2.py3-
↪none-any.whl
Collecting WebOb>=1.7.0 (from pyramid)
  Downloading https://files.pythonhosted.org/packages/b5/74/
↪a9aaec7ca6c94a58e379a9c95255a2b2017514948054c72c0d1a25953348/WebOb-1.8.3-py2.py3-
↪none-any.whl (113kB)
    100% || 122kB 3.8MB/s
Collecting repoze.lru>=0.4 (from pyramid)
  Downloading https://files.pythonhosted.org/packages/b0/30/
↪6cc0c95f0b59ad4b3b9163bff7cdcf793cc96fac64cf398ff26271f5cf5e/repoze.lru-0.7-py3-
↪none-any.whl
Collecting hupper (from pyramid)
  Downloading https://files.pythonhosted.org/packages/70/b7/
↪4013ae11e977d4a38141ecba1c754f8b0a826b182de0c5c6fb780ede9834/hupper-1.3.1-py2.
↪py3-none-any.whl
Collecting venusian>=1.0a3 (from pyramid)
  Downloading https://files.pythonhosted.org/packages/2f/c2/
↪3d122e19287ed7d73f03821cef87e53673f27d41cae54ee3a46e92b147e2/venusian-1.1.0-py2.
↪py3-none-any.whl
Installing collected packages: zope.deprecation, zope.interface, PasteDeploy,␣
↪plaster, plaster-pastedeploy, translationstring, WebOb, repoze.lru, hupper,␣
↪venusian, pyramid
Successfully installed PasteDeploy-1.5.2 WebOb-1.8.3 hupper-1.3.1 plaster-1.0␣
↪plaster-pastedeploy-0.6 pyramid-1.9.2 repoze.lru-0.7 translationstring-1.3␣
↪venusian-1.1.0 zope.deprecation-4.3.0 zope.interface-4.6.0
You are using pip version 10.0.1, however version 18.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip'␣
↪command.
```

Проверим что пакет установился командой *list*:

```
(myenv) C:\Users\user>conda list
# packages in environment at C:\Users\user\Anaconda3\envs\myenv:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
#
certifi                   2018.10.15               py37_0
hupper                    1.3.1                     <pip>
PasteDeploy               1.5.2                     <pip>
pip                       10.0.1                   py37_0
plaster                   1.0                       <pip>
plaster-pastedeploy       0.6                       <pip>
pyramid                   1.9.2                     <pip>
python                    3.7.1                h33f27b4_3
repoze.lru                0.7                       <pip>
setuptools                40.4.3                   py37_0
sqlite                    3.20.1           vc14hf772eac_1  []
translationstring         1.3                       <pip>
vc                        14.1                 h0510ff6_4  []
venusian                  1.1.0                     <pip>
vs2015_runtime            14.15.26706          h3a45250_0  []
WebOb                     1.8.3                     <pip>
wheel                     0.32.2                   py37_0
wincertstore              0.2                      py37_0
zope.deprecation          4.3.0                     <pip>
zope.interface            4.6.0                     <pip>
```

Пример

См.также:

- https://docs.pylonsproject.org/projects/pyramid/en/latest/narr/project.html
- https://docs.pylonsproject.org/projects/pyramid/en/latest/narr/cookiecutters.html

Для примера создадим стартовую Веб-страницу при помощи фреймворка *Pyramid*. Для её создания будем использовать готовый шаблон проекта, который можно установить при помощи программы *cookiecutter*. Скачаем *cookiecutter*:

```
(myenv) C:\Users\user\Project\pyramid_test>pip install cookiecutter
Collecting cookiecutter
  Downloading https://files.pythonhosted.org/packages/16/99/
↪1ca3a75978270288354f419e9166666801cf7e7d8df984de44a7d5d8b8d0/cookiecutter-1.6.0-
↪py2.py3-none-any.whl (50kB)
    100% || 51kB 584kB/s
Collecting requests>=2.18.0 (from cookiecutter)
  Downloading https://files.pythonhosted.org/packages/f1/ca/
↪10332a30cb25b627192b4ea272c351bce3ca1091e541245cccbace6051d8/requests-2.20.0-py2.
↪py3-none-any.whl (60kB)
    100% || 61kB 1.5MB/s
```

(continues on next page)

(продолжение с предыдущей страницы)

```
Collecting poyo>=0.1.0 (from cookiecutter)
  Downloading https://files.pythonhosted.org/packages/e0/16/
↪e00e3001007a5e416ca6a51def6f9e4be6a774bf1c8486d20466f834d113/poyo-0.4.2-py2.py3-
↪none-any.whl
Collecting click>=5.0 (from cookiecutter)
  Downloading https://files.pythonhosted.org/packages/fa/37/
↪45185cb5abbc30d7257104c434fe0b07e5a195a6847506c074527aa599ec/Click-7.0-py2.py3-
↪none-any.whl (81kB)
    100% || 81kB 6.8MB/s
Collecting jinja2>=2.7 (from cookiecutter)
  Downloading https://files.pythonhosted.org/packages/7f/ff/
↪ae64bacdfc95f27a016a7bed8e8686763ba4d277a78ca76f32659220a731/Jinja2-2.10-py2.py3-
↪none-any.whl (126kB)
    100% || 133kB 8.9MB/s
Collecting future>=0.15.2 (from cookiecutter)
  Downloading https://files.pythonhosted.org/packages/85/aa/
↪ba2e24dcb889d7e98733f87515d80b3512418b80ba79d82d2ddcd43fadf3/future-0.17.0.tar.
↪gz (827kB)
    100% || 829kB 3.1MB/s
Collecting whichcraft>=0.4.0 (from cookiecutter)
  Downloading https://files.pythonhosted.org/packages/ab/c6/
↪eb4d1dfbb68168bb01c4394420e5e71d5851e64b910838aa0f14ebd5c7a0/whichcraft-0.5.2-
↪py2.py3-none-any.whl
Collecting jinja2-time>=0.1.0 (from cookiecutter)
  Downloading https://files.pythonhosted.org/packages/6a/a1/
↪d44fa38306ffa34a7e1af09632b158e13ec89670ce491f8a15af3ebcb4e4/jinja2_time-0.2.0-
↪py2.py3-none-any.whl
Collecting binaryornot>=0.2.0 (from cookiecutter)
  Downloading https://files.pythonhosted.org/packages/24/7e/
↪f7b6f453e6481d1e233540262ccbfcf89adcd43606f44a028d7f5fae5eb2/binaryornot-0.4.4-
↪py2.py3-none-any.whl
Collecting urllib3<1.25,>=1.21.1 (from requests>=2.18.0->cookiecutter)
  Downloading https://files.pythonhosted.org/packages/8c/4b/
↪5cbc4cb46095f369117dcb751821e1bef9dd86a07c968d8757e9204c324c/urllib3-1.24-py2.
↪py3-none-any.whl (117kB)
    100% || 122kB 4.1MB/s
Collecting idna<2.8,>=2.5 (from requests>=2.18.0->cookiecutter)
  Downloading https://files.pythonhosted.org/packages/4b/2a/
↪0276479a4b3caeb8a8c1af2f8e4355746a97fab05a372e4a2c6a6b876165/idna-2.7-py2.py3-
↪none-any.whl (58kB)
    100% || 61kB 4.7MB/s
Collecting chardet<3.1.0,>=3.0.2 (from requests>=2.18.0->cookiecutter)
  Downloading https://files.pythonhosted.org/packages/bc/a9/
↪01ffebfb562e4274b6487b4bb1ddec7ca55ec7510b22e4c51f14098443b8/chardet-3.0.4-py2.
↪py3-none-any.whl (133kB)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    100% || 143kB 7.6MB/s
Requirement already satisfied: certifi>=2017.4.17 in␣
↪c:\users\user\anaconda3\envs\myenv\lib\site-packages (from requests>=2.18.0->
↪cookiecutter) (2018.10.15)
Collecting MarkupSafe>=0.23 (from jinja2>=2.7->cookiecutter)
  Downloading https://files.pythonhosted.org/packages/4d/de/
↪32d741db316d8fdb7680822dd37001ef7a448255de9699ab4bfcbdf4172b/MarkupSafe-1.0.tar.
↪gz
Collecting arrow (from jinja2-time>=0.1.0->cookiecutter)
  Downloading https://files.pythonhosted.org/packages/e0/86/
↪4eb5228a43042e9a80fe8c84093a8a36f5db34a3767ebd5e1e7729864e7b/arrow-0.12.1.tar.gz␣
↪(65kB)
    100% || 71kB 2.0MB/s
Collecting python-dateutil (from arrow->jinja2-time>=0.1.0->cookiecutter)
  Downloading https://files.pythonhosted.org/packages/2f/e9/
↪b02e8a1a8c53a55a4f37df1e8e111539d0a3e76828bcd252947a5200b797/python_dateutil-2.7.
↪4-py2.py3-none-any.whl (211kB)
    100% || 215kB 2.9MB/s
Collecting six>=1.5 (from python-dateutil->arrow->jinja2-time>=0.1.0->cookiecutter)
  Downloading https://files.pythonhosted.org/packages/67/4b/
↪141a581104b1f6397bfa78ac9d43d8ad29a7ca43ea90a2d863fe3056e86a/six-1.11.0-py2.py3-
↪none-any.whl
Building wheels for collected packages: future, MarkupSafe, arrow
  Running setup.py bdist_wheel for future ... done
  Stored in directory:␣
↪C:\Users\user\AppData\Local\pip\Cache\wheels\fc\5b\ec\2983c4a6e3692d1315f44d6480c6abdd8585d964
  Running setup.py bdist_wheel for MarkupSafe ... done
  Stored in directory:␣
↪C:\Users\user\AppData\Local\pip\Cache\wheels\33\56\20\ebe49a5c612fffe1c5a632146b16596f9e646767
  Running setup.py bdist_wheel for arrow ... done
  Stored in directory:␣
↪C:\Users\user\AppData\Local\pip\Cache\wheels\a3\dd\b2\d3b8d22e8136164c2e2c36ed42392531957cdf9c
Successfully built future MarkupSafe arrow
Installing collected packages: urllib3, idna, chardet, requests, poyo, click,␣
↪MarkupSafe, jinja2, future, whichcraft, six, python-dateutil, arrow, jinja2-time,
↪ binaryornot, cookiecutter
Successfully installed MarkupSafe-1.0 arrow-0.12.1 binaryornot-0.4.4 chardet-3.0.4␣
↪click-7.0 cookiecutter-1.6.0 future-0.17.0 idna-2.7 jinja2-2.10 jinja2-time-0.2.
↪0 poyo-0.4.2 python-dateutil-2.7.4 requests-2.20.0 six-1.11.0 urllib3-1.24␣
↪whichcraft-0.5.2
You are using pip version 10.0.1, however version 18.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip'␣
↪command.
```

При помощи *cookiecutter* развернем самый простой шаблон Веб-сайта который имеется в фреймворке *Pyramid*:

```
(myenv) C:\Users\user\Project\pyramid_test>cookiecutter gh:Pylons/pyramid-
↪cookiecutter-starter
project_name [Pyramid Scaffold]: myfirstapp
repo_name [myfirstapp]:
Select template_language:
1 - jinja2
2 - chameleon
3 - mako
Choose from 1, 2, 3 (1, 2, 3) [1]: 1

================================================================================
Documentation: https://docs.pylonsproject.org/projects/pyramid/en/latest/
Tutorials:     https://docs.pylonsproject.org/projects/pyramid_tutorials/en/latest/
Twitter:       https://twitter.com/PylonsProject
Mailing List:  https://groups.google.com/forum/#!forum/pylons-discuss
Welcome to Pyramid.  Sorry for the convenience.
================================================================================

Change directory into your newly created project.
    cd myfirstapp

Create a Python virtual environment.
    py -3 -m venv env

Upgrade packaging tools.
    env\Scripts\pip install --upgrade pip setuptools

Install the project in editable mode with its testing requirements.
    env\Scripts\pip install -e ".[testing]"

Run your project's tests.
    env\Scripts\pytest

Run your project.
    env\Scripts\pserve development.ini
```

Проект создается в отдельной директории `myfirstapp`.

```
(myenv) C:\Users\user\Project\pyramid_test>dir
 Том в устройстве C не имеет метки.
 Серийный номер тома: 480D-DE95

 Содержимое папки C:\Users\user\Project\pyramid_test

26.10.2018  16:30    <DIR>          .
26.10.2018  16:30    <DIR>          ..
```

(continues on next page)

(продолжение с предыдущей страницы)

```
26.10.2018  16:30    <DIR>          myfirstapp
               0 файлов              0 байт
               3 папок  31 729 090 560 байт свободно
```

Что бы запустить проект необходимо прежде установить его в окружение, перейдем в директорию проекта и запустим стандартную команду установки пакетов из исходников.

```
(myenv) C:\Users\user\Project\pyramid_test>cd myfirstapp
(myenv) C:\Users\user\Project\pyramid_test\myfirstapp>pip install -e .
Obtaining file:///C:/Users/user/Project/pyramid_test/myfirstapp
Requirement already satisfied: plaster_pastedeploy in␣
↪c:\users\user\anaconda3\envs\myenv\lib\site-packages (from myfirstapp==0.0) (0.6)
Requirement already satisfied: pyramid in␣
↪c:\users\user\anaconda3\envs\myenv\lib\site-packages (from myfirstapp==0.0) (1.9.
↪2)
Collecting pyramid_jinja2 (from myfirstapp==0.0)
  Downloading https://files.pythonhosted.org/packages/21/30/
↪fdd0b9a365a60c9e56ae4730c8839eae603f7a87696df14dbd4f980acf35/pyramid_jinja2-2.7-
↪py2.py3-none-any.whl (70kB)
    100% || 71kB 421kB/s
Collecting pyramid_debugtoolbar (from myfirstapp==0.0)
  Downloading https://files.pythonhosted.org/packages/6f/9a/
↪933267076461c1fd6f4f8b0715ecf037dbe622180d0b77e7ea605a32b51b/pyramid_
↪debugtoolbar-4.5-py2.py3-none-any.whl (345kB)
    100% || 348kB 2.3MB/s
Collecting waitress (from myfirstapp==0.0)
  Downloading https://files.pythonhosted.org/packages/ee/af/
↪ac32a716d64e56561ee9c23ce45ee2865d7ac4e0678b737d2f5ee49b5fd6/waitress-1.1.0-py2.
↪py3-none-any.whl (114kB)
    100% || 122kB 3.7MB/s
Requirement already satisfied: PasteDeploy>=1.5.0 in␣
↪c:\users\user\anaconda3\envs\myenv\lib\site-packages (from plaster_pastedeploy->
↪myfirstapp==0.0) (1.5.2)
Requirement already satisfied: plaster>=0.5 in␣
↪c:\users\user\anaconda3\envs\myenv\lib\site-packages (from plaster_pastedeploy->
↪myfirstapp==0.0) (1.0)
Requirement already satisfied: zope.interface>=3.8.0 in␣
↪c:\users\user\anaconda3\envs\myenv\lib\site-packages (from pyramid->
↪myfirstapp==0.0) (4.6.0)
Requirement already satisfied: translationstring>=0.4 in␣
↪c:\users\user\anaconda3\envs\myenv\lib\site-packages (from pyramid->
↪myfirstapp==0.0) (1.3)
Requirement already satisfied: zope.deprecation>=3.5.0 in␣
↪c:\users\user\anaconda3\envs\myenv\lib\site-packages (from pyramid->
↪myfirstapp==0.0) (4.3.0)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
Requirement already satisfied: setuptools in␣
↪c:\users\user\anaconda3\envs\myenv\lib\site-packages (from pyramid->
↪myfirstapp==0.0) (40.4.3)
Requirement already satisfied: WebOb>=1.7.0 in␣
↪c:\users\user\anaconda3\envs\myenv\lib\site-packages (from pyramid->
↪myfirstapp==0.0) (1.8.3)
Requirement already satisfied: hupper in␣
↪c:\users\user\anaconda3\envs\myenv\lib\site-packages (from pyramid->
↪myfirstapp==0.0) (1.3.1)
Requirement already satisfied: repoze.lru>=0.4 in␣
↪c:\users\user\anaconda3\envs\myenv\lib\site-packages (from pyramid->
↪myfirstapp==0.0) (0.7)
Requirement already satisfied: venusian>=1.0a3 in␣
↪c:\users\user\anaconda3\envs\myenv\lib\site-packages (from pyramid->
↪myfirstapp==0.0) (1.1.0)
Requirement already satisfied: MarkupSafe in␣
↪c:\users\user\anaconda3\envs\myenv\lib\site-packages (from pyramid_jinja2->
↪myfirstapp==0.0) (1.0)
Requirement already satisfied: Jinja2>=2.5.0 in␣
↪c:\users\user\anaconda3\envs\myenv\lib\site-packages (from pyramid_jinja2->
↪myfirstapp==0.0) (2.10)
Collecting Pygments (from pyramid_debugtoolbar->myfirstapp==0.0)
  Downloading https://files.pythonhosted.org/packages/02/ee/
↪b6e02dc6529e82b75bb06823ff7d005b141037cb1416b10c6f00fc419dca/Pygments-2.2.0-py2.
↪py3-none-any.whl (841kB)
    100% || 849kB 1.9MB/s
Collecting pyramid-mako>=0.3.1 (from pyramid_debugtoolbar->myfirstapp==0.0)
  Downloading https://files.pythonhosted.org/packages/f1/92/
↪7e69bcf09676d286a71cb3bbb887b16595b96f9ba7adbdc239ffdd4b1eb9/pyramid_mako-1.0.2.
↪tar.gz
Collecting Mako>=0.8 (from pyramid-mako>=0.3.1->pyramid_debugtoolbar->
↪myfirstapp==0.0)
  Downloading https://files.pythonhosted.org/packages/eb/f3/
↪67579bb486517c0d49547f9697e36582cd19dafb5df9e687ed8e22de57fa/Mako-1.0.7.tar.gz␣
↪(564kB)
    100% || 573kB 1.5MB/s
Building wheels for collected packages: pyramid-mako, Mako
  Running setup.py bdist_wheel for pyramid-mako ... done
  Stored in directory:␣
↪C:\Users\user\AppData\Local\pip\Cache\wheels\08\5f\98\3dfc5a39bcb3fd094897db7f394eb13768cdf472
  Running setup.py bdist_wheel for Mako ... done
  Stored in directory:␣
↪C:\Users\user\AppData\Local\pip\Cache\wheels\15\35\25\dbcb848832ccb1a4b4ad23f529badfd3bce9bf88
Successfully built pyramid-mako Mako
Installing collected packages: pyramid-jinja2, Pygments, Mako, pyramid-mako,␣
↪pyramid-debugtoolbar, waitress, myfirstapp
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  Running setup.py develop for myfirstapp
Successfully installed Mako-1.0.7 Pygments-2.2.0 myfirstapp pyramid-debugtoolbar-4.
↪5 pyramid-jinja2-2.7 pyramid-mako-1.0.2 waitress-1.1.0
You are using pip version 10.0.1, however version 18.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip'␣
↪command.
```

Проверяем что все поставилось:

```
(myenv) C:\Users\user\Project\pyramid_test\myfirstapp>conda list
# packages in environment at C:\Users\user\Anaconda3\envs\myenv:
#
arrow                     0.12.1                    <pip>
binaryornot               0.4.4                     <pip>
certifi                   2018.10.15               py37_0
chardet                   3.0.4                     <pip>
Click                     7.0                       <pip>
cookiecutter              1.6.0                     <pip>
future                    0.17.0                    <pip>
hupper                    1.3.1                     <pip>
idna                      2.7                       <pip>
Jinja2                    2.10                      <pip>
jinja2-time               0.2.0                     <pip>
Mako                      1.0.7                     <pip>
MarkupSafe                1.0                       <pip>
myfirstapp                0.0                       <pip>
numpy                     1.15.3                    <pip>
PasteDeploy               1.5.2                     <pip>
pip                       10.0.1                   py37_0
plaster                   1.0                       <pip>
plaster-pastedeploy       0.6                       <pip>
poyo                      0.4.2                     <pip>
Pygments                  2.2.0                     <pip>
pyramid                   1.9.2                     <pip>
pyramid-debugtoolbar      4.5                       <pip>
pyramid-jinja2            2.7                       <pip>
pyramid-mako              1.0.2                     <pip>
python                    3.7.1                h33f27b4_3
python-dateutil           2.7.4                     <pip>
repoze.lru                0.7                       <pip>
requests                  2.20.0                    <pip>
setuptools                40.4.3                   py37_0
six                       1.11.0                    <pip>
sqlite                    3.20.1           vc14hf772eac_1  []
translationstring         1.3                       <pip>
```

(continues on next page)

(продолжение с предыдущей страницы)

```
urllib3                   1.24                     <pip>
vc                        14.1                 h0510ff6_4  []
venusian                  1.1.0                    <pip>
vs2015_runtime            14.15.26706          h3a45250_0  []
waitress                  1.1.0                    <pip>
WebOb                     1.8.3                    <pip>
wheel                     0.32.2                  py37_0
whichcraft                0.5.2                    <pip>
wincertstore              0.2                     py37_0
zope.deprecation          4.3.0                    <pip>
zope.interface            4.6.0                    <pip>
```

Последний шаг это запуск самого Веб-приложения, после его установки в окружение должна появиться команда `pserve` она позволяет запускать `WSGI` приложения которым и является наш проект. Давайте попробуем это сделать:

```
(myenv) C:\Users\user\Project\pyramid_test\myfirstapp>pserve development.ini --
↪reload
Starting monitor for PID 1144.
Starting server in PID 1144.
Serving on http://DESKTOP-9JPISDO:6543
Serving on http://DESKTOP-9JPISDO:6543
```

Заходим на http://localhost:6543/

4.1.2 Виртуальное окружение

См.также:

https://docs.python.org/3/library/venv.html

```
$ python3 -m venv .env
$ source .env/bin/activate
$ which python
```

4.1.3 Управление пакетами в Python

Установка pip в Ubuntu

См.также:

- https://ru.wikipedia.org/wiki/Advanced_Packaging_Tool

- http://help.ubuntu.ru/wiki/\T2A\cyrr\T2A\cyru\T2A\cyrk\T2A\cyro\T2A\cyrv\T2A\cyro\T2A\cyrd\T2A\cyrs\T2A\cyrt\T2A\cyrv\T2A\cyro_\T2A\cyrp\T2A\cyro_ubuntu_server/\T2A\cyru\T2A\cyrp\T2A\cyrr\T2A\cyra\T2A\cyrv\T2A\cyrl\T2A\cyre\T2A\cyrn\T2A\cyri\T2A\cyre_\T2A\cyrp\T2A\cyra\T2A\cyrk\T2A\cyre\T2A\cyrt\T2A\cyra\T2A\cyrm\T2A\cyri/apt-get
- https://pip.pypa.io/en/latest/installing.html
- http://en.wikipedia.org/wiki/Pip_(package_manager)

Новые версии Ubuntu

```
$ sudo apt-get install python-pip python-dev build-essential
$ sudo pip install --upgrade pip
$ sudo pip install pyramid
$ pcreate -t alchemy MyProgect
```

Старые версии Ubuntu

```
$ sudo apt-get install python-setuptools python-dev build-essential
$ sudo easy_install pip
$ sudo pip install --upgrade pip
$ sudo pip install pyramid
$ pcreate -t alchemy MyProgect
```

Пакетный менеджер pip

```
$ pip uninstall django # Удаление пакета
$ pip install pyramid  # Установка пакета
```

```
$ pip install pyramid -U # Обновление
$ pip install pyramid --upgrade
$ pip install pip -U # Обновление самого pip
```

```
$ pip install pyramid --user # Установка локально, для этого пользователя
```

```
$ pip install -r requirements.txt # Установка из файла
```

```
$ pip install git+https://github.com/pylons/pyramid       # Установка по ссылке
$ pip install git+https://bitbucket.org/zzzeek/sqlalchemy # Установка по ссылке
```

Установка пакетов из исходных кодов

Копирует проект в PYTHONPAH

```
$ git clone git@github.com:myint/rstcheck.git
$ cd rstcheck
$ pip install .
```

Симлинк на директорию. Требуется для разработки, чтобы не устанавливать заново, после каждого изменения в проекте.

```
$ git clone git@github.com:myint/rstcheck.git
$ cd rstcheck
$ pip install -e .
```

4.1.4 Контекстный менеджер

См.также:

- **PEP 343**

```
1 with file("/tmp/foo", "w") as foo:
2     print >> foo, "Hello!"
```

Эквивалентно

```
1 foo = file("/tmp/foo", "w")
2 try:
3     print >> foo, "Hello!"
4 finally:
5     foo.close()
```

4.1.5 Форматированные строки

http://pyformat.info/

4.1.6 Декораторы

См.также:

- **PEP 318**
- http://lgiordani.com/blog/2015/04/23/python-decorators-metaprogramming-with-style/

Декоратор подменяет функцию, например мы можем подменить функцию `foo` на ноль.

```
def zero(func):
    return 0

@zero
def foo():
    return "Hi"

print(foo)  # 0
print(foo())  # Вызовет ошибку как-будто мы хотим вызвать ноль 0()
```

Это равносильно следующему коду:

```
def foo():
    return "Hi"

foo = 0

print(foo)  # 0
print(foo())  # Вызовет ошибку как-будто мы хотим вызвать ноль 0()
```

Подменим функцию на другую:

```
def zero(func):
    return lambda: 0

@zero
def foo():
    return "Hi"

print(foo())  # 0
```

Теперь `foo` это `lambda: 0`, а `foo()` соответственно `0`. Это равносильно следующему коду:

```
def foo():
    return "Hi"

foo = lambda: 0
print(foo())  # 0
```

И более практичный пример, дополним нашу функцию:

```
def world(func):
    return lambda: func() + " World!"

@world
```

(continues on next page)

(продолжение с предыдущей страницы)

```
def foo():
    return "Hi"

print(foo())  # Hi World!

@world
def hello():
    return "Hello"

print(hello())  # Hello World!
```

Этот пример уже сложнее переписать:

```
def foo():
    return "Hi"

foo = lambda: foo() + " World!"
print(foo())  # RuntimeError: maximum recursion depth exceeded
```

```
def foo():
    return "Hi"

hello_world = lambda: foo() + " World!"
print(bar())  # Hello World!
```

4.1.7 Декораторы для корутин в asyncio

См.также:

Декораторы

Декораторы для асинхронных функций пишутся как и для обычных только возвращать нужно корутину, а не функцию.

Для примера обычная функция:

```
def plusplus(func):
    def wrapped():
        return func() + 1
    return wrapped


@plusplus
def one():
    return 1
```

(continues on next page)

(продолжение с предыдущей страницы)

```
print(one())  # return 2


@plusplus
@plusplus
@plusplus
@plusplus
def one():
    return 1

print(one())  # now return 5
```

В декораторе наша обертка над функцией (`wrapped`) стала корутиной:

```
import timeit

import asyncio


def plusplus(func):
    async def wrapped():
        await asyncio.sleep(1)
        return await func() + 1
    return wrapped


@plusplus
async def one():
    await asyncio.sleep(2)
    return 1

loop = asyncio.get_event_loop()

start = timeit.default_timer()
print(loop.run_until_complete(one()))  # return 2
stop = timeit.default_timer()
print(stop - start)  # minimum 3 seconds


@plusplus
@plusplus
@plusplus
@plusplus
async def one():
```

(continues on next page)

(продолжение с предыдущей страницы)

```
31      await asyncio.sleep(2)
32      return 1
33
34  start = timeit.default_timer()
35  print(loop.run_until_complete(one()))  # return 5
36  stop = timeit.default_timer()
37  print(stop - start)  # minimum 6 seconds
```

Более практичный пример это функция `json_response` для вьюх в aiohttp. Идея взята из презентации (http://igordavydenko.com/talks/lvivpy-4/#slide-31).

```
import ujson
import asyncio
from aiohttp import web


def json_response(data, **kwargs):
    kwargs.setdefault('content_type', 'application/json')
    return web.Response(text=ujson.dumps(data), **kwargs)


async def index(request):
   return json_response({"Hello": "World"})
```

Все хорошо но ретурнов во вьюхе может быть много и тогда оборачивать каждый в `json_response` довольно неудобно. Чтобы решить эту проблему создадим декоратор `json_view`.

```
def json_view(func):
    async def wrapped(request):
        return json_response(await func(request))
    return wrapped
```

Теперь можно писать так:

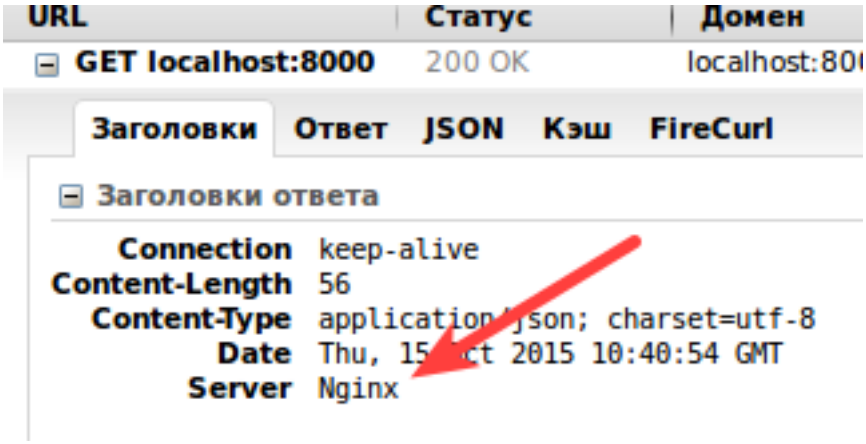
```
@json_view
async def index(request):
   if somethink:
      return {"Somethink": "happens"}
   else:
      return {"else": "happens"}
   return {"Hello": "World"}
```

Класс `aiohttp.web.Response` позволяет задавать различные параметры типа заголовков и статуса ответа. Перепишем наш декоратор таким образом чтобы он умел принимать эти параметры:

```
def json_view_arg(**kwargs):
    def wrap(func):
        async def wrapped(request):
            return json_response(await func(request), **kwargs)
        return wrapped
    return wrap
```

Теперь можно задать, например, кастомный заголовок ответа `Server`:

```
@json_view_arg(headers={"Server": "Nginx"})
async def index(request):
   return {"Hello": "World"}
```



И в заключение то же в виде класса-декоратора:

```
class JsonView(object):

    def __init__(self, **kwargs):
        self.kwargs = kwargs

    def __call__(self, func):
        async def wrapped(request):
            return json_response(await func(request), **self.kwargs)
        return wrapped
```

```
@JsonView(headers={"Server": "Nginx"})
async def index(request):
   return {"Hello": "World"}
```

4.2 Генераторы

См.также:

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/function*
- https://ru.wikipedia.org/wiki/\T2A\CYRP\T2A\cyrr\T2A\cyro\T2A\cyrd\T2A\cyro\T2A\cyrl\T2A\cyrzh\T2A\cyre\T2A\cyrn\T2A\cyri\T2A\cyre_(\T2A\cyri\T2A\cyrn\T2A\cyrf\T2A\cyro\T2A\cyrr\T2A\cyrm\T2A\cyra\T2A\cyrt\T2A\cyri\T2A\cyrk\T2A\cyra)

4.2.1 Python

См.также:

- **PEP 255**

```
def id_maker():
    index = 0
    while True:
        yield index
        index += 1

gen = id_maker()

print(next(gen))
print(next(gen))
print(next(gen))
```

Запуск:

```
$ python gen.py
0
1
2
```

```
class CSVFile(object):
    def __init__(self, path, sep=','):
        self.path = path
        self.sep = sep

    def __iter__(self):
        with open(self.path) as f:
            for l in f:
                yield l.split(self.sep)

csv_generator = CSVFile('sample.csv')

```

(continues on next page)

(продолжение с предыдущей страницы)

```
for row in csv_generator:
    print(row)
```

```
1997,Ford,E350,"ac, abs, moon",3000.00
1999,Chevy,"Venture ""Extended Edition""","",4900.00
1996,Jeep,Grand Cherokee,"MUST SELL! air, moon roof, loaded",4799.00
```

```
$ python csv_gen.py
['1997', 'Ford', 'E350', '"ac', ' abs', ' moon"', '3000.00\n']
['1999', 'Chevy', '"Venture ""Extended Edition"""', '""', '4900.00\n']
['1996', 'Jeep', 'Grand Cherokee', '"MUST SELL! air', ' moon roof', ' loaded"',
↪'4799.00\n']
```

4.2.2 JavaScript

```
function* idMaker(){
    var index = 0;
    while(true)
        yield index++;
}

var gen = idMaker();

console.log(gen.next().value); // 0
console.log(gen.next().value); // 1
console.log(gen.next().value); // 2
```

Запуск:

```
$ iojs gen.js
0
1
2
```

4.3 Текстовые редакторы

4.3.1 Visual Studio Code

См.также:

https://www.visualstudio.com/ru-ru/products/code-vs.aspx

Visual Studio Code отличный выбор для начинающего программиста, имеет необходимый минимум:

- неплохую документацию
- автодополнение кода (с использованием IntelliSense)
- подсветка синтаксиса
- встроенный отладчик
- расширение функционала за счет плагинов
- управление системой контроля версий git
- кроссплатформенный
- бесплатный, с открытым исходным кодом

Также редактор адаптирован для Веб-разработки и вполне подойдет для серьезных проектов как основной инструмент редактирования кода.

Установка

См.также:

https://code.visualstudio.com/docs/setup/setup-overview

Linux

См.также:

https://code.visualstudio.com/docs/setup/linux

1. Скачиваем дистрибутив для своей ОС https://code.visualstudio.com/download
2. Для Linux существуют два типа пакетов, самых популярных форматов, rpm и deb.

   Установка в Ubuntu/Debian:

   ```
   $ sudo dpkg -i <file>.deb
   ```

   CentOS/Fedora:

   ```
   $ sudo yum install <file>.rpm
   ```

   Fedora $>$ 22 версии:

   ```
   $ sudo dnf install <file>.rpm
   ```

3. После установки можно запустить редактор следующей командой:

```
$ code
```

Nix

Пакетный менеджер Nix работает на любом Linux дистрибутиве, содержит богатую базу уже готовых пакетов, в том числе и vscode.

1. Установка пакетного менеджера:

```
$ curl https://nixos.org/nix/install | sh
```

2. Установка Visual Studio Code:

```
$ nix-env -i vscode
```

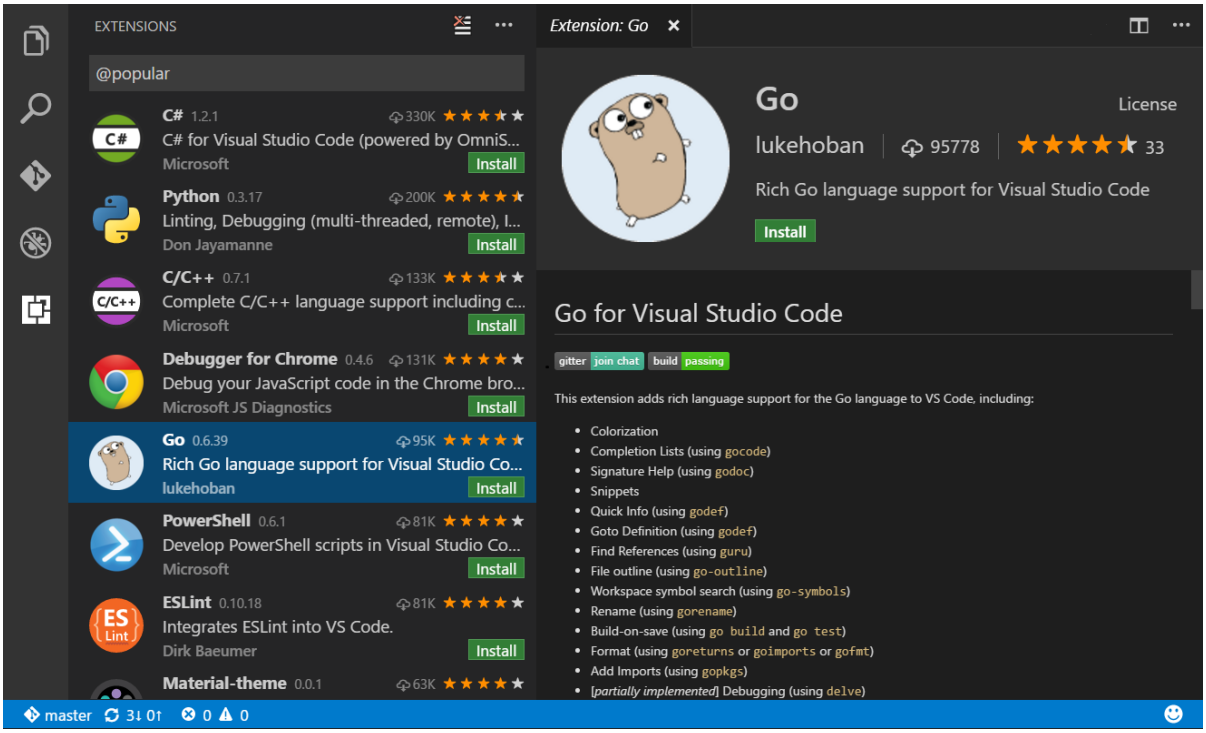
Плагины

См.также:

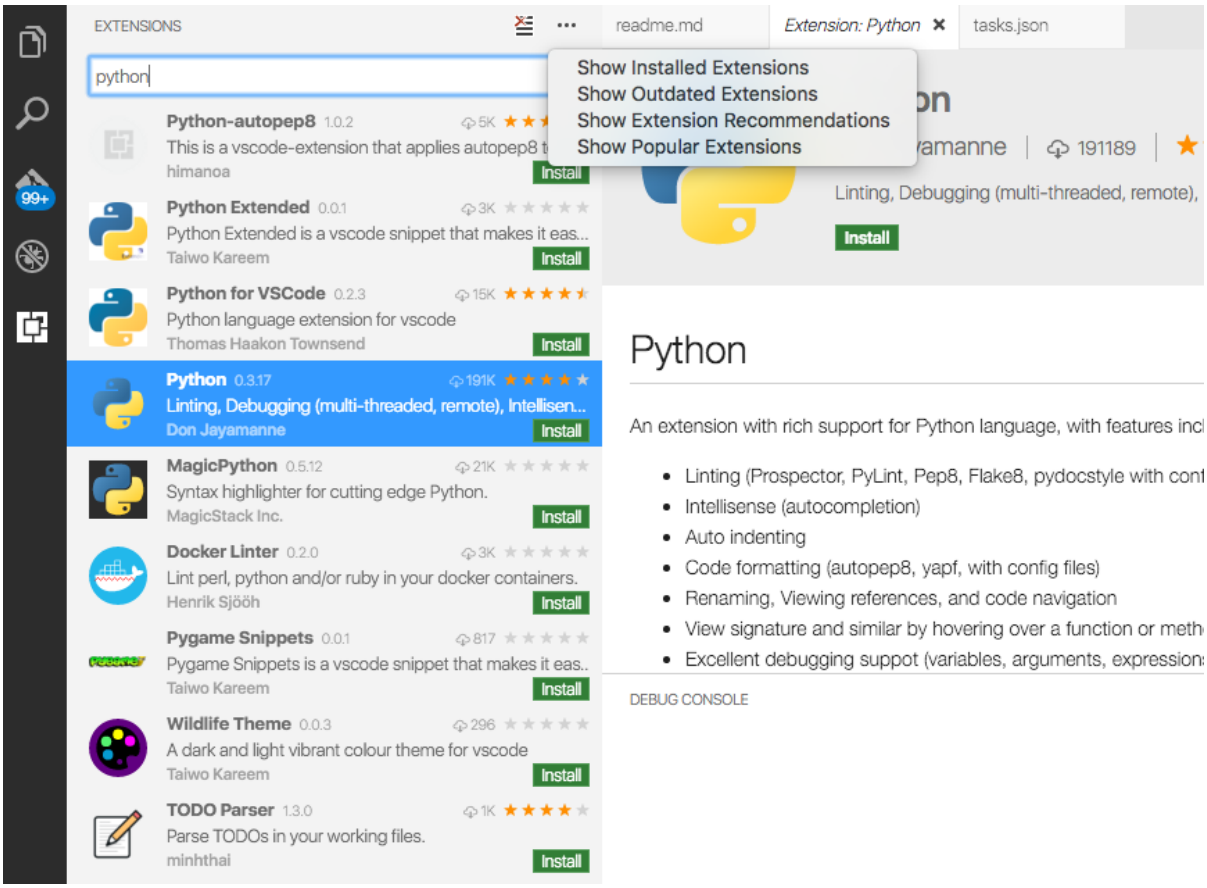
https://code.visualstudio.com/docs/editor/extension-gallery

Редактор имеет возможность расширения функционала за счет плагинов и удобный интерфейс их установки, доступный по нажатию кнопки:

Из списка можно выбрать любой плагин и установить, после чего он применит свои настройки к редактору.



Расширения можно искать введя название или ключевые слова в строке поиска, например *Python*.



Существует огромное количество расширений для *Go*, *C#*, *C/C++*, *Nix*, *Haskell*, *Python*, *JS*, *TypeScript* и др.

Python

См.также:

https://code.visualstudio.com/docs/languages/python

После установки плагина *Python* нам становятся доступны многие функции:

- Автодополнение кода
- Проверка синтаксиса
- Отладка
- Подсказки
- Переход к определению функции, класса и прочее

Автодополнение

Работает при наборе по нажатию `Ctrl` + `Space`.

Проверка синтаксиса

Показывает ошибки в коде:

Работает если установлены Python пакеты *Pylint*, *Pep8* или *Flake8*.

Совет:

```
$ pip install -U --user pylint pep8 flake8
```

Отладка

См.также:

https://code.visualstudio.com/docs/editor/debugging

Встроенный в редактор отладчик позволяет отлаживать код визуально, устанавливать точки останова мышкой и просматривать переменные в отдельном окне. Это похоже на отладку в различных IDE, таких как QtCreator или Wingware.

Также избавляет программиста писать мучительные строки типа *printf* или *import pdb;pdb.set_trace();*.

Настройки

См.также:

https://code.visualstudio.com/docs/customization/userandworkspace

Настройки хранятся в формате *JSON* и доступны из меню `File->Preferences->User Settings`.

Шрифт

Шрифт задается в настройках `File->Preferences->User Settings`:

```
// Place your settings in this file to overwrite the default settings
{
    // Controls the font size.
    "editor.fontSize": 16
}
```

Автодополнение через <Tab>

Более привычно дополнять код по клавише `<Tab>`. Для этого необходимо открыть настройки пользователя `File->Preferences->User Settings` и прописать опцию `editor.tabCompletion`:

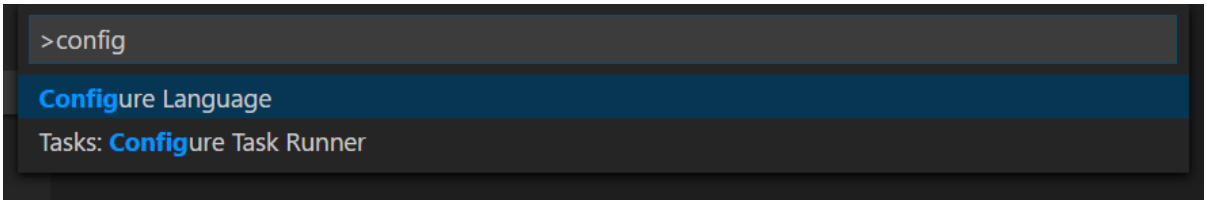
```
// Place your settings in this file to overwrite the default settings
{
    // Controls the font size.
    "editor.fontSize": 16,
    // Insert snippets when their prefix matches. Works best when 'quickSuggestions
↪' aren't enabled.
    "editor.tabCompletion": true
}
```

Язык

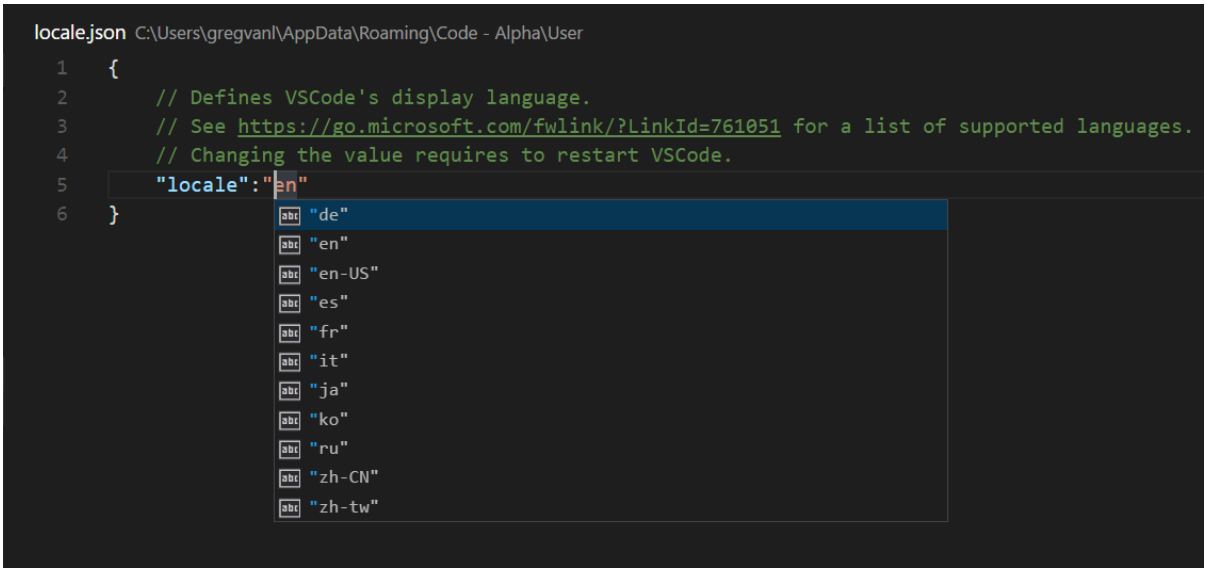
См.также:

https://code.visualstudio.com/docs/customization/locales

1. Открываем командную строку `Ctrl` + `Shift` + `P`
2. Вводим команду *Configure Language*



3. Меняем локаль на нужную, например `ru`:




```
{
    // Defines VS Code's display language.
    "locale": "ru"
}
```

Тема

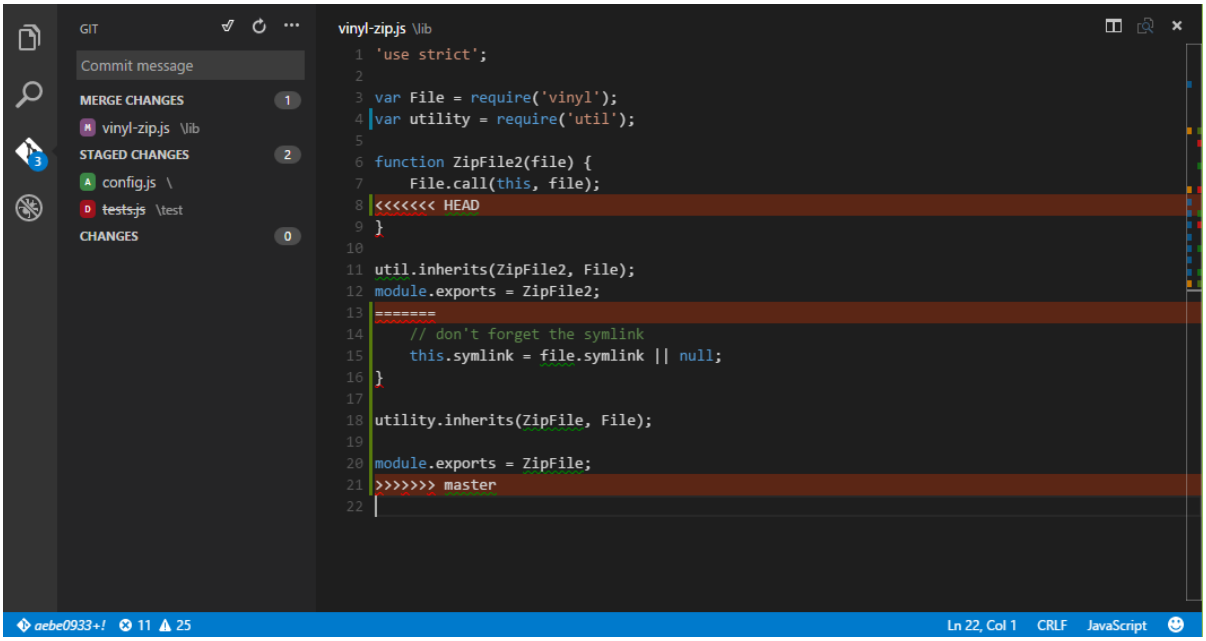
Цветовое оформление задается в настройках `File->Preferences->Color Theme`.

Git

См.также:

https://code.visualstudio.com/docs/editor/versioncontrol

Умеет подсвечивать изменения в файлах с предыдущего коммита, выполнять команды *git* и отслеживать состояние, например какая текущая ветка.



Python скрипты

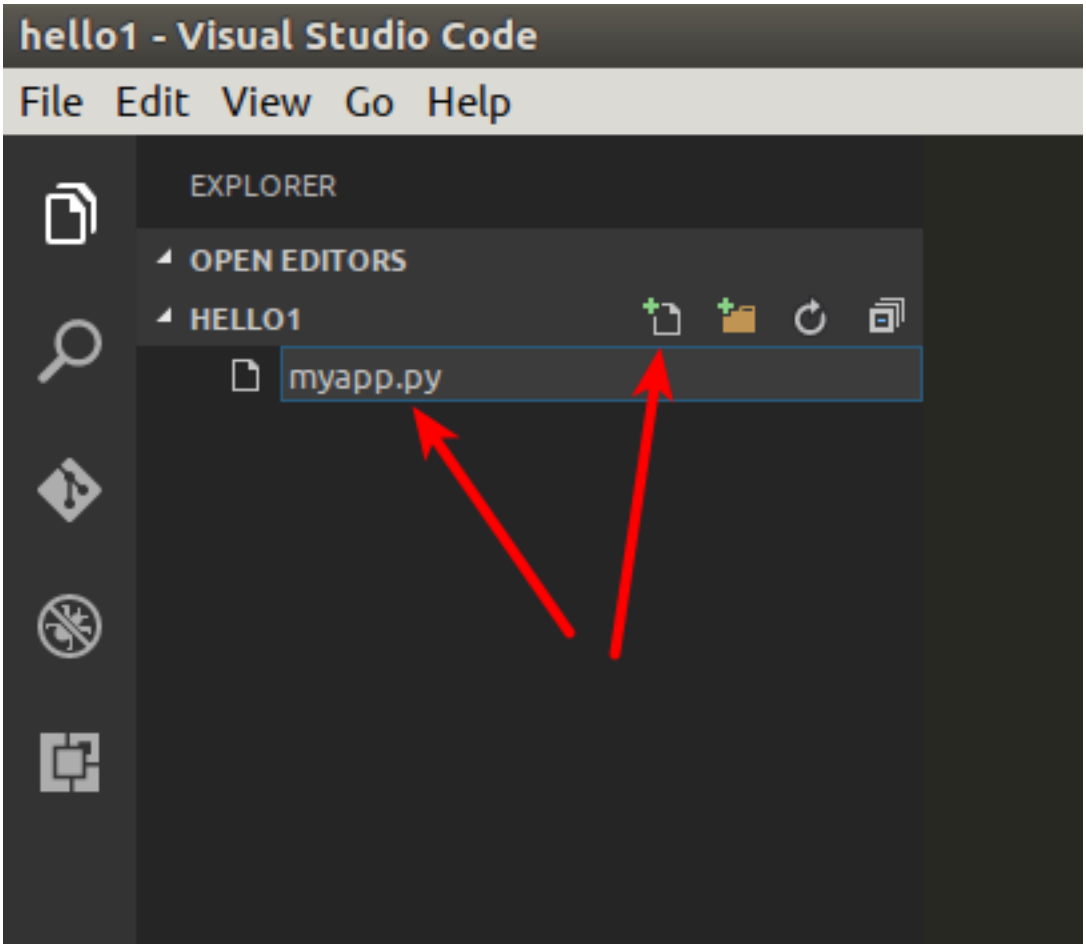
См.также:

http://trypyramid.com

Visual Studio Code требует для отладки открывать не просто файл, а директорию. Это необходимо, чтобы в этом каталоге сохранить локальные настройки редактора. Такая директория будет считаться проектом для редактора.

Для примера, создадим директорию *hello1* и откроем в редакторе `File->Open Folder...`

Создадим в этой директории файл *myapp.py*:

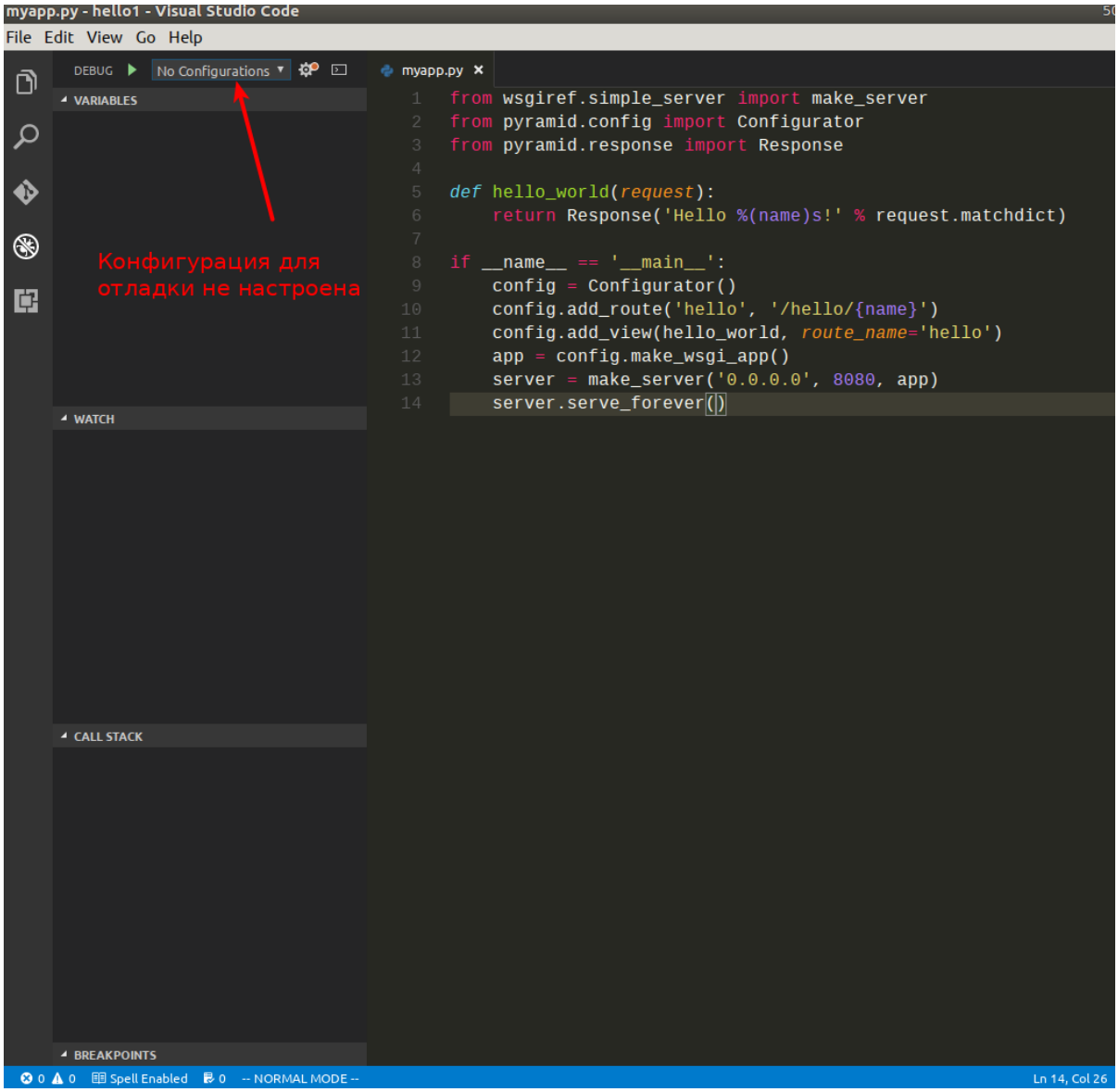


Добавим в файл пример с сайта http://trypyramid.com

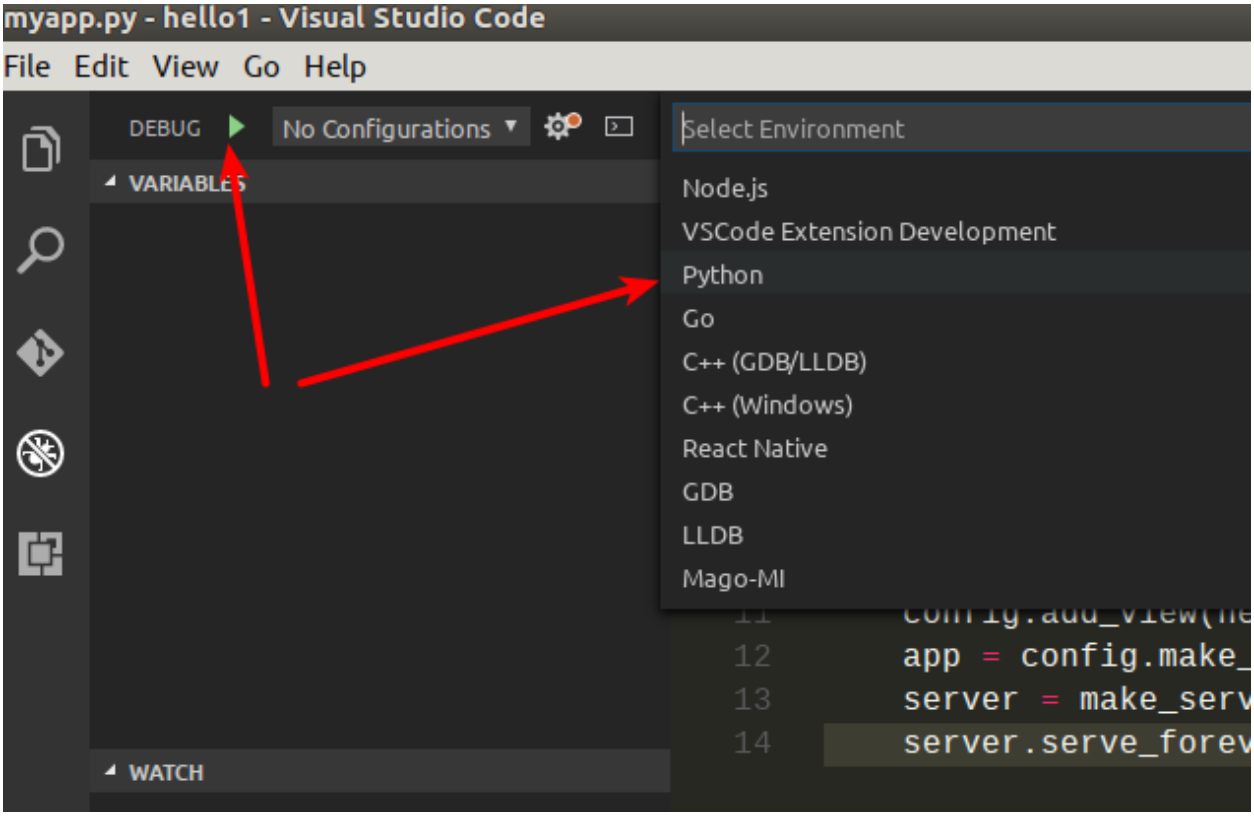
```
from wsgiref.simple_server import make_server
from pyramid.config import Configurator
from pyramid.response import Response

def hello_world(request):
    return Response('Hello %(name)s!' % request.matchdict)

config = Configurator()
config.add_route('hello', '/hello/{name}')
config.add_view(hello_world, route_name='hello')
app = config.make_wsgi_app()
server = make_server('0.0.0.0', 8080, app)
server.serve_forever()
```

Для запуска приложения, заходим в режим отладки по нажатию на кнопку:

.



Пока у нас нет никаких настроек отладки/запуска проекта, но при первом запуске редактор предложит их выбрать из существующих шаблонов.



Шаблон *Python* создает настройки в файле *launch.json* в локальной директории, которые выглядят примерно так:

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "Python",
            "type": "python",
            "request": "launch",
            "stopOnEntry": true,
            "pythonPath": "${config.python.pythonPath}",
            "program": "${file}",
            "debugOptions": [
                "WaitOnAbnormalExit",
                "WaitOnNormalExit",
                "RedirectOutput"
            ]
        },
        {
            "name": "Python Console App",
            "type": "python",
            "request": "launch",
            "stopOnEntry": true,
```

(continues on next page)

(продолжение с предыдущей страницы)

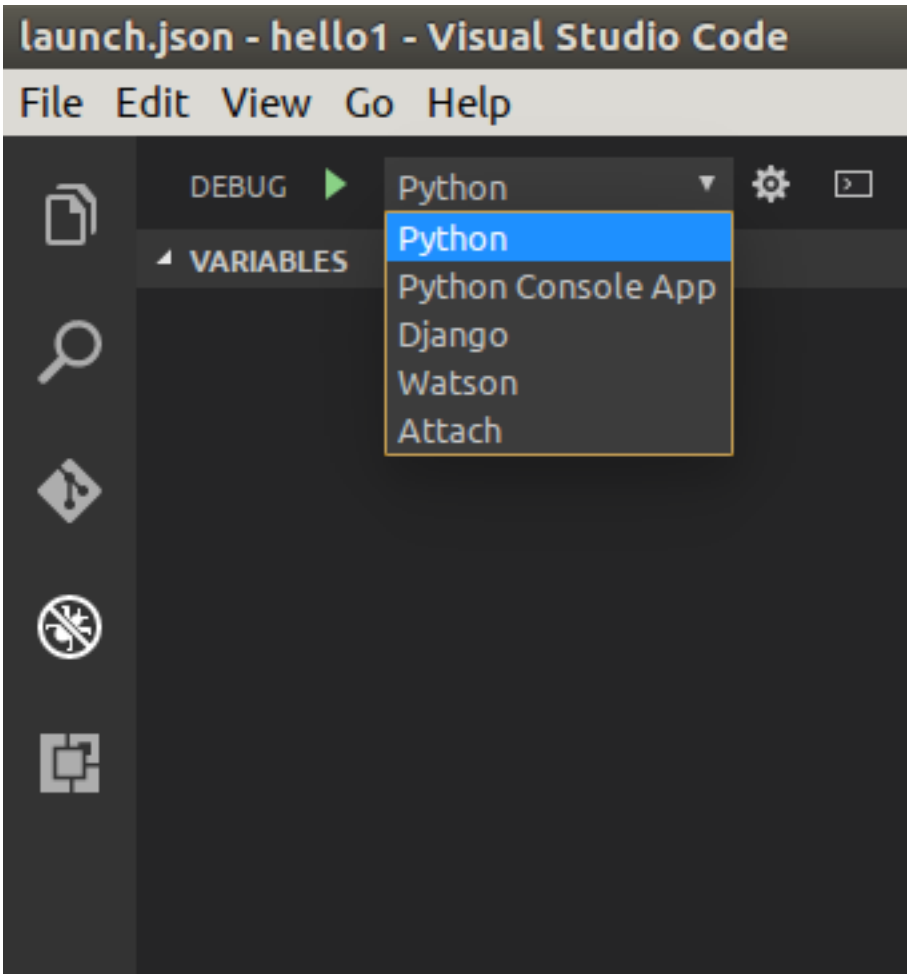
```
            "pythonPath": "${config.python.pythonPath}",
            "program": "${file}",
            "externalConsole": true,
            "debugOptions": [
                "WaitOnAbnormalExit",
                "WaitOnNormalExit"
            ]
        },
        {
            "name": "Django",
            "type": "python",
            "request": "launch",
            "stopOnEntry": true,
            "pythonPath": "${config.python.pythonPath}",
            "program": "${workspaceRoot}/manage.py",
            "args": [
                "runserver",
                "--noreload"
            ],
            "debugOptions": [
                "WaitOnAbnormalExit",
                "WaitOnNormalExit",
                "RedirectOutput",
                "DjangoDebugging"
            ]
        },
        {
            "name": "Watson",
            "type": "python",
            "request": "launch",
            "stopOnEntry": true,
            "pythonPath": "${config.python.pythonPath}",
            "program": "${workspaceRoot}/console.py",
            "args": [
                "dev",
                "runserver",
                "--noreload=True"
            ],
            "debugOptions": [
                "WaitOnAbnormalExit",
                "WaitOnNormalExit",
                "RedirectOutput"
            ]
        },
        {
```

(continues on next page)

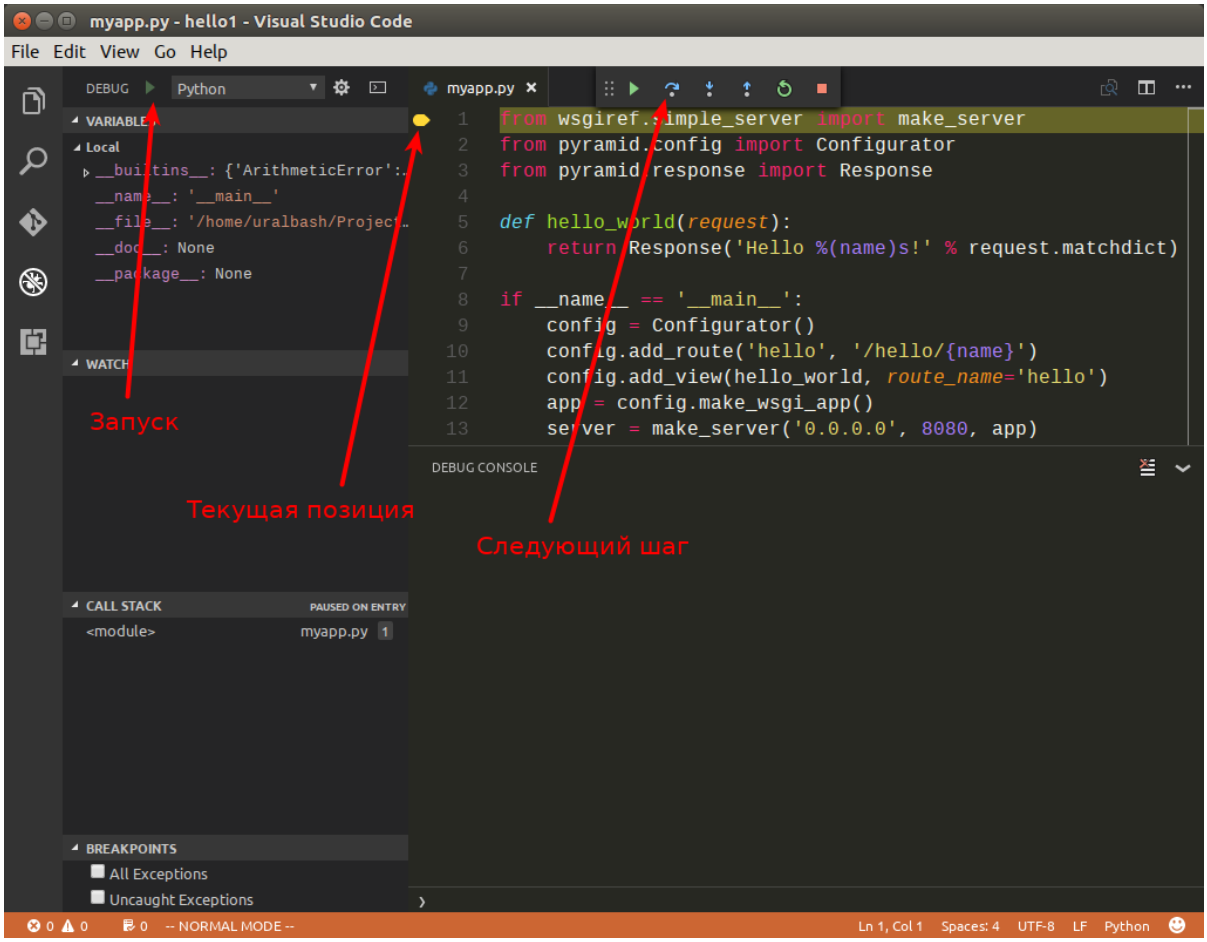
(продолжение с предыдущей страницы)

```
            "name": "Attach",
            "type": "python",
            "request": "attach",
            "localRoot": "${workspaceRoot}",
            "remoteRoot": "${workspaceRoot}",
            "port": 3000,
            "secret": "my_secret",
            "host": "localhost"
        }
    ]
}
```

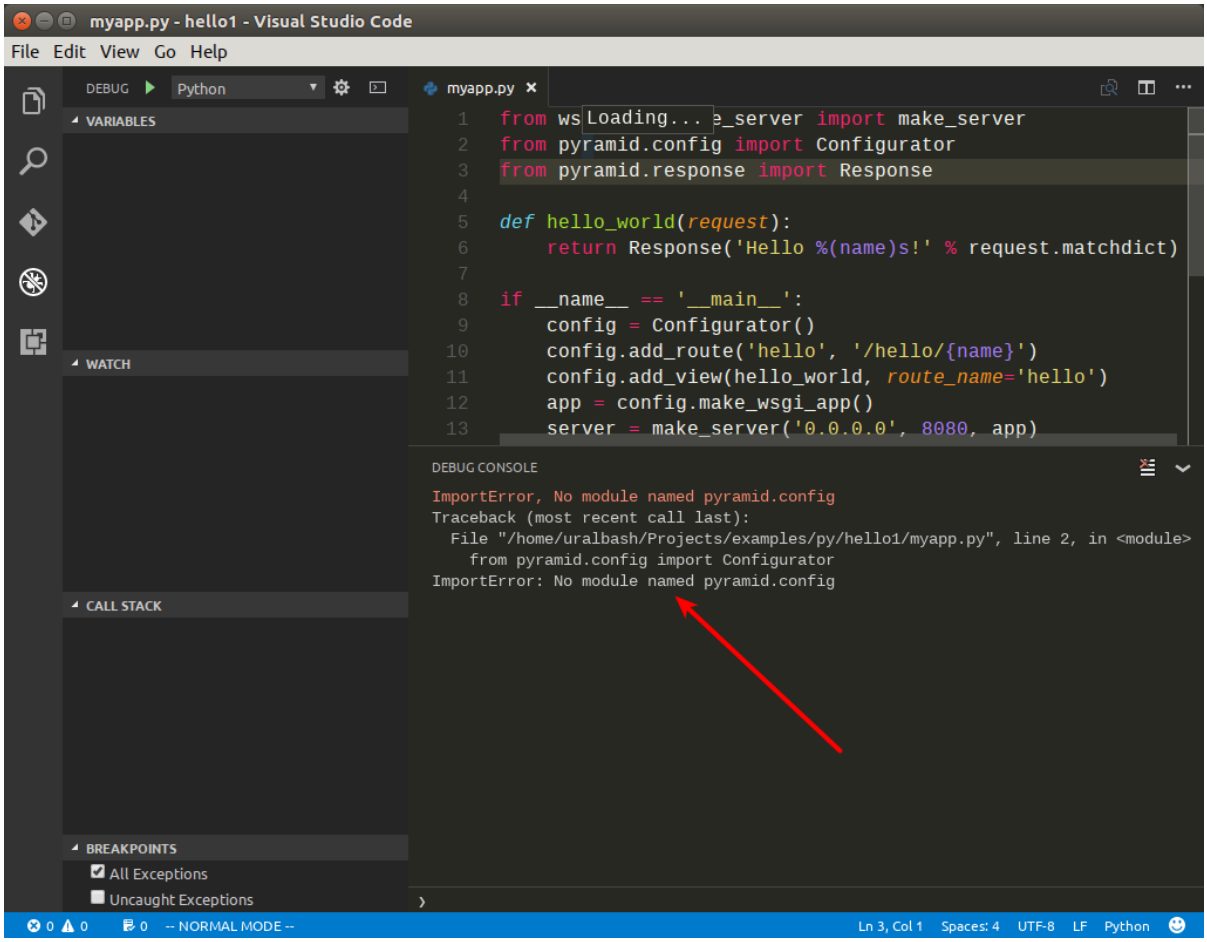
Это универсальный шаблон, который добавляет несколько вариантов запуска приложений. Нас будет интересовать первый вариант `Python`, просто запускающий python файл.



Запущенное приложение останавливается на первой строчке, что позволяет нам продолжать выполнение программы по шагам.



После выполнения второй строки, интерпретатор выдаст ошибку `ImportError: No module named pyramid.config`. Это происходит из-за того что в нашем *Python* окружении не установлен модуль *pyramid*.



Решить эту проблему можно двумя способами:

1. Установить *Pyramid* в глобальное окружение.

```
$ pip install --user pyramid
```

2. Создать виртуальное окружение, установить в нем *Pyramid* и прописать его в настройках Visual Studio Code.

См.также:

Как создать *Виртуальное окружение*

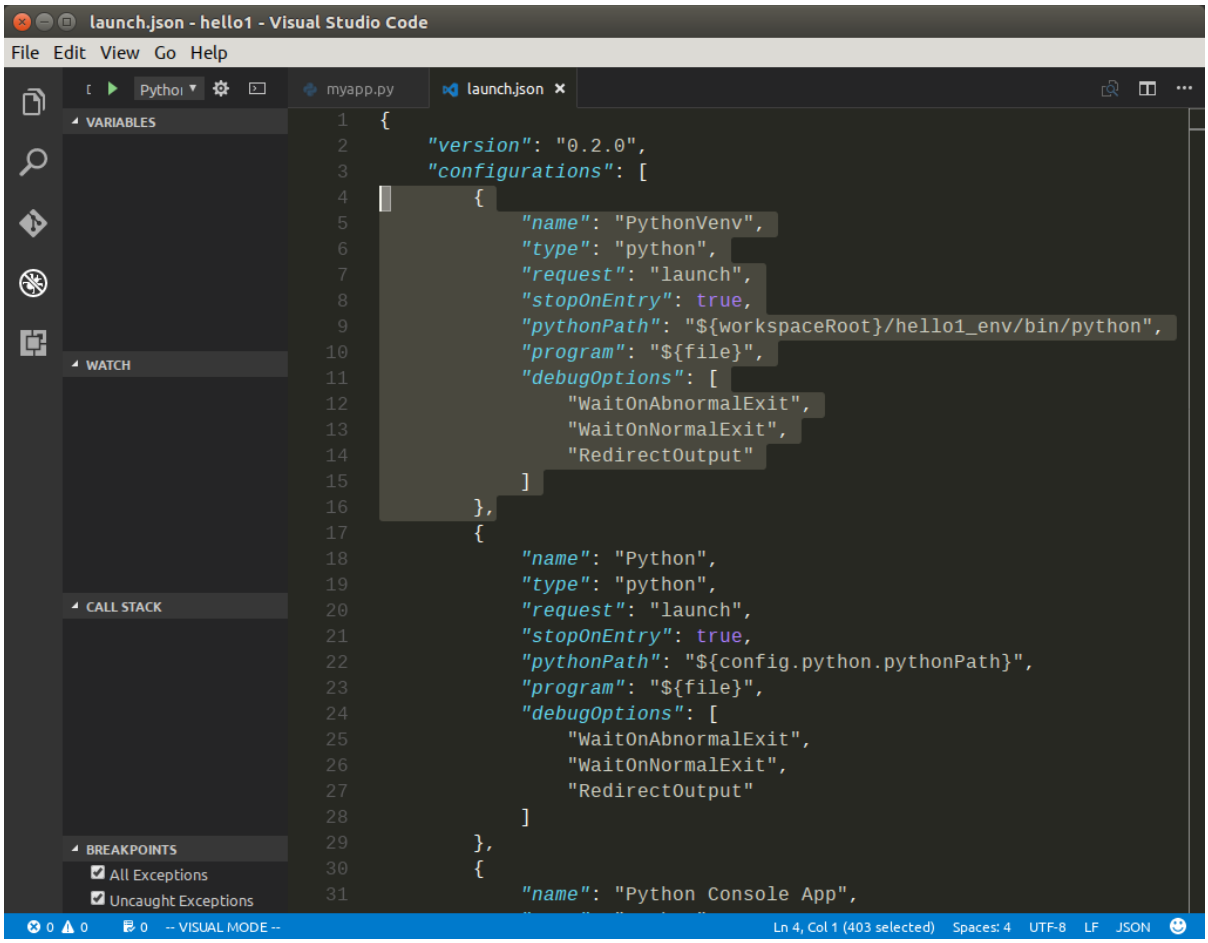
- Создаем виртуальное окружение:

```
$ cd /path/to/hello1/
$ pyvenv hello1_env
$ source ./hello1_env/bin/activate
```

- Устанавливаем *Pyramid*:

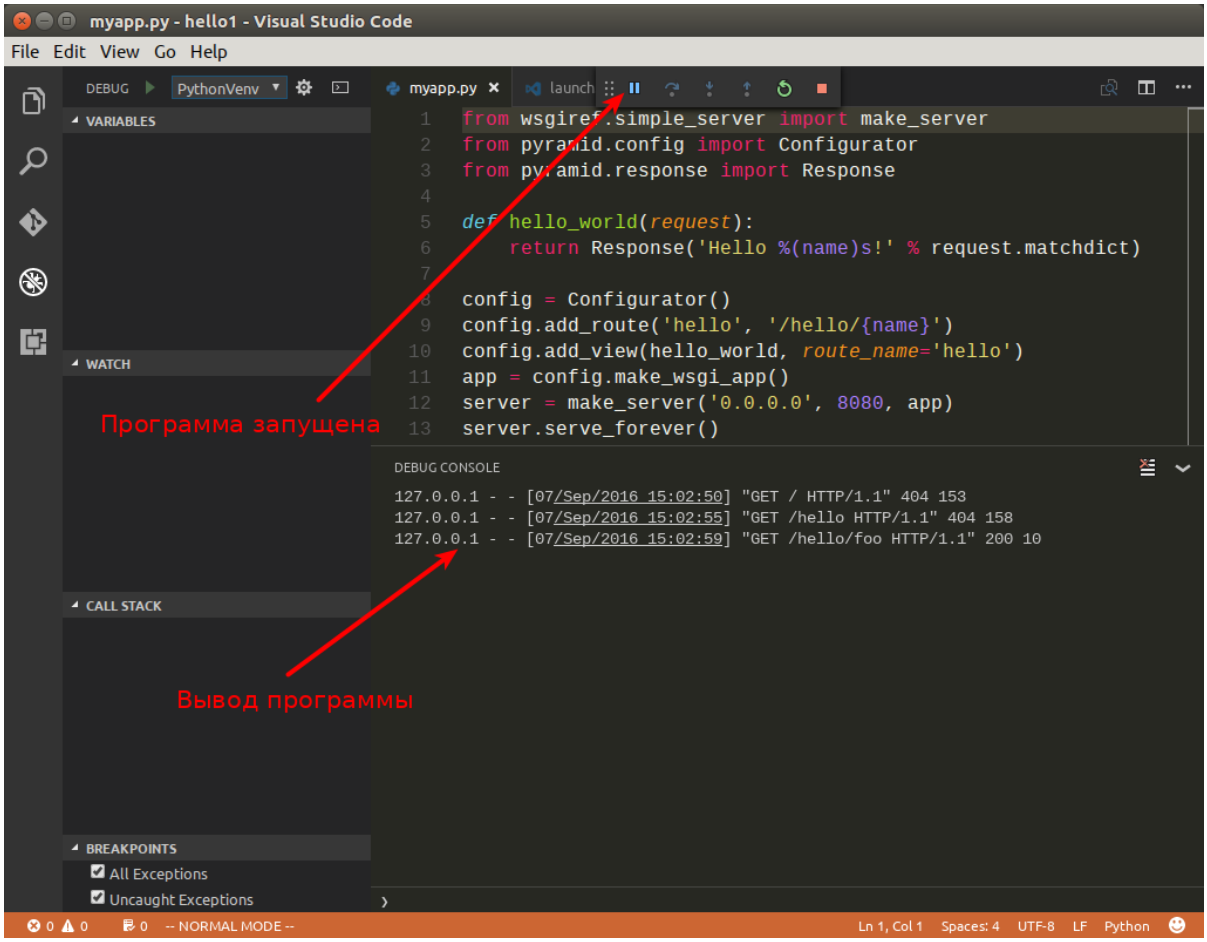
```
(hello1_env)$ pip install pyramid
```

- Прописываем путь до виртуального окружения в настройках проекта Visual Studio Code (файл *launch.json*):

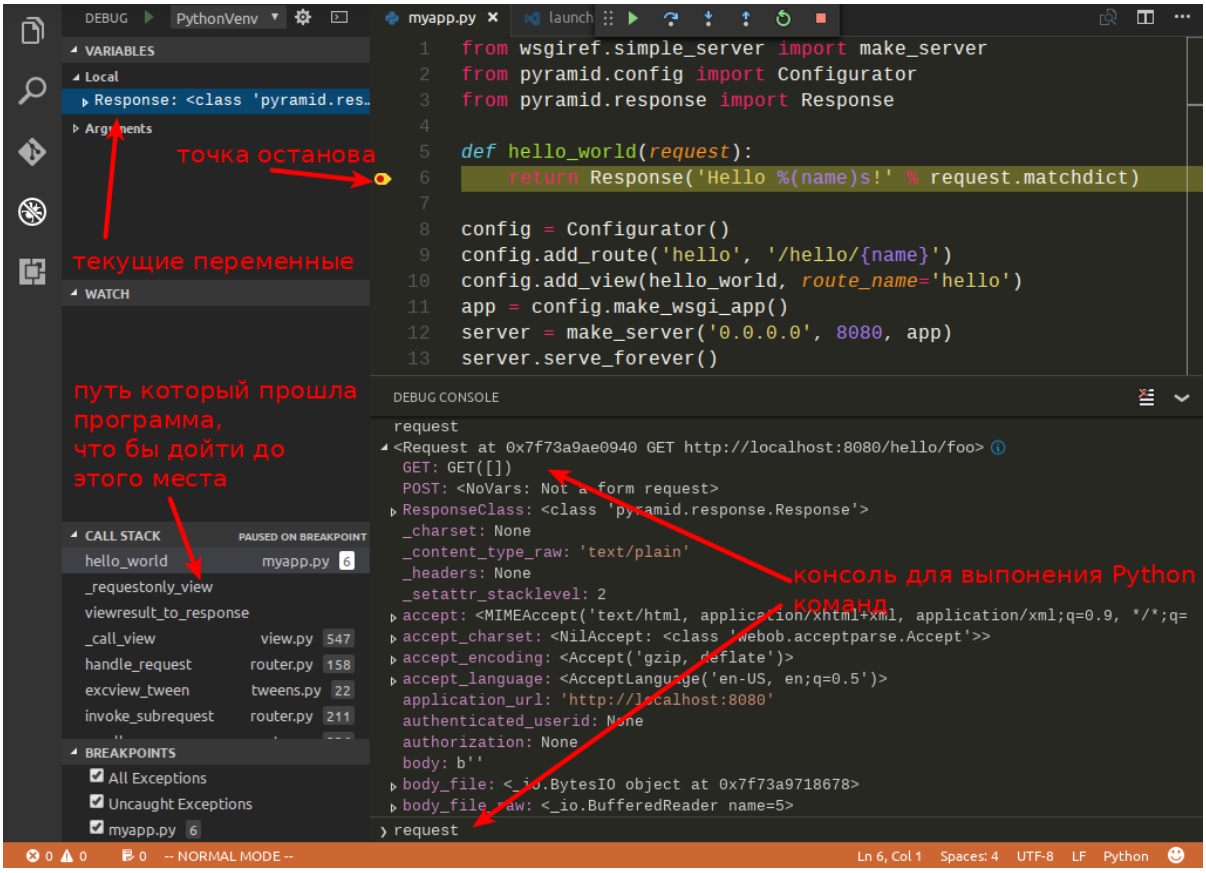



```
{
    "name": "PythonVenv",
    "type": "python",
    "request": "launch",
    "stopOnEntry": true,
    "pythonPath": "${workspaceRoot}/hello1_env/bin/python",
    "program": "${file}",
    "debugOptions": [
        "WaitOnAbnormalExit",
        "WaitOnNormalExit",
        "RedirectOutput"
    ]
}
```

После этого появится возможность запускать наш скрипт в локальном виртуальном окружении. Запущенная программа будет доступна по адресу http://localhost:8080/hello/foo. В консоле отладчика можно наблюдать ее вывод.

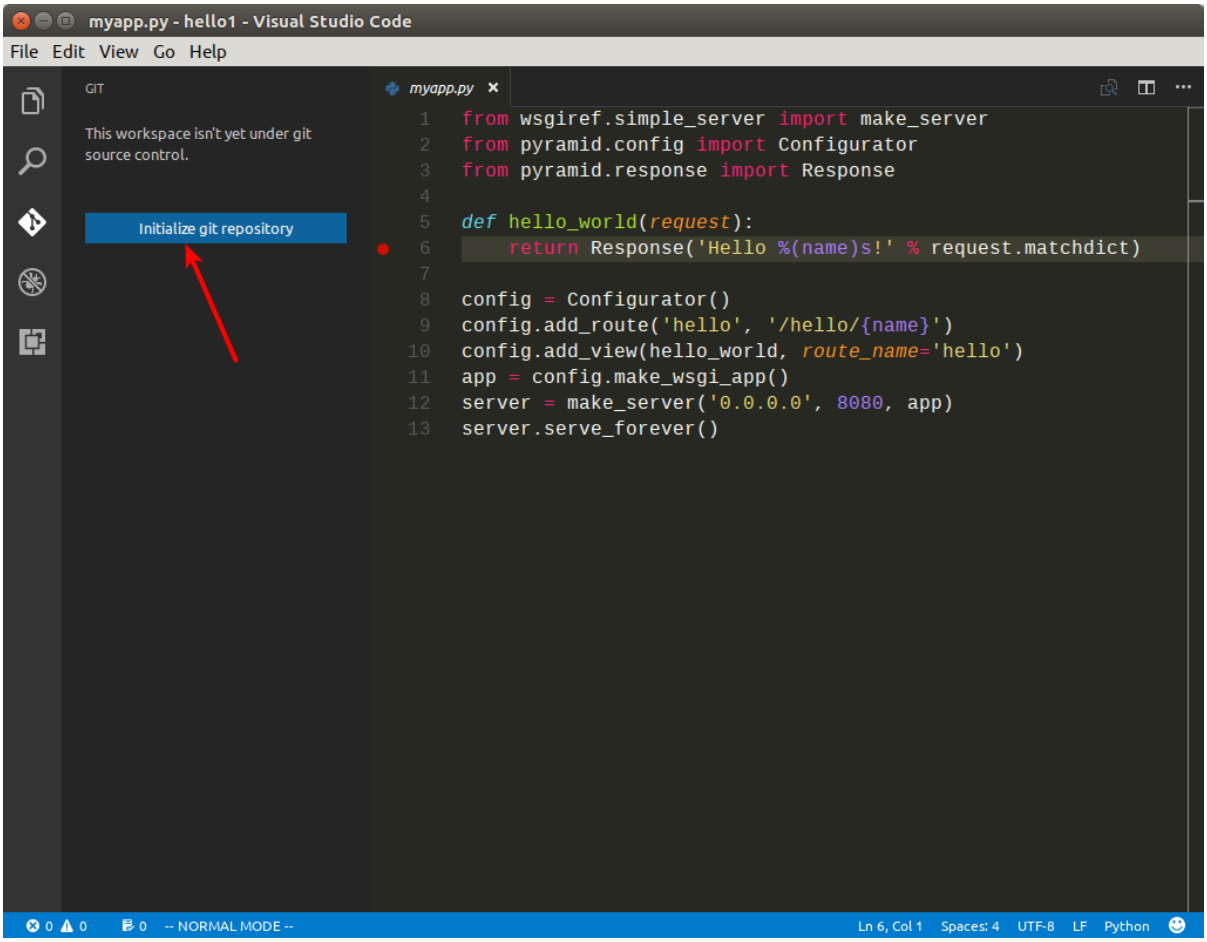


Поставим точку останова внутри функции `hello_world`, в строке 6. Это позволит нам остановить программу при запуске этой функции. После запуска, программа будет нормально работать, пока мы не зайдем по адресу http://localhost:8080/hello/foo, в этом случае запустится функция `hello_world` и выполнение программы прервется, до тех пор пока мы ее не продолжим вручную.



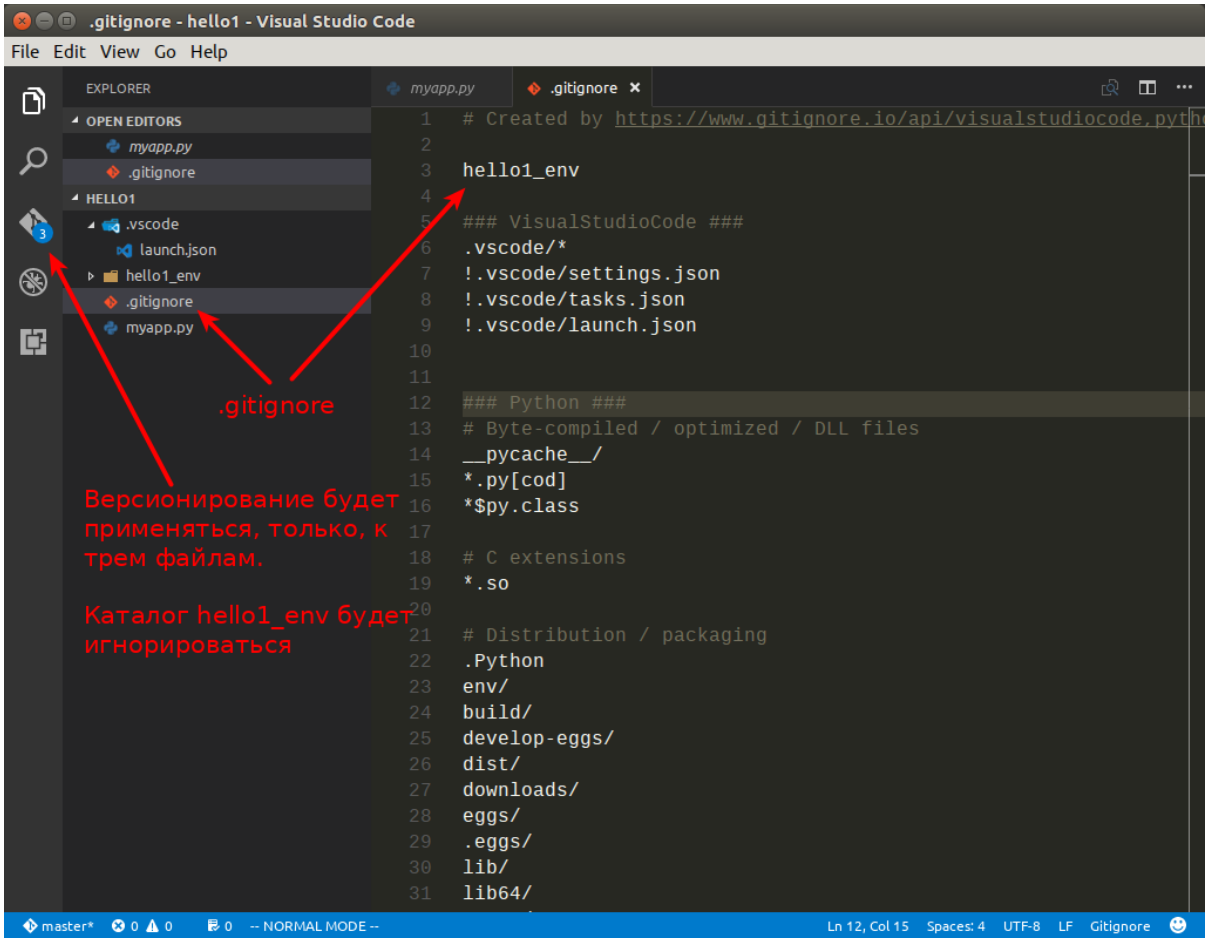
Примерно так выглядит процесс разработки и отладки программ на *Python*. Осталось только инициализировать *git* репозиторий и выложить проект на https://github.com.

1. Инициализируем репозиторий:



2. Добавим файл `.gitignore`:

   Для этого нам потребуется скопировать содержимое https://www.gitignore.io/api/visualstudiocode,python в файл `.gitignore` и добавить туда директорию `hello1_env`, чтобы она не участвовала в процессе создания версий.

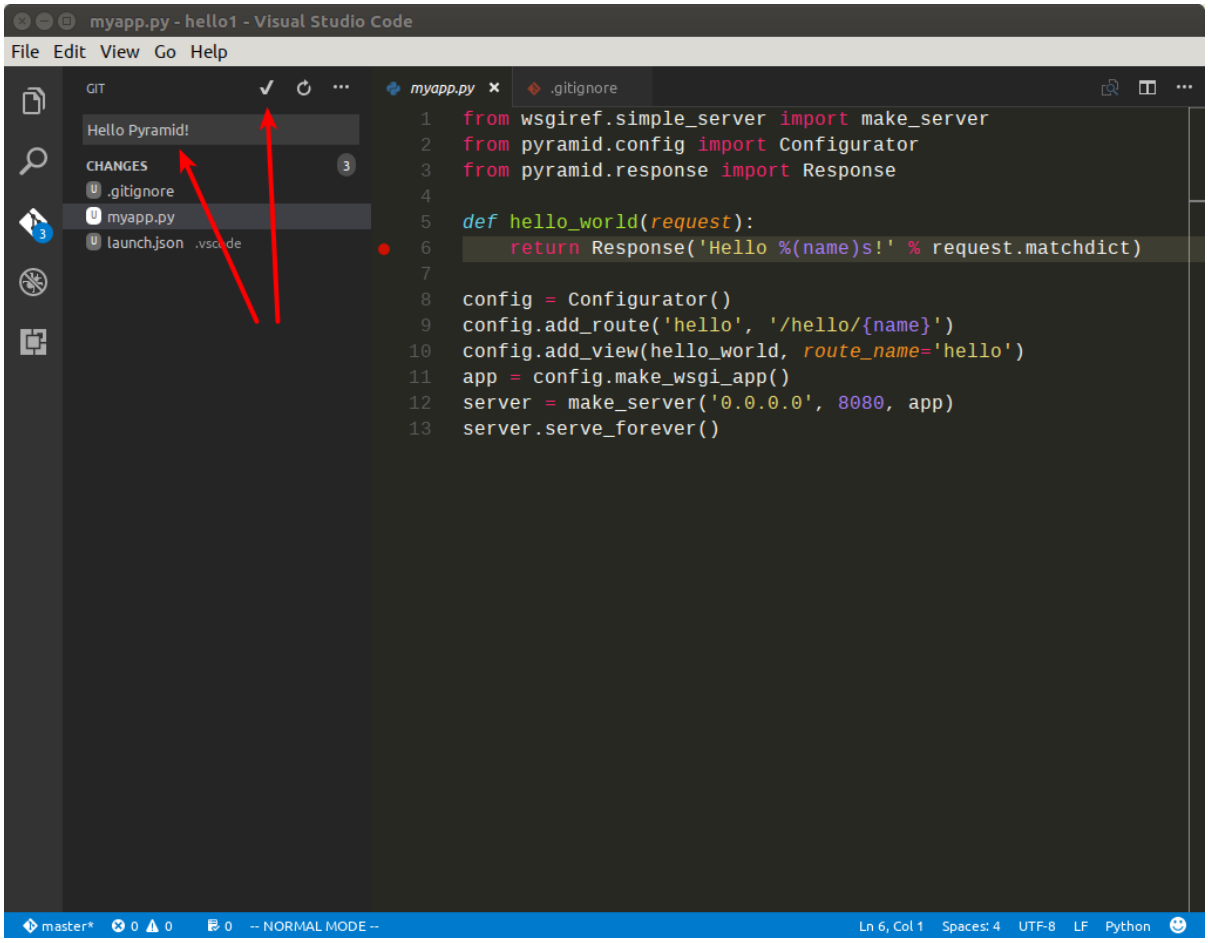



```
# Created by https://www.gitignore.io/api/visualstudiocode,python

hello1_env

### VisualStudioCode ###
.vscode/*
!.vscode/settings.json
!.vscode/tasks.json
!.vscode/launch.json


### Python ###
# Byte-compiled / optimized / DLL files
__pycache__/
*.py[cod]

...
```

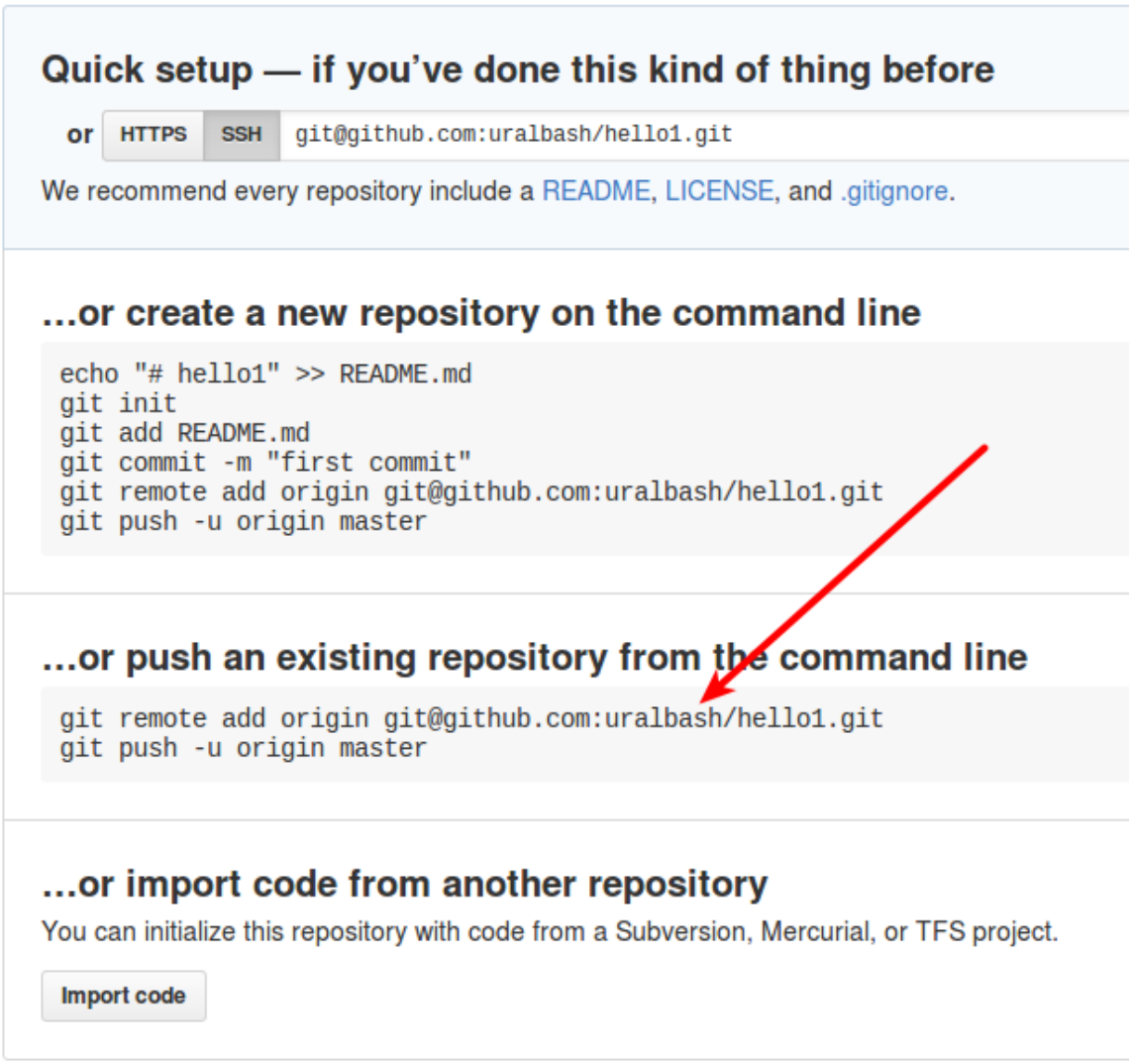
3. Создаем первый коммит

   Для создания коммита требуется ввести комментарий и нажать на кнопку в виде

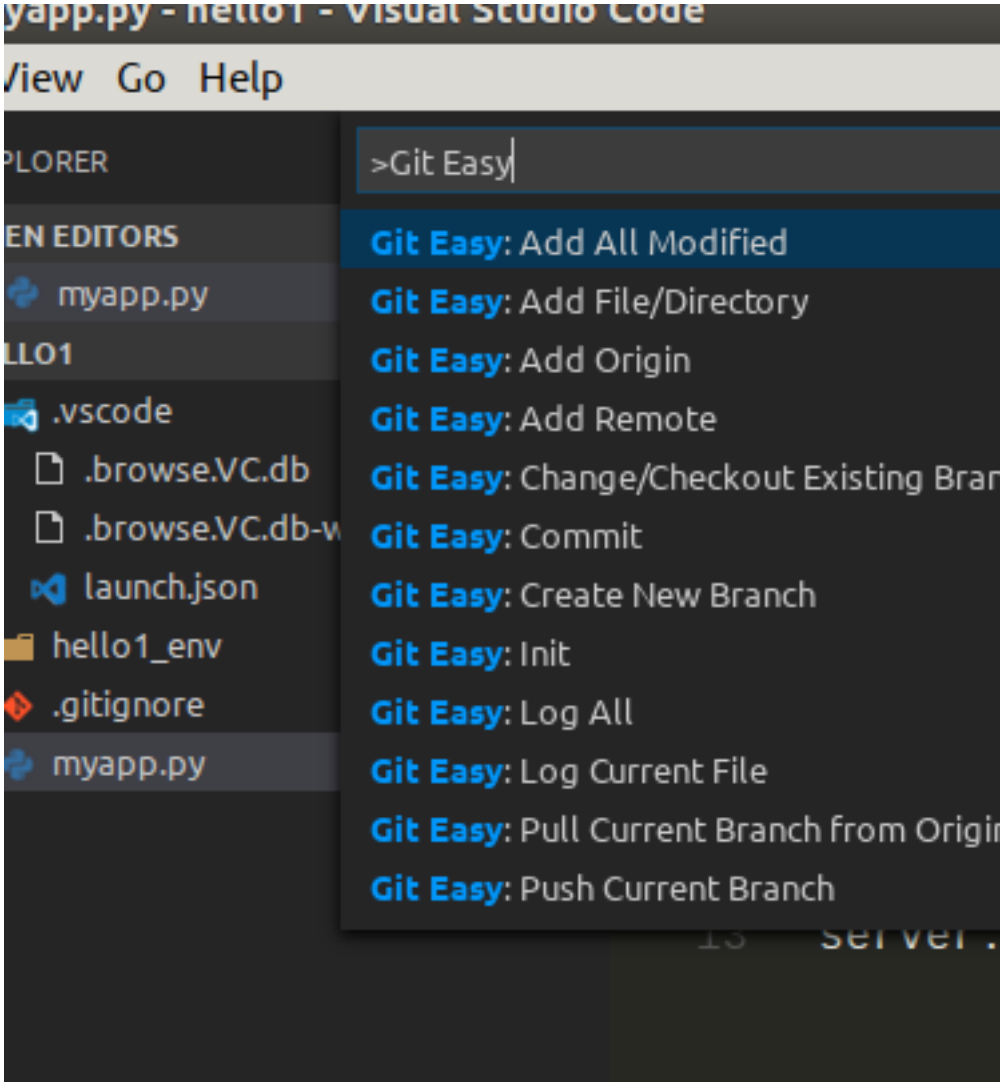
галочки.

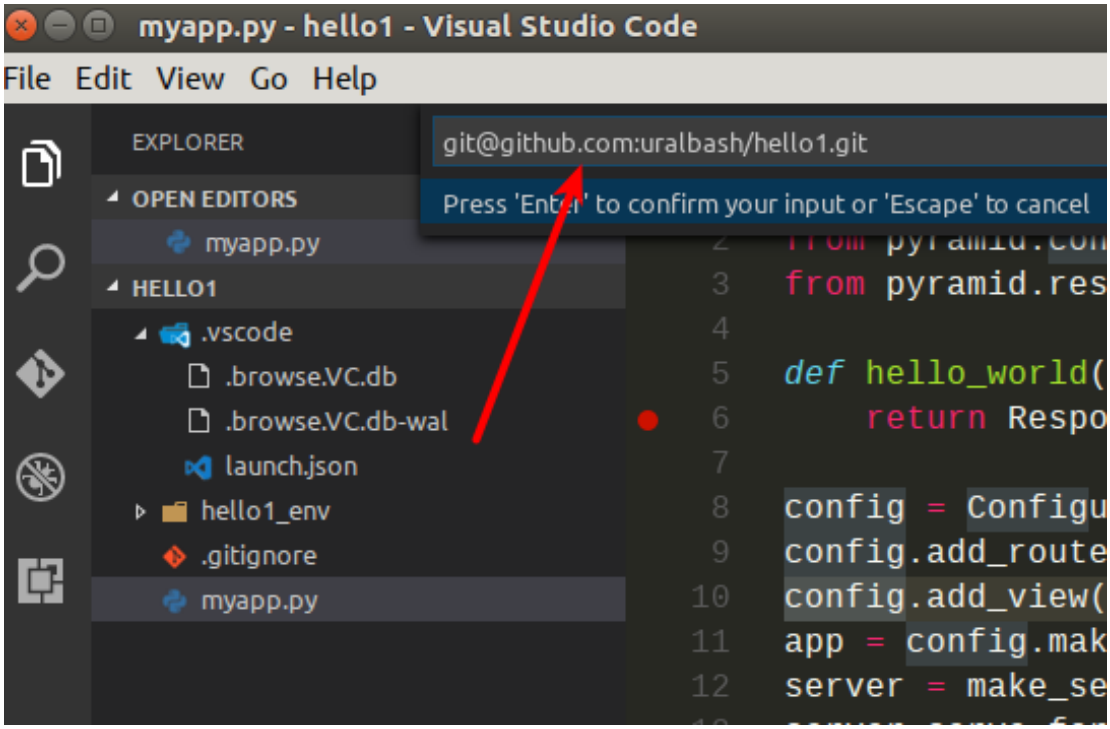


4. Отправляем изменения на https://github.com
   - Добавляем плагин *Git Easy* в проект
   - Создаем репозиторий на GitHub

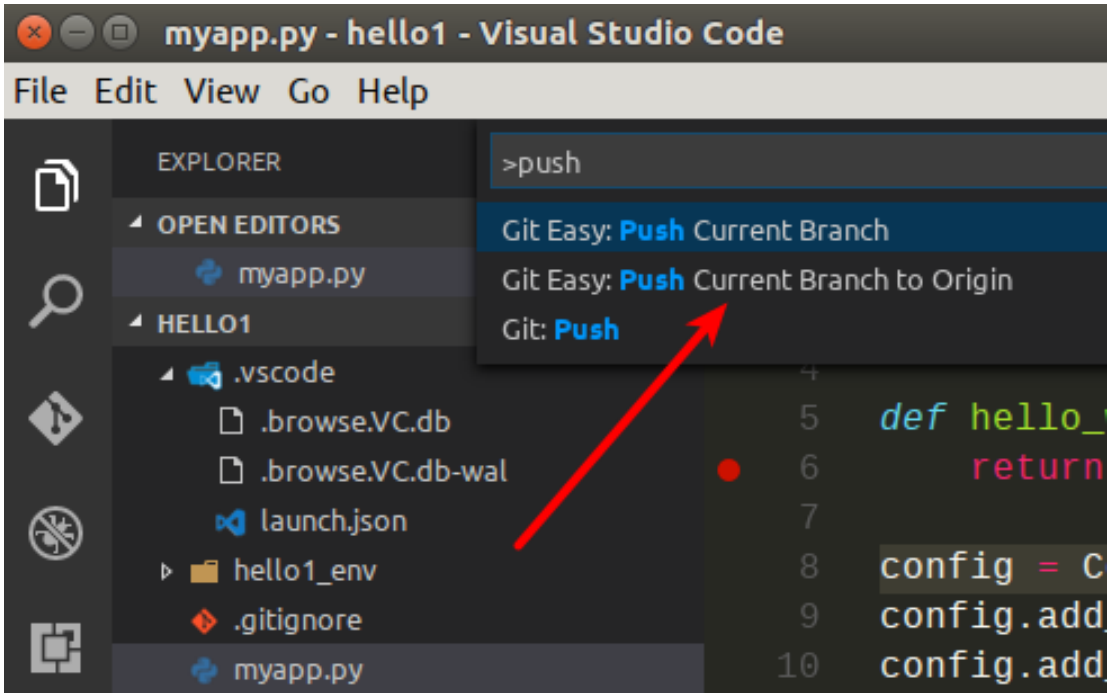


- Прописываем путь до гитхаба в нашем проекте, при помощи команды `Git Easy:Add Orign`


View Go Help
>Git Easy
Git Easy: Add All Modified
Git Easy: Add File/Directory
Git Easy: Add Origin
Git Easy: Add Remote
Git Easy: Change/Checkout Existing Bran
Git Easy: Commit
Git Easy: Create New Branch
Git Easy: Init
Git Easy: Log All
Git Easy: Log Current File
Git Easy: Pull Current Branch from Origi
Git Easy: Push Current Branch
myapp.py
.vscode
.browse.VC.db
launch.json
hello1_env
.gitignore
myapp.py

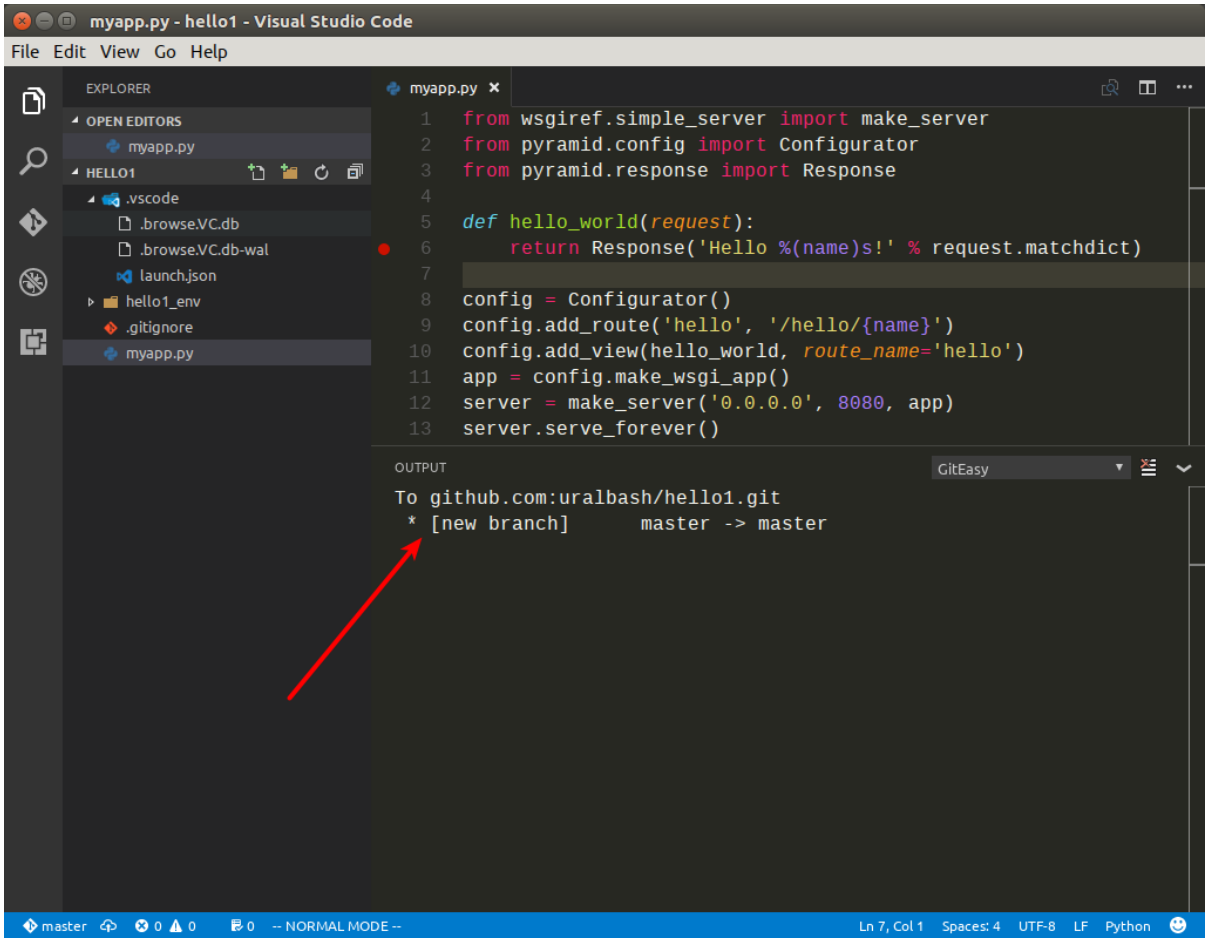


- Отправляем изменения на *GitHub*, при помощи команды `Git Easy:Push Current Branch to Origin`

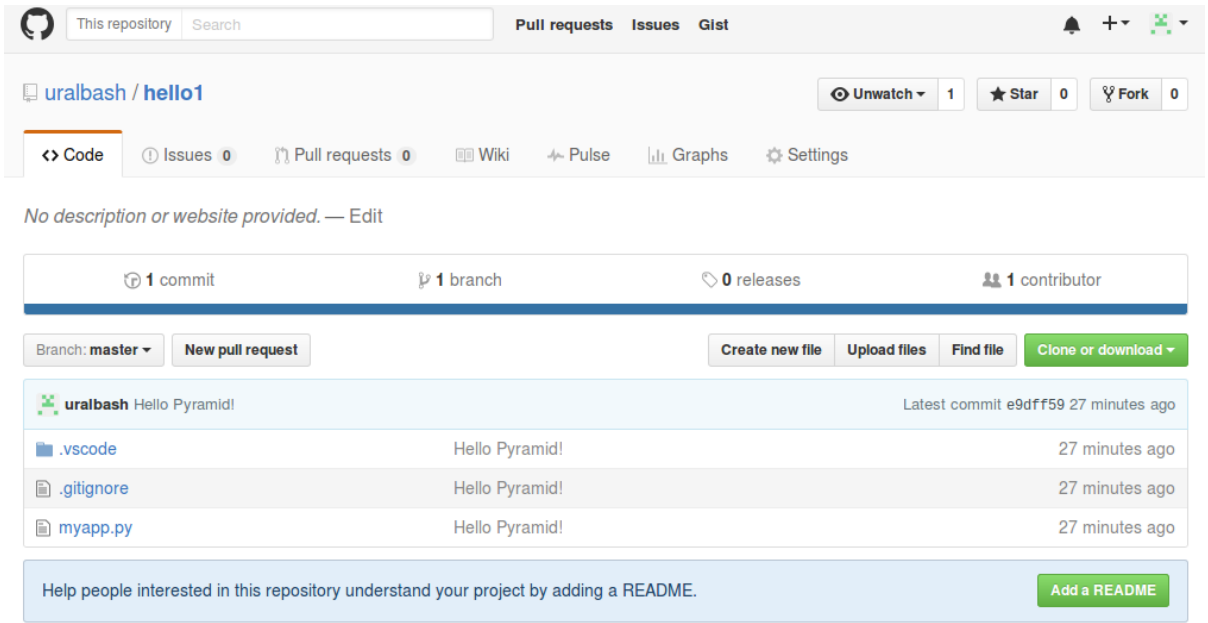


При успешном выполнении команды, мы должны увидеть сообщение типа:

```
To github.com:uralbash/hello1.git
* [new branch]      master -> master
```



Файлы будут доступны по адресу https://github.com/uralbash/hello1



Для того чтобы проверка синтаксиса заработала, необходимо создать файл `.vscode/`

settings.json и переопределить в нем глобальные настройки для нашего проекта:

```
{
    "editor.fontSize": 18,

    //Python
    "python.pythonPath": "${workspaceRoot}/hello1_env/bin/python",

    // Whether to lint Python files using pylint.
    "python.linting.pylintEnabled": true,

    // Whether to lint Python files using pep8
    "python.linting.pep8Enabled": true,

    // Whether to lint Python files using flake8
    "python.linting.flake8Enabled": true
}
```

Pyramid

См.также:

http://docs.pylonsproject.org/projects/pyramid/en/latest/narr/project.html

Фреймворк *Pyramid* имеет несколько стартовых шаблонов, которые нужны для того, чтобы не начинать писать код с нуля. Рассмотрим как создать шаблон с БД *sqlite* + *SQLAlchemy* и настроить его в Visual Studio Code.

Для начала создадим директорию *hello2* и виртуальное окружение *hello2_ env*:

```
$ mkdir hello2
$ cd hello2/
$ pyvenv hello2_env
$ source hello2_env/bin/activate
$ pip install pyramid
```

См.также:

http://docs.pylonsproject.org/projects/pyramid/en/latest/pscripts/index.html

После установки *Pyramid*, в окружении появляется команда pcreate. С ее помощью создадим проект по шаблону:

```
$ pcreate -t alchemy .
$ ls
CHANGES.txt  development.ini  hello2  hello2_env  MANIFEST.in  production.ini ␣
↪pytest.ini  README.txt  setup.py
```

Устанавливаем его как *Python* пакет:

```
$ pip install -e .
$ pserve development.ini
Starting server in PID 17311.
Serving on http://localhost:6543
```

После запуска, становится доступен адрес http://localhost:6543

```
Pyramid is having a problem using your SQL database.  The problem
might be caused by one of the following things:

1.  You may need to run the "initialize_hello2_db" script
    to initialize your database tables.  Check your virtual
    environment's "bin" directory for this script and try to run it.

2.  Your database server may not be running.  Check that the
    database server referred to by the "sqlalchemy.url" setting in
    your "development.ini" file is running.

After you fix the problem, please restart the Pyramid application to
try it again.
```

Но так-как БД еще не создана, отображается страница с подсказкой как ее инициализировать:

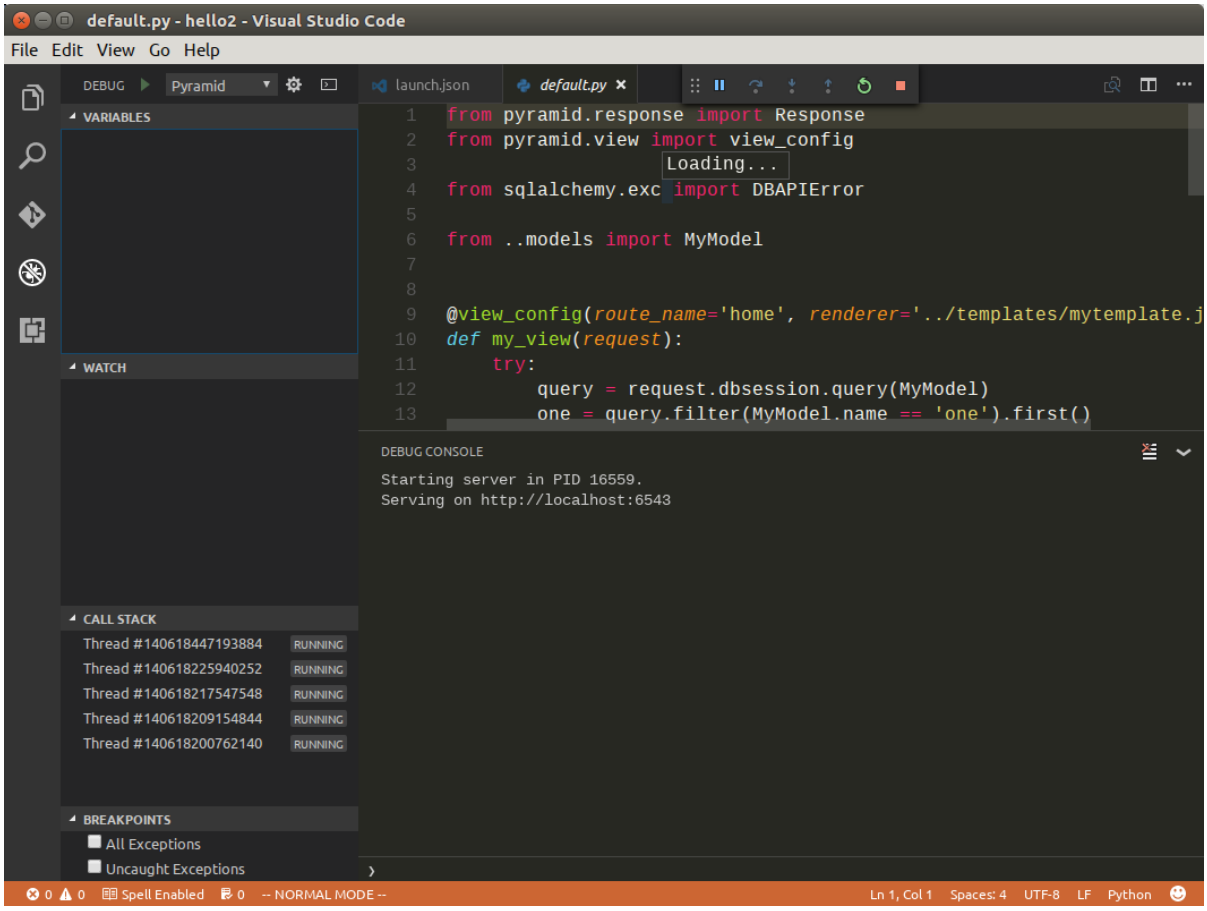
```
$ initialize_hello2_db development.ini
```

Теперь мы увидим стартовую страницу шаблона *alchemy*.

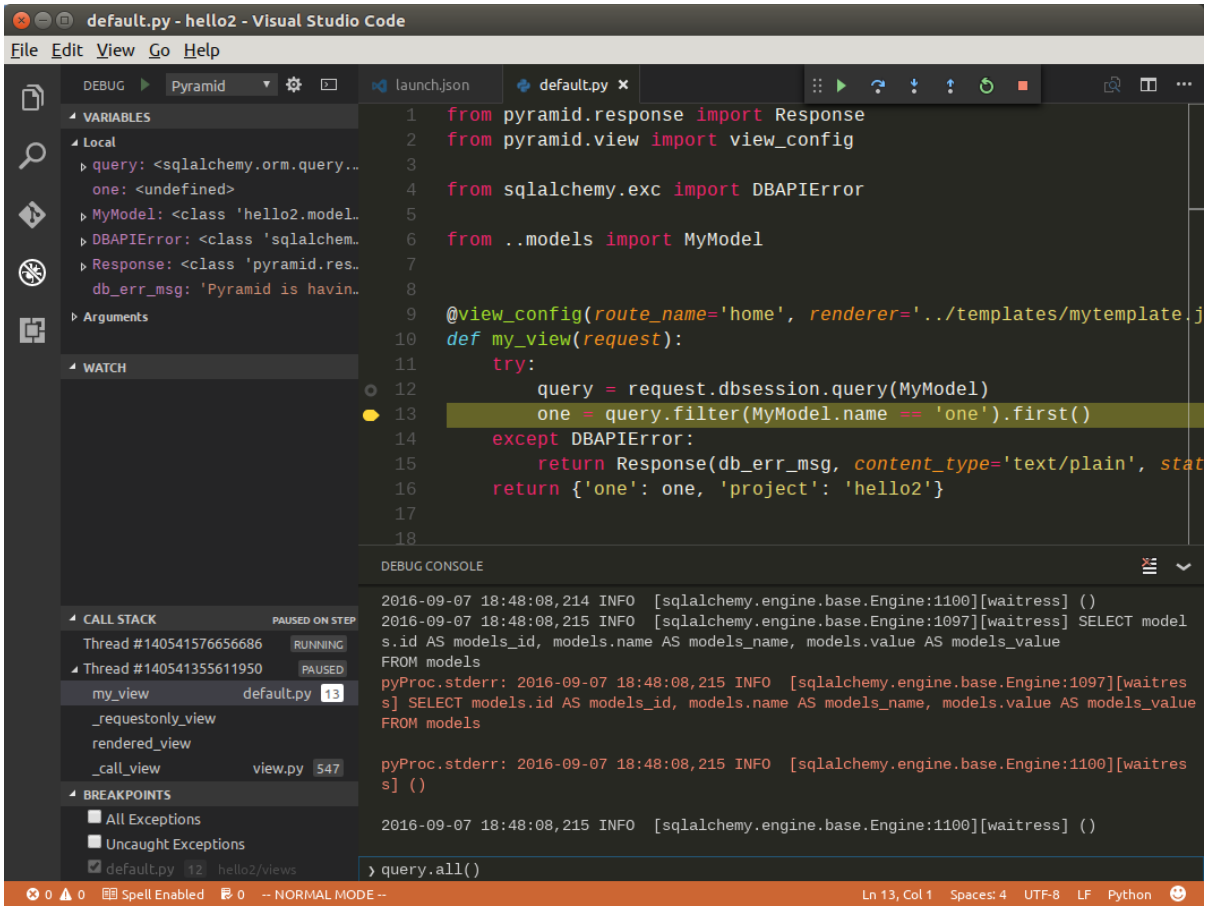
Проект на пирамиде запускается при помощи утилиты `pserve`. Добавим конфигурацию для *Pyramid* в файл настроек `launch.json`, чтобы можно было запускать/отлаживать приложение из редактора:

```
{
    "version": "0.2.0",
    "configurations": [{
        "name": "Pyramid",
        "type": "python",
        "request": "launch",
        "stopOnEntry": true,
        "pythonPath": "${workspaceRoot}/hello2_env/bin/python",
        "program": "${workspaceRoot}/hello2_env/bin/pserve",
        "args": ["${workspaceRoot}/development.ini"],
        "debugOptions": [
            "WaitOnNormalExit",
            "RedirectOutput"
        ]
    }]
}
```

Попробуем запустить:

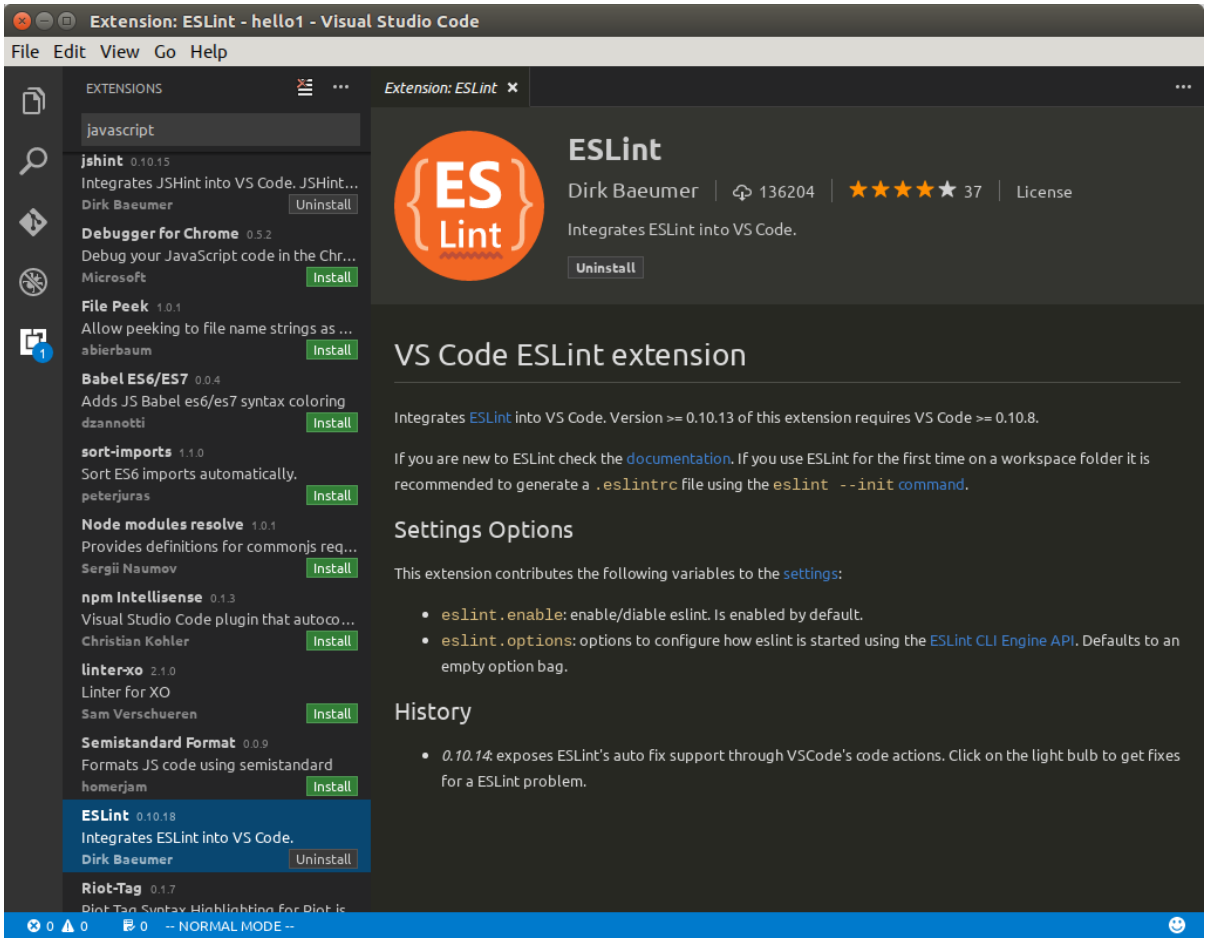


Поставим точку останова в функции `my_view` в файле `hello2/views/default.py`.



После обновления страницы http://localhost:6543 в браузере, программа остановит свое выполнение в этой точке, а браузер будет ждать пока мы не закончим отладку и не продолжим выполнение вручную.

JavaScript



4.3.2 Vim

4.3.3 Notepad++

Скачайте инсталятор https://notepad-plus-plus.org/download/ и установите редактор.

При наборе текста вы столкнетесь с проблемой, что нажатие на клавишу <Tab> вставляет символ табуляции заместо 4 пробелов, как это принято при написании Python программ.

Поменять это поведение можно в пункте меню `Опции->Настройки->Настройки Табуляции`.

В результате нажатие клавиши <Tab> будет подменяться четырьмя пробелами.

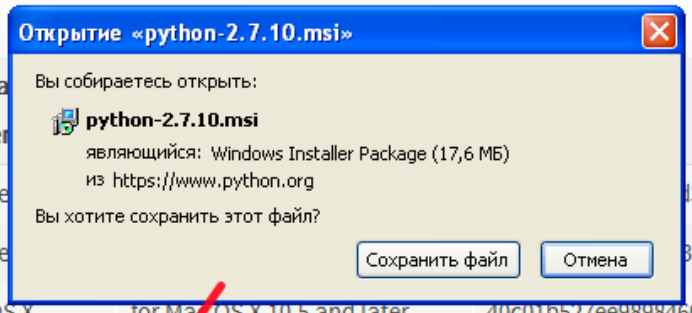
Python 2.7.10

Release Date: 2015-05-23

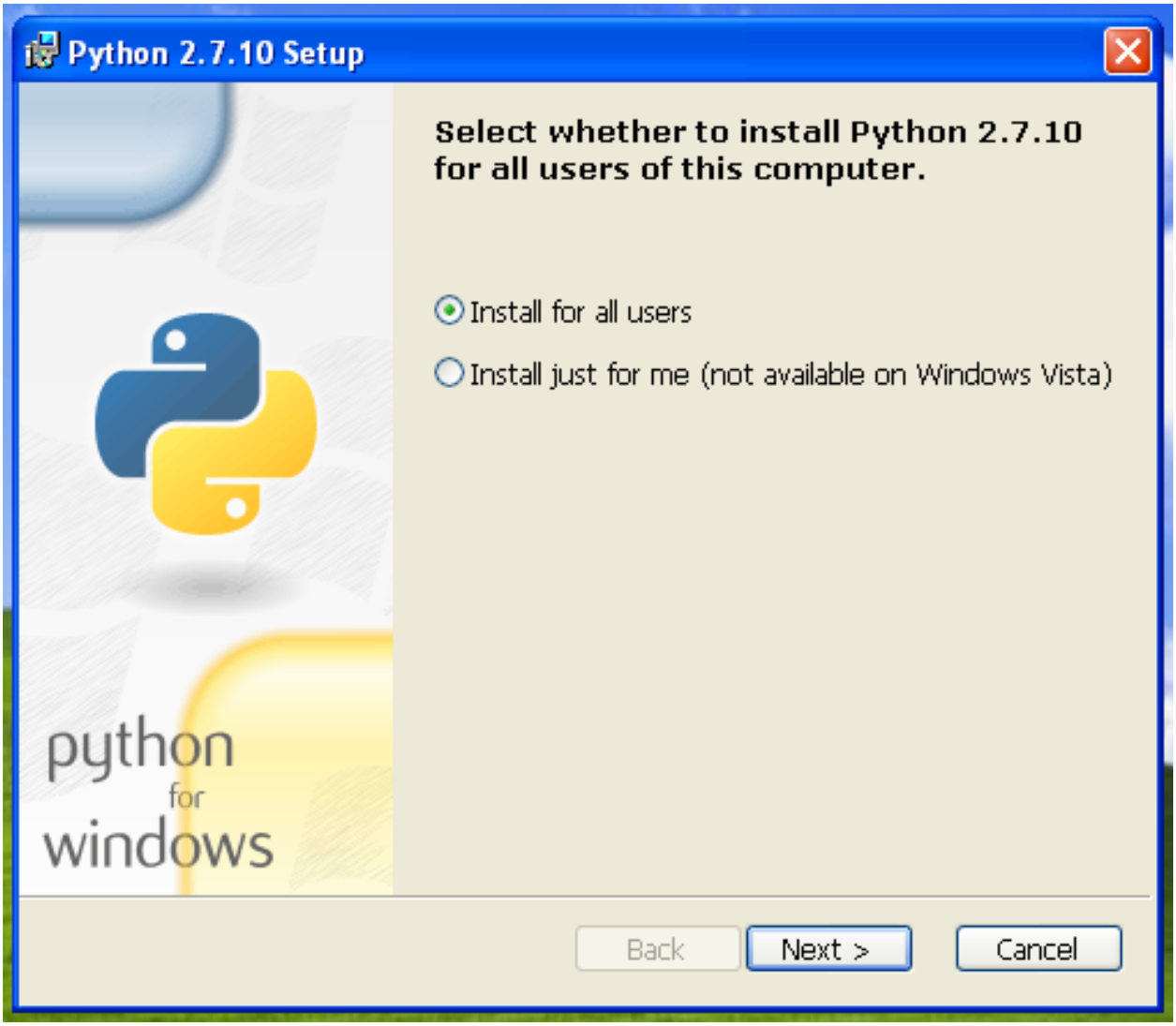
Python 2.7.10 is a bug fix release of the Python 2.7.x series.

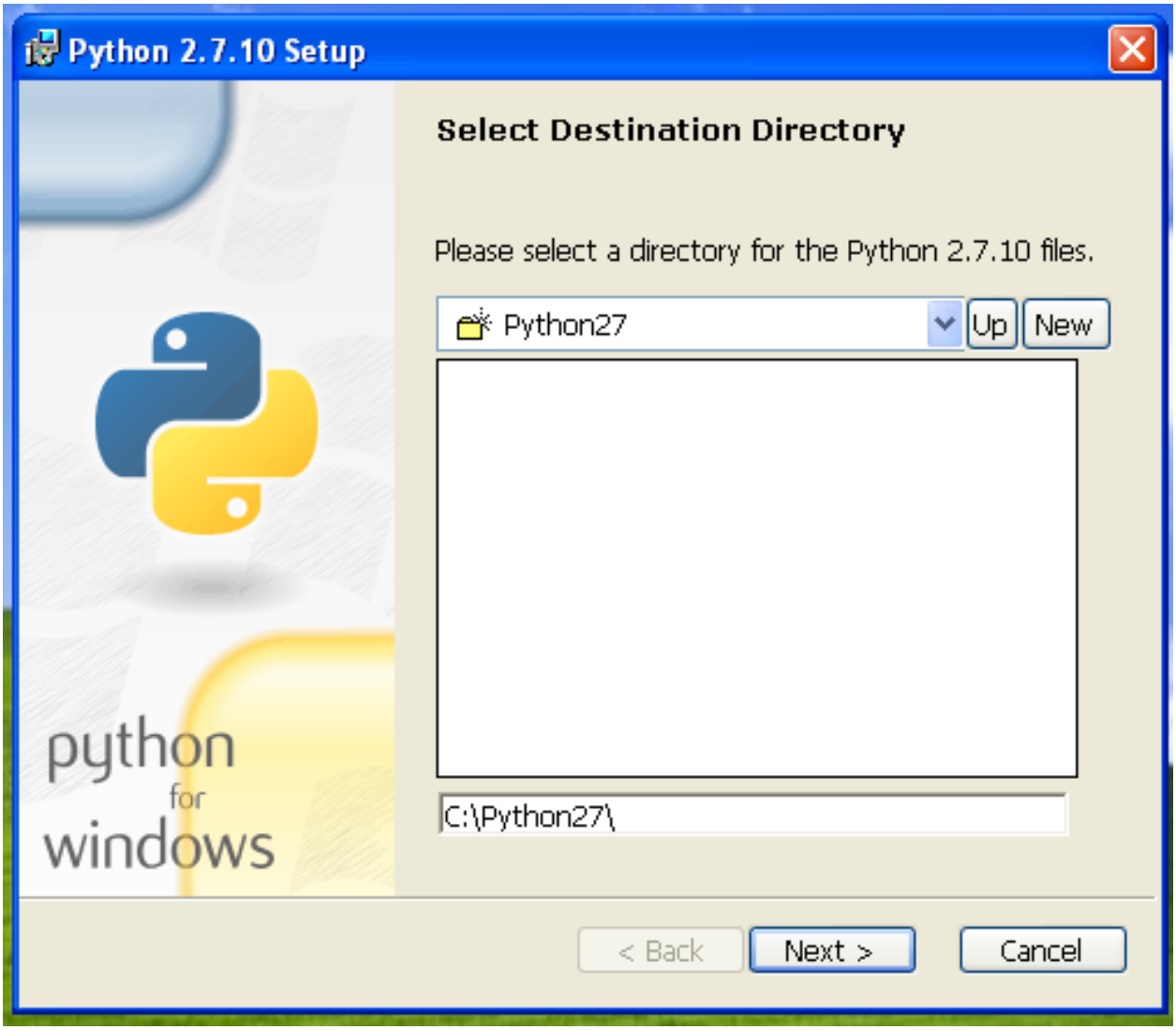
Full Changelog

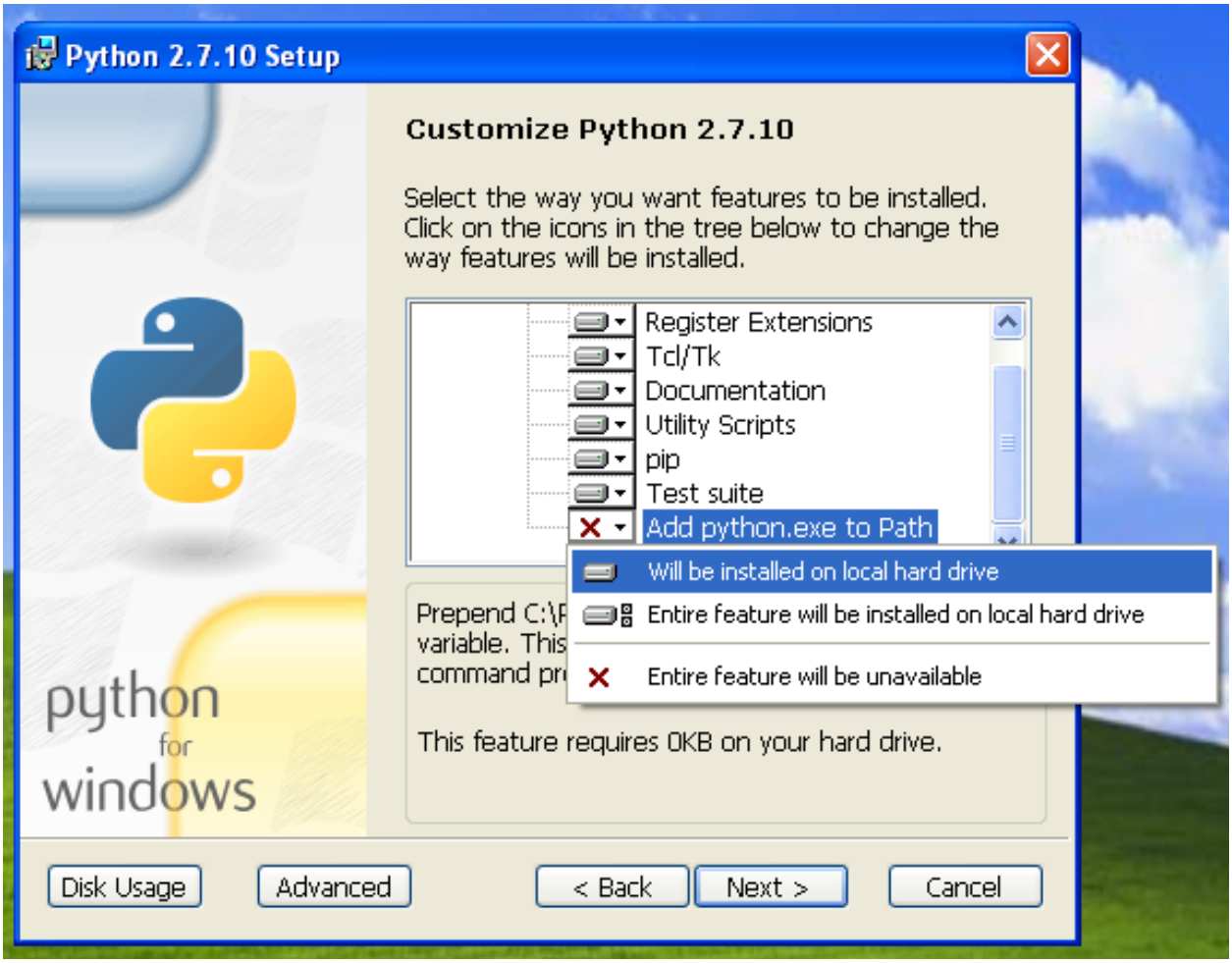
Files



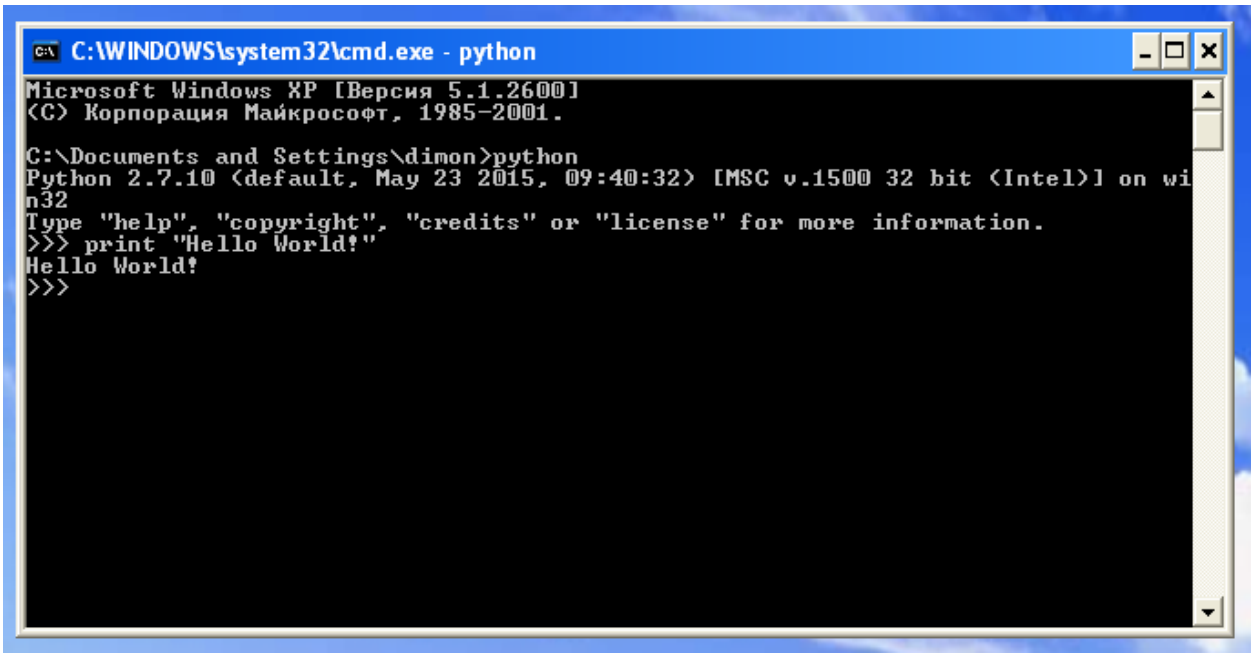
Version	Operating System		
Gzipped source tarball	Source		d38e2108c6b42521
XZ compressed source tarball	Source		3db5db12b268ac6
Mac OS X 32-bit i386/PPC installer	Mac OS X	for Mac OS X 10.5 and later	40c01b527ee9898460f8cd515f1c1651
Mac OS X 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later	3a5419361628c542f5fc28691eb7b773
Windows debug information files	Windows		44c155e72ddae4bfface20932ea2f5cf
Windows debug information files for 64-bit binaries	Windows		2460724a7ce7a736e7b5e3ee44879e53
Windows help file	Windows		5798437100884d987a57626e11d2c618
Windows x86-64 MSI installer	Windows	for AMD64/EM64T/x64, not Itanium processors	35f5c301beab341f6f6c9785939882ee
Windows x86 MSI installer	Windows		4ba2c79b103f6003bc4611c837a08208

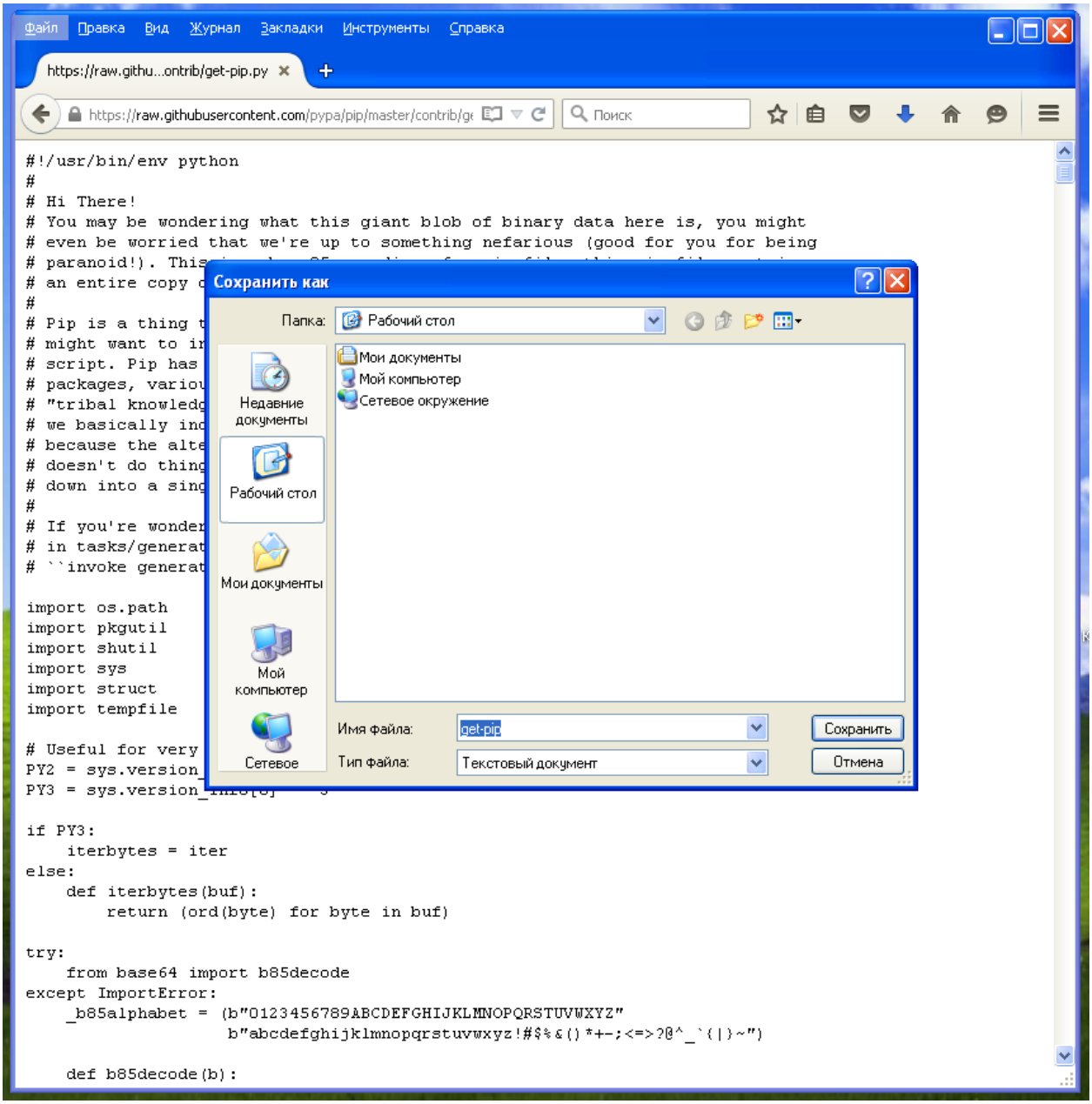

Python 2.7.10 Setup
Select whether to install Python 2.7.10 for all users of this computer.
Install for all users
Install just for me (not available on Windows Vista)
python
for
windows
Back
Next >
Cancel

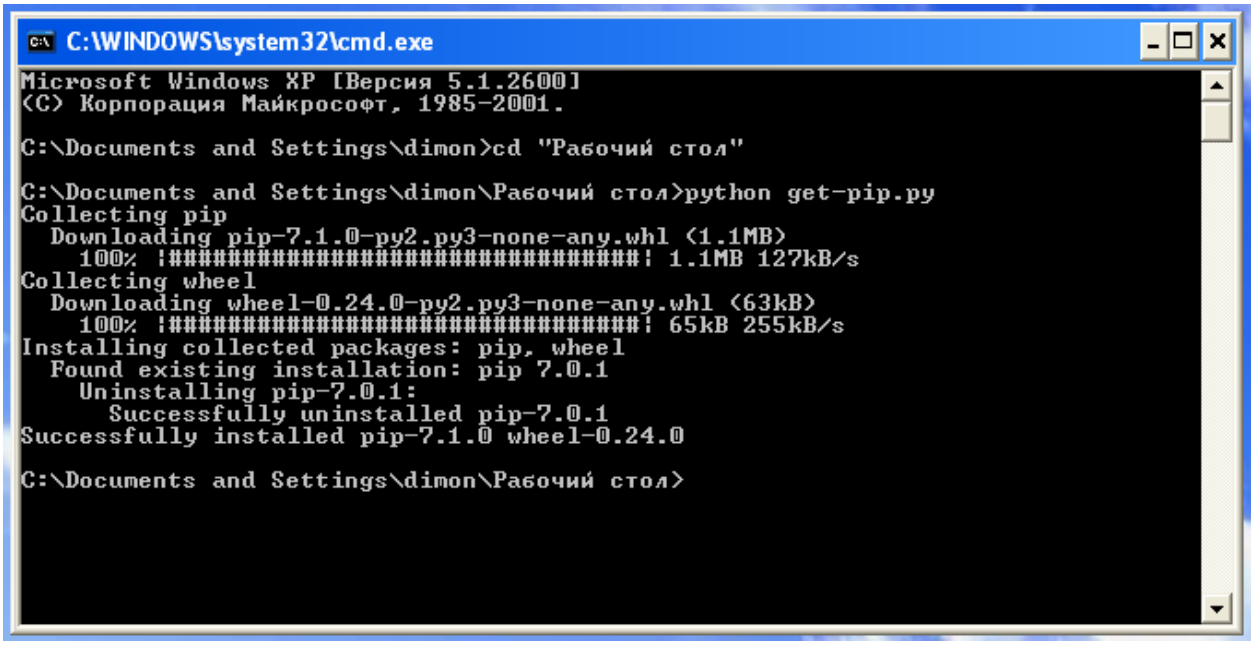

Python 2.7.10 Setup
Select Destination Directory
Please select a directory for the Python 2.7.10 files.
Python27
Up
New
C:\Python27\
python
for
windows
< Back
Next >
Cancel

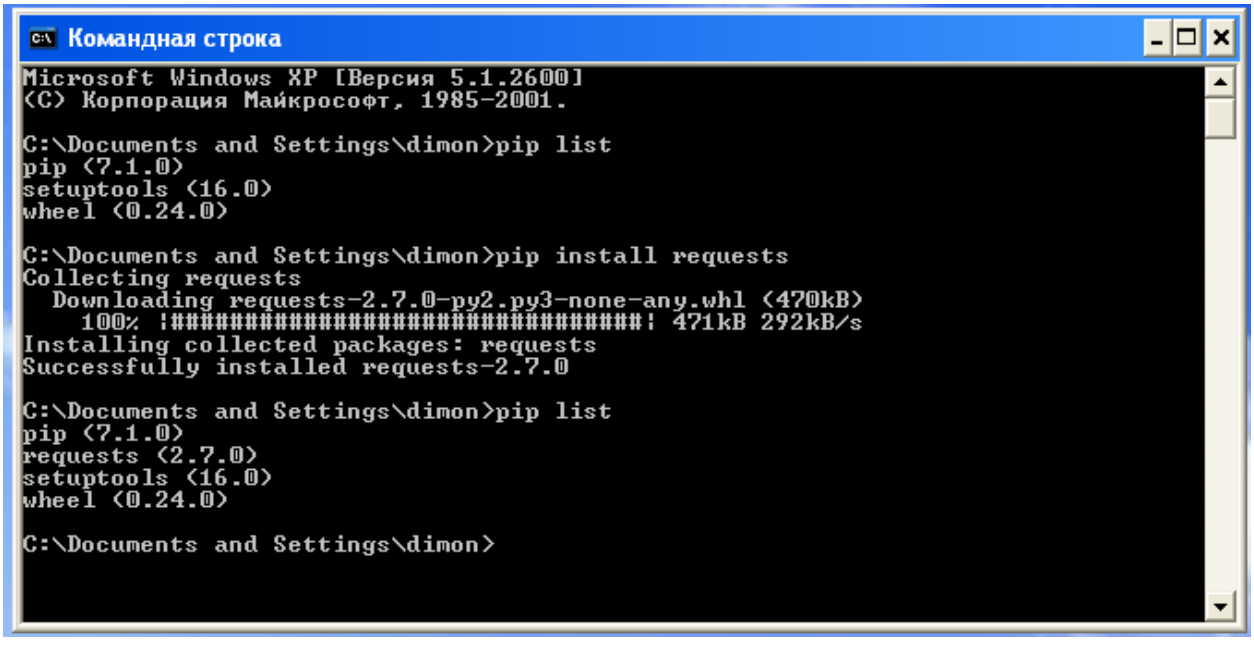

Python 2.7.10 Setup
Customize Python 2.7.10
Select the way you want features to be installed.
Click on the icons in the tree below to change the
way features will be installed.
Register Extensions
Tcl/Tk
Documentation
Utility Scripts
pip
Test suite
Add python.exe to Path
Will be installed on local hard drive
Entire feature will be installed on local hard drive
Entire feature will be unavailable
This feature requires 0KB on your hard drive.
python
for
windows
Disk Usage
Advanced
< Back
Next >
Cancel

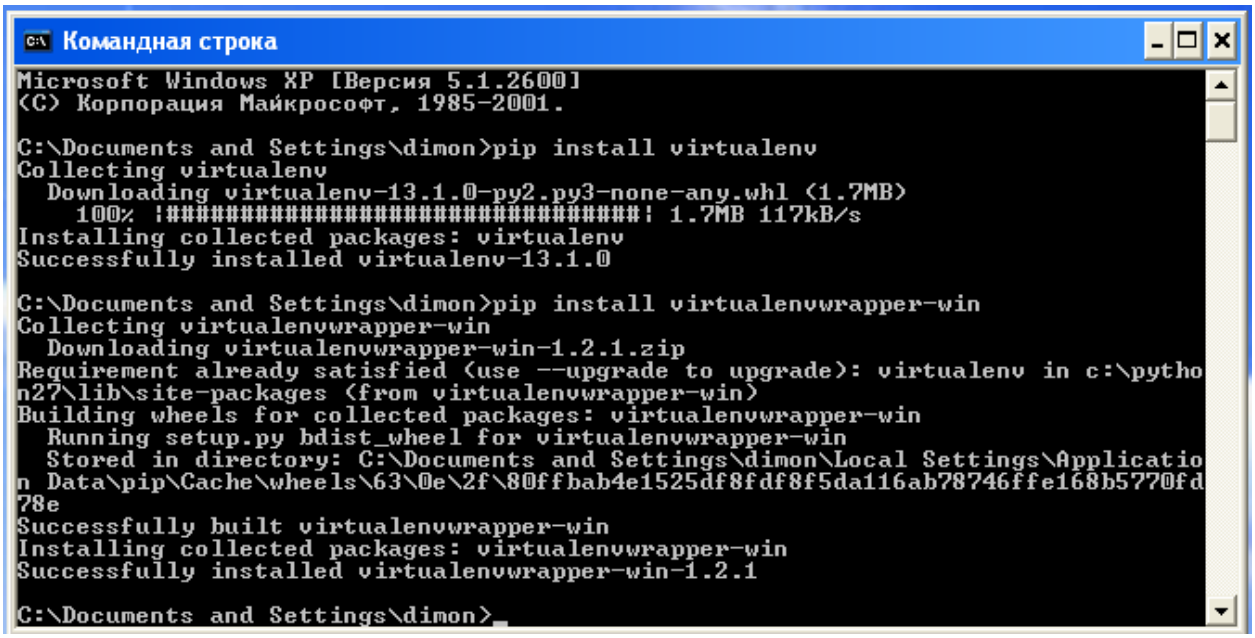

Python 2.7.10 Setup
Complete the Python 2.7.10 Installer
Special Windows thanks to:
Mark Hammond, without whose years of freely shared Windows expertise, Python for Windows would still be Python for DOS.
python
for
windows
Click the Finish button to exit the Installer.
< Back
Finish
Cancel

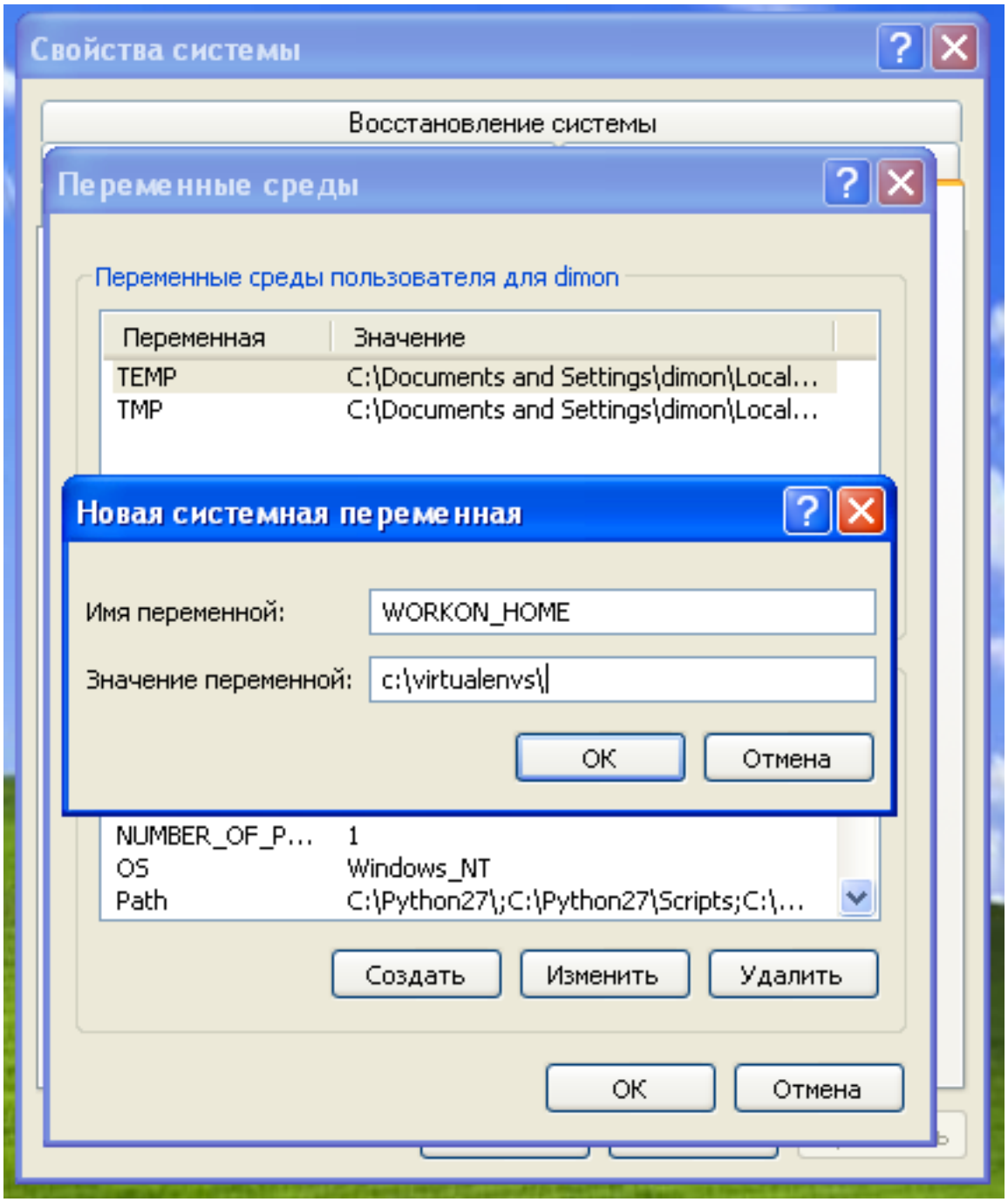

C:\WINDOWS\system32\cmd.exe - python
Microsoft Windows XP [Версия 5.1.2600]
(C) Корпорация Майкрософт, 1985-2001.
C:\Documents and Settings\dimon>python
Python 2.7.10 (default, May 23 2015, 09:40:32) [MSC v.1500 32 bit (Intel)] on wi
n32
Type "help", "copyright", "credits" or "license" for more information.
>>> print "Hello World!"
Hello World!
>>>

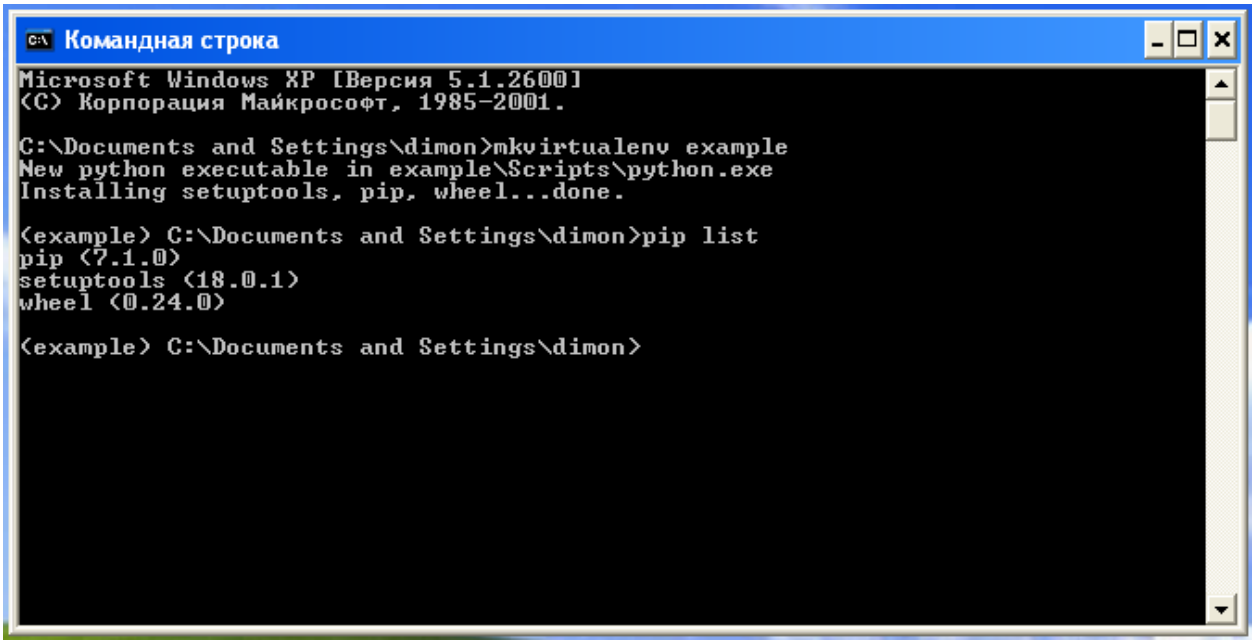

Файл Правка Вид Журнал Закладки Инструменты Справка
https://raw.githu...ontrib/get-pip.py
https://raw.githubusercontent.com/pypa/pip/master/contrib/ge
Поиск
#!/usr/bin/env python
#
Hi There!
You may be wondering what this giant blob of binary data here is, you might
even be worried that we're up to something nefarious (good for you for being
paranoid!). This
an entire copy
#
Pip is a thing
might want to
script. Pip has
packages, variou
"tribal knowledg
we basically inc
because the alte
doesn't do thing
down into a sing
#
If you're wonder
in tasks/generat
``invoke generat
import os.path
import pkgutil
import shutil
import sys
import struct
import tempfile
Useful for very
PY2 = sys.version_
PY3 = sys.version_
if PY3:
iterbytes = iter
else:
def iterbytes(buf):
return (ord(byte) for byte in buf)
try:
from base64 import b85decode
except ImportError:
_b85alphabet = (b"0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"
b"abcdefghijklmnopqrstuvwxyz!#$%&()*+-;<=>?@^_`{|}~")
def b85decode(b):
Сохранить как
Папка: Рабочий стол
Мои документы
Мой компьютер
Сетевое окружение
Недавние документы
Рабочий стол
Мои документы
Мой компьютер
Сетевое
Имя файла: get-pip
Тип файла: Текстовый документ
Сохранить
Отмена

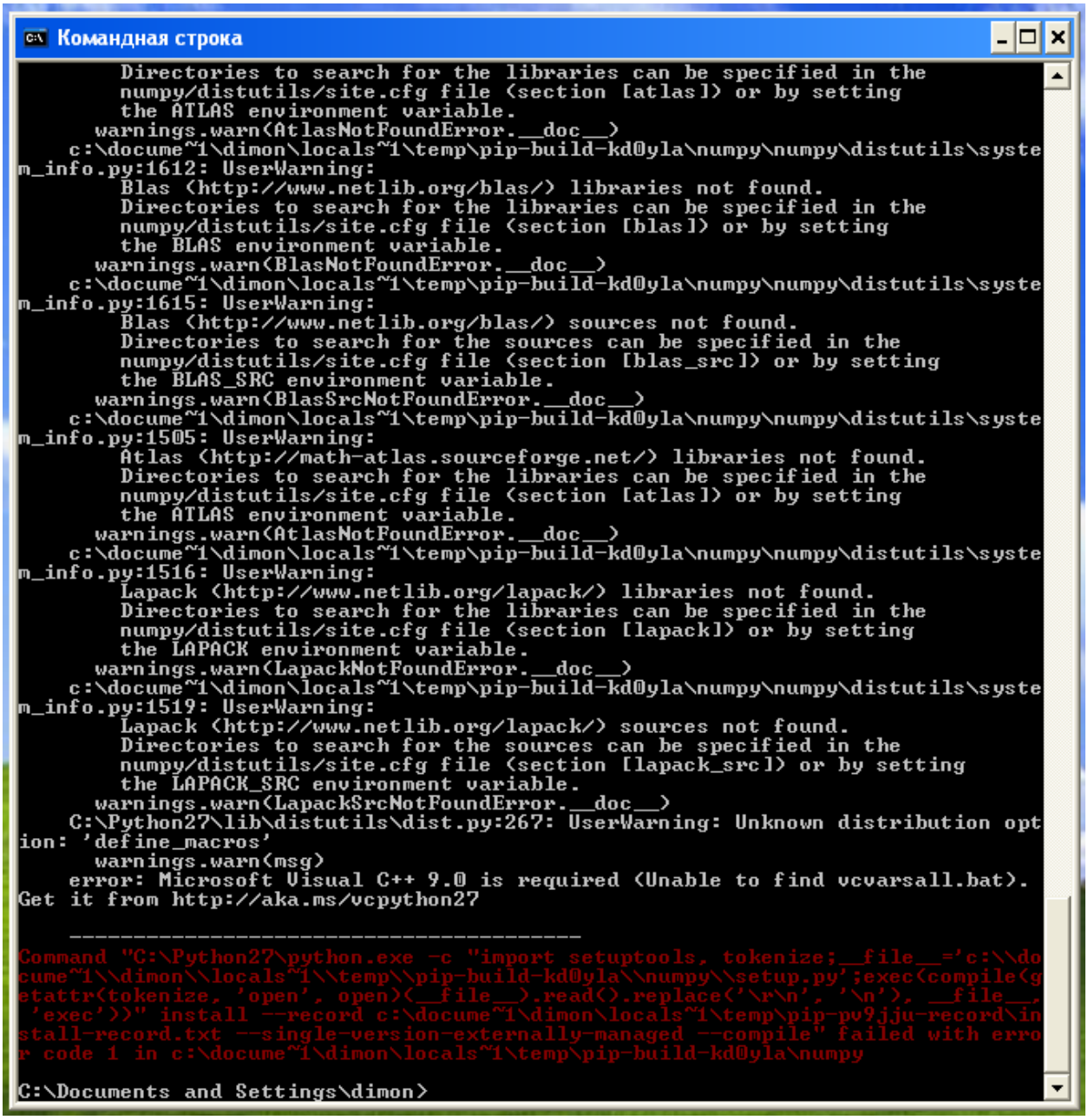

C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Версия 5.1.2600]
(C) Корпорация Майкрософт, 1985-2001.
C:\Documents and Settings\dimon>cd "Рабочий стол"
C:\Documents and Settings\dimon\Рабочий стол>python get-pip.py
Collecting pip
Downloading pip-7.1.0-py2.py3-none-any.whl (1.1MB)
100% 1.1MB 127kB/s
Collecting wheel
Downloading wheel-0.24.0-py2.py3-none-any.whl (63kB)
100% 65kB 255kB/s
Installing collected packages: pip, wheel
Found existing installation: pip 7.0.1
Uninstalling pip-7.0.1:
Successfully uninstalled pip-7.0.1
Successfully installed pip-7.1.0 wheel-0.24.0
C:\Documents and Settings\dimon\Рабочий стол>


Командная строка
Microsoft Windows XP [Версия 5.1.2600]
(C) Корпорация Майкрософт, 1985-2001.
C:\Documents and Settings\dimon>pip list
pip (7.1.0)
setuptools (16.0)
wheel (0.24.0)
C:\Documents and Settings\dimon>pip install requests
Collecting requests
Downloading requests-2.7.0-py2.py3-none-any.whl (470kB)
100% 471kB 292kB/s
Installing collected packages: requests
Successfully installed requests-2.7.0
C:\Documents and Settings\dimon>pip list
pip (7.1.0)
requests (2.7.0)
setuptools (16.0)
wheel (0.24.0)
C:\Documents and Settings\dimon>

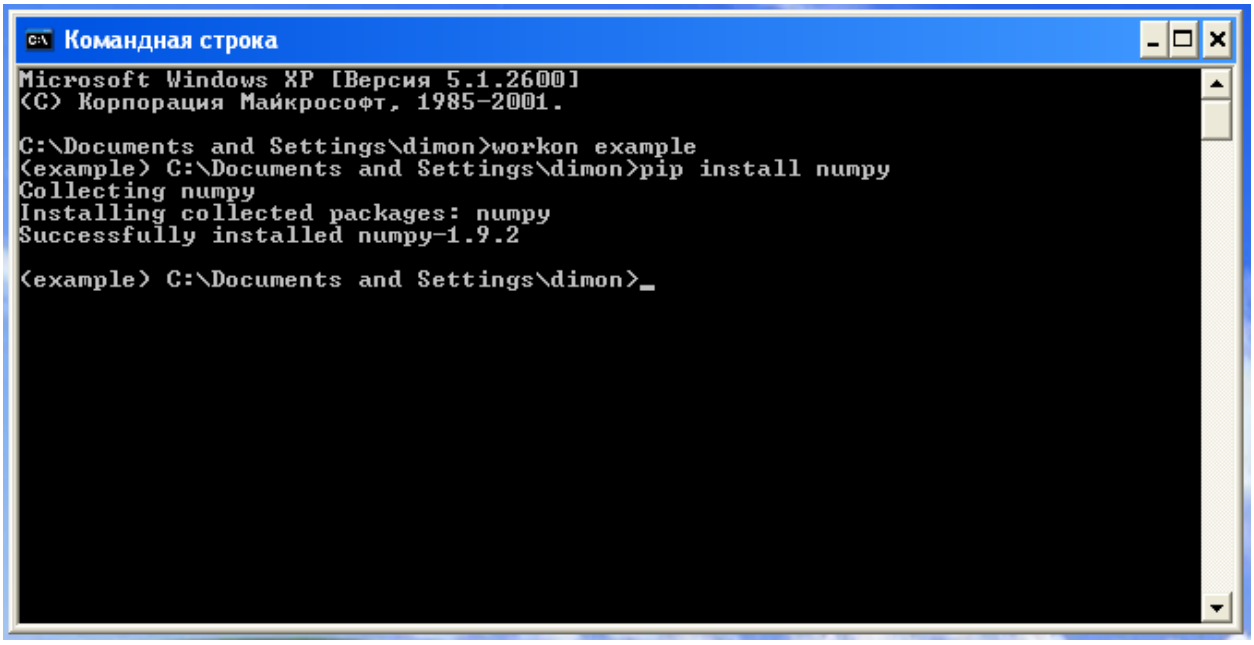

Командная строка
Microsoft Windows XP [Версия 5.1.2600]
(C) Корпорация Майкрософт, 1985-2001.
C:\Documents and Settings\dimon>pip install virtualenv
Collecting virtualenv
Downloading virtualenv-13.1.0-py2.py3-none-any.whl (1.7MB)
100% |################################| 1.7MB 117kB/s
Installing collected packages: virtualenv
Successfully installed virtualenv-13.1.0
C:\Documents and Settings\dimon>pip install virtualenvwrapper-win
Collecting virtualenvwrapper-win
Downloading virtualenvwrapper-win-1.2.1.zip
Requirement already satisfied (use --upgrade to upgrade): virtualenv in c:\pytho
n27\lib\site-packages (from virtualenvwrapper-win)
Building wheels for collected packages: virtualenvwrapper-win
Running setup.py bdist_wheel for virtualenvwrapper-win
Stored in directory: C:\Documents and Settings\dimon\Local Settings\Applicatio
n Data\pip\Cache\wheels\63\0e\2f\80ffbab4e1525df8fdf8f5da116ab78746ffe168b5770fd
78e
Successfully built virtualenvwrapper-win
Installing collected packages: virtualenvwrapper-win
Successfully installed virtualenvwrapper-win-1.2.1
C:\Documents and Settings\dimon>


Свойства системы
Восстановление системы
Переменные среды
Переменные среды пользователя для dimon
Переменная
Значение
TEMP
C:\Documents and Settings\dimon\Local...
TMP
C:\Documents and Settings\dimon\Local...
Новая системная переменная
Имя переменной:
WORKON_HOME
Значение переменной:
c:\virtualenvs\
OK
Отмена
NUMBER_OF_P...
1
OS
Windows_NT
Path
C:\Python27\;C:\Python27\Scripts;C:\...
Создать
Изменить
Удалить
OK
Отмена

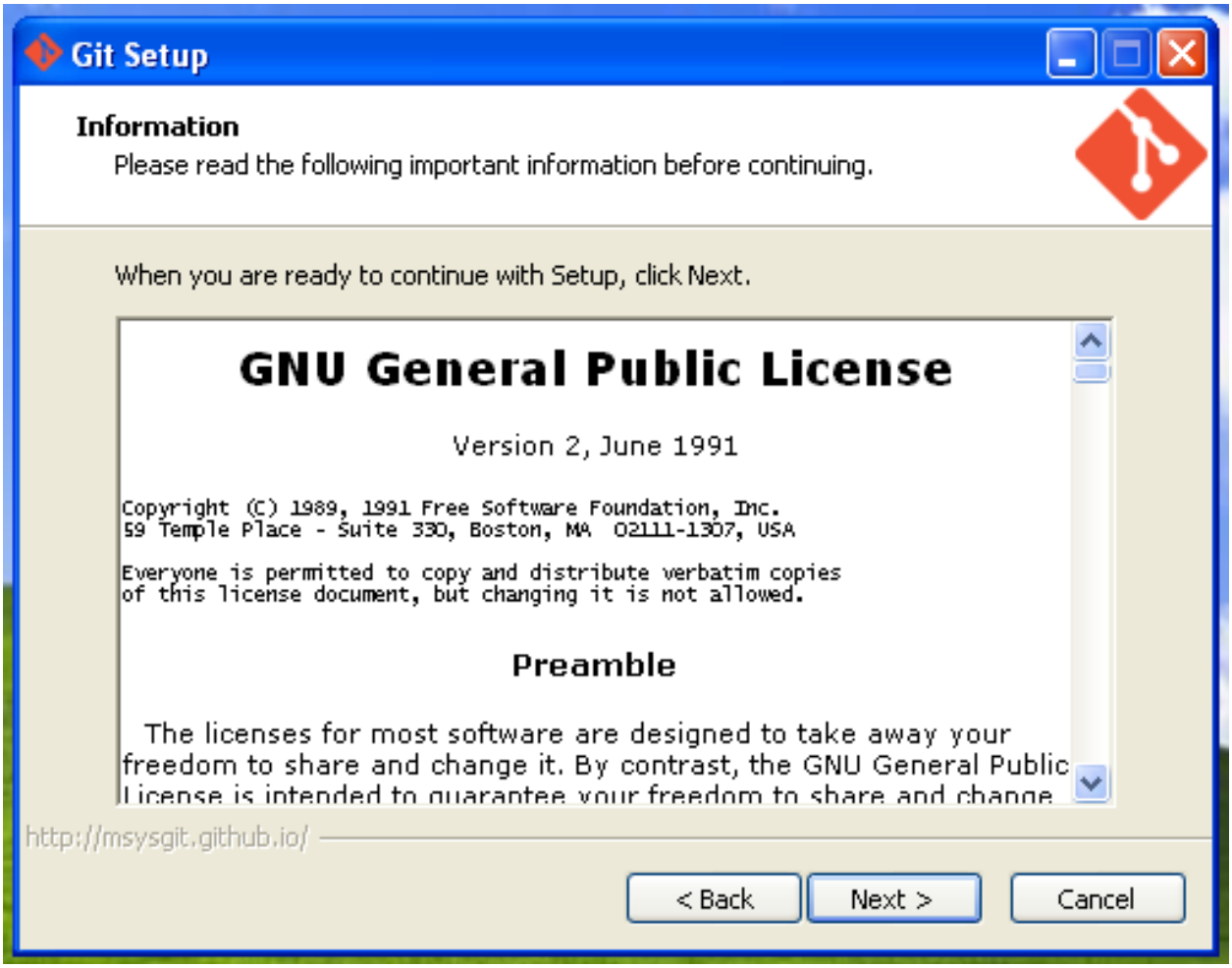

Командная строка
Microsoft Windows XP [Версия 5.1.2600]
(C) Корпорация Майкрософт, 1985-2001.
C:\Documents and Settings\dimon>mkvirtualenv example
New python executable in example\Scripts\python.exe
Installing setuptools, pip, wheel...done.
(example) C:\Documents and Settings\dimon>pip list
pip (7.1.0)
setuptools (18.0.1)
wheel (0.24.0)
(example) C:\Documents and Settings\dimon>

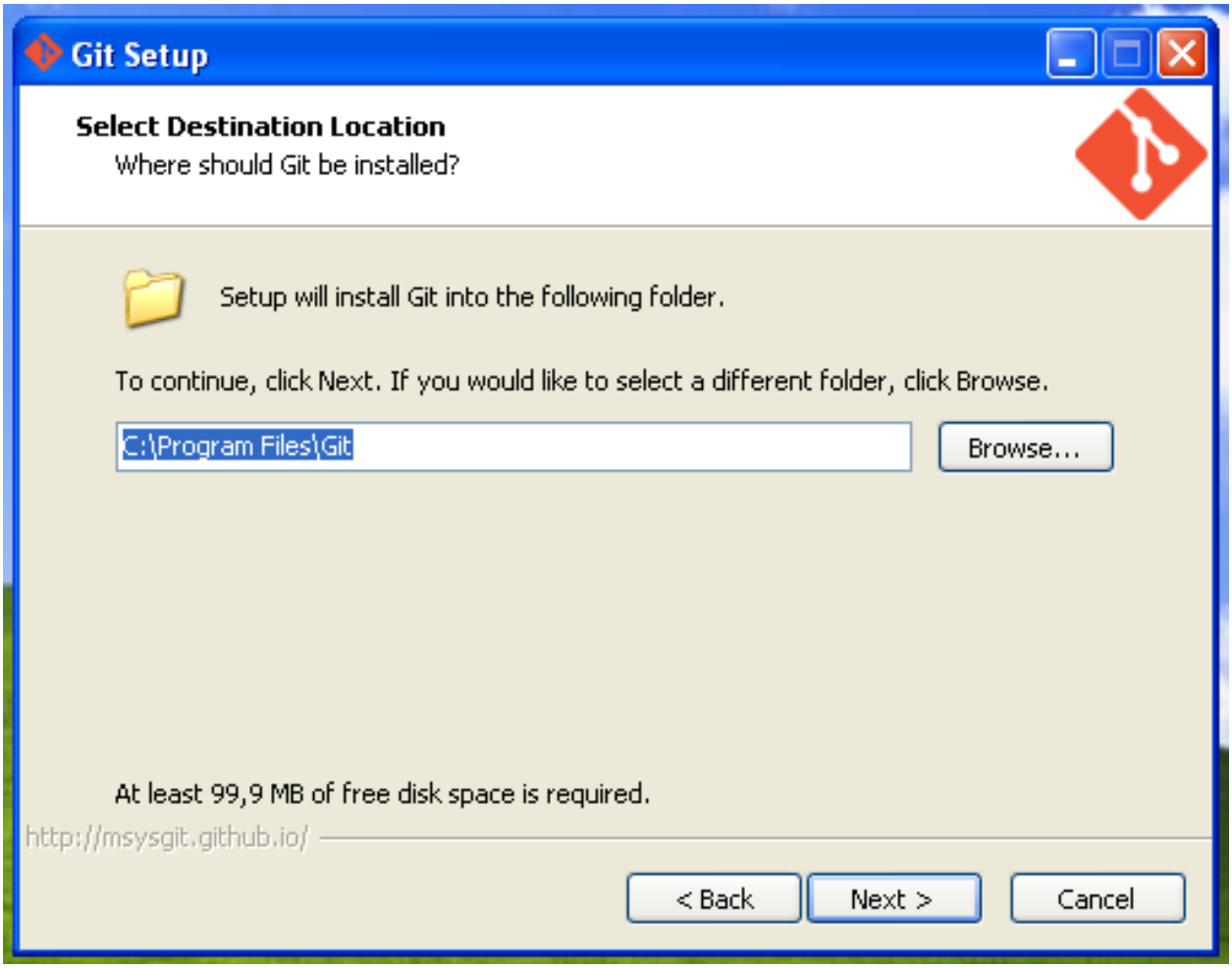

Командная строка
Directories to search for the libraries can be specified in the
numpy/distutils/site.cfg file (section [atlas]) or by setting
the ATLAS environment variable.
warnings.warn(AtlasNotFoundError.__doc__)
c:\docume~1\dimon\locals~1\temp\pip-build-kd0yla\numpy\numpy\distutils\system_info.py:1612: UserWarning:
Blas (http://www.netlib.org/blas/) libraries not found.
Directories to search for the libraries can be specified in the
numpy/distutils/site.cfg file (section [blas]) or by setting
the BLAS environment variable.
warnings.warn(BlasNotFoundError.__doc__)
c:\docume~1\dimon\locals~1\temp\pip-build-kd0yla\numpy\numpy\distutils\system_info.py:1615: UserWarning:
Blas (http://www.netlib.org/blas/) sources not found.
Directories to search for the sources can be specified in the
numpy/distutils/site.cfg file (section [blas_src]) or by setting
the BLAS_SRC environment variable.
warnings.warn(BlasSrcNotFoundError.__doc__)
c:\docume~1\dimon\locals~1\temp\pip-build-kd0yla\numpy\numpy\distutils\system_info.py:1505: UserWarning:
Atlas (http://math-atlas.sourceforge.net/) libraries not found.
Directories to search for the libraries can be specified in the
numpy/distutils/site.cfg file (section [atlas]) or by setting
the ATLAS environment variable.
warnings.warn(AtlasNotFoundError.__doc__)
c:\docume~1\dimon\locals~1\temp\pip-build-kd0yla\numpy\numpy\distutils\system_info.py:1516: UserWarning:
Lapack (http://www.netlib.org/lapack/) libraries not found.
Directories to search for the libraries can be specified in the
numpy/distutils/site.cfg file (section [lapack]) or by setting
the LAPACK environment variable.
warnings.warn(LapackNotFoundError.__doc__)
c:\docume~1\dimon\locals~1\temp\pip-build-kd0yla\numpy\numpy\distutils\system_info.py:1519: UserWarning:
Lapack (http://www.netlib.org/lapack/) sources not found.
Directories to search for the sources can be specified in the
numpy/distutils/site.cfg file (section [lapack_src]) or by setting
the LAPACK_SRC environment variable.
warnings.warn(LapackSrcNotFoundError.__doc__)
C:\Python27\lib\distutils\dist.py:267: UserWarning: Unknown distribution option: 'define_macros'
warnings.warn(msg)
error: Microsoft Visual C++ 9.0 is required (Unable to find vcvarsall.bat). Get it from http://aka.ms/vcpython27

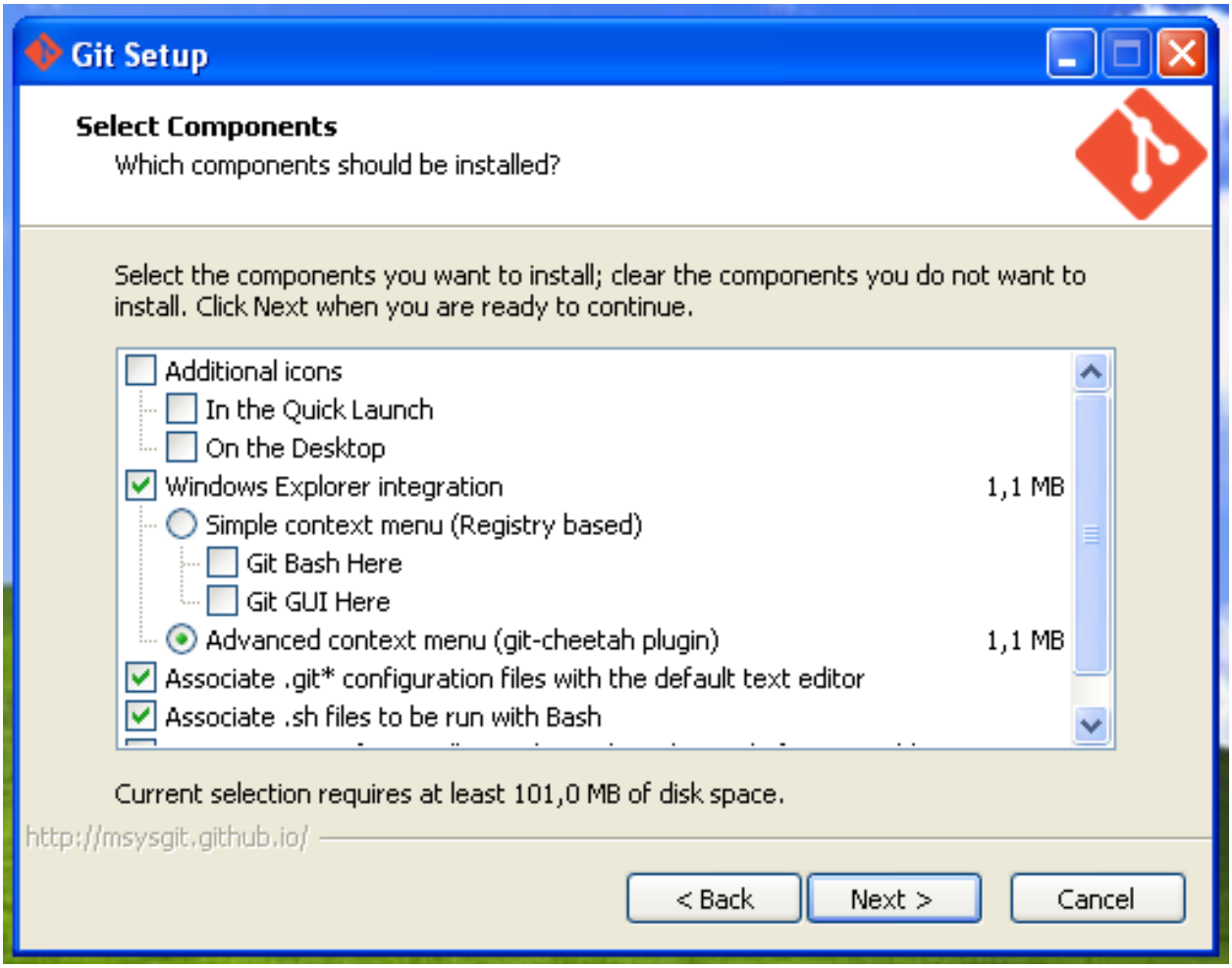
Command "C:\Python27\python.exe -c "import setuptools, tokenize;__file__='c:\\docume~1\\dimon\\locals~1\\temp\\pip-build-kd0yla\\numpy\\setup.py';exec(compile(getattr(tokenize, 'open', open)(__file__).read().replace('\r\n', '\n'), __file__, 'exec'))" install --record c:\docume~1\dimon\locals~1\temp\pip-pv9jju-record\install-record.txt --single-version-externally-managed --compile" failed with error code 1 in c:\docume~1\dimon\locals~1\temp\pip-build-kd0yla\numpy
C:\Documents and Settings\dimon>

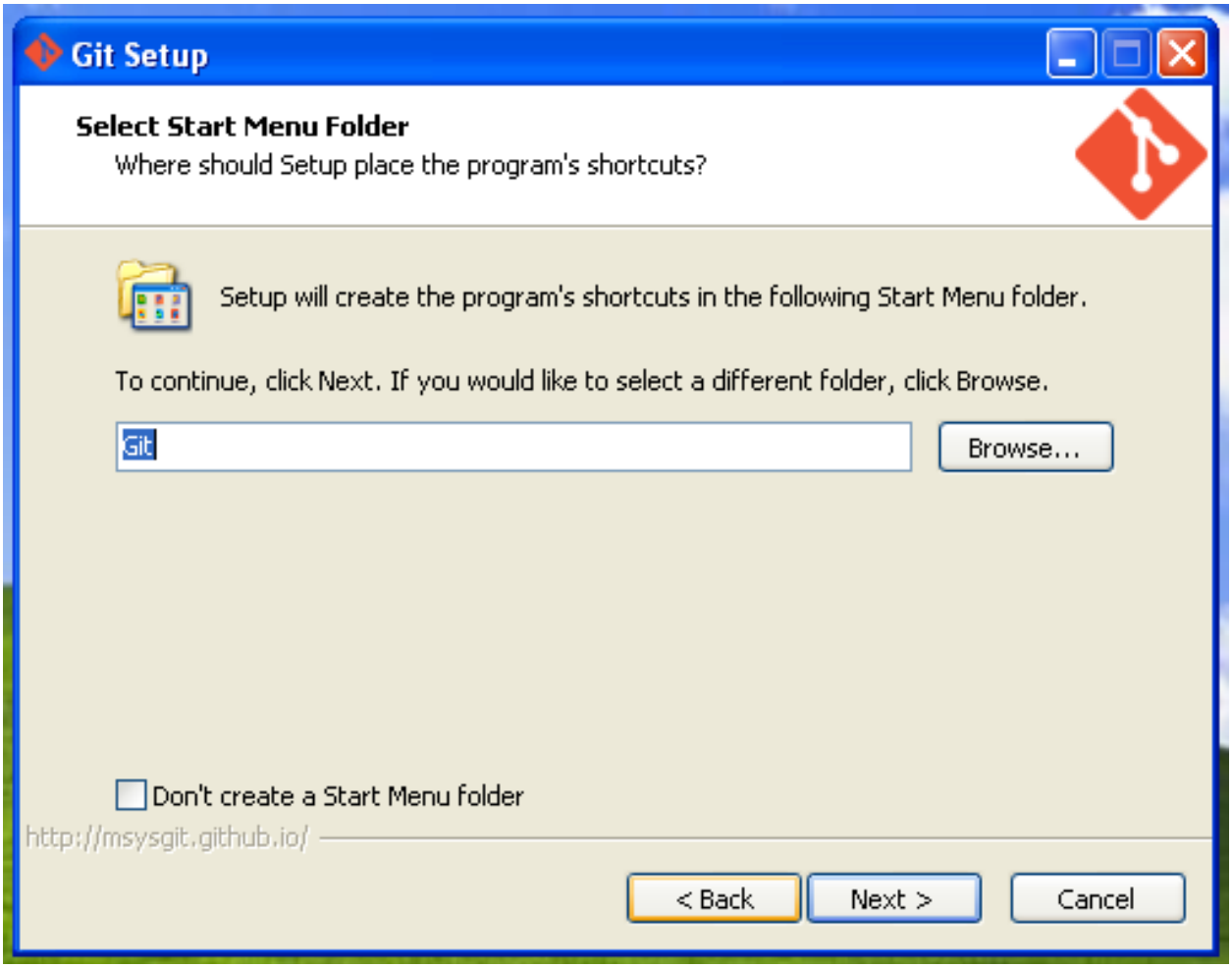

Командная строка
Microsoft Windows XP [Версия 5.1.2600]
(C) Корпорация Майкрософт, 1985-2001.
C:\Documents and Settings\dimon>workon example
(example) C:\Documents and Settings\dimon>pip install numpy
Collecting numpy
Installing collected packages: numpy
Successfully installed numpy-1.9.2
(example) C:\Documents and Settings\dimon>

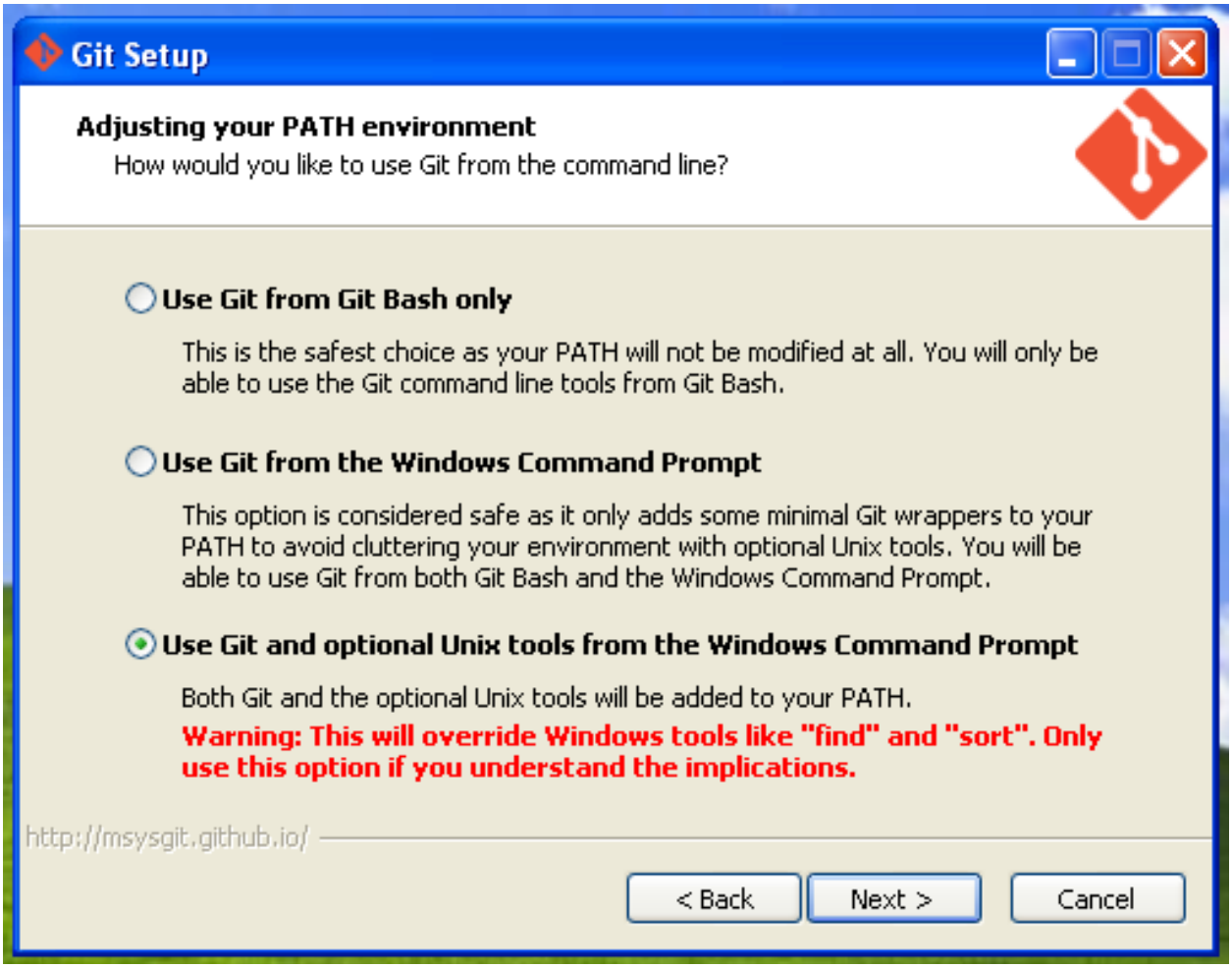

Git Setup
Welcome to the Git Setup Wizard
This will install Git version 1.9.5-preview20150319 on your computer.
It is recommended that you close all other applications before continuing.
Click Next to continue, or Cancel to exit Setup.
git
Next >
Cancel

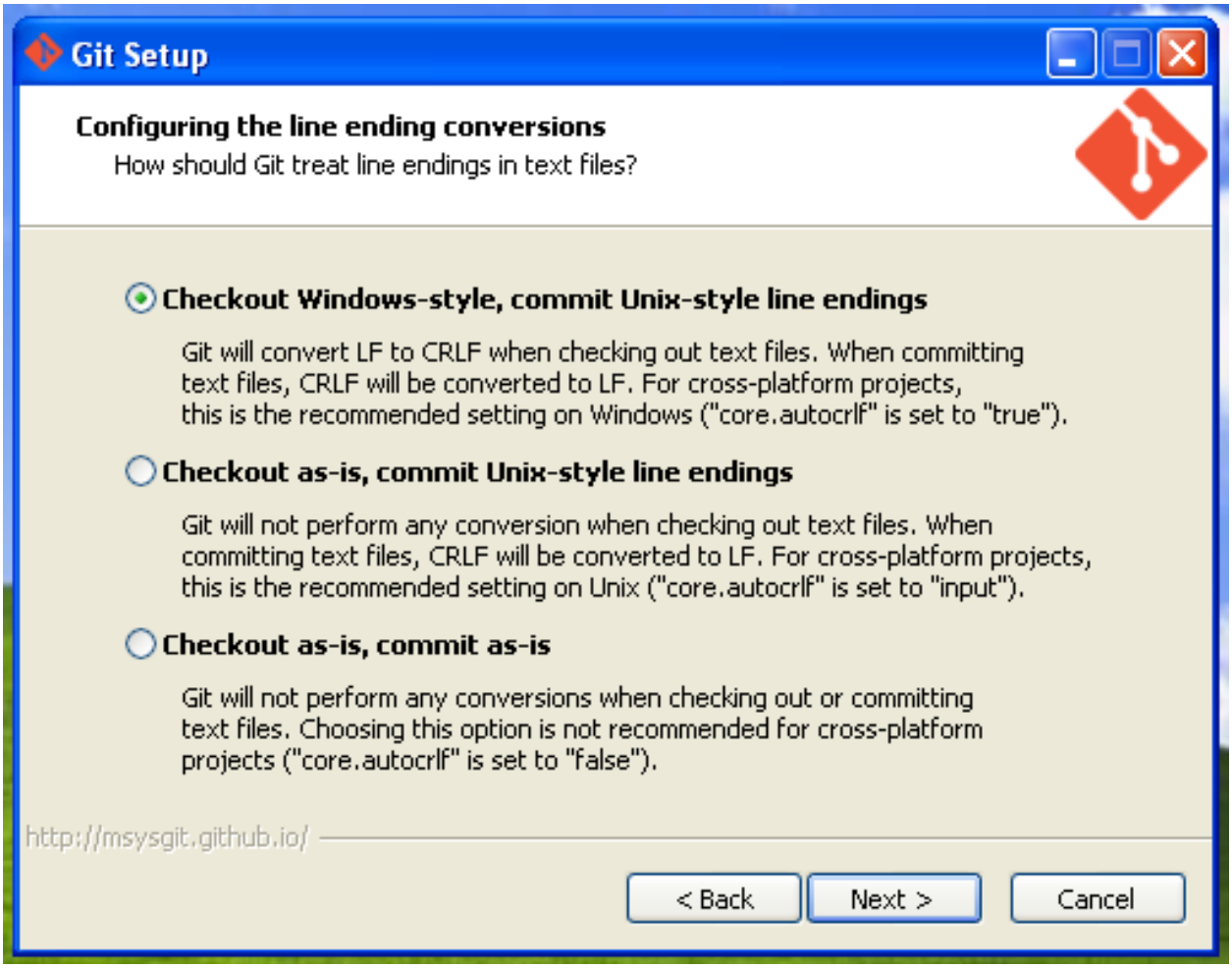

Git Setup
Information
Please read the following important information before continuing.
When you are ready to continue with Setup, click Next.
GNU General Public License
Version 2, June 1991
Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place - Suite 330, Boston, MA 02111-1307, USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.
Preamble
The licenses for most software are designed to take away your
freedom to share and change it. By contrast, the GNU General Public
License is intended to guarantee your freedom to share and change
http://msysgit.github.io/
< Back
Next >
Cancel

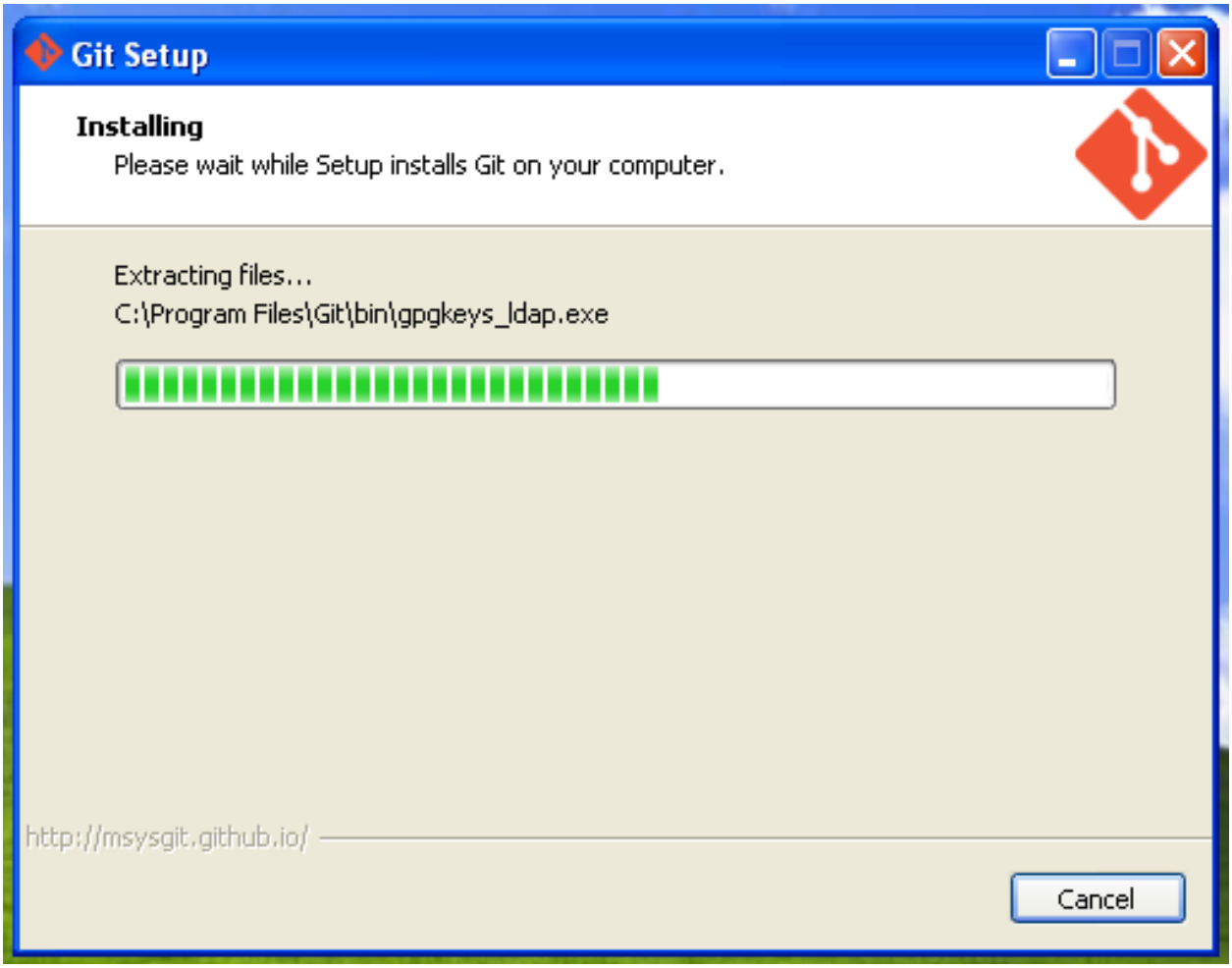

Git Setup
Select Destination Location
Where should Git be installed?
Setup will install Git into the following folder.
To continue, click Next. If you would like to select a different folder, click Browse.
C:\Program Files\Git
Browse...
At least 99,9 MB of free disk space is required.
http://msysgit.github.io/
< Back
Next >
Cancel

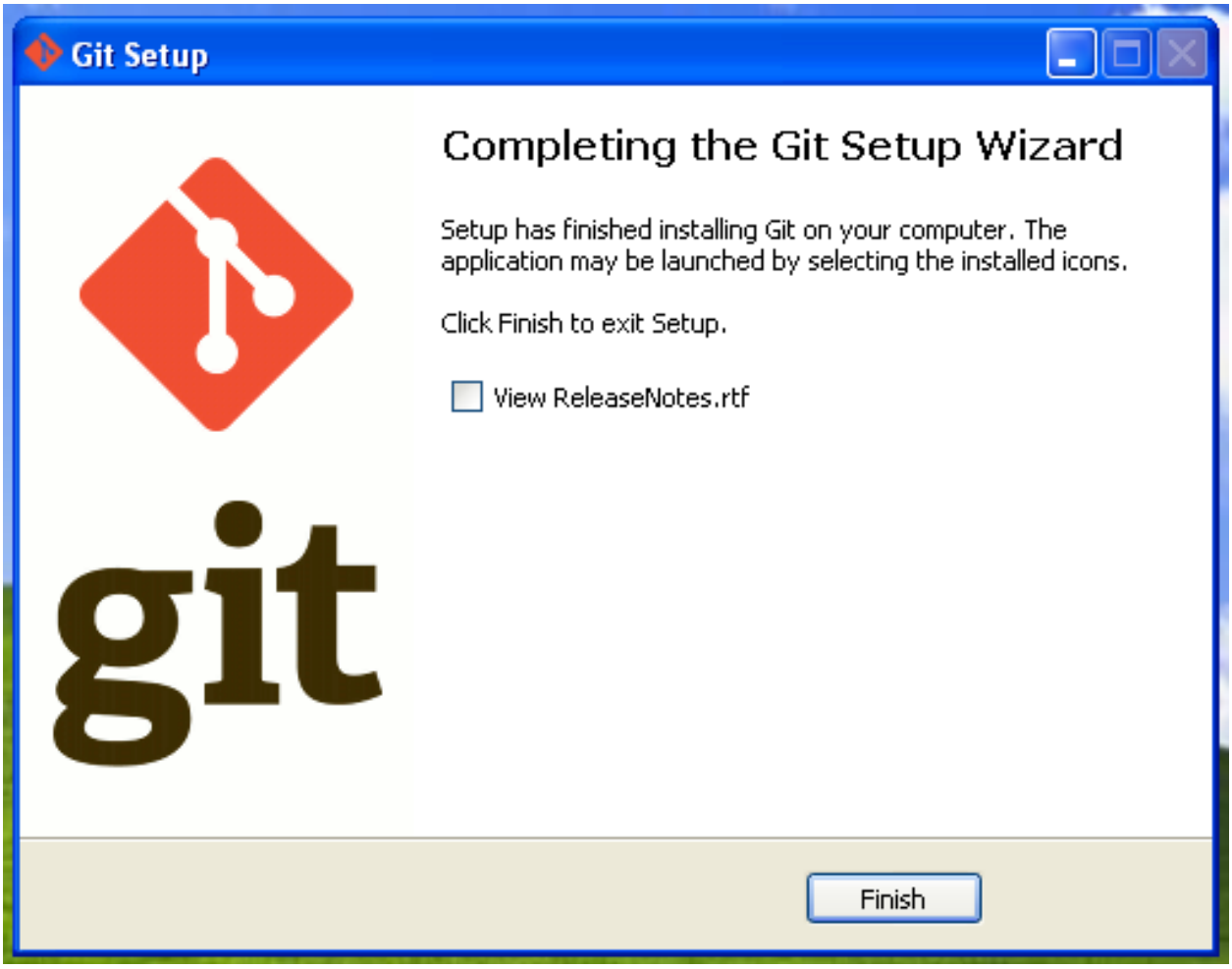

Git Setup
Select Components
Which components should be installed?
Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.
Additional icons
In the Quick Launch
On the Desktop
Windows Explorer integration
1,1 MB
Simple context menu (Registry based)
Git Bash Here
Git GUI Here
Advanced context menu (git-cheetah plugin)
1,1 MB
Associate .git* configuration files with the default text editor
Associate .sh files to be run with Bash
Current selection requires at least 101,0 MB of disk space.
http://msysgit.github.io/
< Back
Next >
Cancel

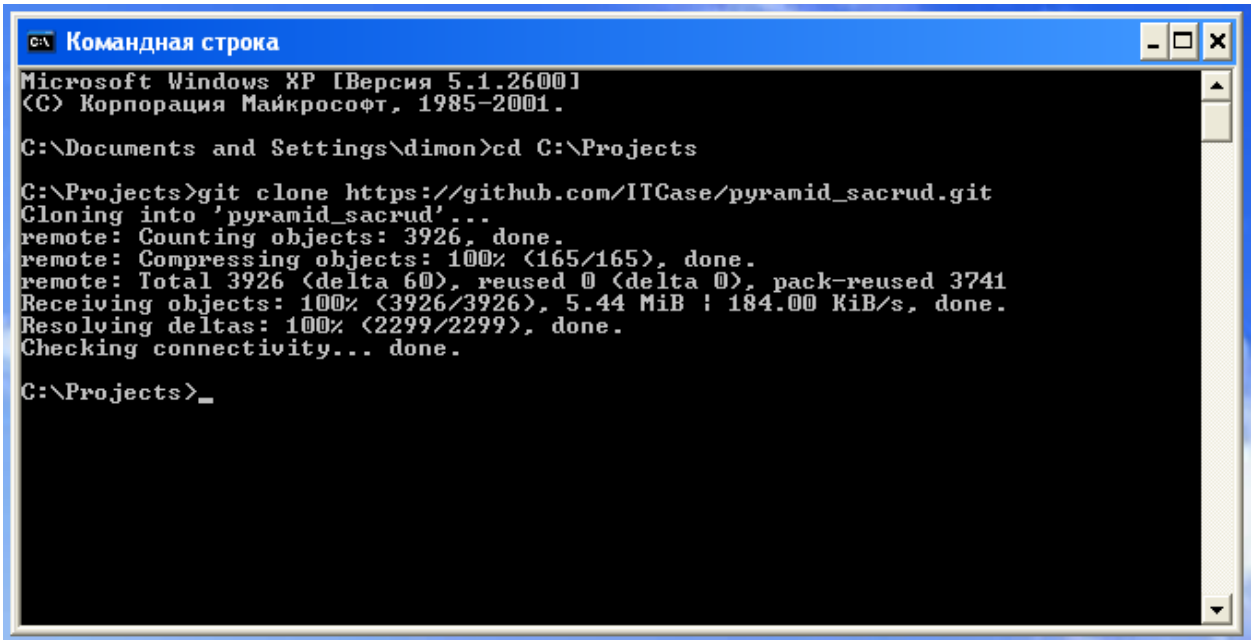

Git Setup
Select Start Menu Folder
Where should Setup place the program's shortcuts?
Setup will create the program's shortcuts in the following Start Menu folder.
To continue, click Next. If you would like to select a different folder, click Browse.
Git
Browse...
Don't create a Start Menu folder
http://msysgit.github.io/
< Back
Next >
Cancel

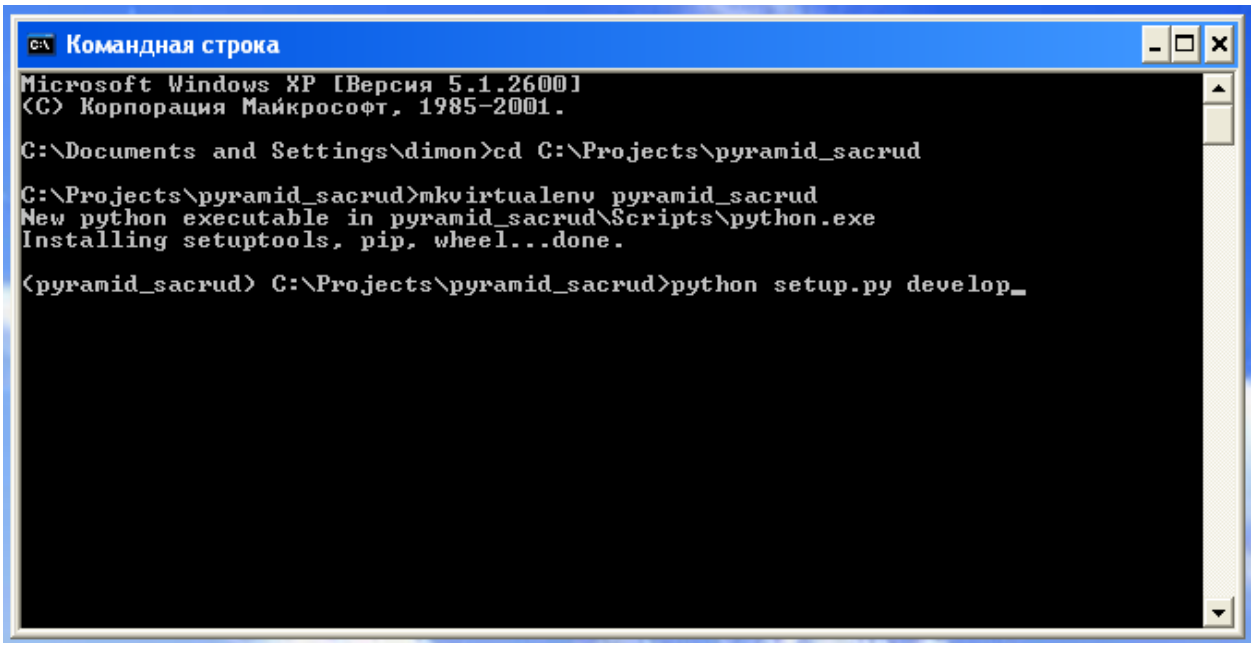

Git Setup
Adjusting your PATH environment
How would you like to use Git from the command line?
Use Git from Git Bash only
This is the safest choice as your PATH will not be modified at all. You will only be able to use the Git command line tools from Git Bash.
Use Git from the Windows Command Prompt
This option is considered safe as it only adds some minimal Git wrappers to your PATH to avoid cluttering your environment with optional Unix tools. You will be able to use Git from both Git Bash and the Windows Command Prompt.
Use Git and optional Unix tools from the Windows Command Prompt
Both Git and the optional Unix tools will be added to your PATH.
Warning: This will override Windows tools like "find" and "sort". Only use this option if you understand the implications.
http://msysgit.github.io/
< Back
Next >
Cancel

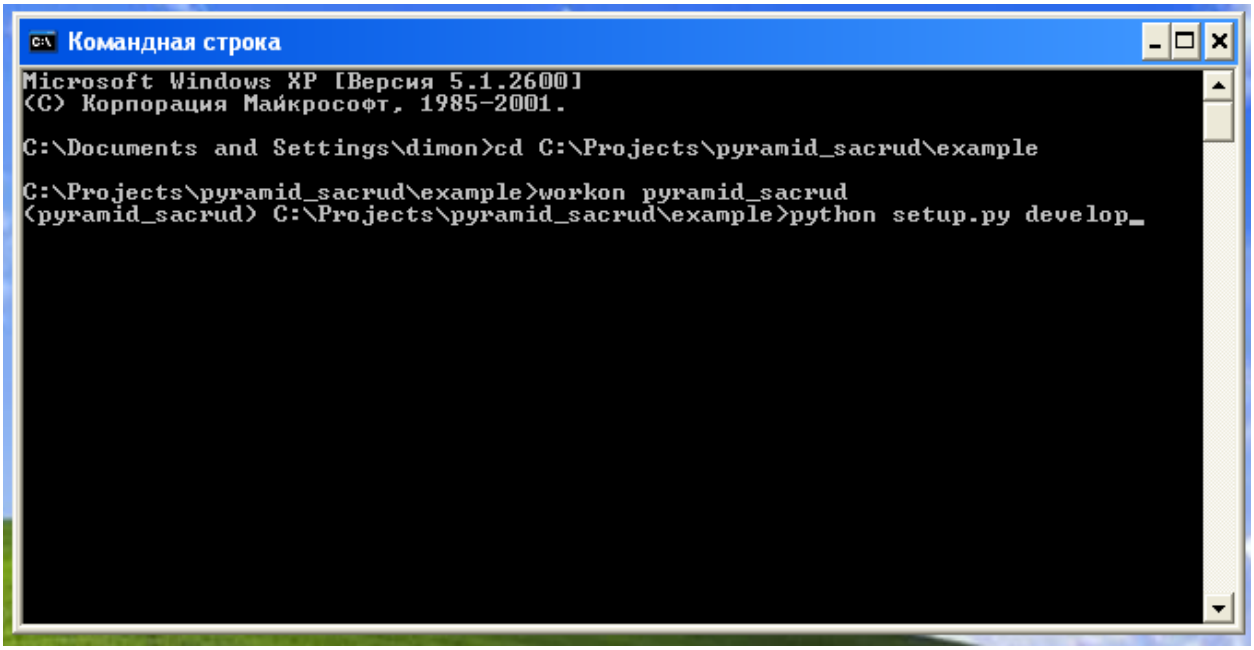

Git Setup
Configuring the line ending conversions
How should Git treat line endings in text files?
Checkout Windows-style, commit Unix-style line endings
Git will convert LF to CRLF when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Windows ("core.autocrlf" is set to "true").
Checkout as-is, commit Unix-style line endings
Git will not perform any conversion when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Unix ("core.autocrlf" is set to "input").
Checkout as-is, commit as-is
Git will not perform any conversions when checking out or committing text files. Choosing this option is not recommended for cross-platform projects ("core.autocrlf" is set to "false").
http://msysgit.github.io/
< Back
Next >
Cancel

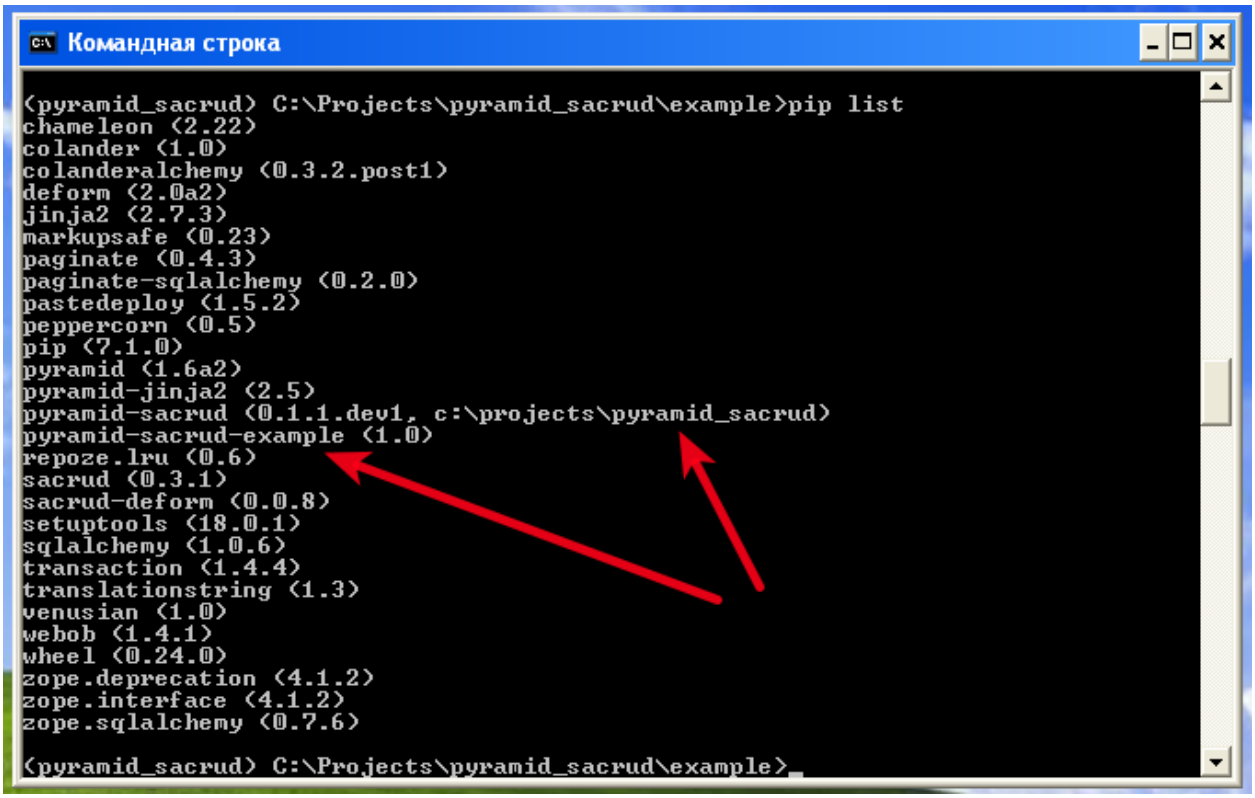

Git Setup
Installing
Please wait while Setup installs Git on your computer.
Extracting files...
C:\Program Files\Git\bin\gpgkeys_ldap.exe
http://msysgit.github.io/
Cancel

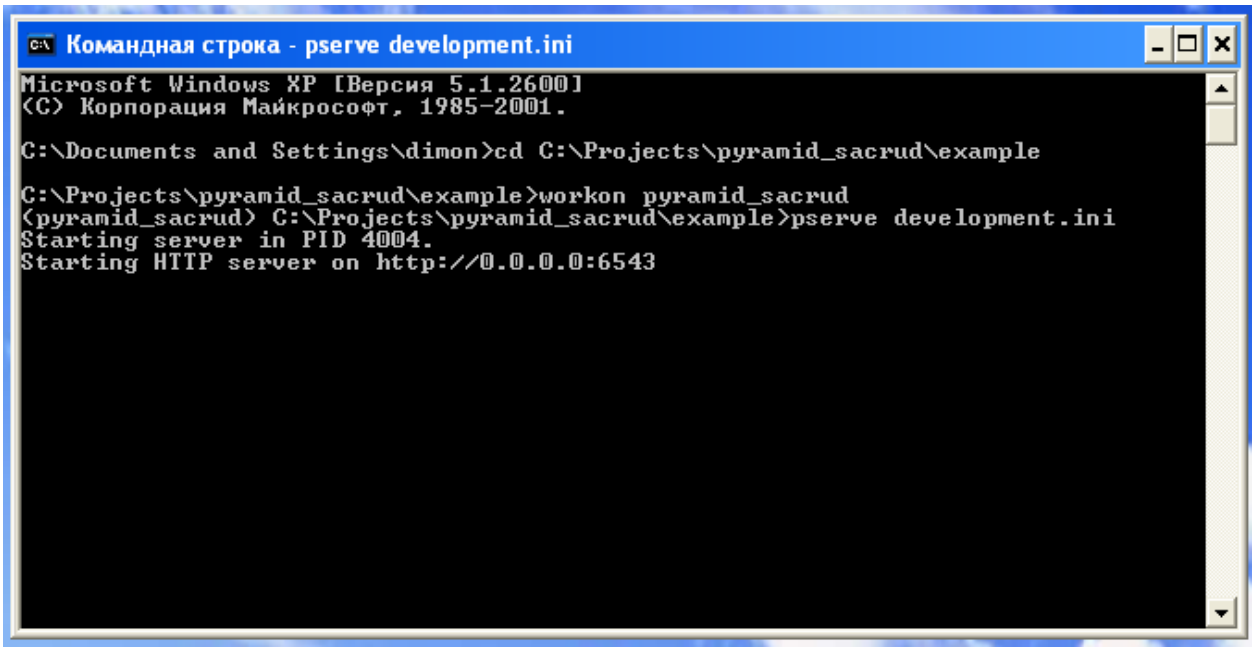

Git Setup
Completing the Git Setup Wizard
Setup has finished installing Git on your computer. The application may be launched by selecting the installed icons.
Click Finish to exit Setup.
View ReleaseNotes.rtf
git
Finish

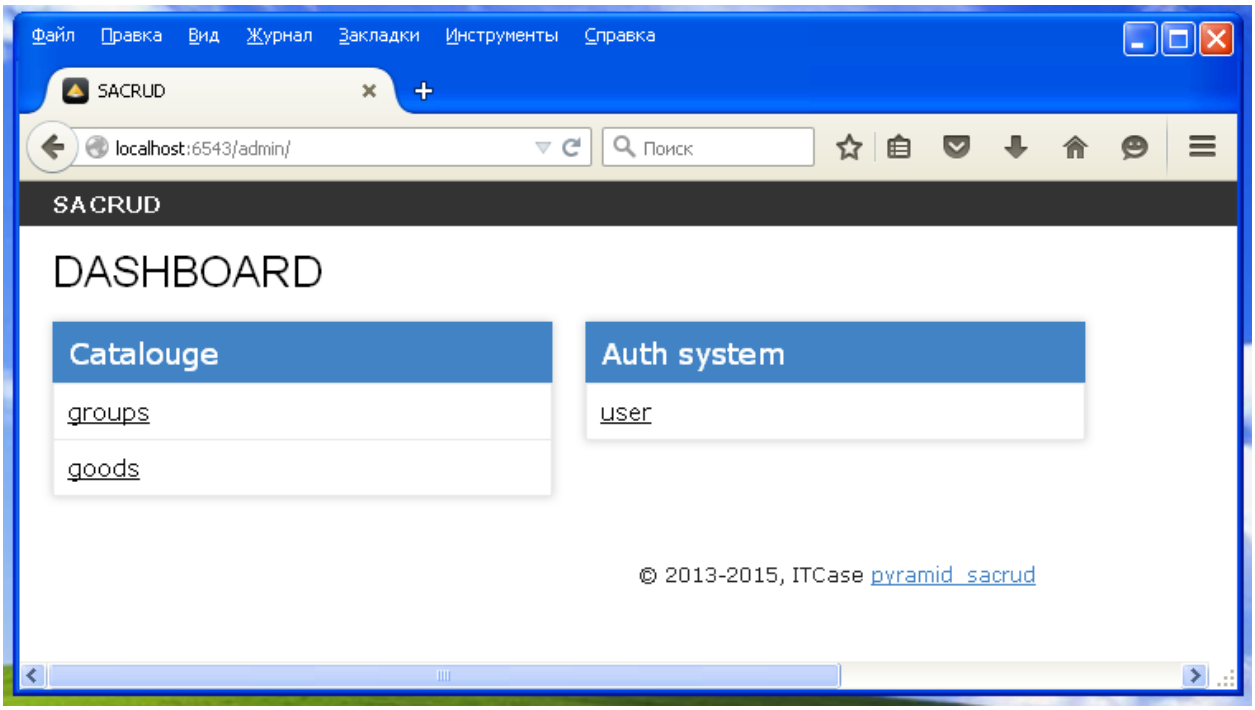

Командная строка
Microsoft Windows XP [Версия 5.1.2600]
(C) Корпорация Майкрософт, 1985-2001.
C:\Documents and Settings\dimon>cd C:\Projects
C:\Projects>git clone https://github.com/ITCase/pyramid_sacrud.git
Cloning into 'pyramid_sacrud'...
remote: Counting objects: 3926, done.
remote: Compressing objects: 100% (165/165), done.
remote: Total 3926 (delta 60), reused 0 (delta 0), pack-reused 3741
Receiving objects: 100% (3926/3926), 5.44 MiB | 184.00 KiB/s, done.
Resolving deltas: 100% (2299/2299), done.
Checking connectivity... done.
C:\Projects>

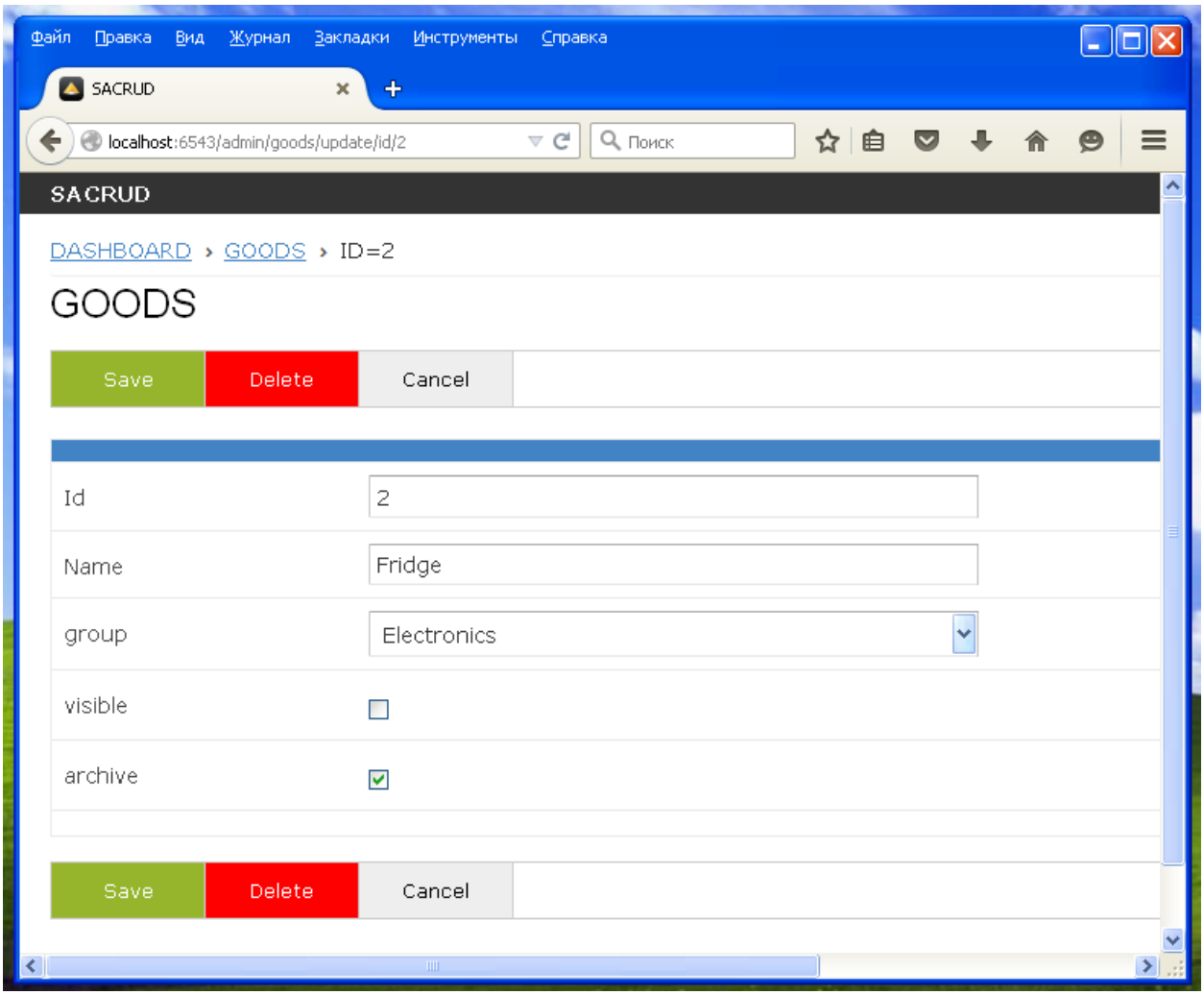

Командная строка
Microsoft Windows XP [Версия 5.1.2600]
(C) Корпорация Майкрософт, 1985-2001.
C:\Documents and Settings\dimon>cd C:\Projects\pyramid_sacrud
C:\Projects\pyramid_sacrud>mkvirtualenv pyramid_sacrud
New python executable in pyramid_sacrud\Scripts\python.exe
Installing setuptools, pip, wheel...done.
(pyramid_sacrud) C:\Projects\pyramid_sacrud>python setup.py develop

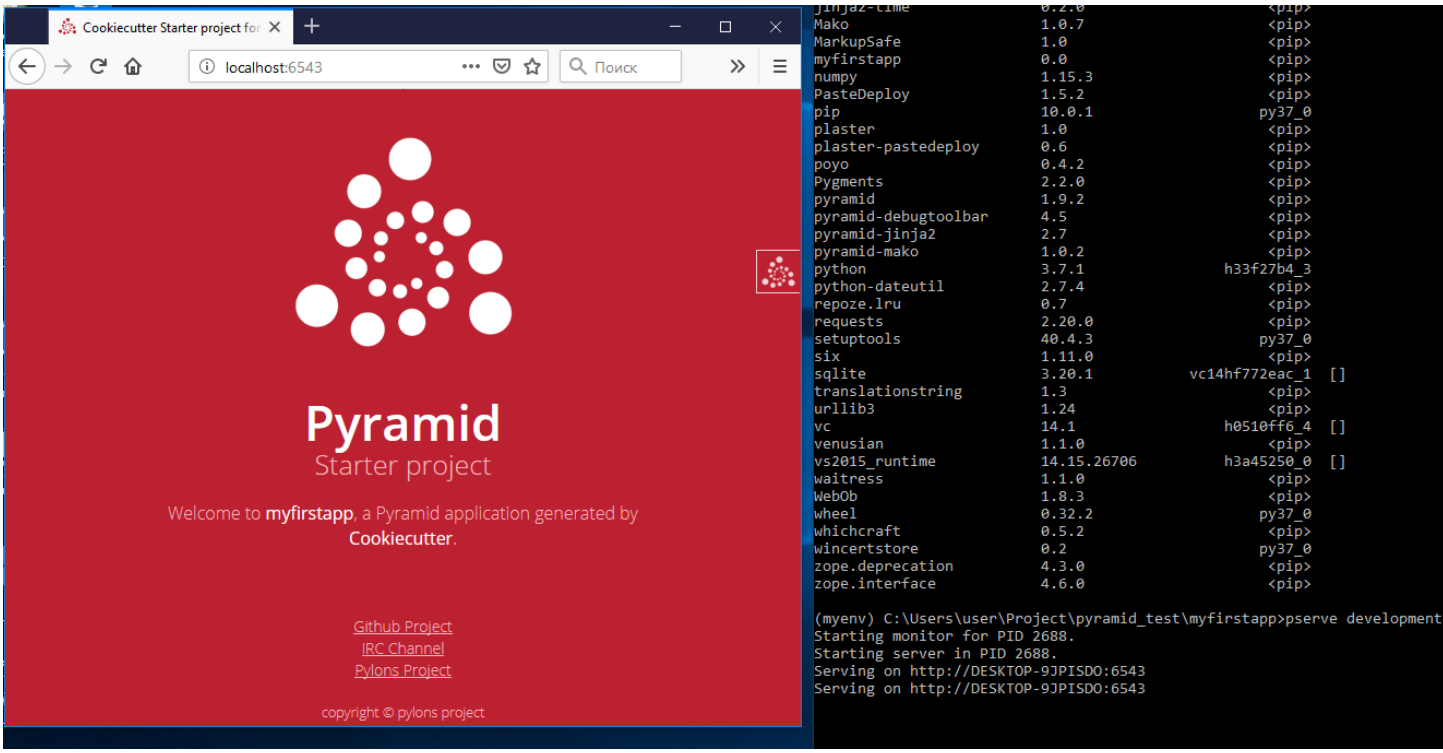

Командная строка
Microsoft Windows XP [Версия 5.1.2600]
(C) Корпорация Майкрософт, 1985-2001.
C:\Documents and Settings\dimon>cd C:\Projects\pyramid_sacrud\example
C:\Projects\pyramid_sacrud\example>workon pyramid_sacrud
(pyramid_sacrud) C:\Projects\pyramid_sacrud\example>python setup.py develop


Командная строка
(pyramid_sacrud) C:\Projects\pyramid_sacrud\example>pip list
chameleon (2.22)
colander (1.0)
colanderalchemy (0.3.2.post1)
deform (2.0a2)
jinja2 (2.7.3)
markupsafe (0.23)
paginate (0.4.3)
paginate-sqlalchemy (0.2.0)
pastedeploy (1.5.2)
peppercorn (0.5)
pip (7.1.0)
pyramid (1.6a2)
pyramid-jinja2 (2.5)
pyramid-sacrud (0.1.1.dev1, c:\projects\pyramid_sacrud)
pyramid-sacrud-example (1.0)
repoze.lru (0.6)
sacrud (0.3.1)
sacrud-deform (0.0.8)
setuptools (18.0.1)
sqlalchemy (1.0.6)
transaction (1.4.4)
translationstring (1.3)
venusian (1.0)
webob (1.4.1)
wheel (0.24.0)
zope.deprecation (4.1.2)
zope.interface (4.1.2)
zope.sqlalchemy (0.7.6)
(pyramid_sacrud) C:\Projects\pyramid_sacrud\example>


Командная строка - pserve development.ini
Microsoft Windows XP [Версия 5.1.2600]
(C) Корпорация Майкрософт, 1985-2001.
C:\Documents and Settings\dimon>cd C:\Projects\pyramid_sacrud\example
C:\Projects\pyramid_sacrud\example>workon pyramid_sacrud
(pyramid_sacrud) C:\Projects\pyramid_sacrud\example>pserve development.ini
Starting server in PID 4004.
Starting HTTP server on http://0.0.0.0:6543


Файл Правка Вид Журнал Закладки Инструменты Справка
SACRUD
localhost:6543/admin/
Поиск
SACRUD
DASHBOARD
Catalouge
groups
goods
Auth system
user
© 2013-2015, ITCase pyramid_sacrud


Файл Правка Вид Журнал Закладки Инструменты Справка
SACRUD
localhost:6543/admin/goods/update/id/2
Поиск
SACRUD
DASHBOARD › GOODS › ID=2
GOODS
Save
Delete
Cancel
Id
2
Name
Fridge
group
Electronics
visible
archive
Save
Delete
Cancel



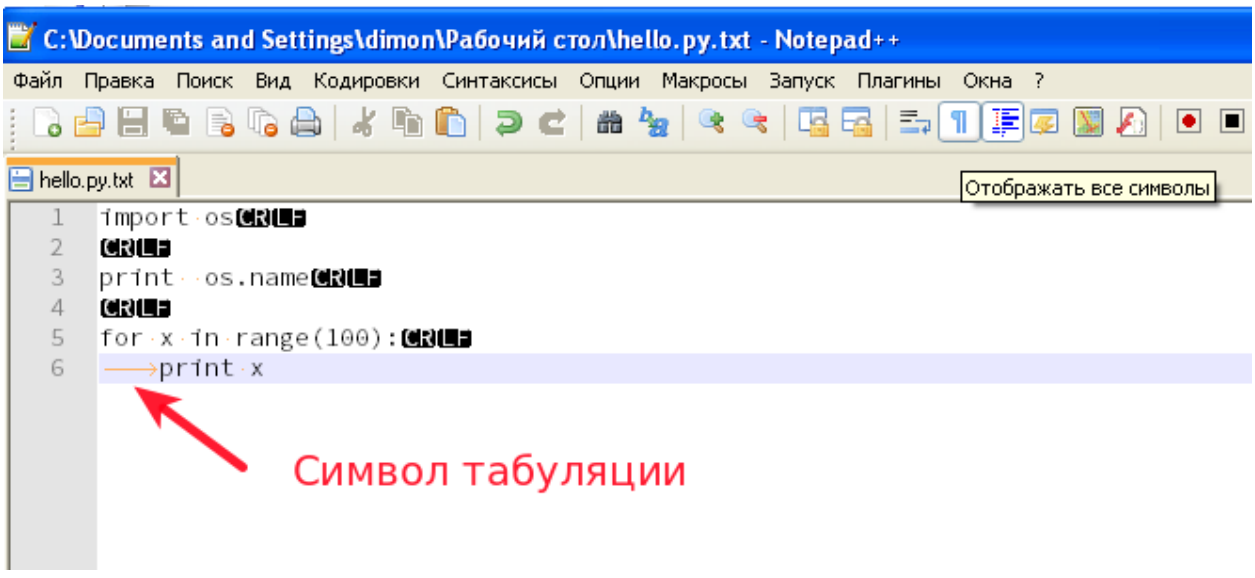


Рис. 1: По умолчанию в Notepad++ клавиша <Tab> вставляет символ табуляции.

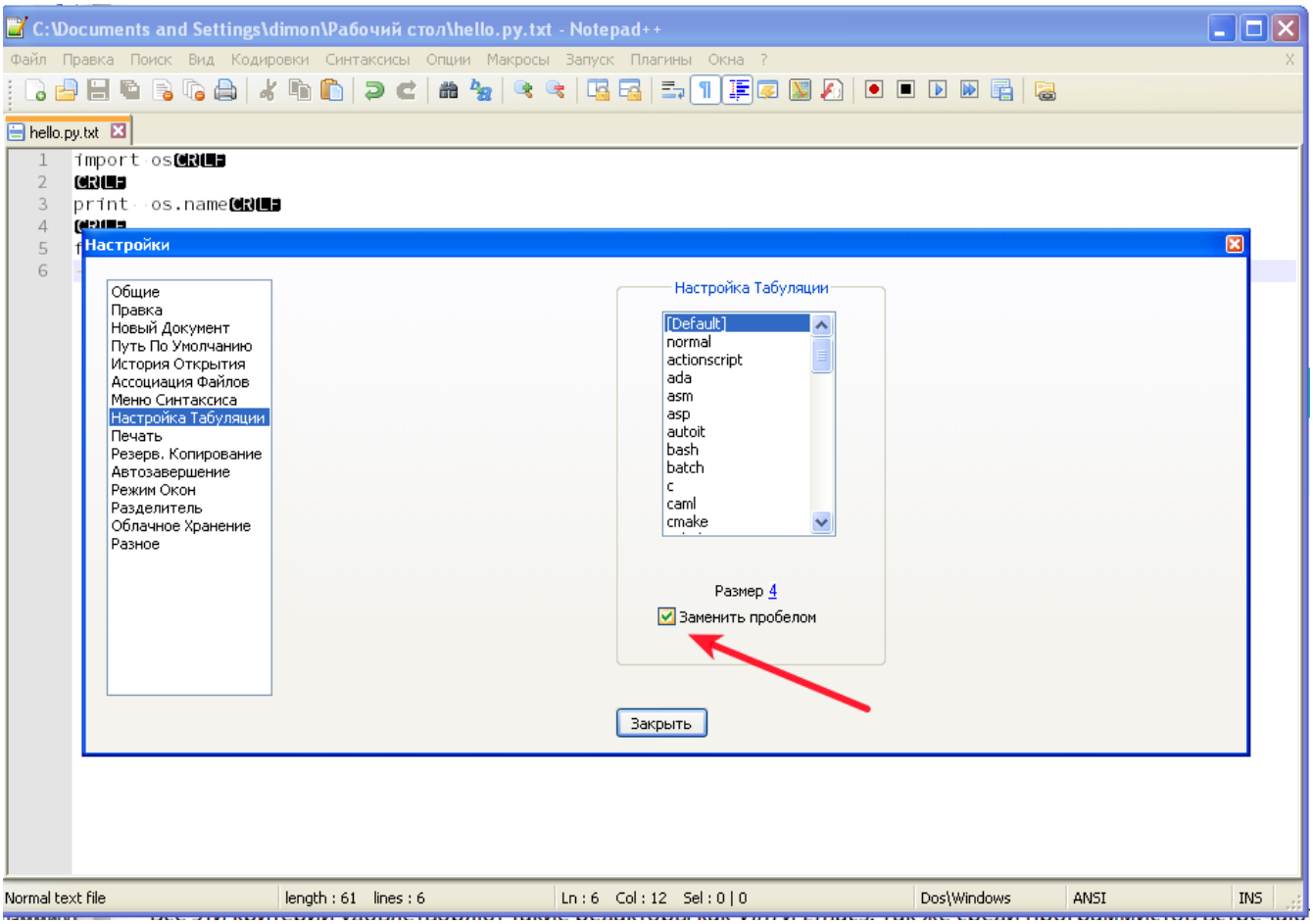
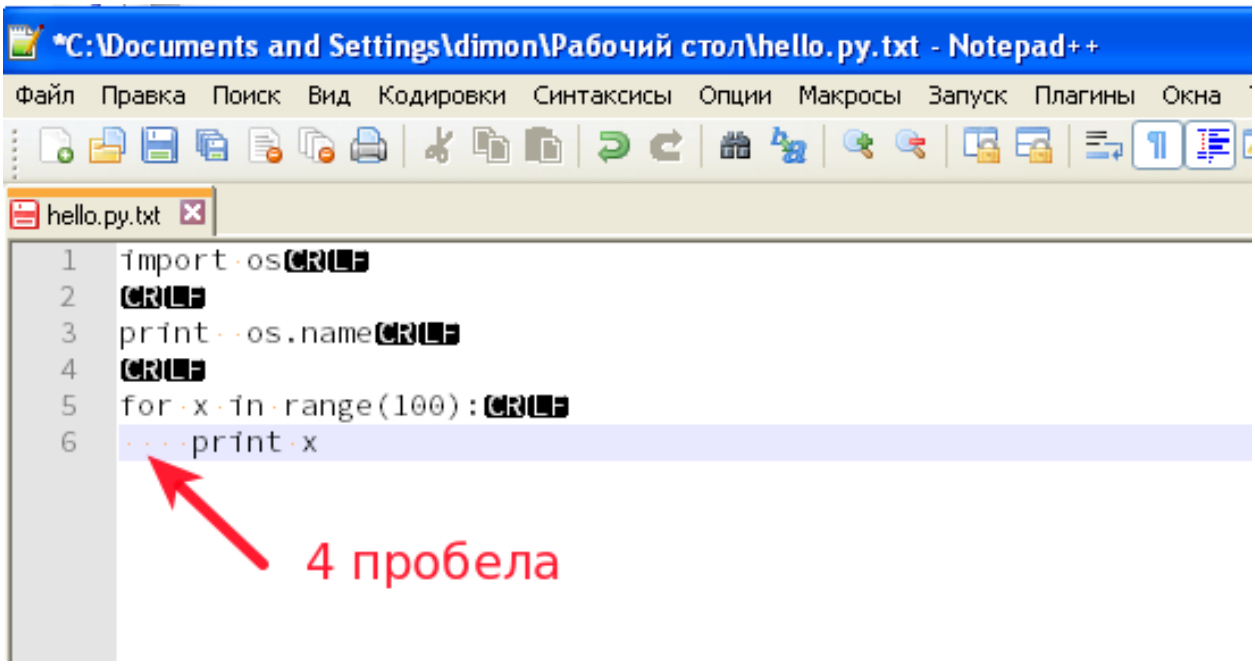


Рис. 2: Настройки табуляции в Notepad++.



Алфавитный указатель

Symbols

A

B

C

D

E

F

G

S

T

U

V

W

Z