Содержание

1. О Django

Django – веб-фреймворк для перфекционистов с дедлайнами.

Основной принцип – DRY – Don’t Repeat Yourself (Не Повторяйся!).

Django является свободным фреймворком для веб-приложений на языке Python. Сайт на Django строится из одного или нескольких приложений, которые рекомендуется делать отчуждаемыми и подключаемыми. В отличие от других фреймворков обработчики URL в Django конфигурируются явно при помощи регулярных выражений, а не выводятся автоматически из структуры моделей контроллеров.

Для работы с базой данных Django использует собственный ORM (Object-Relational Mapping), в котором модель данных описывается классами Python, и по ней генерируется схема базы данных.

Возможности Django:

- ORM, API доступа к БД с поддержкой транзакций;
- Встроенный интерфейс администратора с уже имеющимися переводами на многие языки;
- Диспетчер URL на основе регулярных выражений;
- Расширяемая система шаблонов с тегами и наследованием;
- Система кеширования;
- Интернационализация;
- Подключаемая архитектура приложений, которые можно устанавливать на любые Django-сайты;
- Шаблоны функций контроллеров;
- Авторизация и аутентификация, подключение внешних модулей аутентификации: LDAP, OpenID и тп;
- Библиотека для работы с формами (наследование, построение форм по существующей БД).

2. Шаблон разработки

Приложения, реализованные на платформе Django, работают на основе шаблона MVC (Model-View-Controller, Модель-Представление-Контроллер).

Шаблон MVC описывает такую архитектуру программного обеспечения, в которой модель данных приложения, пользовательский интерфейс и управляющая логика разделены на три отдельных элемента так, что модификация одного из компонентов оказывает минимальное воздействие на остальные. Данный шаблон разделяет данные, их представление и обработку действий пользователя на три отдельных компонента:

- Модель (model), которая предоставляет данные(для view) и реагирует на запросы(от controller), изменяя свое состояние;

- Представление (view), отвечающее за отображение информации;
- Контроллер (controller), который интерпретирует данные от пользователя и информирует модель и представление о необходимости соответствующей реакции.

3. Как работает система, написанная на Django

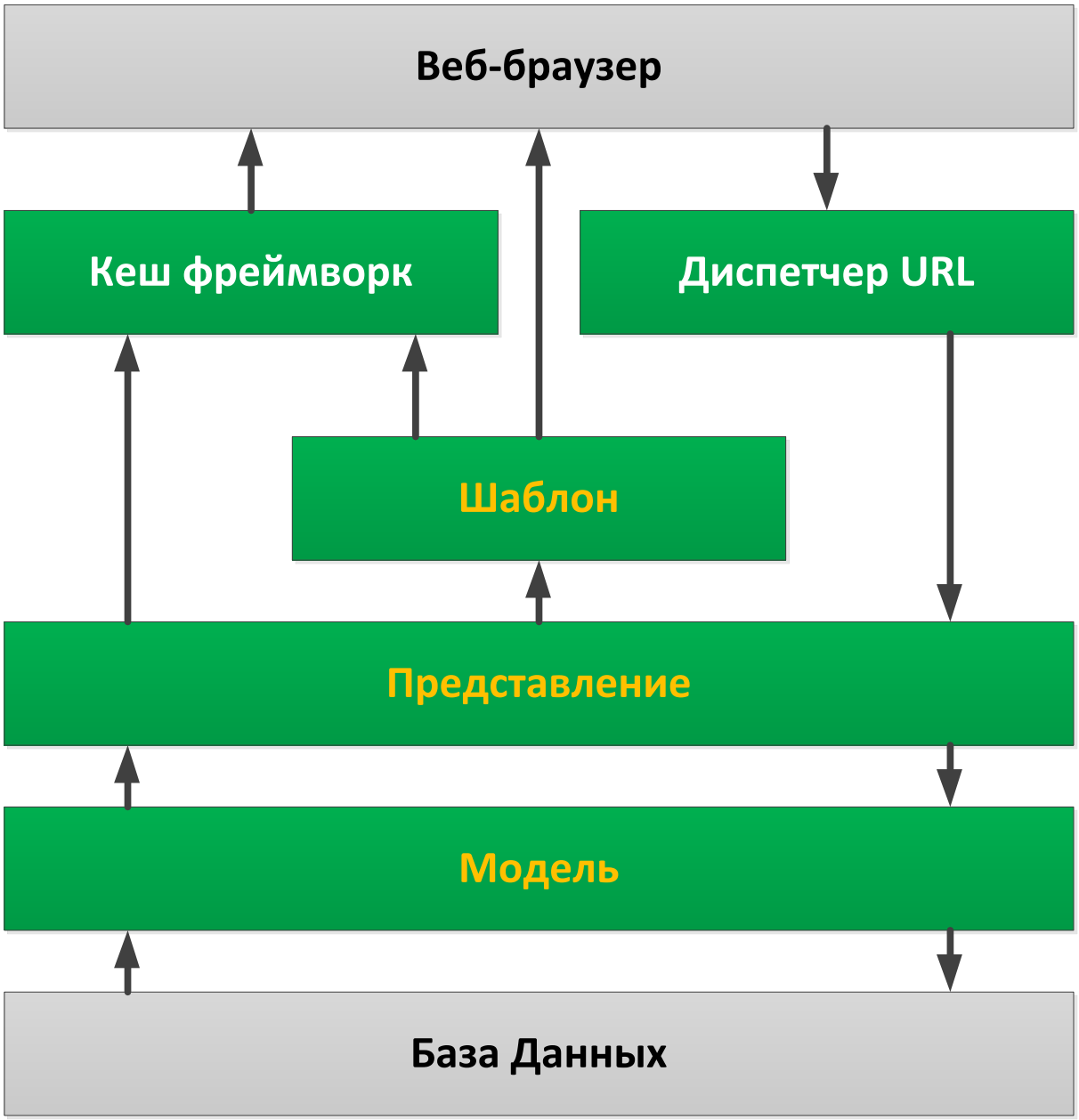

Рис. 1. Схема работы Django-приложения

Файл *models.py* описывает таблицу базы данных.

```
from django.db import models
class Book(models.Model):
    name = models.CharField(max_length=50)
    pub_date = models.DateField()
```

Файл *views.py* описывает логику приложения.

```
from django.shortcuts import render_to_response
from models import Book
def latest_books(request):
    book_list = Book.objects.order_by('-pub_date')[:10]
    return render_to_response('latest_books.html', {'book_list': book_list})
```

Файл *urls.py* описывает соответствие URL логике приложения.

```
from django.conf.urls.defaults import *
import views
urlpatterns = patterns('',
    (r'^latest/$', views.latest_books),
)
```

Файл *latest_books.html* описывает HTML шаблон, используемый при выводе страницы.

```
<html><head><title>Книги</title></head>
<body>
<h1>Книги</h1>
<ul>
{% for book in book_list %}
<li>{{ book.name }}</li>
{% endfor %}
</ul>
</body></html>
```

Рассмотрим на примере:

Файл *models.py* содержит описание таблицы базы данных, представленное в виде класса Python. Такой класс называется моделью. С помощью данного класса вы можете создавать, получать, обновлять и удалять записи в таблице вашей базы данных, используя простой код на языке Python вместо использования повторяющихся SQL команд.

Файл *views.py* содержит логику отображения страницы в функции *latest_books()*. Такая функция называется представлением.

Файл *urls.py* определяет какое именно представление будет вызвано для URL, заданного в виде шаблона. В данном случае URL */latest/* будет обработано функцией *latest_books()*. Другими словами, если имя вашего домена *example.com*, то любой доступ к *http://example.com/latest/* будет обработан функцией *latest_books()*.

Файл *latest_books.html* является HTML шаблоном, который описывает дизайн страницы. Он использует шаблонный язык с основными логическими операторами — *{% for book in book_list %}*.

Рассмотрим подробнее, что происходит, когда пользователь набирает, например, http://something.com/example/.

При начальном запуске сервера соответствующий скрипт ищет в текущем каталоге (в том же, где находится *manage.py*) файл *settings.py*, где указаны основные настройки проекта. Оттуда он считывает основные параметры, такие как DATABASE_NAME (параметры соединения с БД), TEMPLATE_DIRS (директории с html-шаблонами). Самым главным параметром является ROOT_URLCONF, который указывает Django, какой модуль Python следует использовать в качестве привязки для данного сайта.

В текущем приложении параметр установлен в значение

ROOT_URLCONF = `myproject.urls` , которое соответствует файлу *myproject/urls.py*.

Когда приходит запрос на определенный URL - */example/* - Django загружает файл привязок, указанный параметром ROOT_URLCONF. Затем проверяет каждый шаблон этого файла по порядку, сравнивая запрошенный URL с шаблонами, пока не найдет подходящий:

urls.py:

```
(r'^example/$', example_noargs),
```

Если совпадение найдено, Django вызывает функцию представления, ассоциированную с шаблоном (*example_noargs*), передавая ей HttpRequest в качестве первого аргумента:

views.py:

```
def example_noargs(request):
    ....
    return render_to_response('example.html',locals())
```

Функция представления должна возвращать HttpResponse. В данном случае она обращается к шаблону, хранящему в директории, указанной в TEMPLATE_DIRS:

templates/example.html:

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

{% block title %}Example page{% endblock %}
```

{% block content %}

* This is example page *

{% endblock %}

Она загружает базовую страницу, а затем – страницу – наследника.

4. Установка Django

Полный пакет (Python, Apache, MySQL, PostgreSQL, Python, SQLite) - http://bitnami.org/stack/djangostack.

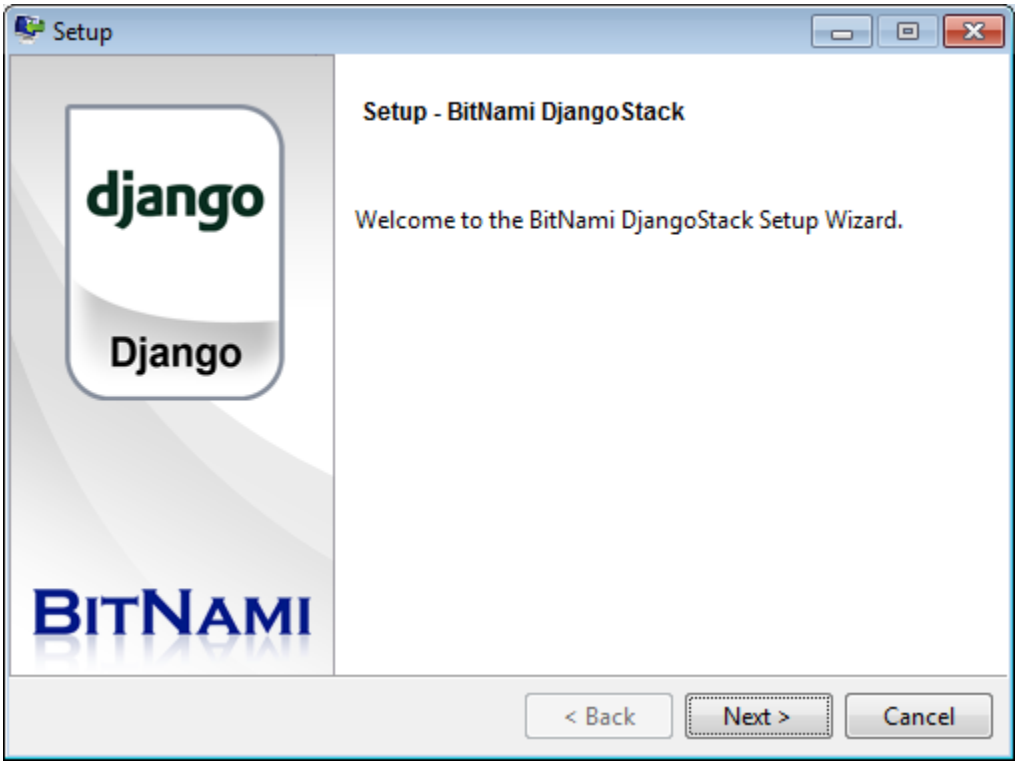

Рис. 2 Установка DjangoStack

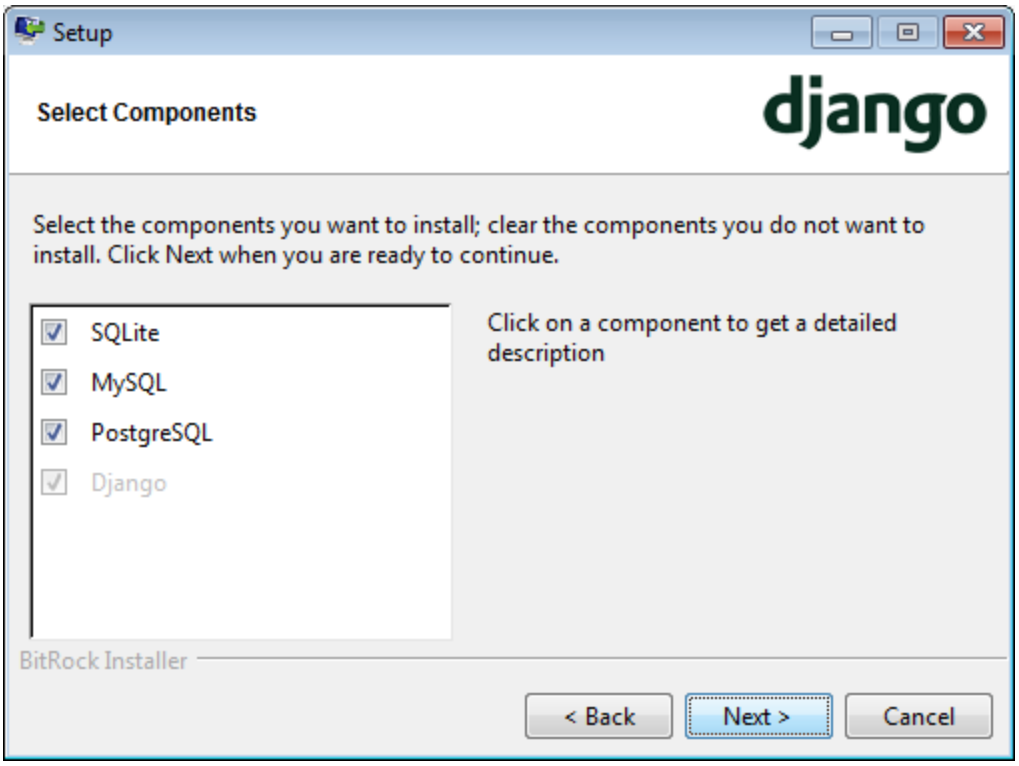

Рис. 3 Установка DjangoStack. Выбор СУБД

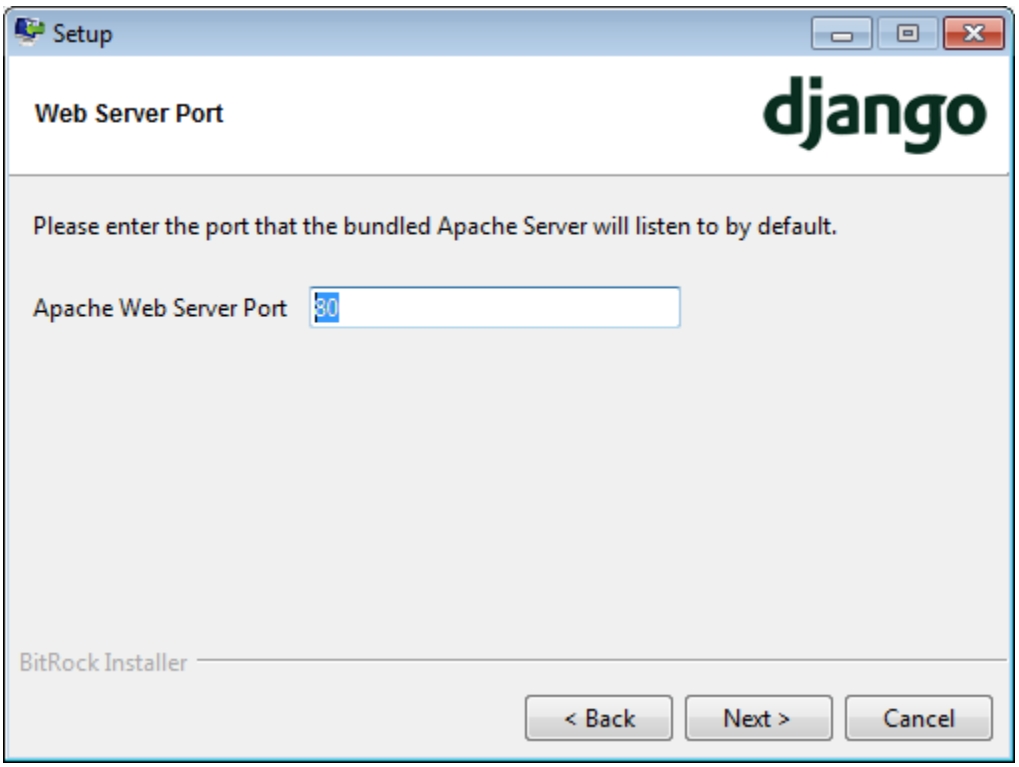

Рис. 4 Установка DjangoStack. Настройка порта

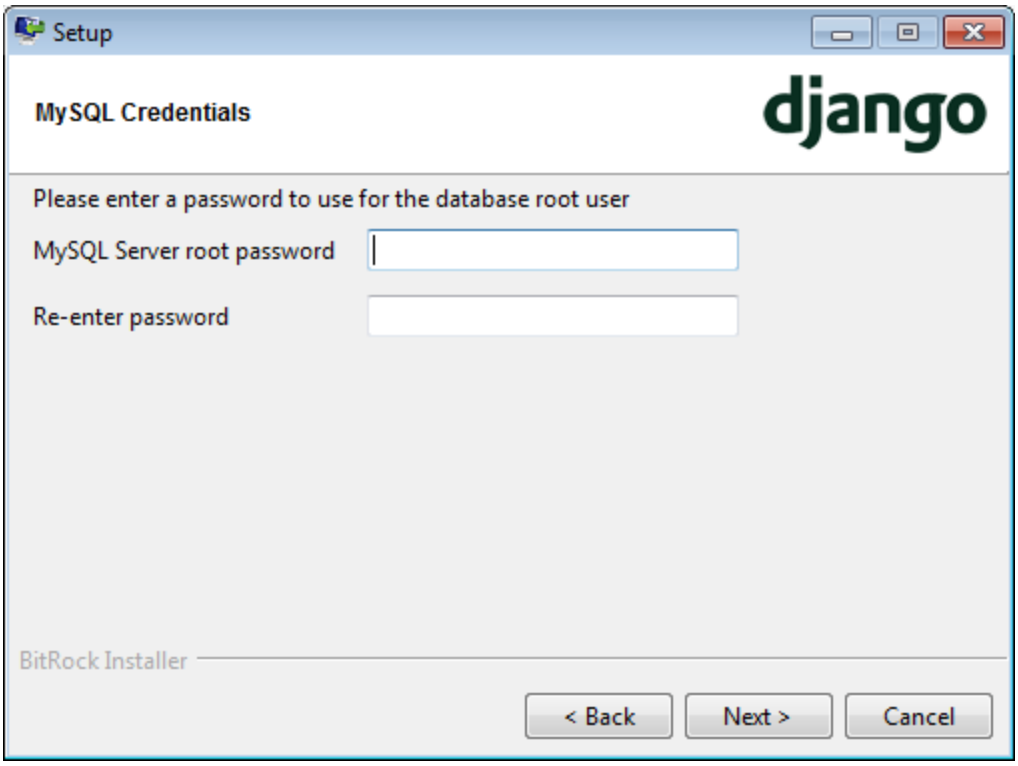

Рис. 5 Установка DjangoStack. Настройка логина и пароля для MySQL

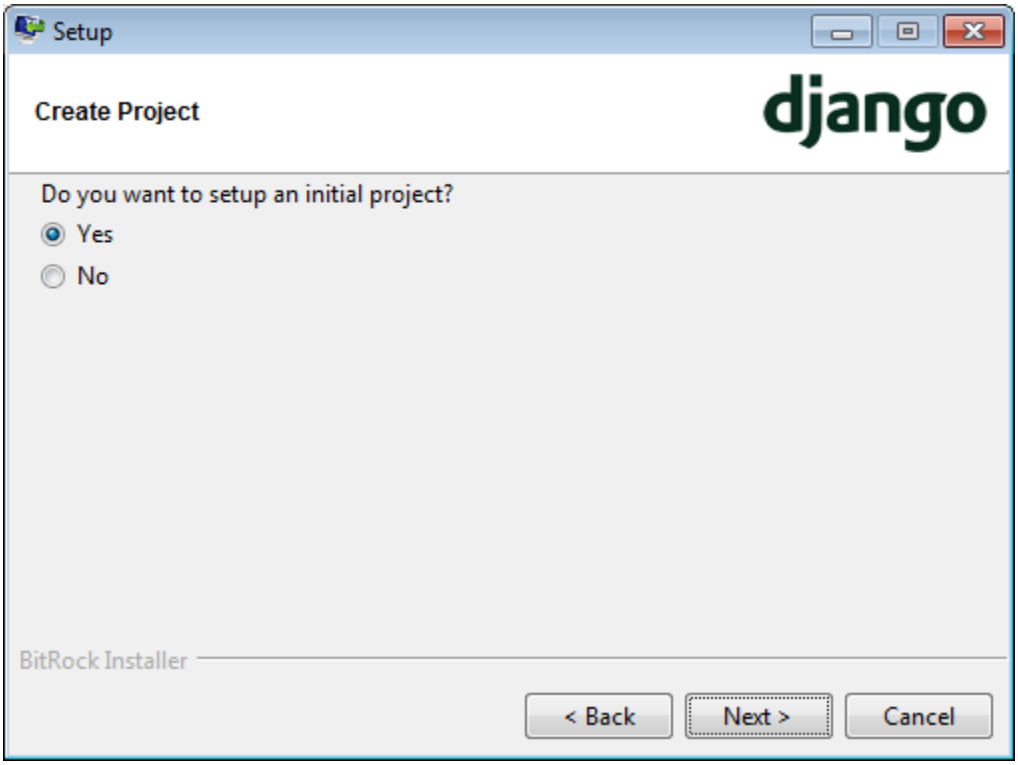

Рис. 6 Установка DjangoStack. Создание проекта

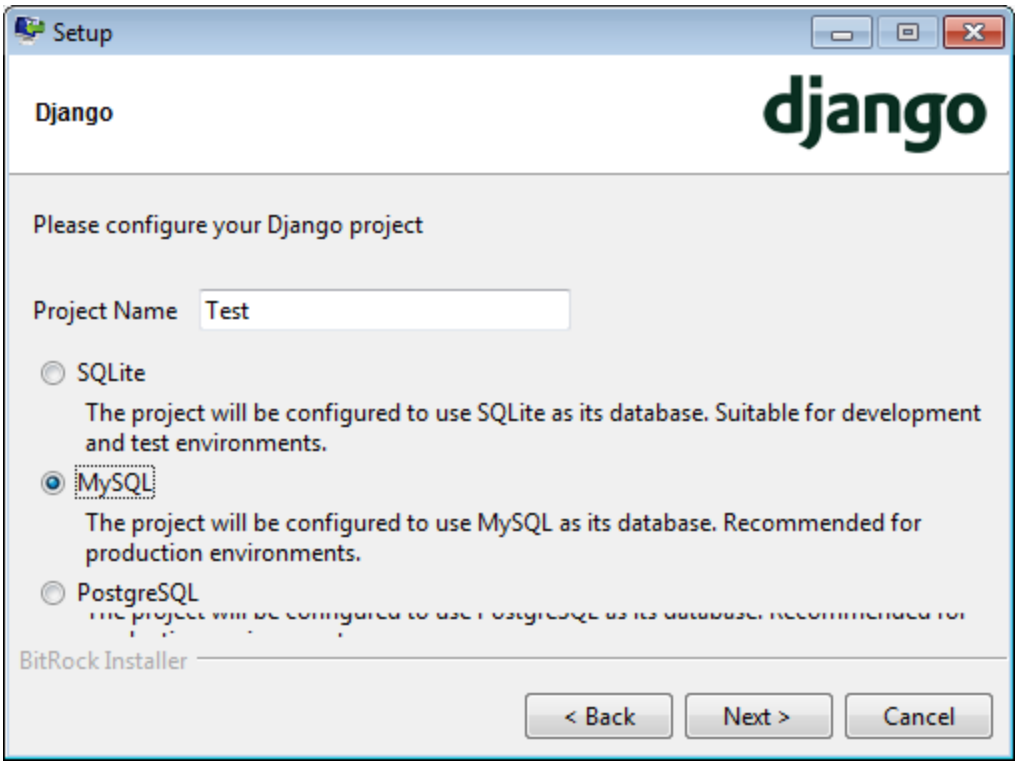

Рис. 7 Установка DjangoStack. Настройка параметров проекта

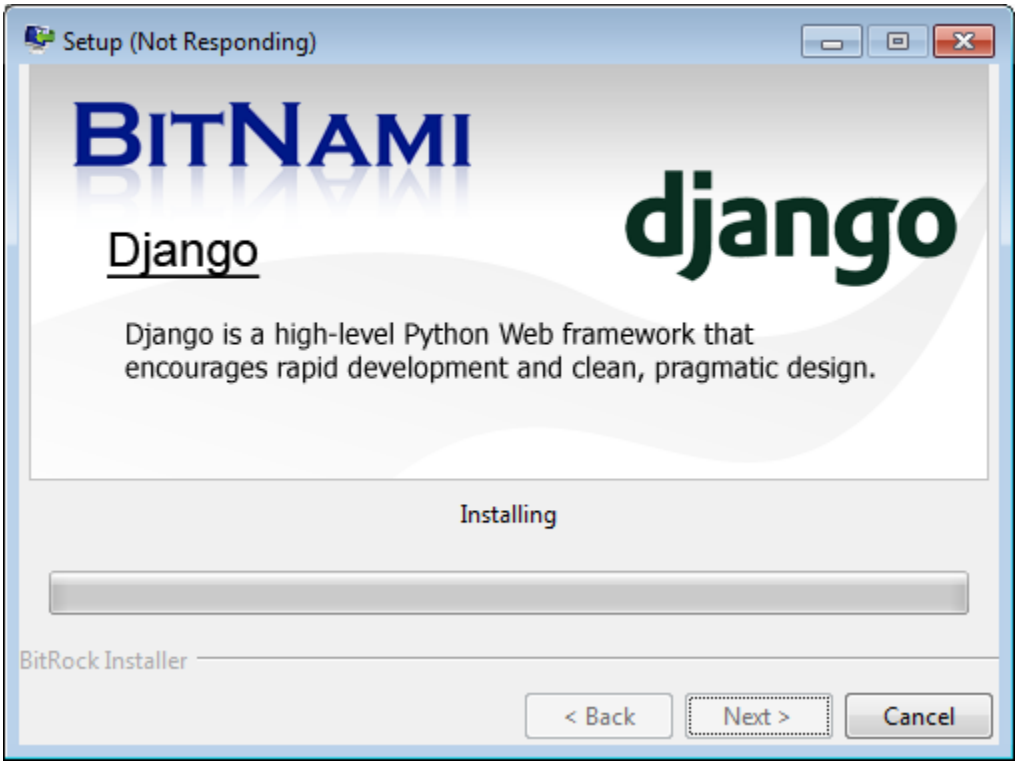

Рис. 8 Установка DjangoStack. Процесс инсталляции

Рис. 9 Установка DjangoStack. Завершение

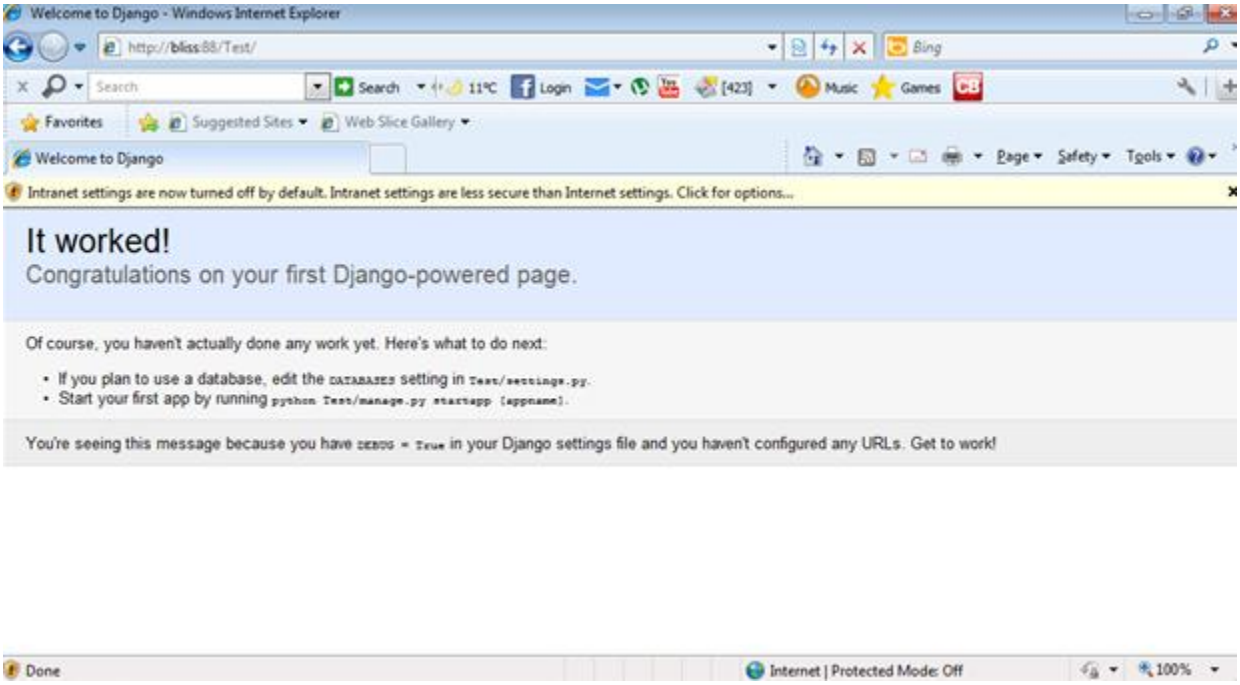

Рис. 10 Запуск первого проекта

Проверим, что Django установлен.

Запустим консоль. Пока не уставновлены переменные окружения, зайдем в корневую директорию python (BitName DjangoStack\python). Затем в командной строке наберем *python*.

Попробуем затем импортировать модуль django: *import django.* Если django был успешно установлен, то модуль импортируется без каких-либо ошибок. При желании можно узнать версию установленного django: *django.VERSION*

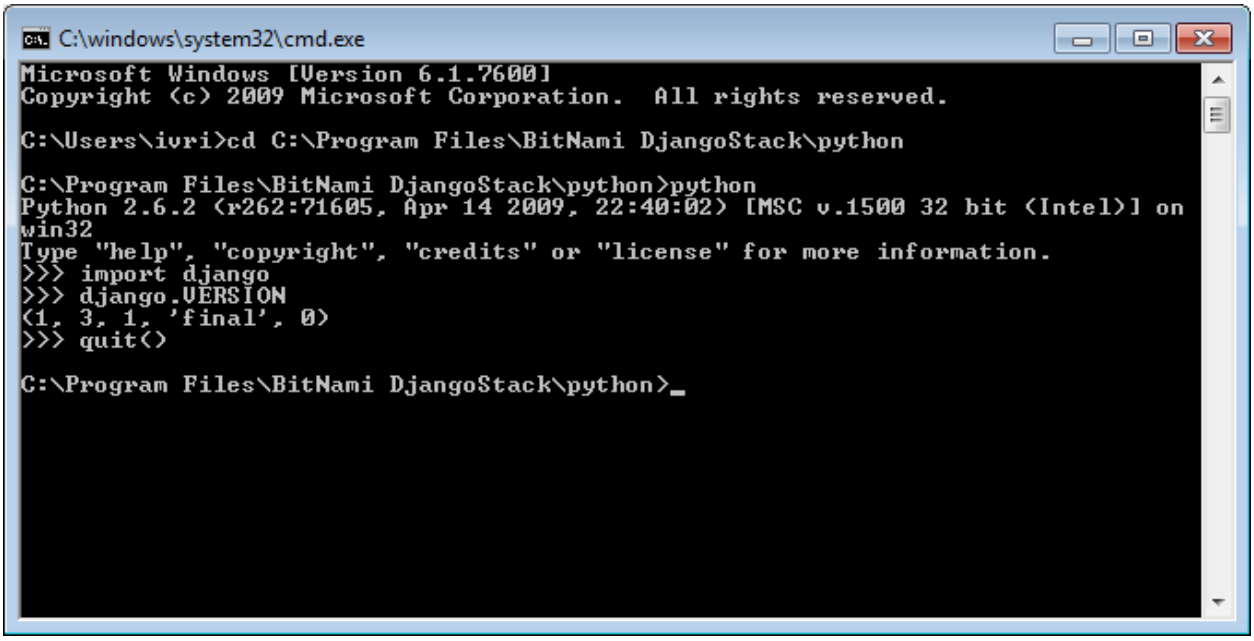

Рис. 11 Запрос версии Django

5. Установка переменных окружения

Для последующего удобства при работе с Python и Django установим переменные окружения (пути до корневых директорий Python и Django):

set PATH=%PATH%;C:\Program Files\BitNami DjangoStack\python
set PATH=%PATH%; C:\Program Files\BitNami DjangoStack\apps\django\django\bin

6. Создание проекта

Попробуем самостоятельно создать проект.
Перейдите в каталоге, где вы планируете размещать свои проекты на Django.
Запустите скрипт:

django-admin.py startproject cutesite

Посмотрим на результаты работы команды. Команда создаст поддиректорию под названием *cutesite:*

cutesite/
__init__.py
manage.py
settings.py
urls.py

Опишем назначение каждого файла:

__init__.py: Файл необходим для того, чтобы Python рассматривал данный каталог как пакет, т.е., как группу модулей. Это пустой файл и обычно вам не требуется добавлять что-либо в него.

manage.py: Это утилита командной строки, которая позволяет вам взаимодействовать с проектом различными методами. Наберите python manage.py help для получения информации о возможностях утилиты. Вы не должны изменять содержимое данного файла, он создан в данном каталоге в целях удобства.

settings.py: Настройки для текущего проекта Django. Посмотрите на содержимое файла, чтобы иметь представление о типах доступных параметров и их значениях по умолчанию.

urls.py: Описания URL для текущего проекта Django, так сказать «оглавление» для вашего сайта. На момент создания должен быть пустым (FIXME: по-моему авторы гонят.)

Несмотря на свой небольшой размер, эти файлы формируют работоспособное приложение Django.

7. Запуск сервера разработки

Сервер разработки Django (также называемый «runserver», по имени команды, которая его запускает) — это встроенный лёгкий веб сервер, который вы можете использовать в процессе разработки вашего сайта.

Для запуска следует перейти в каталог проекта (*cd cutesite*) и выполнить команду:

python manage.py runserver

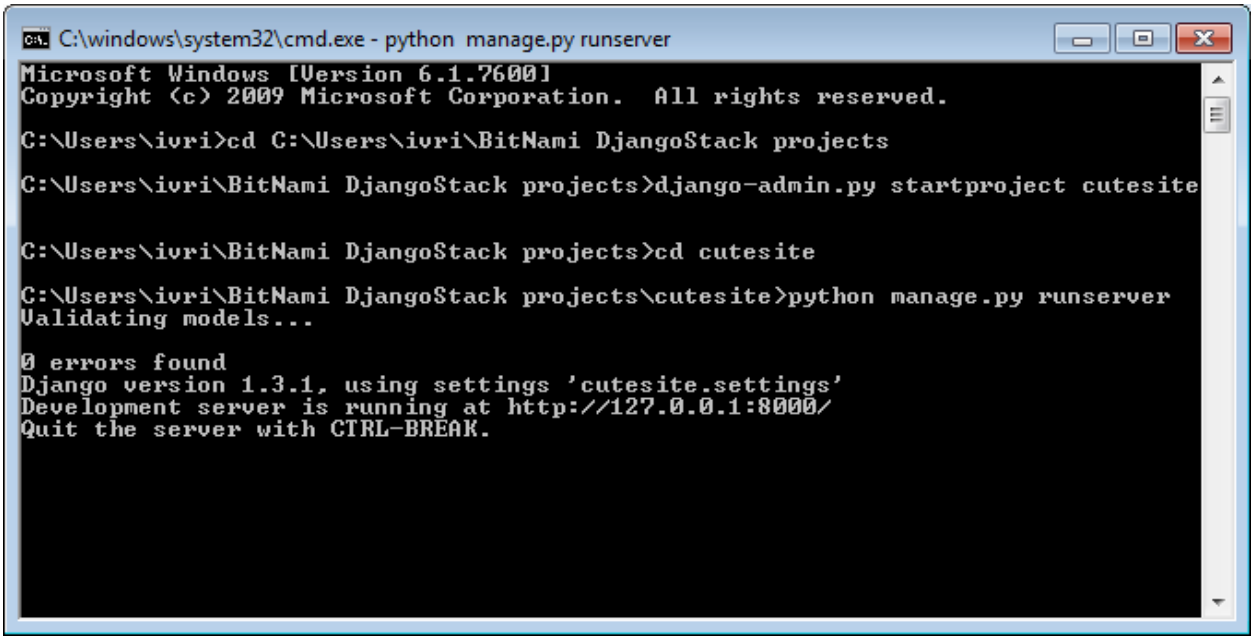

Рис. 12 Запуск сервера разработки

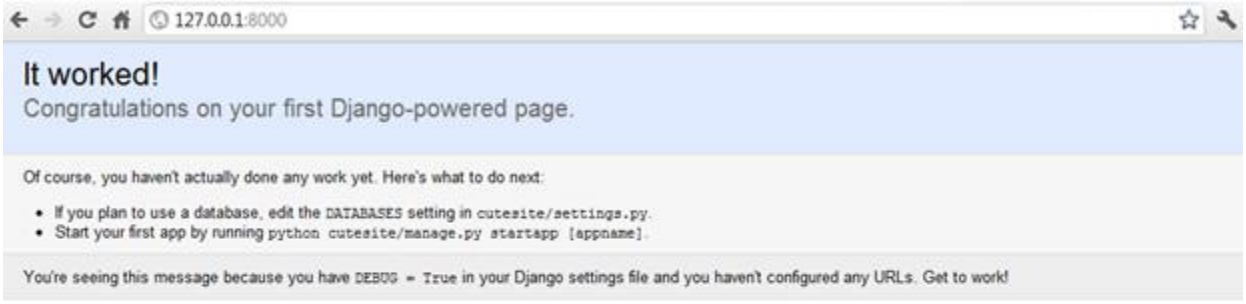

Рис. 13 Первый проект. Страница по умолчанию

По умолчанию, команда *runserver* запускает сервер разработки на порту *8000*, принимая только локальные соединения. Если требуется изменить порт, его можно указать в аргументах командной строки:

python manage.py runserver 8000

8. Первое приложение Hello, World

Первое представление

В основной директории проекта создадим файл *views.py* .

views.py:

```
# -*- coding: utf-8 -*-
from django.http import HttpResponse

def hello(request):
    return HttpResponse("Здравствуй, Мир")
```

Пройдёмся по коду строка за строкой:

Сначала мы импортируем класс *HttpResponse*, который определён в модуле *django.http*. Нам необходимо импортировать этот класс, так как он используется в нашем коде.

Затем мы определяем функцию *hello* — функцию представления.

Первая привязка

Для того, чтобы связать представление (страницу) с конкретным адресом (урлом) нужно добавить соответствующую привязку.

Файл привязки URL можно рассматривать как таблицу с содержанием вашего сайта. Проще говоря, этот файл определяет соответствие между URL и функциями представления, которые должны быть вызваны для этих URL. Именно так вы указываете Django: «Для данного URL, вызывай этот код, а для этого URL вызывай вот этот код.» Например, «Если кто-нибудь посетит URL */foo/*, вызывай функцию представления *foo_view()*, которая расположена в модуле *views.py*.»

Когда вы запустили *django-admin.py startproject*, скрипт автоматически создал файл привязки: *urls.py*. По умолчанию, он выглядит примерно так:

```
from django.conf.urls.defaults import patterns, include, url

# Uncomment the next two lines to enable the admin:
# from django.contrib import admin
# admin.autodiscover()

urlpatterns = patterns('',
    # Examples:
    # url(r'^$', 'cutesite.views.home', name='home'),
    # url(r'^cutesite/', include('cutesite.foo.urls')),

    # Uncomment the admin/doc line below to enable admin documentation:
    # url(r'^admin/doc/', include('django.contrib.admindocs.urls')),
```

```
    # Uncomment the next line to enable the admin:
    # url(r'^admin/', include(admin.site.urls)),
)
```

Этот стандартный файл привязки включает несколько закомментированных блоков, просто раскомментируйте их, если вам нужен соответствующий функционал. Если проигнорировать закомментированные строки, то останется только это:

```
from django.conf.urls.defaults import *

urlpatterns = patterns('',
)
```

Давайте пройдём по каждой строчке этого кода:

Первая строка импортирует все объекты из модуля *django.conf.urls.defaults*, который содержит всю инфраструктуру для работы с файлами привязки. Там же находится функция *patterns*.

Вторая строка вызывает функцию *patterns* и сохраняет результат в переменную *urlpatterns*. Функция *patterns* принимает единственный аргумент — пустую строку.

Основным элементом данного кода является переменная *urlpatterns*, которую Django ожидает найти в вашем файле привязки. Эта переменная определяет соответствие между списком URL и кодом, который обрабатывает данные URL. По умолчанию, как вы можете видеть, файл привязки пуст — ваше Django приложение ещё не настроено. (Именно так Django определяет, что следует показывать страницу «Welcome to Django». Если ваш файл привязки пуст, Django предполагает, что вы только что начали новый проект и, следовательно, отображает данное сообщение.)

Для того, чтобы добавить URL и представление в файл привязки, просто добавьте кортеж, объединяющий URL и функцию представления. Ниже показана привязка для нашего представления *hello*:

```
from django.conf.urls.defaults import *
from cutesite.views import hello

urlpatterns = patterns('',
    ('^hello/$', hello),
)
```

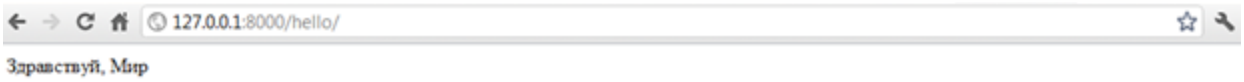

Рис. 14 Первая страница

9. Создание динамического контента

Добавим во *views.py* функцию

```
import datetime
def current_datetime(request):
    now = datetime.datetime.now()
    html = "<html><body>It is now %s.</body></html>" % now
    return HttpResponse(html)
```

После добавления новой функции в *views.py*, следует добавить шаблон в *urls.py*, чтобы указать Django какой именно URL должен обрабатываться с помощью этой функции представления. Подойдёт что-нибудь подобное */time/*:

```
from django.conf.urls.defaults import *
from cutesite.views import hello, current_datetime

urlpatterns = patterns('',
    ('^hello/$', hello),
    ('^time/$', current_datetime),
)
```

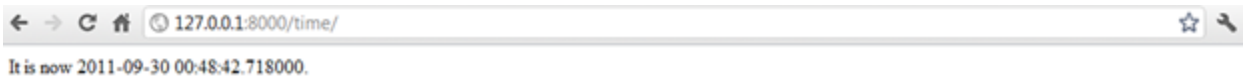

Рис. 15 Страница с динамическим контентом

10.Создание динамических URL

Давайте создадим третье представление, которое отображает текущую дату и время со смещением на указанное количество часов. Цель — реализовать функциональность, с помощью которой сайт будет по URL */time/plus/1/* отдавать текущее время сдвинутое на час вперёд, а по URL */time/plus/2/* — на два часа вперёд и так далее.

Для этого добавим еще один шаблон схемы URL в *urls.py*:

```
(r'^time/plus/(\d{1,2})/$', hours_ahead),
```

Символ *r* в начале регулярного выражения указывает, что строка является «сырой» — в её содержимом не следует интерпретировать обратные слеши. В обычной строке Python, обратные слеши используются для экранирования особых символов, например, \n — односимвольная строка. Если предварить её символом r, сделав её «сырой», Python не будет выполнять экранирование — таким образом, r'\n' станет двухсимвольной строкой, содержащей обратный слеш и символ n.

Выражение */\d{1,2}* означает любое число однозначное или двузначное число.

Подробнее о регулярных выражениях : *http://ru.wikibooks.org/wiki/Регулярные_выражения* .

Скобки определяют те значения, которые будут передаваться в функцию *hours_ahead* в качестве параметров.

Представление *hours_ahead* очень похоже на представление *current_datetime*, которое мы написали ранее, с одним только отличием: оно принимает дополнительный аргумент — количество часов смещения. Вот его код:

```
from django.http import Http404, HttpResponse
import datetime

def hours_ahead(request, offset):
    try:
        offset = int(offset)
    except ValueError:
        raise Http404()
    dt = datetime.datetime.now() + datetime.timedelta(hours=offset)
    html = "<html><body>In %s hour(s), it will be %s.</body></html>" %
(offset, dt)
    return HttpResponse(html)
```

Рассмотрим каждую строчку этого кода.

Функция представления, *hours_ahead*, принимает два параметра: *request* и *offset*.

Параметр *request* является объектом *HttpRequest*, аналогичным используемому в представлениях *hello* и *current_datetime*. Повторим ещё раз: каждое представление всегда принимает объект *HttpRequest* в качестве первого аргумента.

Параметр *offset* является строкой, выделенной скобками в шаблоне URL. Например, если запрошенный URL был */time/plus/3/*, тогда offset будет содержать строку '3'. Если */time/plus/21/* — строка '21'. Следует отметить, что выделенные значения всегда будут строками, не целыми, даже если строка составлена только из цифр.

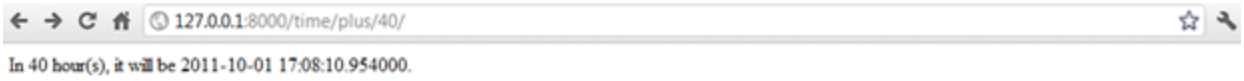

Рис. 16 Динамический URL

11.Использование шаблонов

Шаблон Django — это строка текста, которая предназначена для разделения представления документа от его данных. Шаблон определяет места подстановки и различные виды основной логики (шаблонные теги), которая управляет отображением документа. Обычно шаблоны используются для создания HTML, но шаблоны Django также способны участвовать в генерации любого текстового формата.

Django предоставляет удобный и мощный API для загрузки шаблонов с диска с целью удаления избыточности в вызовах подгрузки шаблонов и в самих шаблонах.

Для использования этого API сначала требуется указать среде разработки где располагаются шаблоны. Это делать надо в файле конфигурации проекта — settings.py.

В *settings.py* есть параметр TEMPLATE_DIRS, который указывает механизму шаблонной системы Django пути, по которым следует искать шаблоны. Укажите требуемый каталог, в котором вы будете хранить шаблоны, например так:

```
TEMPLATE_DIRS = (
   'C:/..cutesite/templates',
)
```

Надо отметить несколько моментов:

Вы можете указать любой каталог, который доступен на чтение пользователю, от которого работает ваш веб-сервер. Если вы не можете придумать такое место, мы рекомендуем создать каталог *templates* в каталоге вашего проекта.

Если ваш параметр TEMPLATE_DIRS содержит только один каталог, не забудьте запятую в конце строки с каталогом шаблонов!

Неправильно:

```
# Missing comma!
TEMPLATE_DIRS = (
   '/home/django/mysite/templates'
)
```

Правильно:

```
# Comma correctly in place.
TEMPLATE_DIRS = (
   '/home/django/mysite/templates',
)
```

Дело в том, что Python требует наличия запятой в одноэлементном кортеже, чтобы отличать его от обычного выражения окружённого скобками. На этом обычно попадаются новички.

Если вы используете Windows, включите букву диска и используйте Unix-стиль с прямыми слэшами вместо обратных, вот так:

```
TEMPLATE_DIRS = (
   'C:/www/django/templates',
)
```

Проще всего использовать абсолютные пути (т.е. пути до каталогов, которые начинаются от корня файловой системы). Если вы желаете быть более гибким и независимым в данном вопросе, вы можете воспользоваться тем, что файлы настроек в Django — это всего лишь код на языке Python и содержимое TEMPLATE_DIRS можно изменять динамически, например:

```
import os.path

TEMPLATE_DIRS = (
   os.path.join(os.path.dirname(__file__), 'templates').replace('\\','/'),
)
```

Этот пример использует «волшебную» переменную *__file__*, которая автоматически заменяется именем файла модуля Python, в котором располагается данный код. Она получает имя каталога, в котором содержится *settings.py (os.path.dirname),* соединяет его с *templates* кроссплатформенным способом (*os.path.join*) и затем проверяет, что используются только прямые слеши, а обратных нет (в случае Windows).

Следующим шагом будет создание файла представления *current_datetime.html*, в который перенесем основную статическую информацию.

Файл поместим в директорию templates.

current_datetime.html:

```
<html><body>It is now {{ current_date }}</body></html>
```

Current_datetime в данном случае является переменной.

В шаблонах Django можно использовать также такие теги, как if-else, for, ifequal, а также фильтрацию данных (подробнее - http://djbook.ru/ch04s03.html).

Изменим соответствующую функцию в файле *views.py*:

Views.py

```
from django.shortcuts import render_to_response
...
def current_datetime(request):
   now = datetime.datetime.now()
   return render_to_response('current_datetime.html', {'current_date': now})
```

В функции *current_datetime* мы всё ещё вычисляем *now*, но загрузка шаблона, создание контекста, обработка шаблона и создание объекта HttpResponse — обо всём этом заботится функция *render_to_response()*. Так как эта функция возвращает объект HttpResponse, мы просто возвращаем это значение в представление.

Первым аргументом функции *render_to_response()* должно быть имя используемого шаблона. Вторым аргументом, если он есть, должен быть словарь, используемый при создании контекста для этого шаблона. Если вы не предоставите второй аргумент, функция будет использовать пустой словарь.

Примечание:

Если вы один из таких ленивых разработчиков и вам нравится краткий код, вы можете воспользоваться встроенной в Python функцией *locals()*. Она возвращает словарь, хранящий все локальные переменные и их соответствующие значения (речь идёт о всех переменных, определённых в текущей области видимости). Следовательно, предыдущую функцию представления можно переписать так:

```
def current_datetime(request):
    current_date = datetime.datetime.now()
    return render_to_response('current_datetime.html', locals())
```

Здесь, вместо ручного ввода контекстного словаря, как это было раньше, вы передали значение функции *locals()*, которое содержало все переменные определённые в этой точке выполнения функции. Как результат, мы переименовали переменную *now* в *current_date*, потому что это то имя, которое шаблон ожидает.

12.Наследование шаблонов

Часто бывает ситуация, когда структура сайта статично, а содержание динамично, то есть теги остаются, но их содержимое меняется.

Для решения подобной проблемы в Django имеется механизм наследования шаблонов.

Первым шагом является определение базового шаблона — основы вашей страницы, которую позже будут заполнять дочерние шаблоны. Ниже представлен базовый шаблон для нашего текущего примера:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html lang="en">
<head>
    <title>{% block title %}{% endblock %}</title>
</head>
<body>
    <h1>My helpful timestamp site</h1>
    {% block content %}{% endblock %}
    {% block footer %}
    <hr>
    <p>Thanks for visiting my site.</p>
    {% endblock %}
</body>
</html>
```

Этот шаблон, который мы назвали *base.html*, определяет основу HTML документа, которую мы будем использовать для каждой страницы на сайте. Это уже задача дочерних шаблонов заполнить или добавить, или не трогать содержимое этих блоков.

Здесь мы используем тег, который раньше не определяли: *{% block %}*. Всё, что такие теги делают — указывают шаблонной системе, что дочерние шаблоны могут переопределять эту часть основного шаблона.

Имея основной шаблон, мы можем внести изменения в существующий *current_datetime.html*, чтобы использовать его:

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

{% block title %}The current time{% endblock %}

{% block content %}
<p>It is now {{ current_date }}.</p>
{% endblock %}
```

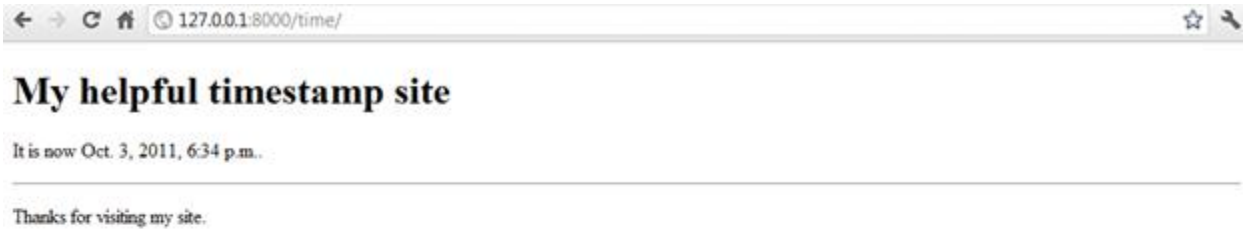

Рис. 17 Страница-наследник

Задание

1. Установить Django (например, пакет DjangoStack)
2. Создать приложение Hello, World
3. Создать страницу /your_last_name/ , при загрузке которой будут отображаться ваши ФИ.
4. Динамический контент и URL: создать страницу /your_last_name/number/ , где number – любое трехзначное число. При загрузке страницы должна отображаться ваша фамилия, вариант и значение, равное вариант+number.
5. Наследование шаблонов: Для страниц /your_last_name/ и /your_last_name/number/ создать базовый шаблон и применить наследование шаблонов.

В отчет: основные скриншоты и код. Присылайте вместе с проектом.