The Django Web Framework

Hamid Zarrabi-Zadeh
Web Programming – Fall 2013
Outline

- Introduction
- Getting Started
  - Creating a Project
  - Running the Development Server
- Creating a Blog Application
- The Admin Site
Introduction

- Django is a high-level Python Web framework that encourages rapid development and clean design.
- Born in 2003 at the Lawrence Journal-World newspaper by Adrian Holovaty and Simon Willison.
- Released publicly under BSD license in July 2005.
Django Features

- Django is a "web framework for perfectionist with deadlines"
- Its primary goal is to ease the creation of complex, database-driven websites
- Django emphasizes reusability and "pluggability" of components, rapid development, and the principle of Don't Repeat Yourself (DRY)
- Django makes it easier to build better web apps more quickly with less code
Django Architecture

• Django follows the Model–View–Controller architectural pattern

• The core Django MVC framework consists of
  – an object-relational mapper which mediates between data models (defined as Python classes) and a relational database (Model)
  – a system for processing requests with a web templating system (View)
  – a regular-expression-based URL dispatcher (Controller)
Components

- A lightweight, standalone web server for development and testing
- A template system that utilizes the concept of inheritance borrowed from object-oriented programming
- A form serialization and validation system which can translate between HTML forms and values suitable for storage in the database
- A caching framework which can use any of several cache methods
- Support for middleware classes which can intervene at various stages of request processing and carry out custom functions
Components (cont'd)

- An internationalization system, including translations of Django's own components into a variety of languages
- An internal dispatcher system which allows components of an application to communicate events to each other via pre-defined signals
- A serialization system which can produce and read XML/JSON representations of Django model instances
- A system for extending the capabilities of the template engine
- An interface to Python's built-in unit test framework
Django Versions

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9</td>
<td>Nov 2005</td>
</tr>
<tr>
<td>1.0</td>
<td>Sept 2008</td>
</tr>
<tr>
<td>1.4</td>
<td>Mar 2012</td>
</tr>
<tr>
<td>1.5</td>
<td>Feb 2013 (Python 3 Support)</td>
</tr>
<tr>
<td>1.6</td>
<td>Nov 2013</td>
</tr>
</tbody>
</table>
Getting Started
Installing Django

• Download Django from djangoproject.com
• Install it:
  – `python setup.py install`

  Hint: on Windows, add the following to PATH:
  `C:\Python32;C:\Python32\Scripts;`

• Verify it

```python
>>> import django
>>> django.VERSION
(1, 6, 0, 'final', 0)
```
Example: Building a Blog

- Django describes itself as “the Web framework for perfectionists with deadlines”
- So, let’s put ourselves on deadline and see how fast we can produce a simple blog using Django
- We will address the perfectionist side later
Creating a Project

- The easiest way to organize your Django code is to use what Django calls a project.

- A project is a directory, containing settings for an instance of Django.

- Django comes with a utility called `django-admin.py` to streamline tasks such as the creation of the project directories.

```$ cd directory
$ python django-admin.py startproject mysite```
Project Structure

• The project directory looks like this:

```
mysite/
    manage.py
mysite/
    __init__.py
    settings.py
    urls.py
    wsgi.py
```

• We can rename this directory and put it anywhere (outside web server's document root)

• `mysite` is a package, so we can use dot-notation to access pieces of our project, e.g., `mysite.urls`
Project Files

• Besides `__init__.py`, the project contains the followings files:
  – `manage.py`: a utility for working with this Django project
  – `settings.py`: a file containing default settings for your project, including database information, debugging flags, and other important variables
  – `urls.py`: a configuration file that maps URL patterns to actions your applications perform
  – `wsgi.py`: an entry-point for WSGI-compatible web servers to serve your project
The Development Server

- Django comes with a built-in lightweight Web server written purely in Python
- It helps you develop things rapidly, without having to deal with configuring a production server – such as Apache – until you’re ready for production
- It automatically detects when you make changes to your Python source files and reloads those modules
Running the Server

• Running the development server is as simple as issuing a single command:
  – python manage.py runserver

• Check the result on your browser
Creating a Blog Application
Creating an Application

• Now that we have a project, we can create applications (or “apps”) within it
  – `python manage.py startapp blog`

• It creates the following directory

```python
blog/
    __init__.py
    admin.py
    models.py
    tests.py
    views.py
```

• To activate the app, we need to add its name 'blog' to `INSTALLED_APPS` in `settings.py`
Designing Models

• We define the data structures of our blog in `model.py`

```python
from django.db import models

class BlogPost(models.Model):
    title = models.CharField(max_length=150)
    body = models.TextField()
    timestamp = models.DateTimeField()
```

• That’s a complete model, representing a `BlogPost` object with three fields

• Django automatically adds an auto-increment unique `id` for each model by default
Setting Up the Database

- Django by default uses SQLite as database backend.
- It is fast, widely available, and stores its database as a single file in the filesystem.
- Access controls are simply file permissions.
- It is a popular choice for testing, and even for deployment in scenarios with no much simultaneous writing.
- You can change database setting in settings.py.
Creating the Tables

• Now, run `syncdb` to create the model tables in your database
  – `python manage.py syncdb`

• `syncdb` can be called as often as you like, and it will only ever create the tables that don’t exist
The Admin Site
The Admin App

- Django has a built-in admin app
- The app is installed by default in Django 1.6
- It provides an administrative interface to create, read, update, and delete content
- You can try out the admin site instantly at
  - [http://127.0.0.1:8000/admin/](http://127.0.0.1:8000/admin/)
Adding Blog to Admin

• We can tell the admin that Blog objects have an admin interface, by modifying blog/admin.py as follows:

```python
from django.contrib import admin
from blog.models import BlogPost

admin.site.register(BlogPost)
```
Object Representations

- Once you add new posts to the blog, you see each item listed with an awkward name BlogPost object.
- You can make the objects' representations nicer by adding a `__str__` function to BlogPost class.

```python
class BlogPost(models.Model):
    title = models.CharField(max_length=150)
    body = models.TextField()
    timestamp = models.DateTimeField()

    def __str__(self):
        return self.title
```
Customizing Admin

• The admin site is very flexible
• It has several options for dealing with customizing the interface
• All options are defined in ModelAdmin subclass

```python
from django.contrib import admin
from blog.models import BlogPost

class BlogPostAdmin(admin.ModelAdmin):
    list_display = ('title', 'timestamp')
    fields = (('title', 'timestamp'), 'body')
    search_fields = ['title', 'body']

admin.site.register(BlogPost, BlogPostAdmin)
```
References

• Django Official Website
  – http://djangoproject.com/

• Django 1.6 Documentation
  – http://docs.djangoproject.com/en/1.6/

• Python Web Development with Django
  – By Jeff Forcier, Paul Bissex, Wesley Chun