The Django Web Framework – Part II

Hamid Zarrabi-Zadeh
Web Programming – Fall 2013
Outline

• Overview
• Making the Front Page
  – Creating the Template
  – Creating View Functions
  – Configuring URL Patterns
• Finishing Touches
• Wrap It Up
Overview

- We will continue the Blog application
- What we did in last session:
  - installed Django
  - created the project mysite
  - created the Blog app, and added it to the project
  - designed our model, BlogPost, and synced it to the db
  - set up the admin site, and registered our model onto it
  - started entering data
Overview (cont'd)

• Today, we will focus on building the blog's public interface, namely its front page

• A page in Django has three typical components
  – A template that displays information passed to it (in a Python-dictionary-like object called a Context)
  – A view function that fetches information to be displayed, typically from a database
  – A URL pattern that matches an incoming request with your view function, optionally passing parameters to the view as well
Making the Front Page
Creating a Template

• This is a simple template for displaying a single blog post:

```
<h2>{{ post.title }}</h2>
<p>{{ post.timestamp }}</p>
<p>{{ post.body }}</p>
```

• It’s just HTML, plus special template tags in curly braces

• Django templates can be indeed used for any kind of textual output

• We store templates inside blog/templates/[blog]
Template Variable Tags

- In the previous example, we had three variable tags.
- Inside a variable tag, you can use Python-style dot-notation to access attributes of the objects passed to the template.
- For example, previous template assumes you have passed it a BlogPost object called post.
- The three lines of the template fetch the BlogPost object’s title, timestamp, and body fields, respectively.
Template Code Tags

• We can enhance the template to display multiple blog posts, using `for` block tag

• Unlike variable tags, **block tags** are enclosed in `{% ... %}` pairs

{% for post in posts %}
<h2>{{ post.title }}</h2>
<p>{{ post.timestamp|date:'l, F jS' }}</p>
<p>{{ post.body }}</p>
{% endfor %}
Now we write a `view function` that fetches all our blog posts from the database, and displays them using our template.

```python
from django.template import loader, Context
from django.http import HttpResponse
from blog.models import BlogPost

def index(request):
    posts = BlogPost.objects.all()
    template = loader.get_template('index.html')
    context = Context({'posts': posts})
    return HttpResponse(template.render(context))
```
Rendering Shortcut

- It is a very common idiom to load a template, fill a context and return an `HttpResponse` object with the result of the rendered template.
- Django provides a shortcut, called `render`

```python
from django.shortcuts import render
from blog.models import BlogPost

def index(request):
    posts = BlogPost.objects.all()
    context = { 'posts': posts }
    return render(request, 'index.html', context)
```
Creating a URL Pattern

- The last piece needed for our page to work is a URL
- We can create the needed URL pattern inside the blog package at blog/urls.py

```python
from django.conf.urls import patterns, url
from blog import views

urlpatterns = patterns('',
    url(r'^$', views.index, name='index')
)
```
The last step is to modify the root URLconf at `mysite/urls.py` to include `blog/urls.py`

```python
from django.conf.urls import patterns, url

urlpatterns = patterns('',
    url(r'^blog/', include('blog.urls')),
    url(r'^admin/', include(admin.site.urls)),
)
```
Try It Out

- The page is now ready to see in action at http://localhost:8000/blog/

Hello World!
Nov. 24, 2013, 2:30 p.m.
This is the first post.

My Second Post
Nov. 24, 2013, 2:35 p.m.
This is another post.
Finishing Touches
Base Templates

• Our previous template was simplistic and self-contained
• Most often, we have several pages in our app, and we want all of them to be based on a common template
• The right way in Django is to create a base template, and then extend this template to generate the other, specific templates
Here is a sample base template

```html
<html>
<style type="text/css">
  body { color: #ccc; background: #000;
          padding: 0 5em; margin: 0 }
  h1 { padding: 1em; background: #675 }
  h2 { color: #bf8; border-top: 1px
dotted #fff; margin-top: 2em }
  p { margin: 1em 0 }
</style>
<body>
  <h1>My Blog</h1>
  {% block content %}
  {% endblock %}
</body>
</html>
```
Extending Base Template

- We can now define our template by extending the base

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

{% block content %}
{% for post in posts %}
<h2>{{ post.title }}</h2>
<p>{{ post.timestamp }}</p>
<p>{{ post.body }}</p>
{% endfor %}
{% endblock %}
```
The New Look

My Blog

My Second Post
Sunday, November 24th
Post

Hello World!
Sunday, November 24th
This is the first post.
Date Ordering

- Our blog posts are not being presented in traditional reverse-chronological order
- It is easy to tell Django to do that
  - We can either add it to the query:
    ```python
    BlogPost.objects.all().order_by('-timestamp')
    ```
  - Or add a default ordering to our model:
    ```python
class Meta:
    ordering = ('-timestamp',)
    ```
Template Filters

- its ISO8601 format of the dates is a little nerdy; Let’s humanize it a bit using template filters
- To apply a filter to a variable, simply add it to the end of the variable name using a pipe character

<p>{{ post.timestamp|date }}</p>

- You can even customize the filter by passing argument to it

<p>{{ post.timestamp|date:'l, F jS' }}</p>
Wrap It Up
The Big Picture

- Django encompasses a large number of concepts, features, and tools
- Before starting to learn the details, you need to grasp an overall view of the whole framework and its architecture
Overall Architecture
Core Philosophies of Django

- Separating the Layers (MVC)
- Try to Be Pythonic
- Don’t Repeat Yourself (DRY)
- Loose Coupling and Flexibility
- Rapid Development
  - generic views, shortcuts, middleware, etc.
References

- Django Documentation

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