Blogapp: A review of the MTV model

Chapter 5

Objectives

- In this chapter we'll build a Blog application that allows users to create, edit, and delete posts.
- The homepage will list all blog posts and there will be a dedicated detail page for each individual post.

Initial Setup

- Our initial setup involves the following steps:
  - create a new directory for our code called blog
  - install Django in a new virtual environment
  - create a new Django project called blog_project
  - create a new pages app called blog
  - perform a migration to set up the database
  - update settings.py
- On the command line make sure you're not working in an existing virtual environment.
- You can tell if there's anything in parentheses before your command line prompt. If you are simply type exit to leave it.

Database Models

- Assume each post has a title, author, and body. We can turn this into a database model by opening the blog/models.py file and entering the code below:

```
# blog/models.py

from django.db import models

class Post(models.Model):
    title = models.CharField(max_length=200)
    author = models.ForeignKey(
        'auth.User',
        on_delete=models.CASCADE,
    )
    body = models.TextField()
    def __str__(self):
        return self.title
```

For title we're limiting the length to 200 characters

For the author field we're using a ForeignKey which allows for a many-to-one relationship. The reference is to the built-in User model thatDjango provides for authentication.

For body we're using a TextField which automatically expands as needed

Migration record and Migrate

- Now that our new database model is created we need to create a new migration record for it and migrate the change into our database. This two-step process can be completed with the commands below:

```
python manage.py makemigrations blog
python manage.py migrate blog
```

- Our database is configured! Next, use Django Admin to access the data.

Admin

- First create a superuser account by typing the command below and following the prompts to set up an email and password.

```
python manage.py createsuperuser
```

- Now start running the Django admin page. Login with your new superuser account.
- Our new post model is missing! What step have we missed?
- We forgot to update blog/admin.py

```
# blog/admin.py
from django.contrib import admin
from .models import Post
admin.site.register(Post)
```

Add blog posts via Django Admin

- Let’s add two blog posts. Make sure to add an “author” to each post too since by default all model fields are required.
- If you want to change this requirement, you could add field options to our model to make a given field optional or fill it with a default value.
- Now that our database model is complete we need to create the necessary views, URLs, and templates so we can display the information on our web application.

URLs

- We’ll first configure our project-level URLConfs and then our app-level URLConfs in order to display our blog posts on the homepage.
  1. Update our project-level urls.py file so that it knows to forward all requests directly to the blog app.
  2. Create a new app-level urls.py file (i.e. blog/urls.py) as follows:

```
# blog_project/urls.py
from django.contrib import admin
from django.urls import path, include
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('blog.urls')),
]
```

```
# blog/urls.py
from django.urls import path
from . import views
urlpatterns = [
    path('', views.BlogListView.as_view(), name='home'),
]
```

Accordingly, we will create a class-based view named “BlogListView”

Views

- In our views file, create a class-based view named “BlogListView” (as indicated in the URLConfs and add the code below to display the contents of our Post model using ListView.

```
# blog/views.py
from django.views.generic import ListView
from . models import Post

class BlogListView(ListView):
    model = Post
    template_name = 'home.html'
```

- The top two lines we import ListView and our database model Post.
- Then we subclass ListView and add links to our model and template

- The next step is to create the template “home.html” as indicated in the last line of the view.

Templates

- As we already saw in Chapter 4, we can inherit from other templates to keep our code clean.
- Thus we’ll start off with a base.html file and a home.html file that inherits from it.
- Then later when we add templates for creating and editing blog posts, they too can inherit from base.html.
- Start by creating our project-level templates directory with the two template files.
- Then update settings.py so Django knows to look there for our templates: 'DIRS': [os.path.join(BASE_DIR, 'templates')],

base.html

```
<!-- templates/base.html -->
<html>
  <head>
    <title>Django blog</title>
  </head>
  <body>
    <header>
      <h1><a href="/">Django blog</a></h1>
   </header>
   <div class="container">
     {% block content %}
     {% endblock content %}
  </body>
 </html>
```

Note that code between {% block content %} and {% endblock content %} can be filled by other templates.

home.html

- This template extends base.html and then wraps our desired code with content blocks.
- We use the Django Templating Language to set up a simple for loop for each blog post.
- Note that object_list comes from ListView and contains all the objects in our view.
- If you refresh the page on the browser, you can see it's working

```
<!-- templates/home.html -->
{% extends 'base.html' %}
{% block content %}
  {% for post in object_list %}
    <div class="post-entry">
      <h2><a href="">{{ post.title }}</a></h2>
      <p>{{ post.body }}</p>
    </div>
  {% endfor %}
{% endblock content %}
```

Individual blog pages - view

- To add the functionality for individual blog pages, we need to create a new view, url, and template.
- Do you notice a pattern in development with Django now?
- Start with the view. We can use the generic class-based DetailView to simplify things.
- At the top of the file add DetailView to the list of imports and then create our new view called BlogDetailView.
- Next, create the template 'post_detail.html'.

```
class BlogDetailView(DetailView):
    model = Post
    template_name = 'post_detail.html'
```

Individual blog pages - template

- Content of post_detail.html:
- It displays the title and body from our context object, which DetailView makes accessible as post, which is the lowercased name of our model.
- The last missing step is to add a new URLConf for our view.

```
<!-- templates/post_detail.html -->
{% extends 'base.html' %}
{% block content %}
  <div class="post-entry">
    <h2>{{ post.title }}</h2>
    <p>{{ post.body }}</p>
  </div>
{% endblock content %}
```

Individual blog pages - URLConfs

- Add the following line to our app-level urls.py

```
path('post/<int:pk>/', views.BlogDetailView.as_view(), name='post_detail'),
```

- All blog post entries will start with post/. Next is the primary key for our post entry which will be represented as an integer <int:pk>.
- Django automatically adds an auto-incrementing primary key to our database models.
- So while we only declared the fields title, author, and body on our Post model, Django also added another field called id, which is our primary key. We can access it as either id or pk.
- On the browser, you'll see a dedicated page for our first blog post.

Update home.html

- Currently in home.html our link is empty: <a href="">.
- Update it to <a href="{% url 'post_detail' post.pk %}">.
- We start off by telling our Django template we want to reference a URLConf by using the code {% url ... %}.
- WhichURL? The one named post_detail, which is the name we gave BlogDetailView in our URLConf.
- post_detail in our URLConf expects to be passed an argument pk representing the primary key for the blog post. We pass it into the URLConf by adding it in the template as post.pk.
- To confirm everything works, refresh the main page at your browser and click on the title of each blog post to confirm the new links work.

Conclusion

- We used DetailView for the first time to create a detailed individual view of each blog post entry.
- In the next section Chapter 6: Blog app with forms, we'll add forms so we don't have to use the Django admin at all for these changes.