Module – 3

Learning Objectives

- **Task - Rendering the content on html templates.**
- Class – based views and Function – based views.
- Creating Templates.
- urls.py – views.py – html-template.
- **Task – Identifying the common content, place it in a common template. Adding a navigation bar to project.**
- Extending templates.
- **Task – Adding project logo image.**
- Adding images.
- Applying css and bootstrap.
- Add a dynamically generating title to your web pages
- Testing using SimpleTestCase.

Class-Based Views and Function-Based Views

Early versions of Django only shipped with function-based views, but developers soon found themselves repeating the same patterns over and over again. Write a view that lists all objects in a model. Write a view that displays only one detailed item from a model. And so on. Function-based generic views were introduced to abstract these patterns and streamline development of common patterns. However, there was no easy way to extend or customize these views. As a result, Django introduced class-based generic views that make it easy to use and also extend views covering common use cases.

blog/views.py

```
from django.shortcuts import render
from django.http import HttpResponse
from django.views.generic import TemplateView

# Returning a response directly
def homePageView(request):
    return HttpResponse('Hello, World!')

# Rendering a template
# Function based views
def postListView(request):
    return render(request, 'listposts.html')

# Class Based Views
class PostListView(TemplateView):
    template_name = 'listposts.html'
```

blog/urls.py

```
from django.urls import path
from blog import views

urlpatterns = [
    # Class Based
    path('', views.PostListView.as_view() , name = 'listposts'), # url-pattern - /blog

    # Function Based
    path('', views.postListView, name = 'listposts'), # url-pattern - /blog
]
```

MyBlog/urls.py

```
from django.contrib import admin
from django.urls import path, include
from django.views.generic import TemplateView

urlpatterns = [
    path('admin/', admin.site.urls),

# Returning template directly
    path('', TemplateView.as_view(template_name = "home.html"), name= 'homepage'),
    path('about/', TemplateView.as_view(template_name = "about.html"), name= 'aboutpage'),
    path('contact/', TemplateView.as_view(template_name = "contact.html"), name= 'contactpage'),
```

```
# including app specific urls.py
    path('blog/', include('blog.urls')),
    path('accounts/', include('accounts.urls')),
]
```

Templates

Every web framework needs a convenient way to generate HTML files. In Django, the approach is to use templates so that individual HTML files can be served by a view to a web page specified by the URL. It's worth repeating this pattern since you'll see it over and over again in Django development: Templates, Views, and URLs. The URLs control the initial route, the entry point into a page, the views contain the logic or the "what", and the template has the HTML. For web pages that rely on a database model, it is the view that does much of the work to decide what data is available to the template. So: Templates, Views, URLs. This pattern will hold true for **every Django web page you make**.

By default, Django looks for the templates directory to retrieve the html template. The name of the directory containing all the template files should be specified in settings.py under "TEMPLATES" list.

MyBlog/ settings.py

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': ['templates'],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

This specifies that all the template files will be placed in templates directory. This directory can be placed in main project folder or inside app folder.

templates/home.html

```
<html>
<body>
<h1>Welcome to My Blog!!!</h1>
<!-- <a href="/about"> About us</a> | <a href="/contact"> Contact us</a> -->
<a href="{% url 'homepage' %}"> Home</a> | <a href="{% url 'aboutpage' %}"> About us</a> | <a
        href="{% url 'contactpage' %}"> Contact us</a>
</body>
</html>
```

Note: It is completely up to the developer where to place the templates directory. Also, developer has freedom to choose between class-based views or function-based views.

Extending Templates

The real power of templates is their ability to be extended. If you think about most web sites, there is content that is repeated on every page (header, footer, etc). Let's create a base.html file containing a header with links to our two pages.

templates/base.html

```
<html>
<body>
<h1>Welcome to My Blog!!!</h1>
<!-- <a href="/about"> About us</a> | <a href="/contact"> Contact us</a> -->
<a href="{% url 'homepage' %}"> Home</a> | <a href="{% url 'aboutpage' %}"> About us</a> | <a
        href="{% url 'contactpage' %}"> Contact us</a>
</body>
</html>
```

templates/about.html

```
{% extends 'base.html' %}
{% block base%}
<h2>About us content</h2>
{% endblock%}
```

Static Files

Just as we did with our templates folder we need to update settings.py to tell Django where to look for static files. We can update settings.py with a one-line change for STATICFILES_DIRS. Add it at the bottom of the file below the entry for STATIC_URL.

MyBlog/settings.py

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/2.0/howto/static-files/

STATIC_URL = '/static/'

STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')]
```

Now create a folders /static/css and add a new base.css file in it.

static/css/base.css

```
body {
  font-family: 'Source Sans Pro', sans-serif;
  font-size: 18px;
}

header {
  border-bottom: 1px solid #999;
```

```
margin-bottom: 2rem;
  display: flex;
}
.
.
.
```

Adding Images

Create a folders /static/media and store all images there.

MyBlog/scripts.py – All bootstrap related includes

```
{% load static %}

<html>
<head>
    <title>My Blog - {% block title %}
        {% endblock %}</title>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js"></script>
    <link href='//fonts.googleapis.com/css?family=Lobster&subset=latin,latin-ext' rel='stylesheet'
type='text/css'>
    <link rel="stylesheet" href="{% static 'css/base.css' %}">
</head>
<body>
{% block scripts_content %}
{% endblock %}

</body>
</html>
```

MyBlog/base.py – Navigation bar

```
{% extends 'scripts.html' %}
{% load static %}
{% block scripts_content %}

<div class="content container">
    <div class="page-header">
        <img src="{% static 'media/headerimg.png' %}" width="1110" height="150"/>
        <div class="progress" style="height: 2px">
            <div class="progress-bar bg-warning" style="width:100%; height: 5px"></div>
        </div>
        <nav class="navbar navbar-expand bg-dark navbar-dark">
            <div class="container justify-content-md-start">
                <ul class="navbar-nav">
                    <a href="{% url 'homepage' %}"> Home</a>
                       |   
                    <a href="{% url 'aboutpage' %}"> About us</a>
                       |   
```

```
                    <a href="{% url 'contactpage' %}"> Contact us</a>
                </ul>
            </div>
            <div class="container justify-content-md-end">
                <ul class="navbar-nav">
                    <a href="{% url 'loginpage' %}"> Login</a>
                </ul>
            </div>
        </nav>
    </div>
    <div class="progress" style="height: 2px">
        <div class="progress-bar bg-warning" style="width:100%"></div>
    </div>
    <br>
    {% block base%}
    {% endblock%}
    <div class="fixed-bottom" align="center">
        <div class="progress" style="height: 2px">
            <div class="progress-bar bg-warning" style="width:100%; height: 5px"></div>
        </div>
        <div class="bg-dark">
            Copyright Django@PetaaBytes
        </div>
    </div>
</div>
{% endblock %}
```

Page without navigation bar but extends scripts.py (Add a dynamically generating title to your web pages)

```
{% extends 'scripts.html' %}
{%block title%}Demo Page {%endblock%}
{% block scripts_content %}

<div class="content container">
    <div class="page-header">
    My Demo page
    </div>
</div>
{% endblock %}
```

Tests

Writing tests is important because it automates the process of confirming that the code works as expected. In a smaller app, we can manually look and see that the pages exist and contain the intended content. But as Django project grows in size there can be hundreds if not thousands of individual web pages and the idea of manually going throw each page is not possible. Further, whenever we make changes to the code–adding new features, updating existing ones, deleting unused areas of the site–we want to be sure that we have not inadvertently broken some other piece of the site. Automated tests let us write one time how we expect a specific piece of our project to behave and then let the computer do the checking for us. Fortunately, Django comes with robust, built-in testing tools for writing and running tests.

blog/tests.py

```
from django.test import SimpleTestCase

class SimpleTests(SimpleTestCase):

    def test_home_page_status_code(self):
        response = self.client.get('/')
        self.assertEqual(response.status_code, 200)

    def test_about_page_status_code(self):
        response = self.client.get('/about/')
        self.assertEqual(response.status_code, 200)
```

We're using SimpleTestCase here since we aren't using a database. If we were using a database, we'd instead use TestCase. Then we perform a check if the status code for each page is 200, which is the standard response for a successful HTTP request.

Running Test cases

```
D:\Django\MyBlog>python manage.py test
Creating test database for alias 'default'...
System check identified no issues (0 silenced).

----------------------------------------------------------------------
Ran 0 tests in 0.000s

OK
Destroying test database for alias 'default'...
```