Exercise 2.4: Django Views and Templates

Learning Goals

- Summarize the process of creating views, templates, and URLs
- Explain how the “V” and “T” parts of MVT architecture work
- Create a frontend page for your web application

Reflection Questions

1. Do some research on Django views. In your own words, use an example to explain how Django views work.

Django Views - Overview

In Django, **views** are responsible for handling requests and returning responses. They act as the link between your **models** (data) and **templates** (HTML), deciding which data to present and how.

There are two main types of views:

1. **Function-Based Views (FBVs)**:
   Simple Python functions that process requests and return responses.
2. **Class-Based Views (CBVs)**:
   Views written as classes, offering more reusable and modular code.

How Django Views Work - Example with `render`

In **Function-Based Views**, `render()` is often used to combine a template with data and return an HTML response. It simplifies the process of loading a template, passing context (data), and returning a rendered page.

Example:

1. **View Logic**:
   The view retrieves data (e.g., from a model or form), then passes it to a template.

```
# views.py
from django.shortcuts import render

def hello_view(request):
    context = {'message': 'Hello, world!'}
```

```
    return render(request, 'hello.html', context)
```

- **`render()`**: Combines the request, template (`hello.html`), and context (`{'message': 'Hello, world!'}`).

2. **Template (HTML)**:
   The template (`hello.html`) uses Django's Template Language (DTL) to display the passed data:

```
<!-- hello.html -->
<!DOCTYPE html>
<html>
<head><title>Hello Page</title></head>
<body>
    <h1>{{ message }}</h1>
</body>
</html>
```

3. **URL Configuration**:
   This view is mapped to a URL in `urls.py`:

```
# urls.py
from django.urls import path
from .views import hello_view

urlpatterns = [
    path('hello/', hello_view),
]
```

4. **Request and Response**:
   When a user visits `/hello/`, the `hello_view` function runs, rendering the `hello.html` template with the message, and returning an HTML response that says "**Hello, world!**".

2. Imagine you're working on a Django web development project, and you anticipate that you'll have to reuse lots of code in various parts of the project. In this scenario, will you use Django function-based views or class-based views, and why?

    a. I'd use class-based views as they allow for easy reuse once classes are defined, also to extend if needed based on inheritance. For repeating generic components, this is the ideal option.

3. Read Django's documentation on the Django template language and make some notes on its basics.

1. **Purpose**:
   DTL separates presentation from logic, rendering dynamic content in HTML with minimal backend logic.
2. **Syntax**:
   - **Variables**: Use `{{ variable }}` to display values, e.g., `{{ user.username }}`.
   - **Filters**: Modify output using `|`, e.g., `{{ name|lower }}`. Common filters: `date`, `upper`, `length`.
   - **Tags**: Control flow with `{% %}`.
     - **if**: `{% if user.is_authenticated %}...{% endif %}`
     - **for**: `{% for item in items %}...{% endfor %}`
3. **Template Inheritance**:
   - **extends**: `{% extends "base.html" %}` for inheritance.
   - **block**: `{% block content %}...{% endblock %}` for overriding sections.
4. **Escaping**:
   Variables are auto-escaped for security. Use `|safe` for raw HTML.
5. **Custom Tags & Filters**:
   Developers can create custom tags and filters using Python code.