Django Basics

Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. At its core, Django follows the Model-View-Controller (MVC) pattern, which is central to organizing your code and separating different concerns within your web application. However, in Django, this pattern is often referred to as Model-View-Template (MVT).

1. Understanding the MVC (Model-View-Controller) Pattern in Django

- **Model**: Represents the data structure. The model is responsible for managing the data of the application. In Django, this is defined using Python classes, and it typically corresponds to a table in your database.
- **View**: Handles the logic and interacts with the model to retrieve data. In Django, the "view" refers to the logic that processes requests and returns responses, typically in the form of HTML pages.
- **Controller**: The controller in the traditional MVC pattern manages the user input, interacting with the model and the view. In Django, this role is somewhat merged between the framework's URL dispatcher and view functions.
- **Template**: The template in Django is where the presentation logic resides, separating the HTML/CSS from the Python code. This allows you to define how data is presented to the user.

So, in Django:

- **Model** corresponds to the database.
- **View** handles the logic and response to user requests.
- **Template** is responsible for rendering the HTML.

2. Setting Up a Django Project

Setting up a Django project involves a few steps, and Django provides commands to simplify this process.

Step 1: Install Django

First, you need to install Django. You can do this using `pip`:

bash

```
pip install django
```

Step 2: Create a Django Project

Create a new Django project using the `django-admin` command:

bash

```
django-admin startproject myproject
```

This creates a directory structure like this:

markdown

```
myproject/
    manage.py
    myproject/
        __init__.py
        settings.py
        urls.py
        asgi.py
        wsgi.py
```

- **manage.py**: A command-line utility that lets you interact with this Django project.
- **settings.py**: Configuration for your Django project, including database settings, installed apps, and middleware.
- **urls.py**: The URL configuration module, mapping URLs to views.
- **wsgi.py and asgi.py**: Entry points for WSGI/ASGI-compatible web servers to serve your project.

Step 3: Run the Development Server

You can run Django's built-in development server to test your project:

bash

```
python manage.py runserver
```

Visit `http://127.0.0.1:8000/` in your web browser, and you should see the Django welcome page, indicating that your project is set up correctly.

3. Creating a Django App

Django projects are composed of multiple apps. An app is a self-contained module that can handle a specific functionality or feature.

Step 1: Create an App

Create an app using the `startapp` command:

bash

```
python manage.py startapp myapp
```

This creates a directory structure for the app:

markdown

```
myapp/
    migrations/
    __init__.py
    admin.py
    apps.py
    models.py
    tests.py
    views.py
```

- **models.py**: Define your database models here.
- **views.py**: Define the logic for handling user requests.
- **admin.py**: Register your models with the Django admin interface.
- **migrations/**: Store database migration files.

Step 2: Register the App

You need to register your app in the `settings.py` file:

python

```
# myproject/settings.py
INSTALLED_APPS = [
    ...
    'myapp',
    ...
]
```

4. Managing URLs and Views

Step 1: Define a View

A view function in Django is a Python function that takes a web request and returns a web response.

python

```
# myapp/views.py
from django.http import HttpResponse

def hello_world(request):
    return HttpResponse("Hello, world!")
```

Step 2: Map the View to a URL

To make the view accessible via a URL, you need to map it in `urls.py`.

python

```
# myproject/urls.py
from django.contrib import admin
from django.urls import path
from myapp import views

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

```
    path('hello/', views.hello_world),  # Maps the /hello/ URL
to the hello_world view
]
```

Step 3: Access the View

Run the server and navigate to `http://127.0.0.1:8000/hello/` in your browser. You should see "Hello, world!" displayed on the page.

5. Templates in Django

Templates allow you to separate the HTML from your business logic.

Step 1: Create a Template

Create a directory named `templates` inside your app folder:

markdown

```
myapp/
    templates/
        hello.html
    ...
```

Add the following HTML to `hello.html`:

html

```
<!DOCTYPE html>
<html>
<head>
    <title>Hello World</title>
</head>
<body>
    <h1>{{ message }}</h1>
</body>
</html>
```

Step 2: Render the Template in a View

Update your view to render the template:

python

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

def hello_world(request):
    return render(request, 'hello.html', {'message': 'Hello,
world!'})
```

Step 3: Access the Template View

Now, when you navigate to `http://127.0.0.1:8000/hello/`, Django will render the `hello.html` template with the message "Hello, world!".

Summary

- **MVC/MVT Pattern**: Django follows a Model-View-Template pattern, with models representing the database, views handling the logic, and templates rendering the HTML.
- **Setting Up**: You can set up a Django project using `django-admin`, create apps using `startapp`, and manage URLs and views to handle web requests.
- **Templates**: Django uses templates to separate presentation logic from Python code, allowing for dynamic HTML generation.