DJANGO

Day 76

🧠 What is Django?

Django is a **high-level Python web framework** that allows you to build web applications **quickly**, with **less code**, and by following the **Model-View-Template (MVT)** architectural pattern.

🔑 Key Features:

- **Fast Development** (batteries-included)
- **Secure** (prevents common attacks like SQL injection, CSRF)
- **Scalable** and **reusable**
- **ORM-based** (Object Relational Mapping)
- **Built-in admin interface**
- Used by **Instagram**, **Pinterest**, **Mozilla**, etc.

🆚 Django vs Flask

Feature	Django	Flask
Framework Type	Full-stack framework (batteries included)	Micro-framework (lightweight)
Structure	Built-in structure (MVT)	Developer-defined structure
Admin Panel	Yes (Auto-generated)	No (requires third-party)
ORM	Built-in ORM	Optional (SQLAlchemy, etc.)
Use Case	Large apps, Admin dashboards, APIs	Microservices, lightweight apps
Learning Curve	Slightly steep	Beginner-friendly

⚙️ Install Python and pip

☑ Step 1: Install Python

- Visit https://www.python.org/downloads/
- Download **Python 3.10+**
- During installation: ✅ Check **"Add Python to PATH"**

☑ Step 2: Verify Python Installation

```
python --version
```

or

```
python3 --version
```

☑ Step 3: Check pip (Python's package manager)

```
pip --version
```

If not installed:

- Run: `python -m ensurepip --upgrade`

📦 Install Django

☑ Step 1: Create a virtual environment (optional but recommended)

```
python -m venv myenv
```

☑ Step 2: Activate the virtual environment

- **Windows:**

```
myenv\Scripts\activate
```

- **macOS/Linux:**

```
source myenv/bin/activate
```

☑ Step 3: Install Django

```
pip install django
```

☑ Step 4: Check Django version

```
django-admin --version
```

🛠 Create First Django Project

☑ Step 1: Create a Django project

```
django-admin startproject myproject
```

This creates a folder like:

```
myproject/
├── manage.py
└── myproject/
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    ├── asgi.py
```

```
    └── wsgi.py
```

☑ Explanation of files:

- `manage.py`: Command-line tool to manage the project
- `settings.py`: All configuration (apps, database, etc.)
- `urls.py`: URL routing file
- `wsgi.py/asgi.py`: For deployment (Web Server Gateway Interface / Asynchronous Server Gateway Interface)

▶ Run the Development Server

☑ Step 1: Move inside the project folder

```
cd myproject
```

☑ Step 2: Start the development server

```
python manage.py runserver
```

☑ Output:

```
Watching for file changes with StatReloader
Starting development server at http://127.0.0.1:8000/
```

☑ Step 3: Open in your browser

Go to: http://127.0.0.1:8000/

You will see:

"The install worked successfully! Congratulations!"

☑ Summary of Commands

```
# Install Django
pip install django

# Create a project
django-admin startproject myproject

# Move into project directory
cd myproject

# Run the server
python manage.py runserver
```

🗂 Django Project Structure Explained

When you create a Django project with:

```
django-admin startproject myproject
```

You get the following structure:

```
myproject/
├── manage.py
└── myproject/
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    ├── asgi.py
    └── wsgi.py
```

Let’s understand each file step by step:

◇ `manage.py`

- This is the command-line tool for:
  - Running the development server
  - Applying migrations
  - Creating apps
  - Managing users, DB, etc.

✅ Example:

```
python manage.py runserver
python manage.py migrate
python manage.py createsuperuser
```

◇ `settings.py`

This is the **heart** of your Django project. It contains all project settings like:

- Installed apps
- Middleware
- Databases
- Templates
- Static files
- Security keys, debug, etc.

✅ Key parts:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    ...
]

DATABASES = {
    'default': {
```

```
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / "db.sqlite3",
    }
}

STATIC_URL = 'static/'
```

◇ urls.py

- Controls URL routing
- Connects incoming URLs to views

✅ Example:

```
from django.contrib import admin
from django.urls import path

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

You can add paths from your apps later here.

◇ wsgi.py & asgi.py

Used for deployment:

- **WSGI**: For traditional synchronous servers (e.g., Gunicorn, Apache)
- **ASGI**: For asynchronous support (e.g., WebSockets)

You usually don't edit these directly.

◇ `__init__.py`

Marks this directory as a **Python package**.

🏗 Create a Django App

While `myproject/` is the main project, real functionality goes inside **apps**. An app is a component — like login, blog, cart, etc.

☑ Step 1: Create an app

Inside your project directory:

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

You'll now see:

```
myproject/
├── myapp/
│   ├── admin.py
│   ├── apps.py
│   ├── models.py
│   ├── tests.py
│   ├── views.py
│   ├── urls.py (you create this manually)
│   └── migrations/
```

☑ Step 2: Add `myapp` to `INSTALLED_APPS` in `settings.py`

```
INSTALLED_APPS = [
    ...
    'myapp',
]
```

🗂 Django App Files Explained

File	Purpose
`models.py`	Define your database models (tables) using classes
`views.py`	Handle business logic for HTTP requests
`urls.py`	(Create this manually) App-level URL routes
`admin.py`	Register models to appear in Django admin panel
`apps.py`	Configuration of the app (used for labeling, signals, etc.)
`tests.py`	Unit tests for the app
`migrations/`	Stores migration files (DB schema changes)

☑ Example: Add a Simple View

In `myapp/views.py`:

```
from django.http import HttpResponse

def hello(request):
    return HttpResponse("Hello from MyApp!")
```

Create `myapp/urls.py`:

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

urlpatterns = [
    path('', views.hello),
]
```

In `myproject/urls.py`, include the app URL:

```
from django.contrib import admin
from django.urls import path, include
```

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

Visit http://127.0.0.1:8000/ — you'll see:

"Hello from MyApp!"

☑ Summary of Commands

```
# Create a project
django-admin startproject myproject

# Move into project
cd myproject

# Create an app
python manage.py startapp myapp

# Add app to settings.py
# Create and link app-level urls
```

📌 Overview:

1. What is a **view** in Django?
2. Creating **function-based views (FBVs)**
3. **Mapping URLs** to views
4. Creating and returning **HTML templates**
5. Organizing **template folders** & using `render()`

☑ 1. What is a View?

A **view** in Django is a **Python function or class** that takes an HTTP request and returns an HTTP response.

For beginners, we start with **Function-Based Views (FBVs)**.

☑ 2. Create a Function-Based View

In `myapp/views.py`, write:

```
from django.http import HttpResponse

def home(request):
    return HttpResponse("Hello, this is the Home Page!")
```

☑ 3. Map URL to View

Step 1: Create `urls.py` in `myapp/`

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

urlpatterns = [
    path('', views.home, name='home'),
]
```

Step 2: Include app URLs in project-level `urls.py`

Edit `myproject/urls.py`:

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('myapp.urls')),  # Root URL handled by myapp
]
```

☑ 4. Returning HTML Templates

Instead of returning plain text, let's return an **HTML file**.

Step 1: Create Template Folder

Inside `myapp/`, create:

```
myapp/
└── templates/
    └── myapp/
        └── home.html
```

📄 `home.html`:

```
<!DOCTYPE html>
<html>
<head>
    <title>Home</title>
</head>
<body>
    <h1>Welcome to My Django App</h1>
</body>
</html>
```

Step 2: Update View to Return Template

In `myapp/views.py`:

```
from django.shortcuts import render

def home(request):
    return render(request, 'myapp/home.html')
```

☑ 5. Template Folder Structure Best Practice

- You can organize templates like this:

```
myapp/
└── templates/
    └── myapp/
        ├── home.html
        ├── about.html
```

Why this format?

It avoids name conflicts when multiple apps have the same template name.

☑ 6. Configure Templates in `settings.py`

By default, Django finds templates using:

```
TEMPLATES = [
    {
        ...
        'DIRS': [],  # You can add global template folders here if
needed
        'APP_DIRS': True,  # Enables app-level templates (like
templates/myapp/)
```

```
        ...
    },
]
```

You don't need to change this unless you want a **global template directory**.

☑ Summary of `render()`

```
# render() takes:
# - request object
# - template path
# - optional context (data for template)

return render(request, 'myapp/home.html', context)
```

☑ Full Example Flow:

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

def home(request):
    return render(request, 'myapp/home.html')
# myapp/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('', views.home, name='home'),
]
# myproject/urls.py
urlpatterns = [
    path('', include('myapp.urls')),
```

```
]
```

📄 `myapp/templates/myapp/home.html`

```
<h1>Hello from the Template!</h1>
```

Visit: http://127.0.0.1:8000/

☑ Recap

Concept	Description
View	Python function returning a response
URL	Maps a URL path to a view
Template	HTML file rendered using `render()`
Template Folder	Placed under `myapp/templates/myapp/`

📌 Overview

1. Template Inheritance (`base.html`)
2. `{{ variables }}` and `{% tags %}`
3. `for` loops & `if` conditions in templates
4. Adding and loading Static Files (CSS, JS, images)

🧱 1. Template Inheritance – `base.html`

Django allows you to create a **base layout** and extend it in other templates. This avoids code repetition (DRY principle).

🔧 Create base template:

File: `myapp/templates/myapp/base.html`

```
<!DOCTYPE html>
<html>
<head>
    <title>{% block title %}My App{% endblock %}</title>
    <link rel="stylesheet" href="{% static 'css/style.css' %}">
</head>
<body>
    <header>
        <h1>My Site Header</h1>
    </header>

    <main>
        {% block content %}
        <!-- Content goes here -->
        {% endblock %}
    </main>

    <footer>
        <p>© 2025 My Site</p>
    </footer>
</body>
</html>
```

🔧 Extend base in child template:

File: `myapp/templates/myapp/home.html`

```
{% extends 'myapp/base.html' %}

{% block title %}Home Page{% endblock %}

{% block content %}
    <h2>Welcome, {{ user_name }}</h2>
```

```
{% endblock %}
```

🧩 2. {{ variables }} and {% tags %}

Syntax	Purpose	Example
`{{ var }}`	Output a variable's value	`{{ user_name }}`
`{% tag %}`	Template logic (like loops)	`{% for item in items %}`

Example context in view:

```
def home(request):
    return render(request, 'myapp/home.html', {'user_name':
'Maheshwaran'})
```

🔁 3. Loops & Conditions

🔄 For Loop Example

```
<ul>
{% for product in products %}
    <li>{{ product }}</li>
{% empty %}
    <li>No products found.</li>
{% endfor %}
</ul>
```

🔁 Context in view

```
def home(request):
    return render(request, 'myapp/home.html', {'products': ['Laptop',
```

```
'Mouse', 'Keyboard']})
```

🧠 If Condition Example

```
{% if user %}
    <p>Welcome, {{ user }}!</p>
{% else %}
    <p>Guest user</p>
{% endif %}
```

🎨 4. Static Files (CSS/JS/images)

Static files are non-Python assets like:

- CSS
- JavaScript
- Images

☑️ Step 1: Create static directory inside your app

```
myapp/
├── static/
│   └── css/
│       └── style.css
```

📄 `style.css` (example content):

```
body {
    background-color: #f0f0f0;
    font-family: Arial;
```

```
}
```

☑ Step 2: Load static files in template

At the top of your HTML file:

```
{% load static %}
```

Then link your CSS/JS:

```
<link rel="stylesheet" href="{% static 'css/style.css' %}">
```

☑ Step 3: Confirm static settings in `settings.py`

Django already has this:

```
STATIC_URL = 'static/'
```

During development, Django will serve static files automatically.

☑ Summary

Feature	How It Works
`base.html`	Parent layout for consistent design
`{% extends %}`	Inherit from a base template
`{% block %}`	Sections that child templates can override
`{{ variable }}`	Render dynamic data
`{% for %}` / `{% if %}`	Loop or condition in templates

`{% static %}`	Link static files like CSS/JS/images
`static/` folder	Place static content inside `myapp/static/`