01. Introduction to Django Framework

- What is Django
- MVT architecture
- Django vs Flask
- Installation & setup

What you learn:

- Understand Django framework basics, structure, and environment setup

Introduction to Django Framework

What is Django?

Django is a high-level Python web framework that enables rapid development of secure, scalable, and maintainable web applications.

It follows the principle of "Don't Repeat Yourself (DRY)" and includes many built-in features such as authentication, ORM, and admin interfaces — helping developers focus on writing business logic.

MVT Architecture

Django is based on the Model–View–Template (MVT) architecture, which is similar to MVC but with a few key differences:

- **Model:** Manages the data and database structure using Django's ORM (Object Relational Mapper).
- **View:** Handles business logic and interacts with the model to deliver data to the user.
- Template: Manages the presentation layer (HTML, CSS) that defines how the data is displayed to the user.

Note: This separation of concerns makes Django applications modular, maintainable, and easier to extend.

Django vs Flask

Feature	Django	Flask
Type	Full-stack web framework	Lightweight micro-framework
Architecture	MVT (Model–View–Template)	No predefined structure
Built-in Features	ORM, admin panel, authentication, templates	Minimal; requires external extensions
Best For	Large-scale projects needing structure	Small to medium projects needing flexibility
Learning Curve	Moderate	Easier for beginners

What is Special About Django?

Django stands out among web frameworks because of its power, simplicity, and security. It is designed to help developers build robust web applications quickly and efficiently — without reinventing the wheel.

1. Rapid Development

Django allows developers to build complete web applications in less time thanks to its ready-to-use features like authentication, database management, and admin interfaces.

2. Built-in Admin Interface

One of Django's most unique features is its auto-generated Admin Panel. Once you define models, Django automatically creates a web-based interface to manage your data — no extra coding required.

3. Security

Django protects developers from common security threats such as:

- SQL injection
- Cross-site scripting (XSS)
- Cross-site request forgery (CSRF)
- Clickjacking

Note: It handles security best practices by default, making it one of the safest web frameworks.

4. Scalability and Performance

Django can easily handle high-traffic websites (like Instagram, Pinterest, and Disqus). Its architecture supports scalability through caching, database optimization, and load balancing.

5. "Batteries Included" Philosophy

Django provides almost everything you need:

- ORM (Object Relational Mapper)
- Authentication and authorization
- Template engine
- URL routing
- Form handling
- Session management

Note: This reduces dependency on external libraries.

6. Versatile and Cross-Platform

Django works across different platforms and can be used to build:

- Business web applications
- Content management systems (CMS)

- E-commerce platforms
- Social networks
- AI/ML integrated dashboards

7. Strong Community Support
Django has a large global community, extensive documentation, and continuous updates -- ensuring reliability and long-term support.

Django is special because it is fast, secure, and feature-rich, offering everything you need to build modern web applications efficiently.

Configure django in VSCode:

a. Commands to run to set up Django properly
b. Use virtual environment and proper path
c. Setting Up Django

Complete step by step Guide

Step 1: Check Python and pip Installation (optional)
Make sure Python and pip are installed and added to PATH.

```
>>python --version
>>pip --version
```

Step 2: Create a Project Folder
Create and navigate to your project directory (example: DjangoProjects):

```
>> mkdir DjangoProjects
>> cd DjangoProjects
```

Step 3: Create a Virtual Environment
This isolates your Django project dependencies.

```
>>python -m venv venv
```

This creates a folder named venv inside your project directory.

```
>>Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass
```

Step 4: Activate the Virtual Environment

```
On Windows:
>>venv\Scripts\activate

On macOS/Linux:
source venv/bin/activate
```

When activated, you will see (venv) at the beginning of your command line.

Step 5: Install Django

Now install Django inside your virtual environment:

```
>>pip install Django

Check Django installation path:
>>where django-admin      # On Windows

# or
>>which django-admin      # On macOS/Linux
```

You will see the path, something like:

C:\Users\<YourName>\DjangoProjects\venv\Scripts\django-admin.exe

Step 6: Create a New Django Project

Use the django-admin command (from the virtual environment) to create a project:

```
>>django-admin startproject myproject
```

Now your folder structure looks like:

```
DjangoProjects/
|
|-- venv/
└── myproject/
   |── manage.py
   └── myproject/
      ├── __init__.py
      ├── settings.py
      ├── urls.py
      ├── asgi.py
      └── wsgi.py
```

Step 7: **Navigate to Project Directory**

```
>>cd myproject
```

Step 8: **Run the Development Server**

```
>>python manage.py runserver
```

You will see output like:
Starting development server at http://127.0.0.1:8000/

Step 9: **Open in Browser**

Go to:
http://127.0.0.1:8000/

You will see the Django “Congratulations!” welcome page

Step 10: Deactivate the Virtual Environment (When Done)
deactivate (Ctrl + C)

Django Project structure

1. DjangoProjects/

- This is your main workspace folder — you can create multiple Django projects here.
- It’s just a parent directory to keep your files organized.

2. venv/

• The virtual environment folder.
• It stores all Python packages and dependencies specific to this project (like Django).
• Keeps your system clean and avoids version conflicts between projects.

Note: You never edit files inside this manually.

3. myproject/ (Outer Folder)

• The project container folder created when you ran startproject.
• Contains the manage.py file and another folder with the same name (inner project directory).

4. manage.py

A command-line utility for managing your project.
You use it to run commands like:

```
>> python manage.py runserver              # Start server
>> python manage.py startapp app1          # Create a new app
>> python manage.py makemigrations         # Prepare database migrations
>> python manage.py migrate                # Apply migrations
>> python manage.py createsuperuser        # Create admin user
```

Note: Think of it as your project's control center.

5. Inner myproject/ Folder

This is the actual project package — it contains configuration and core settings that define how your Django project behaves.

This folder consists of most important files:

__init__.py

- Marks this folder as a Python package.
- Without this file, Python wouldn't recognize the folder as importable code.

settings.py

- The heart of your Django project and it contains all the configuration settings, such as:
  - Installed apps
  - Database connection details
  - Static files setup
  - Middleware
  - Templates
  - Security and debug settings

Note: You will modify this file often during development.

urls.py

- Controls your project's URL routing.
- Maps URLs (like /home, /about) to specific views or apps.

Example:

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

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

Note: Think of it as the table of contents for your website's pages.

asgi.py

- Used for Asynchronous Server Gateway Interface.
- Handles asynchronous requests (like real-time apps or WebSockets).

Note: Required when deploying Django on asynchronous servers.

wsgi.py

- Stands for Web Server Gateway Interface.
- It is the entry point for traditional (synchronous) web servers to serve your Django project.

Note: Used when deploying Django on production servers like Apache or Nginx.