
Django Learning Report

Name : Prajwal Bashyal
Institution: Bhaktapur Multiple Campus, BIT
Date : 2025-12-14
Topic : Django – Project structure and core concept

Django Project Structure & Core Concepts

This document is written so it can be directly used **in a Django report** and later **converted into a blog post** with minimal changes.

---

1. Django Project: A Collection of Apps

In Django, a **project** is the overall web application, while an **app** is a modular component that performs a specific function.

- A **project** acts as a container that holds:
    - Multiple Django apps
    - Global configuration files
    - Server interface files

Example:

- Project: `college_portal`
- Apps inside it: `accounts`, `results`, `notices`

This design follows **modular programming**, allowing:

- Reusability of apps
- Better code organization
- Easier maintenance and scalability

---

2. Why Is the Same Project Name Folder Created Twice?

When we run:

```
django-admin startproject myproject
```

Django creates **two folders with the same name**:

```
myproject/
|── manage.py
```

```
|── myproject/
    |── __init__.py
    |── settings.py
    |── urls.py
    |── asgi.py
    |── wsgi.py
```

Explanation:

- **Outer `myproject/` folder**
    - Project root directory
    - Used for managing the project
    - Contains `manage.py`
- **Inner `myproject/` folder**
    - Actual Python package of the project
    - Contains core configuration files

This separation keeps **project management logic** and **application configuration logic** cleanly organized.

---

3. Files Inside the Project Folder (Core Files Explained)

3.1 `__init__.py`

- Marks the directory as a **Python package**
- Allows Django to import project modules
- Usually empty but mandatory for package recognition

3.2 `settings.py`

This is the **main configuration file** of a Django project.

It contains:

- Installed apps (`INSTALLED_APPS`)
- Database configuration
- Middleware

- Templates settings
- Static & media file settings
- Security configurations

Term: `__file__` in `settings.py`

- `__file__` is a Python attribute
- It stores the **absolute path of the current file**

Used like:

```
BASE_DIR = Path(__file__).resolve().parent.parent
```

Purpose:

- Helps Django dynamically locate project directories
- Makes the project portable across systems

3.3 `urls.py`

- Acts as the **URL routing system**
- Maps URLs to views

Example:

```
path('login/', views.login_view)
```

3.4 `wsgi.py`

WSGI stands for **Web Server Gateway Interface**.

Purpose:

- Connects Django with **production web servers** like:
    - Apache
    - Nginx
    - Gunicorn

Local Server vs Live Server

Local Server	Live Server
Used for development	Used in production

Local Server	Live Server
Runs using `runserver`	Runs using WSGI server
Debug mode ON	Debug mode OFF
Not publicly accessible	Publicly accessible

`wsgi.py` is mainly used when deploying Django to a **live server**.

3.5 `asgi.py`

- ASGI = Asynchronous Server Gateway Interface
- Used for **async features** like:
    - WebSockets
    - Real-time notifications

---

4. `manage.py` in Detail

`manage.py` is a **command-line utility** for Django projects.

It allows developers to:

- Run development server
- Apply migrations
- Create apps
- Access Django shell

How It Works:

- Sets the default settings module
- Acts as a wrapper around Django's admin commands

---

5. Command Line Arguments in Django

Command-line arguments are instructions passed to `manage.py`.

Example:

```
python manage.py runserver
```

Here:

- `python` → Python interpreter
- `manage.py` → Django management script
- `runserver` → Command-line argument

Other common commands:

```
python manage.py makemigrations
python manage.py migrate
python manage.py createsuperuser
```

Each argument tells Django **what task to perform**.

---

6. Port Description in Django

When running:

```
python manage.py runserver
```

Django starts the server at:

```
http://127.0.0.1:8000/
```

Port Explanation:

- **127.0.0.1** → Localhost (your own machine)
- **8000** → Port number

A **port** is a logical endpoint that allows multiple services to run on the same system.

Custom Port Example:

```
python manage.py runserver 8080
```

This runs Django on port **8080** instead of 8000.

Ports — *Very Simple Explanation*

Think of your computer like a **house** 🏠.

- The **IP address** is the *house address*
- **Ports** are the *doors* of the house

Each door is used for a **different purpose**.

Why ports are needed?

Many services run on one computer at the same time:

- Website
- Email
- Database
- Django server

Ports tell the computer **which service should receive the request**.

A port is a logical endpoint that helps a computer identify which service or application should receive incoming network requests.

7. Summary

- A Django project is a **collection of apps**
- Duplicate project folders help separate **management** and **configuration**
- Core files handle routing, settings, and server communication
- `manage.py` enables command-line control
- Ports define where the server listens for requests

This structured design makes Django **scalable, maintainable, and production-ready**.