DJANGO NOTES

**What is Django?**

Django is a **high-level Python web framework** used for building secure, scalable, and maintainable web applications. It follows the **Model-View-Template (MVT) architecture** and provides built-in features like authentication, ORM (Object-Relational Mapping), and admin panel.

🔹 **Key Features of Django:**  
**Fast Development** – Reduces repetitive tasks using built-in tools.  
**Security First** – Protects against SQL injection, CSRF, XSS, and more.  
**Scalable** – Used by large applications like Instagram, Pinterest, and Disqus.  
**Batteries Included** – Comes with admin panel, authentication, ORM, and more.

1. Django Architecture: MVT (Model-View-Template)

| **Component** | **Description** |
| --- | --- |
| **Model (M)** | Handles database operations (tables, queries). |
| **View (V)** | Contains business logic and processes user requests. |
| **Template (T)** | Controls the frontend (HTML, CSS, JavaScript). |

**MVT Flow**

1️ **User requests a URL →** Django **views.py** handles it.  
2️ **View interacts with the model →** Fetches data from the database.  
3️ **View sends data to the template →** Renders HTML with dynamic content.

User Request → URLconf → View → Model (DB) → View → Template → Response to User

1. Setting Up Django

🔹 Install Django

pip install Django

🔹 Create a Django Project

django-admin startproject <project\_name>

cd myproject

python manage.py runserver # Run the development server

Open http://127.0.0.1:8000/ to see the Django welcome page.

🔹 Django Project Structure

myproject/

│── manage.py # Django CLI tool

│── myproject/ # Main project directory

│ ├── \_\_init\_\_.py

│ ├── \_\_asgi\_\_.py

│ ├── settings.py # Project settings

│ ├── urls.py # URL routing

│ ├── wsgi.py # Web server gateway

**\_\_init\_\_.py – package installer. It will be an empty file. If not present, it will not work**

**Settings.py – All settings and configurations already will be loaded.**

**Manage.py – each project will have a separate manage.py file**

| **Reason** | **Benefit** |
| --- | --- |
| **✅ Built-in Admin Panel** | **Save time on backend UI** |
| **✅ ORM (Object Relational Mapper)** | **Work with DB using Python** |
| **✅ Form handling** | **Secure and simple** |
| **✅ Scalable and robust** | **Used by large sites like Instagram, Pinterest** |

**Ways to create an app**

1. Django Apps

Django projects consist of multiple **apps** (modular components).

🔹 Create an App

Django-admin startapp <app\_name>

**Django App Structure:**

├── myapp/ ← A Django app

│ ├── migrations/ ← Auto-created DB migration files

│ │ └── \_\_init\_\_.py

│ ├── \_\_init\_\_.py

│ ├── admin.py ← Admin interface settings

│ ├── apps.py ← App config

│ ├── models.py ← Database models

│ ├── tests.py ← Unit tests

│ └── views.py ← Views Business logic (controller)

│

└── db.sqlite3 ← Default database file (if using SQLite)

myapp/

│── migrations/ # Database migrations

│── models.py # Database models

│── views.py # Business logic

│── urls.py # App-specific URLs

│── templates/ # HTML files

│── static/ # CSS, JS, images

🔹 Register App in settings.py

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'myapp', # Add your app here

]

Django **ORM (Object-Relational Mapping)** lets you interact with databases using Python.

**🔹 Define a Model in models.py**

from django.db import models

class Book(models.Model):

title = models.CharField(max\_length=255)

author = models.CharField(max\_length=255)

published\_date = models.DateField()

def \_\_str\_\_(self):

return self.title

🔹 Apply Migrations (Sync with Database)

python manage.py makemigrations

python manage.py migrate

Put the code of settings.py where we connect to database, DB connection string to be given here

DATABASES = {  
 'default': {  
 'ENGINE': 'django.db.backends.mysql',  
 'NAME': 'emp',  
 'USER': 'root',  
 'PASSWORD': 'root'  
 }  
}

myproject/

│

├── manage.py

├── myproject/ ← Project settings directory

│ ├── \_\_init\_\_.py

│ ├── settings.py ← Project-level settings

│ ├── urls.py ← Project-level URL declarations

│ ├── asgi.py ← For asynchronous deployments – asynchronous server gateway interface

│ └── wsgi.py ← For WSGI-compatible web servers – web server gateway interface

│

├── myapp/ ← A Django app

│ ├── migrations/ ← Auto-created DB migration files

│ │ └── \_\_init\_\_.py

│ ├── \_\_init\_\_.py

│ ├── admin.py ← Admin interface settings

│ ├── apps.py ← App config

│ ├── models.py ← Database models

│ ├── tests.py ← Unit tests

│ └── views.py ← View logic (controller)

│

└── db.sqlite3 ← Default database file (if using SQLite)

**Explanation of Key Files**

| **File/Folder** | **Purpose** |
| --- | --- |
| manage.py | Command-line utility to manage the project. |
| settings.py | Contains all settings for your project (DB, apps, templates). |
| urls.py | Root URL config; maps URLs to views. |
| wsgi.py / asgi.py | Entry points for web servers (WSGI for sync, ASGI for async). |
| myapp/ | A Django application inside the project. |
| models.py | Contains database models (ORM). |
| views.py | Functions or classes that define the response for each URL. |
| admin.py | Registers models for Django admin panel. |
| migrations/ | Stores migration files to sync models with the database. |

**Optional Folders You May Add**

| **Folder** | **Purpose** |
| --- | --- |
| templates/ | HTML templates used by views. |
| static/ | CSS, JavaScript, and images. |
| forms.py | Custom Django forms. |
| serializers.py | Used in Django REST Framework to convert models to JSON. |

**Summary Table**

| **Feature** | **wsgi.py** | **asgi.py** |
| --- | --- | --- |
| Full Form | **Web Server Gateway Interface** | **Asynchronous Server Gateway Interface** |
| Protocol | **Synchronous (WSGI)** | **Asynchronous + Synchronous (ASGI)** |
| Use Case | Traditional web apps | Real-time apps (chat, WebSockets) |
| Performance | Slower for async tasks | Better for concurrent connections |
| Web Server Example | Gunicorn, uWSGI | Daphne, Uvicorn |
| Django Version | Default before Django 3.0 | Introduced in Django 3.0+ |

Why ORM:

Instead of writing SQL like this:

SELECT \* FROM students WHERE age > 18;

You can write in Django like this:

Student.objects.filter(age\_\_gt=18)

**How Django ORM Works**

Django ORM translates **Python classes** and their **objects** into **SQL tables and records**.

**Example:**

**1. Model (Python class)**

from django.db import models

class Student(models.Model):

name = models.CharField(max\_length=100)

age = models.IntegerField()

This creates a table in the database like:

| **id** | **name** | **age** |
| --- | --- | --- |
| 1 | Amar | 20 |
| 2 | Seetha | 22 |

**Common ORM operations:**

| **Task** | **ORM Code** |
| --- | --- |
| Get all records | Student.objects.all() |
| Filter records | Student.objects.filter(age=20) |
| Get one record | Student.objects.get(id=1) |
| Add new record | Student.objects.create(name='Anil', age=25) |
| Update a record | student.age = 30; student.save() |
| Delete a record | student.delete() |