DJANGO PROJECT STRUCTURE (BIG PICTURE)

After running:

```
python -m django startproject myproject
```

You get:

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

1. manage.py – Project Controller

What it does
This file is used to:

- run server
- create apps
- apply migrations
- manage the project

Students should remember:

“All Django commands are run using manage.py”

Basic manage.py (default)

```
#!/usr/bin/env python
import os
import sys
```

```
def main():
    os.environ.setdefault('DJANGO_SETTINGS_MODULE',
'myproject.settings')
    try:
        from django.core.management import
execute_from_command_line
    except ImportError:
        raise
    execute_from_command_line(sys.argv)

if __name__ == '__main__':
    main()
```

2. settings.py – Project Configuration File

What it does

- Installed apps
- Database configuration
- Middleware
- Templates
- Static files

Installed Apps

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

When you create your own app:

```
INSTALLED_APPS = [
    ...
    'app1',
```

```
]
```

Database Configuration (default)

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

Static Files

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

Used for:

- CSS
- JS
- Images

3. urls.py – URL Routing File

What it does

- Connects URL → View
- Decides which function runs for which URL

Basic urls.py

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

def home(request):
    return HttpResponse("Welcome to Django")

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

4. wsgi.py – Web Server Gateway Interface

What it does

- Used in production
- Connects Django with web servers like Apache, Gunicorn

Code (don't touch)

```
import os
from django.core.wsgi import get_wsgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE',
'myproject.settings')

application = get_wsgi_application()
```

5. asgi.py – Async Server Gateway Interface

What it does

- Used for async features
- WebSockets
- Real-time apps (chat apps)

```
import os
from django.core.asgi import get_asgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE',
'myproject.settings')

application = get_asgi_application()
```

DJANGO APP STRUCTURE

After running:

```
python manage.py startapp app1
```

You get:

```
app1/
├── admin.py
├── apps.py
├── models.py
├── tests.py
├── views.py
└── migrations/
```

6. views.py – Business Logic

What it does

- Handles request
- Returns response

Basic views.py

```
from django.http import HttpResponse

def home(request):
    return HttpResponse("Hello from Django App")
```

Teaching line
View = Python function that returns output

7. urls.py (inside app – you create this)

Used to keep project clean

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

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

Then connect app URLs in project urls.py:

```
from django.urls import path, include

urlpatterns = [
    path('app1/', include('app1.urls')),
]
```

8. models.py – Database Tables

What it does

- Defines database structure

Example models.py

```
from django.db import models

class Student(models.Model):
    name = models.CharField(max_length=50)
    age = models.IntegerField()
```

Then run:

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

Teaching line

👉 One model = one table

admin.py – Admin Panel Configuration

What it does

- Registers models in Django admin

```
from django.contrib import admin
from .models import Student

admin.site.register(Student)
```

10. apps.py – App Configuration

```
from django.apps import AppConfig

class App1Config(AppConfig):
    name = 'app1'
```

Mostly auto-managed

11. tests.py – Testing File

```
from django.test import TestCase
```

Used for writing test cases (advanced topic)

FINAL TEACHING SUMMARY (VERY IMPORTANT)

File	Purpose
manage.py	Runs project
settings.py	Configuration
urls.py	URL routing
views.py	Logic
models.py	Database
admin.py	Admin panel
wsgi.py	Production
asgi.py	Async features

1. manage.py

This file acts as the **command-line controller** of the Django project.
It is used to run the server, create apps, apply migrations, and execute all Django management commands.
Developers usually **do not modify this file**.

2. settings.py

This file contains **all configuration settings** of the Django project.
It controls installed applications, database connections, middleware, templates, static files, and security settings.
Any change related to project behavior is generally done here.

3. urls.py (Project Level)

This file is responsible for **URL routing** at the project level.
It maps incoming URLs to applications or views and decides which page opens for a given URL.
It acts as the **entry point for all HTTP requests**.

4. wsgi.py

This file is used in **production environments**.
It connects the Django project with traditional web servers such as Apache or Gunicorn.
Developers rarely change this file during normal development.

5. asgi.py

This file is used for **asynchronous features** in Django.
It enables real-time functionality such as WebSockets and async requests.
It becomes important in advanced Django applications like chat systems.

6. views.py

This file contains the **business logic** of the application.
It receives user requests, processes data, and returns responses.
Every web page in Django is ultimately controlled by a view.

7. urls.py (App Level)

This file handles **URL routing for a specific app**.
It keeps the project structure clean by separating app URLs from project URLs.
Requests flow from project URLs to app URLs and then to views.

8. models.py

This file defines the **database structure** using Django models.
Each model represents a table and each field represents a column.
Django uses this file to automatically create and manage database tables.

9. admin.py

This file controls how models appear in the **Django Admin Panel**.
Registering models here allows administrators to add, update, and delete records using the admin interface.
It simplifies backend data management.

10. apps.py

This file stores **application configuration details**.
It helps Django recognize the app and manage it internally.
Most of the time it is auto-generated and not manually edited.

11. tests.py

This file is used to write **test cases** for the application.
It helps verify that the code works correctly and prevents future errors.
Testing is usually introduced after students understand core Django concepts.

Final Summary (One-Line Memory Trick)

- **manage.py** → runs the project
- **settings.py** → project settings
- **urls.py** → URL routing
- **views.py** → logic
- **models.py** → database
- **admin.py** → admin panel

- **wsgi.py** → production server
- **asgi.py** → async support

Project Overview

Assign students a Student Dashboard project using Django. This involves building a web app where users (students) log in to view personalized stats like courses enrolled, grades, attendance, and simple charts. It reinforces models, views, templates, authentication, and basic data visualization for a one-week timeline.

Learning Objectives

- Implement Django models for core entities like User, Course, Grade.
- Handle CRUD operations with class-based views.
- Customize Django admin for data management.
- Add user authentication and dashboard personalization.
- Display data with HTML templates, Bootstrap for UI, and Chart.js for visuals.

Project Features

- User registration/login/logout.
- Student dashboard showing enrolled courses, average grade, attendance percentage.
- Admin panel to add courses, assign grades/attendance.
- Simple charts (e.g., grade trends over time).
- Search/filter courses or grades.

File Structure

Use this standard Django layout for clarity:

```
dashboard_project/
├── manage.py
├── requirements.txt
├── dashboard_project/
```

```
|   ├── __init__.py
|   ├── settings.py
|   ├── urls.py
|   └── wsgi.py
├── core/  # Main app for models/views
|   ├── __init__.py
|   ├── admin.py
|   ├── apps.py
|   ├── migrations/
|   ├── models.py
|   ├── tests.py
|   ├── urls.py
|   ├── views.py
|   ├── forms.py
|   ├── templates/core/
|   |   ├── base.html
|   |   ├── dashboard.html
|   |   ├── login.html
|   |   └── register.html
|   └── static/core/
|       ├── css/style.css
|       └── js/charts.js
└── media/  # User uploads if extended
```

Populate requirements.txt with: Django==5.0, pillow (for images/charts).

Step-by-Step Guidelines

1. Setup: django-admin startproject dashboard_project , cd dashboard_project , python manage.py startapp core . Add 'core' to INSTALLED_APPS in settings.py . Run python manage.py migrate .

2. Models: In models.py , define Course (name, credits), Enrollment (user, course, grade, date). Use OneToMany via ForeignKey. python manage.py makemigrations && python manage.py migrate .

3. Admin: Register models in admin.py with list_display , search_fields for easy data entry.

4. Auth Views: Use LoginView , LogoutView ; create custom DashboardView (LoginRequiredMixin) fetching user data.

5. URLs: Project urls.py includes app URLs. App urls.py : paths for login, dashboard, etc.

6. Templates: base.html with Bootstrap navbar. dashboard.html shows stats, embed Chart.js canvas for grades chart.

7. Charts: Fetch data in view as JSON, use Chart.js in JS for bar/line charts.

8. Static/Media: python manage.py collectstatic . Serve in settings.py for dev.

9. Test/Run: Create superuser, add sample data via admin, python manage.py runserver . Deploy to free host like Render if time allows.

Extensions for Bonus

- Add attendance model with percentage calc.
- Real-time updates via htmx or simple JS fetch.
- Export grades as PDF using ReportLab.

Evaluation Criteria

- Working auth and personalized dashboard.
- Clean UI with responsive design.
- Proper error handling (e.g., no data).
- Git repo with commit history.
- Brief README explaining setup/run.