**Files in startproject: (django-admin startproject projectname)**

manage.py – A command-line tool for managing your Django project (running server, migrations, etc.).

projectname/ – The main project folder containing core settings.

**init**.py – Marks the directory as a Python package.

settings.py – Configuration file (database, installed apps, middleware, etc.).

urls.py – Defines URL patterns for routing.

asgi.py – Entry point for ASGI-compatible web servers.

wsgi.py – Entry point for WSGI-compatible web servers.

**Files in startapp: (python manage.py startapp appname)**

appname/ – The application directory.

**init**.py – Marks the directory as a Python package.

admin.py – Configuration for Django admin panel.

apps.py – Configuration for the app.

models.py – Defines database models.

views.py – Handles request and response logic.

tests.py – Contains unit tests for the app.

urls.py (optional, needs to be created) – Defines app-specific URLs.

migrations/ – Stores database migration files.

**Working with views.py File:**

1. **Basic View (Function-Based View) A simple function-based view returns an HTTP response**.

**Example: Basic View**

In views.py:

python Copy Edit

from django.http import HttpResponse

def home(request):

return HttpResponse("Hello, Django!")

Here:

* request is the HTTP request object.
* HttpResponse sends a simple text response.

Now, link this view to a URL.

In urls.py (inside your app):

python Copy Edit

from django.urls import path

from .views import home

urlpatterns = [

path('', home, name='home'), # Now, accessing '/' shows "Hello, Django!"

]

**2**. **Using Templates (Rendering HTML) Instead of returning plain text, let's render an HTML template.**

**Example: Rendering a Template**

Modify views.py:

python Copy Edit

from django.shortcuts import render

def home(request):

return render(request, 'home.html')

* render() looks for home.html inside the templates folder.

Create a templates/home.html file:

html Copy Edit

<!DOCTYPE html>

<html>

<head>

<title>Home</title>

</head>

<body>

<h1>Welcome to Django!</h1>

</body>

</html>

**3. Passing Data to Template You can pass dynamic data to the template using a context dictionary.**

**Example: Passing Data**

In views.py:

python Copy Edit

def home(request):

context = {

'name': 'John Doe',

'age': 25

}

return render(request, 'home.html', context)

Modify home.html:

html Copy Edit

**<h1>Hello, {{ name }}!</h1>**

**<p>You are {{ age }} years old.</p>**

Now, visiting the page will display:

sql Copy Edit

Hello, John Doe!

You are 25 years old.

**4. Class-Based Views Django also provides Class-Based Views (CBVs), which help organize code better.**

**Example: Using a Class-Based View**

In views.py:

python Copy Edit

from django.views.generic import TemplateView

class HomeView(TemplateView):

template\_name = 'home.html'

Update urls.py:

python Copy Edit

from django.urls import path

from .views import HomeView

urlpatterns = [

path('', HomeView.as\_view(), name='home'),

]

This does the same as before but in a structured way.

1. **Handling Forms (User Input) A view can handle form submissions.**

**Example: Handling a Form**

In views.py:

python Copy Edit

from django.shortcuts import render

from django.http import HttpResponse

def contact(request):

if request.method == 'POST':

name = request.POST.get('name')

return HttpResponse(f"Thank you, {name}!")

return render(request, 'contact.html')

Create templates/contact.html:

html Copy Edit

Top of Form

<form method="post">

{% csrf\_token %}

<label>Name:</label>

<input type="text" name="name">

<button type="submit">Submit</button>

</form>Bottom of Form

Now, entering a name and submitting will display a thank-you message.