Django Syllabus From **18th February**

* **By Digital Pathshala**

**Contents**

[1. Day 1: Introduction to Django 3](#_Toc190886311)

[1.1. What is Django? Why use Django? 3](#_Toc190886312)

[1.2. Setting up a Django project 3](#_Toc190886313)

[1.2.1. Check if Python is Installed 3](#_Toc190886314)

[1.2.2. Install pip (Python Package Manager) 3](#_Toc190886315)

[1.3. Installing and using a virtual environment 4](#_Toc190886316)

[2. Day 2: Django Project Structure 5](#_Toc190886317)

[2.1. Installation of Django on virtual environment 5](#_Toc190886318)

[2.2. Creation of Django Project 5](#_Toc190886319)

[2.3. Running the development server 5](#_Toc190886320)

[2.4. Creating and exploring a Django app 5](#_Toc190886321)

[2.5. Understanding Django’s Folder Structure 5](#_Toc190886322)

[2.6. Role of Key Files 6](#_Toc190886323)

[2.7. Configuring Django Settings 9](#_Toc190886324)

[2.8. Creating and Managing Django Apps 9](#_Toc190886325)

**Week 1: Django Basics (Days 1–6)**

# Day 1: Introduction to Django

## What is Django? Why use Django?

## Setting up a Django project

### Check if Python is Installed

Before installing Django, make sure Python is installed on your system.

For Windows, open Command Prompt (cmd) and run **python --version** or **python3 --version.**

If Python is not installed, download it from the official Python website and install it.

### Install pip (Python Package Manager)

**For Windows:**

Download the pip installer script from: **https://bootstrap.pypa.io/get-pip.py**

Go to the download directory and run the command:

**python get-pip.py**

**or python3 get-pip.py**

Find the Scripts directory inside the Python installation folder and add it to Environment Variables.

**For Mac:**

Run the following command to download the pip installer script:

**curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py**

Execute the script using:

**python3 get-pip.py**

**For Linux:**

Install pip using:

**sudo apt-get install python3-pip python-dev**

Verify pip Installation:

Run the following command to check if pip is installed correctly:

**pip --version**

**or pip3 –version**

## Installing and using a virtual environment

After successfully installing pip, the next step is to set up a **virtual environment** using **venv**. A virtual environment allows you to manage dependencies separately for each project, making it useful for deployment or transferring projects to another system.

**Installing Virtual Environment**

Run the following command to install **venv**:

* **pip install virtualenv**

Each project should have its own **venv** to maintain package isolation.

**To create virtual environment file:**

**python -m venv env**

**Activating the Virtual Environment**

* On Windows (Command Prompt or VS Code Terminal)
  + \venv\_folder\_name\Scripts\activate.bat
  + \venv\_folder\_name\Scripts\activate
* On Mac/Linux (Terminal)
  + source venv\_folder\_name/bin/activate

**Note:** If the activation does not work in **PowerShell**, open **Command Prompt** in VS Code and try again.

Generating packages on requirement.txt file:

pip freeze > requirements.txt

# Day 2: Django Project Structure

## Installation of Django on virtual environment

Once the virtual environment is activated, install Django using:

* **pip install django**

## Creation of Django Project

To create a new Django project, run:

* **django-admin startproject** project\_name : blogapp

**Note: Replace project\_name with your preferred project name.**

## Running the development server

Navigate to the project directory and start the Django server:

* cd project\_name
* **python manage.py runserver**

## Creating and exploring a Django app

Inside your project folder, create a Django app using:

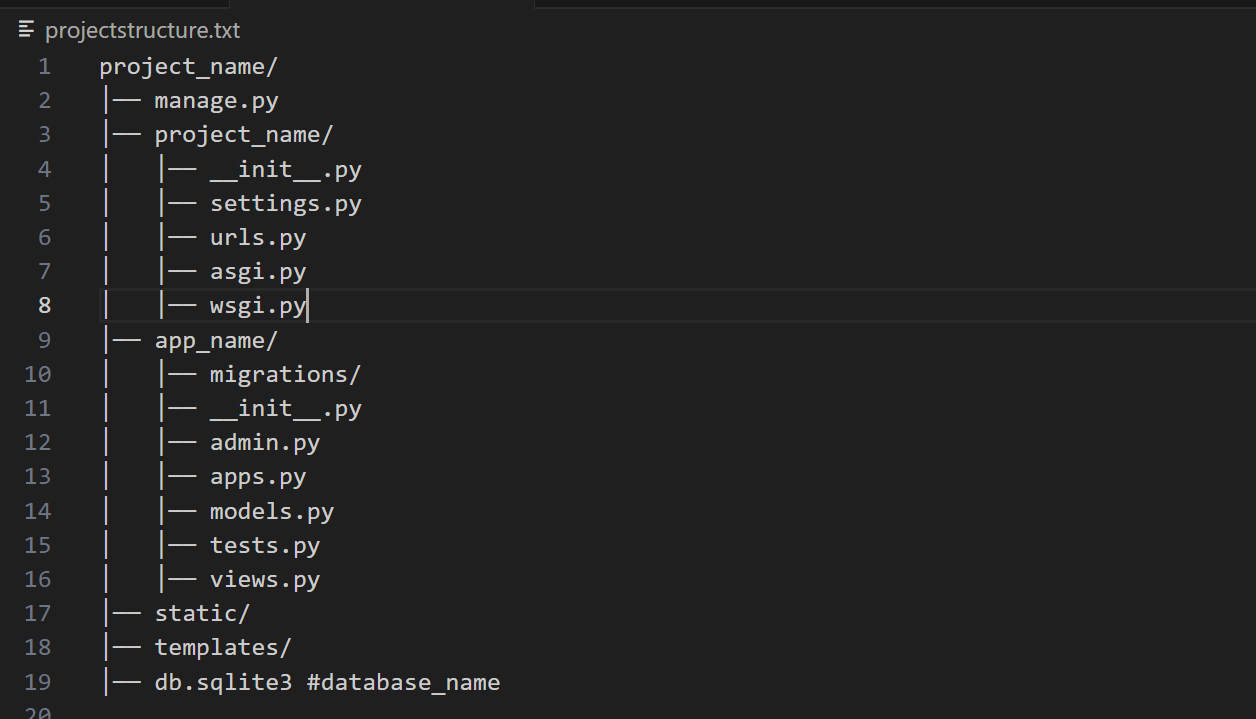
* python manage.py startapp app\_name

For example, to create an app for a **blog website**:

* **python manage.py startapp** blog
* **python manage.py startapp** blog # Handles blog posts
* **python manage.py startapp** users # Manages user authentication
* **python manage.py startapp** comments # Manages comments on blog posts
* **python manage.py startapp** categories # Manages categories for blog posts
* **python manage.py startapp** likes # Manages likes on blog posts and comments

## Understanding Django’s Folder Structure

After creating a Django project (**django-admin startproject** project\_name), the structure will look like this:



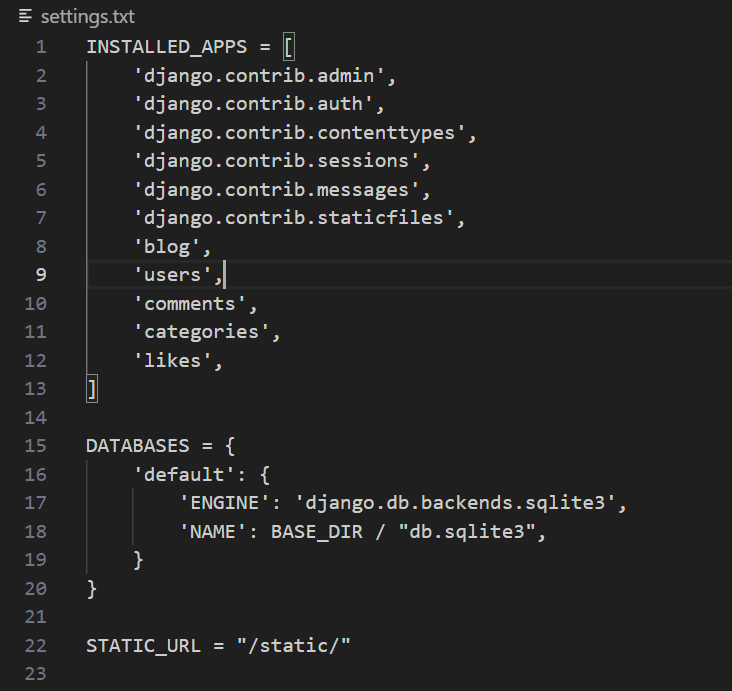
Each file serves a specific role:

* **manage.py**: A command-line utility for managing the Django project.
* **settings.py**: Configures settings like databases, middleware, installed apps, etc.
* **urls.py**: Defines URL patterns and routes to views.
* **wsgi.py / asgi.py**: Entry points for web servers (**Web Server Gateway Interface** (WSGI ) for synchronous apps**, Asynchronous Server Gateway Interface**( ASGI) for async apps).
* **models.py**: Defines database models (tables).
* **views.py**: Contains functions or classes that handle requests and return responses.
* **admin.py**: Registers models to the Django admin panel.
* **apps.py**: Configures app settings.
* **migrations/**: Stores database migration files.

## Role of Key Files

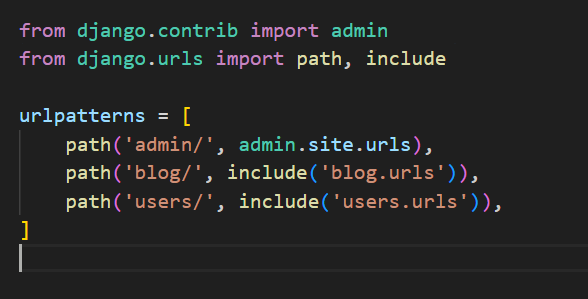
* **settings.py** (Configures Django Settings)

This file manages configurations like installed apps, middleware, databases, static files, etc.



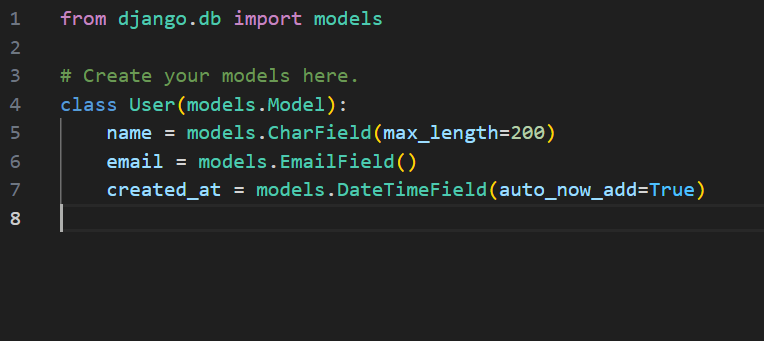
* **urls.py** (Defines URL Routing)

Maps URLs to views.



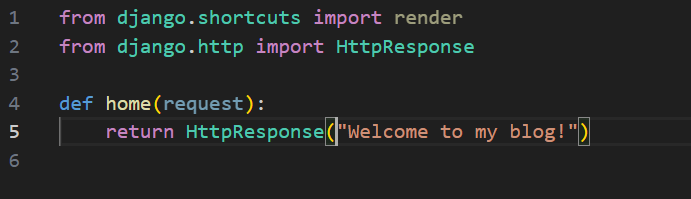
* **models.py** (Defines Database Tables)

Used to create and manage database tables using Django ORM.



* **views.py** (Handles Business Logic)

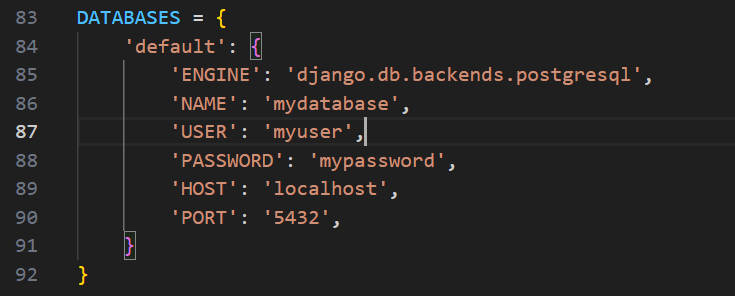
Defines what happens when a user visits a particular URL.



## Configuring Django Settings

Django settings can be modified inside settings.py. Common configurations include:

* Database Configuration

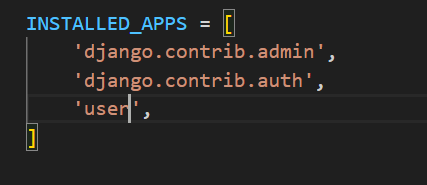


## Creating and Managing Django Apps

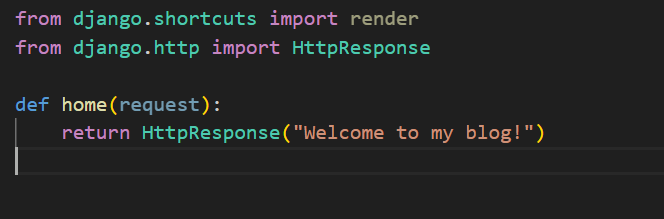
* **Creating an App**

python manage.py startapp app\_name

* Registering an App



* Creating Views



* Creating URLs

A screen shot of a computer code

AI-generated content may be incorrect.

# Day 3: URLs and Views

## How Django processes a request

Django follows a **request-response cycle** when handling HTTP requests. The key steps are:

* **User Makes a Request**
  + A user accesses a URL (e.g., http://localhost:8000/home).
* **Django URL Resolver**
  + Django searches for a matching pattern in urls.py.
* **View Execution**
  + If a matching URL pattern is found, the associated **view function** or **class-based view** (CBV) is executed.
* **Response Generation**
  + view processes the request and returns an HttpResponse (HTML, JSON, etc.).
* **Browser Displays Response**
  + The response is rendered in the user's browser

## Mapping URLs with urls.py

Django uses URL patterns to map requests to specific views. The main urls.py file is typically located in the project folder (next to settings.py).

## Function-based views (FBVs) vs. Class-based views (CBVs)

Django provides two ways to define views:

* Function-Based Views (FBVs)

FBVs use simple functions to handle requests.



* Class-Based Views(CBVs)

CBVs provide reusable and object-oriented views.

View: This is the base class for all CBVs. It dispatches HTTP requests to the appropriate method (get(), post(), etc.).

A screen shot of a computer screen

AI-generated content may be incorrect.

## Django Templates

### Template syntax ({{}} and {% %})

**{{ }}**: This is the syntax used to output variables and dynamic data into HTML. For example:

**{% %}**: This syntax is for control structures like loops and conditionals. For example:

This helps in rendering logic-based content.

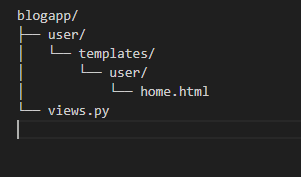
### Rendering templates using render()

The render() function in Django is used to generate an HTTP response by combining a template with context data. This is typically used in views to dynamically render HTML pages.

**Steps to render templates**:

**Create the Folder Structure**

* First, inside the user folder, create a folder called templates.
* Inside the templates folder, create another folder called user (this helps to organize templates by app).
* Inside the user folder, create the HTML file that you want to render (e.g., home.html).



**Create the Template (home.html)**:

* Now, in the home.html file, you can write your HTML content.
* You can use Django’s template syntax to display dynamic data passed from the view.

Example code:

A screenshot of a computer

AI-generated content may be incorrect.

**Create the View Function**:

* In your views.py file, create a function that will render the HTML template using the render() function.
* The render() function combines the template (home.html) with the context (data you want to pass to the template) and the request.

A screen shot of a computer code

AI-generated content may be incorrect.

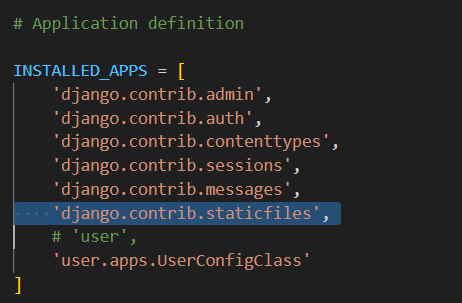


Html file message displaying successfully:

### Using template inheritance (base.html)

### Static files (CSS, JS, images)

Step of using static files:



In settings.py file

Starting of settings.py file:



In the end of the settings.py file:

A screen shot of a computer program

AI-generated content may be incorrect.

Django provides a {% static %} template tag to refer to static files in HTML templates.

Load the static tag at the top of the template:

{% load static %}

<img src="{% static user/images/logo.png' %}" alt="Logo">

**Day 5: Models and Migrations**

* Creating models (models.py)
* Fields and data types
* Running migrations (makemigrations, migrate)
* Django ORM introduction

**Day 6: Querying the Database**

* CRUD operations (create, read, update, delete)
* Filtering and ordering queries
* Using Django’s interactive shell

**Week 2: Django REST Framework (Days 7–12)**

**Day 7: Introduction to REST APIs and DRF**

* What are REST APIs?
* Installing Django REST Framework
* Serializers and views overview

**Day 8: Serializers in DRF**

* Creating and using serializers
* ModelSerializer vs. regular Serializer
* Serializing and deserializing data

**Day 9: Function-Based Views (FBVs)**

* Creating API views using FBVs
* Handling GET, POST, PUT, DELETE requests

**Day 10: Routers and URLs in DRF**

* Setting up API URLs
* Using DefaultRouter in Django

**Day 11: Query Parameters**

* Filtering and searching APIs
* Handling query parameters in views

**Day 12: Class-Based Views (CBVs)**

* Using Django’s generic API views
* Creating ListView, CreateView, UpdateView, and DeleteView

**Week 3: Advanced DRF (Days 13–18)**

**Day 13: Viewsets and Routers**

* What are viewsets?
* Creating API viewsets
* Using routers

**Day 14: Authentication in Django**

* Token-based authentication
* Session authentication
* JWT authentication

**Day 15: Permissions and Access Control**

* Using DRF’s built-in permissions
* Customizing permissions

**Day 16: Relationships and Nested Serializers**

* Serializing related models
* Using ForeignKey, ManyToManyField in serializers

**Day 17: Pagination in Django**

* Implementing pagination in APIs
* Customizing pagination styles

**Day 18: File Uploads in Django**

* Handling file uploads in Django REST
* Storing images and documents

**Week 4: Blog Project (Days 19–24)**

**Day 19: Setting Up the Blog Project**

* Creating a new Django project
* Setting up the database

**Day 20: Building Models for the Blog**

* Creating Post and Comment models
* Running migrations

**Day 21: Implementing Blog Views and URLs**

* Creating blog views
* Connecting URLs

**Day 22: Designing Templates and Static Files**

* Creating and styling templates
* Managing static files

**Day 23: Adding Authentication**

* User login and logout
* Restricting access to views

**Day 24: Implementing CRUD for Blog Posts**

* Creating, editing, and deleting posts