Django Framework

Unit - 5

Introduction to Django Framework

What is Django?

- Django is a high-level, open-source Python web framework.
- It enables rapid development of secure and maintainable web applications.
- It follows the Model-View-Template (MVT) architectural pattern.
- Created to help developers take applications from concept to completion quickly.

Why Use Django?

- Batteries-included: Comes with authentication, ORM, and admin panel.
- Encourages reusable, maintainable, and clean code.
- Highly secure and scalable for production use.
- Supported by a large and active community.

Django Architecture (MVT Pattern)

- Model – Manages the data and database structure.
- View – Contains business logic and handles data processing.
- Template – Manages presentation and layout for users.
- URL Dispatcher – Connects URLs to views (like routing in Flask).

Installing Django

- You can install Django using pip.

  Commands:

  ```
  pip install django
  ```

- Verify installation:

  ```
  python -m django --version
  ```

Creating a Django Project

- Start a new project using the command:

```
django-admin startproject myproject
```

- Navigate to the project folder: `cd myproject`
- Run the development server: `python manage.py runserver`

Django Project Structure

- manage.py – Command-line utility for administrative tasks.
- settings.py – Contains configuration settings.
- urls.py – Defines URL patterns for routing.
- views.py – Contains logic for handling user requests.
- models.py – Defines database models using ORM.

Summary & Advantages

- Django simplifies complex web development tasks.
- Follows the **DRY** (Don’t Repeat Yourself) principle.
- Provides built-in security features like CSRF and SQL injection protection.
- Suitable for small projects and large-scale enterprise applications.
- Django empowers developers to build fast, secure, and scalable web apps.

Django Project Installation in Virtual Environment

Introduction

- Django is a Python framework for building powerful web applications.
- Installing it inside a virtual environment ensures dependency isolation.
- This prevents version conflicts between projects.

What is a Virtual Environment?

- A virtual environment (venv) is an isolated Python workspace.
- It allows different projects to have their own dependencies.
- Example analogy: Each project gets its own clean toolbox.

Creating a Virtual Environment

- Use Python’s built-in venv module.
- Commands:

```
python -m venv venv
```

- This creates a folder named ‘venv’ that stores dependencies.

Activating the Virtual Environment

Activate the environment before installing Django.

Commands:

```
venv\Scripts\activate     # Windows
source venv/bin/activate   # Mac/Linux
```

You’ll see (venv) before your terminal prompt.

Installing Django

Installing Django

Once the virtual environment is active, install Django using pip.

Command:

```
pip install django
```

This installs the latest version of Django in the environment.

Creating and Running a Django Project

Create a new project using the Django admin tool:

```
django-admin startproject myproject

cd myproject

python manage.py runserver
```

Open http://127.0.0.1:8000/ to verify installation.

Deactivating and Managing the Environment

To deactivate the environment:

```
deactivate
```

To list installed packages:

```
pip list
```

Virtual environments help maintain clean and reproducible setups for Django projects.

Phases in Django Project Creation & Creating a Project

Introduction

- Django development follows a structured workflow from setup to deployment.
- Understanding each phase ensures smooth project creation and maintenance.
- Let’s explore the main phases and commands involved in creating a Django project.

Phase 1 – Environment Setup

- Install Python and Django (preferably inside a virtual environment).
- Use pip to install Django:

```
pip install django
```

- Verify installation:

```
python -m django --version
```

Phase 2 – Project Initialization

- A Django project acts as a container for apps and settings.
- Command to create a new project:

```
django-admin startproject myproject
```

- Navigate into your project folder: cd myproject

Phase 3 – App Creation

Each functionality (like blog, user, shop) is built as an app.

Create an app inside the project using:

```
python manage.py startapp myapp
```

Add the app to INSTALLED_APPS in settings.py.

Apps make your project modular and reusable.

Phase 5 – Running the Server & Testing

- To start the development server:
- python manage.py runserver
- Visit http://127.0.0.1:8000/ in your browser.
- Check if the default Django welcome page appears.
- Fix errors and ensure everything runs correctly.

Summary of Phases

- Environment Setup – Install Python and Django.
- Project Initialization – Create the base structure.
- App Creation – Build modular applications.
- Configuration & Database Setup – Configure settings and migrations.
- Running & Testing – Start server and verify setup.
- Django provides a clear, step-by-step workflow for efficient web development.

SIX PHRASE Democratizing Skilling | a Veranda Enterprise

WORKING WITH MODELS

What is a Model?

- Models define the structure of database tables using Python classes.
- Each class variable becomes a column in the table.
- Models provide an abstraction layer between the database and Python code.
- They allow you to perform database operations (insert, update, delete, fetch) without writing SQL queries.
- Models make it easy to maintain and modify the database structure using Django’s migration system.

Creating a Model

models.py

```
from django.db import models


class Student(models.Model):
    name = models.CharField(max_length=50)
    roll_no = models.IntegerField(unique=True)
    marks = models.FloatField()
```

This creates a table named Student with fields name, roll_no, and marks.

Applying Model Changes

Commands:

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

- makemigrations creates migration files; migrate applies them to the database.

Displaying Data from Model

views.py

```
from django.shortcuts import render
from .models import Student
def show_students(request):
    data = Student.objects.all()
    return render(request, 'students.html', {'students': data})
```

Student.objects.all() fetches all records and sends them to the template.

Template for Display

students.html

```
<h2>Student Records</h2>

<ul>

{% for s in students %}

   <li>{{ s.name }} — {{ s.marks }}</li>

{% endfor %}

</ul>
```

The for loop displays each student’s data fetched from the database.

Registering Model in Admin

admin.py

```
from django.contrib import admin

from .models import Student


admin.site.register(Student)
```

Registering allows adding, editing, and deleting Student records via Django Admin.

FORM PROCESSING

What is a Form?

- Forms collect user input.
- Django’s forms module helps create and validate HTML forms easily.
- Forms handle both displaying the input fields and processing the submitted data.
- Django automatically validates user input to ensure the data is correct and secure.
- Forms can be created manually using forms.Form or directly from models using forms.ModelForm.

Creating a Form

forms.py

```
from django import forms

class StudentForm(forms.Form):

    name = forms.CharField(max_length=50)

    roll_no = forms.IntegerField()

    marks = forms.FloatField()
```

Defines a form with three input fields corresponding to student details

Handling Form Submission

views.py

```
from django.shortcuts import render
from .forms import StudentForm
def student_form(request):
    if request.method == "POST":
        form = StudentForm(request.POST)
        if form.is_valid():
            data = form.cleaned_data
            return render(request, 'success.html', data)
    else:
        form = StudentForm()
    return render(request, 'student_form.html',
{'form': form})
```

Explanation:

Form data is validated with is_valid() and then accessed using cleaned_data.

Template for Form

student_form.html

```
<h3>Enter Details</h3>
<form method="POST">
   {% csrf_token %}
   {{ form.as_p }}
   <button type="submit">Submit</button>
</form>
```

{{ form.as_p }} auto-generates HTML inputs; {% csrf_token %} prevents CSRF attacks.

ModelForm Example

forms.py

```
from django.forms import ModelForm
from .models import Student


class StudentModelForm(ModelForm):
    class Meta:
        model = Student
        fields = ['name', 'roll_no', 'marks’]
```

ModelForm automatically maps model fields to form fields.

STATIC AND MEDIA FILES

Static vs Media

- **Static files** → CSS, JS, Images used in website layout.
- **Media files** → Uploaded by users (e.g., photos, PDFs).

Settings Configuration

settings.py

```
STATIC_URL = '/static/'

STATICFILES_DIRS = [BASE_DIR / 'static']


MEDIA_URL = '/media/'

MEDIA_ROOT = BASE_DIR / 'media’
```

Defines the URL and directory paths for static and media files.

Serve Media Files

urls.py

```
from django.conf import settings

from django.conf.urls.static import static

urlpatterns = [

    path('', views.home),

] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Allows Django to serve uploaded media files during development.

Using Static Files

HTML Example

```
{% load static %}

<link rel="stylesheet" href="{% static 'css/style.css' %}">

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

{% load static %} gives access to the static tag for linking CSS and images.

Handling Uploaded Media

models.py

```
class Profile(models.Model):
    name = models.CharField(max_length=50)
    photo = models.ImageField(upload_to='photos/’)
```

upload_to defines the folder where uploaded images are stored inside MEDIA_ROOT.

Upload Form

template.html

```
<form method="POST" enctype="multipart/form-data">
    {% csrf_token %}
    {{ form.as_p }}
    <input type="submit" value="Upload">
</form>
```

enctype="multipart/form-data" is required to upload files via forms.

Summary

- ✓ Templates – Dynamic HTML
- ✓ Models – Database Tables
- ✓ Forms – Input and Validation
- ✓ Static/Media – Design & File Handling

SIX PHRASE
Democratizing Skilling

a Veranda Enterprise

Thank You