Experiment 12: Install Django and Setup a Virtual Environment

Course Outcomes (COs):

- CO1: Apply knowledge of installation and environment setup to isolate dependencies for web development.

Program Outcomes (POs):

- PO1: Engineering knowledge
- PO5: Modern tool usage

Program Specific Outcomes (PSOs):

- PSO1: Apply software development fundamentals

Objective: Learn how to install Django in a virtual environment to isolate project dependencies.

Steps:

1. **Install Python (if not already installed):**
   - Ensure Python 3.x is installed by running:

     ```
     python --version
     ```

   - Download from https://python.org if not installed.
2. **Create a project folder:**

```
mkdir my_django_project
cd my_django_project
```

4. **Create a virtual environment:**

   ```
   python -m venv venv
   ```

   - This creates an isolated environment named `venv`.
5. **Activate the virtual environment:**
   - On Windows:

     ```
     venv\Scripts\activate
     ```

   - On macOS/Linux:

     ```
     source venv/bin/activate
     ```

   - You should see `(venv)` at the beginning of your terminal prompt.
6. **Install Django:**

   ```
   pip install django
   ```

7. **Verify installation:**

```
django-admin --version
```

8. **Start a new Django project:**

```
django-admin startproject mysite .
```

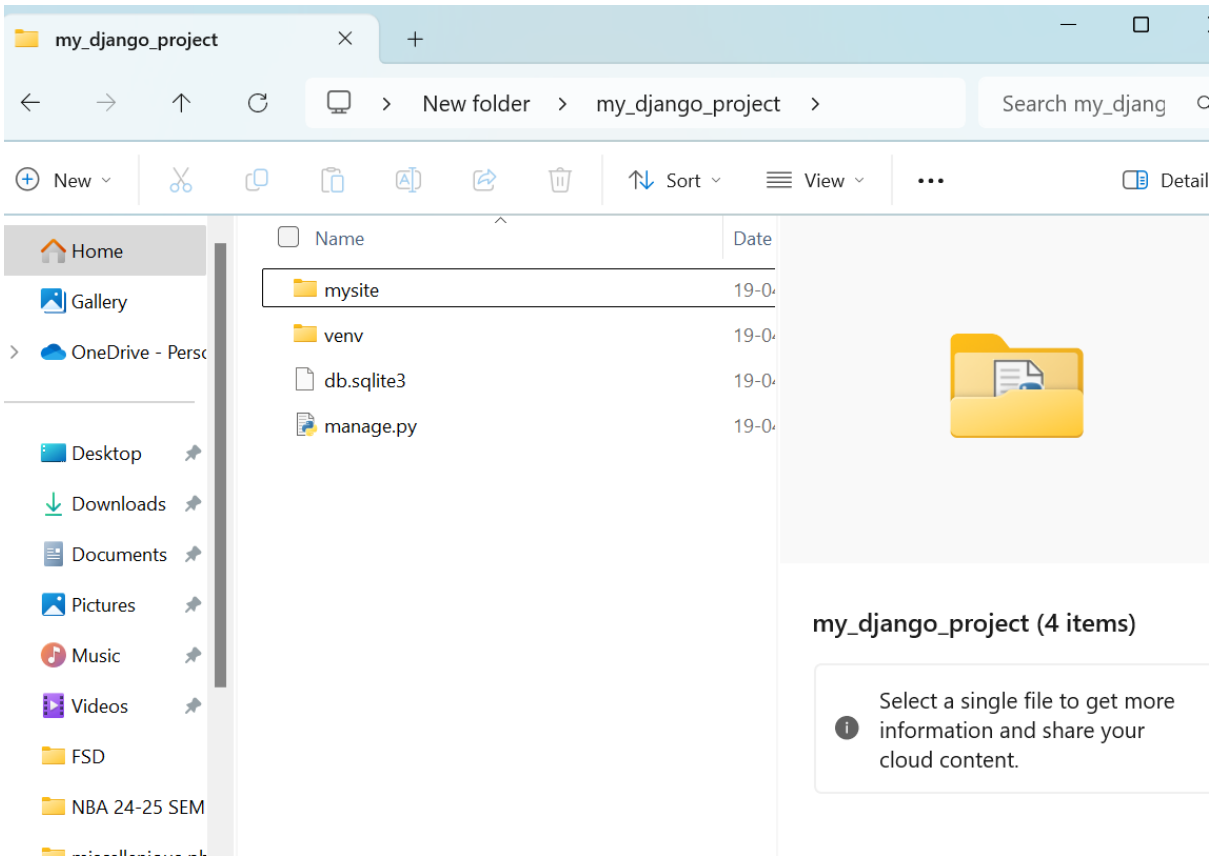
9. **Run the Django server:**

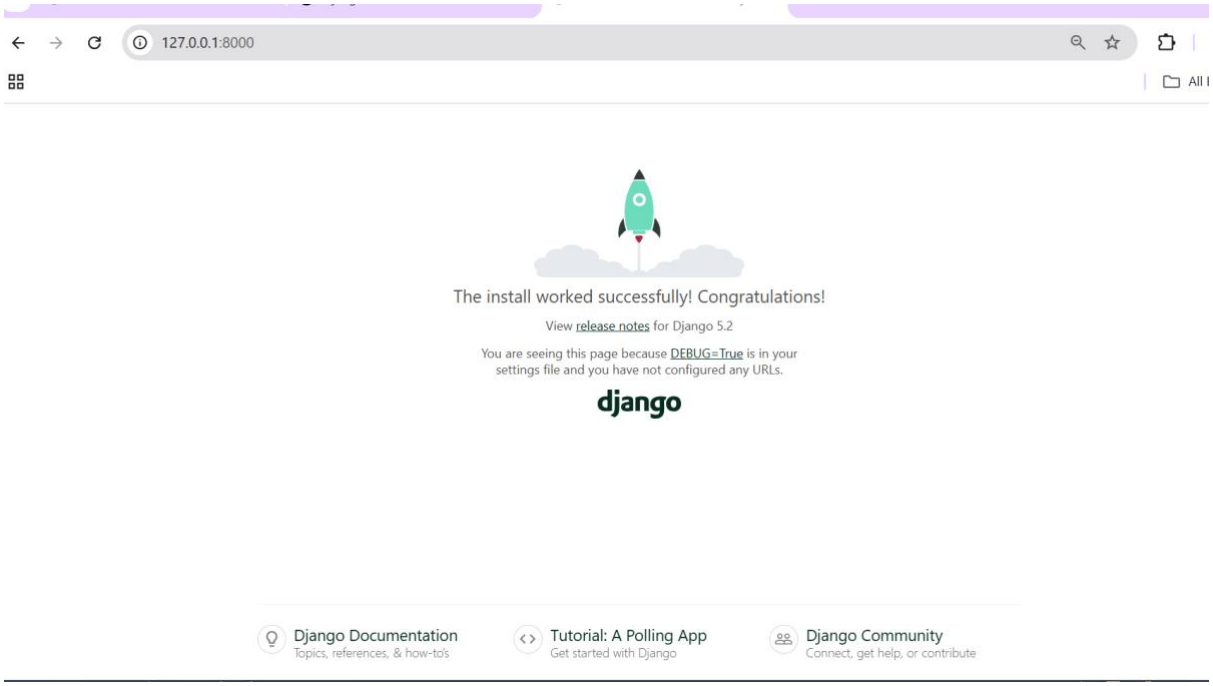
```
python manage.py runserver
```

   - This starts the development server at `http://127.0.0.1:8000/`

Output:

- A Django project structure with folders: `mysite`, `manage.py`, etc.
- Running server message confirming successful setup.






```
(venv) PS C:\Users\DELL-7490\Desktop\New folder\my_django_project> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin,
 auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
April 19, 2025 - 11:01:06
Django version 5.2, using settings 'mysite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

WARNING: This is a development server. Do not use it in a production setting. Use a production WSGI or ASGI server inste
ad.
For more information on production servers see: https://docs.djangoproject.com/en/5.2/howto/deployment/
```

✓ WEEK 13 – Django App with Navigation Menu (Home, Dashboard, Contact Us)

🎯 Goal

Build a Django web application that displays:

- A **navigation bar** using Bootstrap
- Three pages: **Home**, **Dashboard**, and **Contact Us**

🔧 Step-by-Step Instructions

✓ STEP 1: Check Python Version

🔑 In Command Prompt (CMD) or Terminal:

```
bash
```

```
python --version
```

Make sure it shows Python **3.x**

✅ STEP 2: Install Virtual Environment

```
bash
```

```
py -m pip install virtualenv
```

What is `virtualenv`?

`virtualenv` is a **predefined tool** that allows you to create isolated Python environments. This helps in managing dependencies for different projects without causing conflicts between them. It creates a separate directory with its own Python binary and can install packages independently of the global Python environment.

💡 What's happening?

- `py`: Runs Python.
- `-m pip install virtualenv`: Uses Python's package installer (`pip`) to **install a tool** called `virtualenv`.

👉 `virtualenv` lets you **create an isolated space** to work on your project, so it doesn't interfere with other projects or your system Python.

✅ STEP 3: Create Virtual Environment

```
bash
```

```
py -m venv second
```

📁 This will create a folder `second/` with your isolated environment.

💡 What's happening?

- `venv` is a built-in module in Python that helps create **virtual environments**.

- `second`: The name of your virtual environment folder.

💼 This creates a folder named `second/`, which contains:

- A **copy of Python**
- Its own `pip` installer
- Scripts to activate the environment

🎯 So you now have a **clean room** just for this project, without messing up other Python projects on your system.

✅ STEP 4: Activate Virtual Environment

```
bash

.\second\Scripts\activate
```

🔄 If it works, you'll see your terminal change to:

```
scss

(second) C:\Users\YourName>
```

💡 What's happening?

- You're **entering** the virtual environment.
- It changes your terminal so that any Python or `pip` command will run **inside the virtual space** (`second`) and not in the global/system Python.

🎯 Once activated:

- You'll see something like this in the terminal:
  `(second) C:\Users\YourName\ProjectFolder>`
- Now you can safely install Django and other packages just for this project.

✅ STEP 5: Install Django

```
bash

pip install django
```

✅ STEP 6: Create a Django Project

```
bash

django-admin startproject program13
```

This creates:

```
markdown

program13/
├── manage.py
└── program13/
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    ├── asgi.py
    └── wsgi.py
```

✅ STEP 7: Go Into the Project Folder

```
bash

cd program13
```

Now you're in the same folder as `manage.py`

✅ STEP 8: Create a `templates` Folder

📁 Inside this `program13/` folder (where `manage.py` is), create a folder named:

```
nginx

templates
```

Inside `templates`, create these 3 files:

- `home.html`
- `dashboard.html`
- `contactUs.html`

✅ STEP 9: Add HTML Code

📄 In each file, paste this:

home.html

```
html

<!DOCTYPE html>
<html>
<head>
    <title>Home</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.c
ss" rel="stylesheet">
</head>
<body>
    <h1 class="text-center">Welcome to CMRIT</h1>
    <nav class="navbar navbar-expand-lg bg-body-tertiary">
        <div class="container-fluid">
            <a class="navbar-brand" href="#">CMRIT</a>
            <div class="collapse navbar-collapse" id="navbarNav">
                <ul class="navbar-nav">
                    <li><a class="nav-link" href="{% url 'home'
%}">Home</a></li>
                    <li><a class="nav-link" href="{% url 'dashboard'
%}">Dashboard</a></li>
                    <li><a class="nav-link" href="{% url 'contactUs'
%}">Contact Us</a></li>
                </ul>
            </div>
        </div>
    </nav>
</body>
</html>
```

dashboard.html

```
html

<!DOCTYPE html>
<html>
<head><title>Dashboard</title></head>
<body>
    <h1>Welcome to Dashboard</h1>
    <a href="{% url 'home' %}">Back to home</a>
</body>
</html>
```

contactUs.html

```
html

<!DOCTYPE html>
<html>
<head><title>Contact Us</title></head>
<body>
    <h1>CMR Institute of Technology, Medchal, Hyderabad</h1>
    <a href="{% url 'home' %}">Back to home</a>
</body>
</html>
```

✅ STEP 10: Edit `settings.py` to Link Templates

📄 Open: `program13/program13/settings.py`

1. At the top, add:

```
python

import os
```

2. Find the `TEMPLATES` section and change:

```
python

'DIRS': [],
```

➡ To:

```
python

'DIRS': [BASE_DIR / 'program13' / 'templates'],
```

✅ STEP 11: Create `index.py` for Views

📄 Inside the **inner** `program13/` folder (the one with `urls.py`), create a file called `index.py`
Paste this:

```
python

from django.shortcuts import render

def home(request):
    return render(request, 'home.html')

def dashboard(request):
    return render(request, 'dashboard.html')

def contactUs(request):
    return render(request, 'contactUs.html')
```

✅ STEP 12: Update `urls.py`

📄 Open: `program13/program13/urls.py`
Replace everything with this:

```
python

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

```
urlpatterns = [
    path('', index.home, name='home'),
    path('dashboard/', index.dashboard, name='dashboard'),
    path('contactUs/', index.contactUs, name='contactUs'),
]
```

✅ STEP 13: Run the Server

📍 Make sure you're still in the folder where `manage.py` is (outer `program13`):

```
bash

py manage.py runserver
```

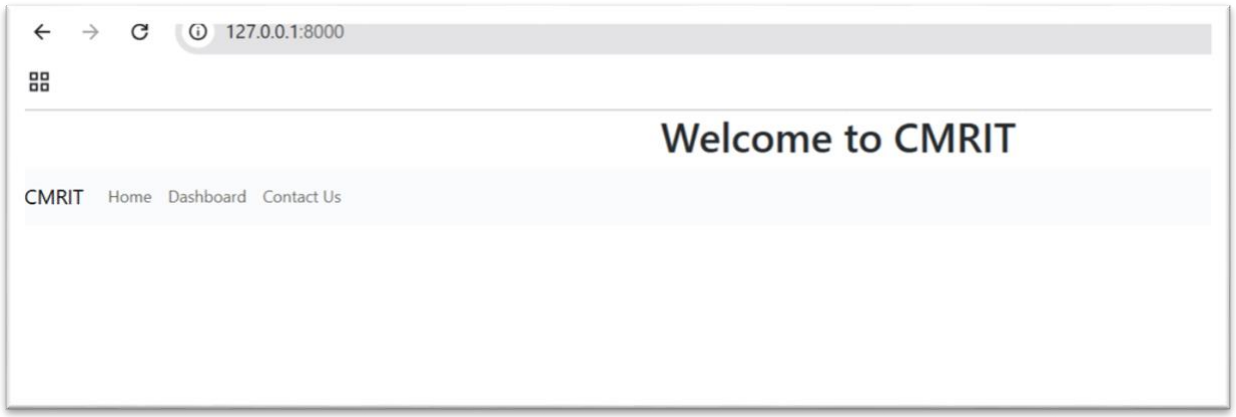
You'll see:

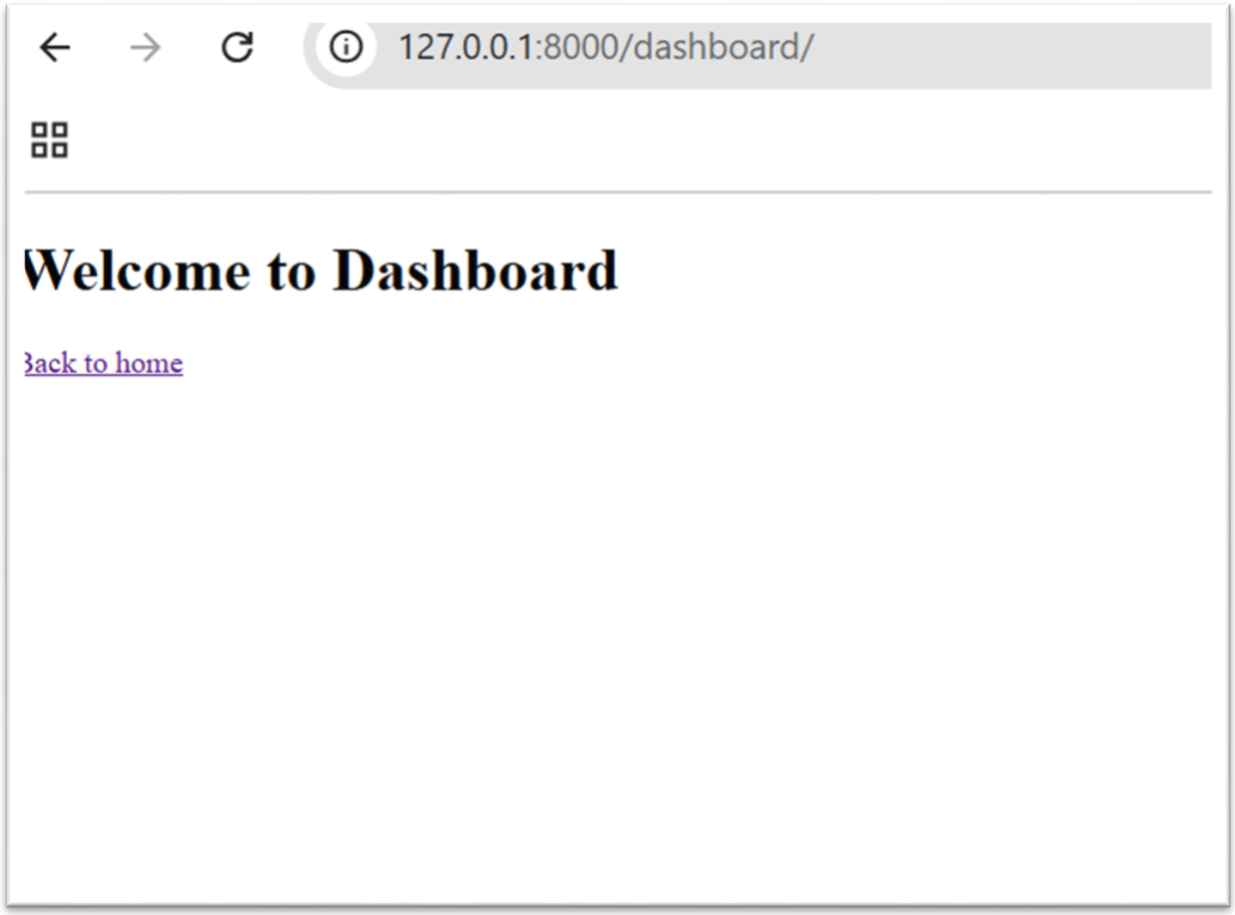
```
nginx

Starting development server at http://127.0.0.1:8000/
```

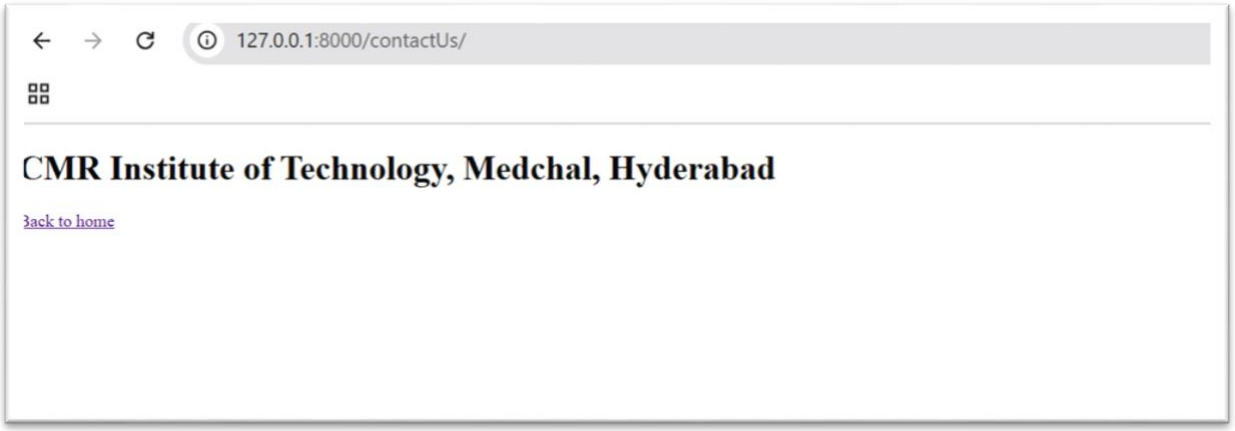
✅ STEP 14: Open in Browser

Visit these in your browser:

Page	URL
Home	http://127.0.0.1:8000/
Dashboard	http://127.0.0.1:8000/dashboard/
Contact Us	http://127.0.0.1:8000/contactUs/




127.0.0.1:8000/dashboard/
Welcome to Dashboard
Back to home


127.0.0.1:8000/contactUs/
CMR Institute of Technology, Medchal, Hyderabad
Back to home

✅ WEEK 14 LAB ACTIVITY: Django CRUD Operations (Create, Read, Update, Delete)

We'll build a simple **Student Management App** where you can:

- Add a new student
- View all students
- Edit a student's record
- Delete a student

✅ STEP 1: Create a Django Project

📍 **Where**: Inside your activated virtual environment in Command Prompt or VS Code Terminal
📌 **Command**:

```
bash

django-admin startproject crud_project
cd crud_project
```

📌 **Why**: This sets up the main project folder with settings, URLs, etc.

📤 **Output**:

- Folder `crud_project/` created
- Files: `manage.py`, and a subfolder also named `crud_project` with `settings.py`, `urls.py`, etc.

✅ STEP 2: Start a Django App

📍 **Where**: Inside the `crud_project` directory
📌 **Command**:

```
bash

python manage.py startapp studentapp
```

📌 **Why**: This creates an app called `studentapp` which will contain the models, views, templates, etc.

📤 **Output**:

- New folder `studentapp/` created with `models.py`, `views.py`, `urls.py` (you will create this manually), etc.

✅ STEP 3: Add `studentapp` to Installed Apps

📍 **Where**: In `crud_project/settings.py`, under `INSTALLED_APPS`

🛠 **Change this**:

```
python

INSTALLED_APPS = [
    ...
    'studentapp',
]
```

📌 **Why**: To tell Django to include this app when running the project.

✅ STEP 4: Create a Model

📍 **Where**: In `studentapp/models.py`

📌 **Code**:

```
python

from django.db import models

class Student(models.Model):
    name = models.CharField(max_length=100)
    email = models.EmailField()
    address = models.TextField()

    def __str__(self):
        return self.name
```

📌 **Why**: This defines the structure of your database table.

✅ STEP 5: Register Model in Admin

📍 **Where**: In `studentapp/admin.py`

📌 **Code**:

```
python

from django.contrib import admin
```

```
from .models import Student

admin.site.register(Student)
```

📌 **Why**: So you can manage student data from Django Admin.

✅ STEP 6: Apply Migrations

📍 **Where**: In terminal

📌 **Commands**:

```
bash

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

📌 **Why**: This creates and updates your database schema.

✅ STEP 7: Create Superuser

📍 **Where**: In terminal

📌 **Command**:

```
bash

python manage.py createsuperuser
```

Follow the prompts (username, password, etc.)

📌 **Why**: To access Django's admin panel to manage data.

✅ STEP 8: Set Up URLs

In `crud_project/urls.py`, include the app URLs:

```
python

from django.contrib import admin
from django.urls import path, include

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

Then create a new file: `studentapp/urls.py`

```
python

from django.urls import path
from . import views

urlpatterns = [
    path('', views.student_list, name='student_list'),
    path('add/', views.add_student, name='add_student'),
    path('edit/<int:id>/', views.edit_student, name='edit_student'),
    path('delete/<int:id>/', views.delete_student, name='delete_student'),
]
```

📌 **Why**: This routes user requests to correct views.

✅ STEP 9: Create Views

🔑 **Where**: In `studentapp/views.py`

```
python

from django.shortcuts import render, redirect
from .models import Student

def student_list(request):
    students = Student.objects.all()
    return render(request, 'student_list.html', {'students': students})

def add_student(request):
    if request.method == 'POST':
        name = request.POST['name']
        email = request.POST['email']
        address = request.POST['address']
        Student.objects.create(name=name, email=email, address=address)
        return redirect('/')
    return render(request, 'add_student.html')

def edit_student(request, id):
    student = Student.objects.get(id=id)
    if request.method == 'POST':
        student.name = request.POST['name']
        student.email = request.POST['email']
        student.address = request.POST['address']
        student.save()
        return redirect('/')
    return render(request, 'edit_student.html', {'student': student})

def delete_student(request, id):
    student = Student.objects.get(id=id)
    student.delete()
    return redirect('/')
```

✅ STEP 10: Create Templates

🔑 **Where**: Create a `templates` folder inside `studentapp/`
Make sure to configure it in `settings.py`:

```
python

import os
TEMPLATES = [
    {
        ...
        'DIRS': [os.path.join(BASE_DIR, 'studentapp', 'templates')],
        ...
    },
]
```

Create these HTML files inside `studentapp/templates/`:

🔻 *student_list.html*

```
html

<h2>Student List</h2>
<a href="{% url 'add_student' %}">Add Student</a>
<ul>
  {% for student in students %}
    <li>
      {{ student.name }} - {{ student.email }}
      <a href="{% url 'edit_student' student.id %}">Edit</a>
      <a href="{% url 'delete_student' student.id %}">Delete</a>
    </li>
  {% endfor %}
</ul>
```

🔻 *add_student.html*

```
html

<h2>Add Student</h2>
<form method="post">
  {% csrf_token %}
  <input type="text" name="name" placeholder="Name"><br>
  <input type="email" name="email" placeholder="Email"><br>
  <textarea name="address" placeholder="Address"></textarea><br>
  <button type="submit">Add</button>
</form>
```

🔻 *edit_student.html*

```
html

<h2>Edit Student</h2>
<form method="post">
  {% csrf_token %}
  <input type="text" name="name" value="{{ student.name }}"><br>
  <input type="email" name="email" value="{{ student.email }}"><br>
  <textarea name="address">{{ student.address }}</textarea><br>
  <button type="submit">Update</button>
</form>
```

✅ STEP 11: Run the Server

🔑 **Where**: Terminal
📌 **Command**:

```
bash
```

```
python manage.py runserver
```

Open browser: http://127.0.0.1:8000

📤 **Output**: A working CRUD app for managing students!