



# Channabasaveshwara Institute of Technology

(Affiliated to VTU, Belgaum & Approved by AICTE, New Delhi)  
(NAAC Accredited & ISO 9001:2015 Certified Institution)  
NH 206 (B.H. Road), Gubbi, Tumkur – 572216. Karnataka.



## Department of Information Science & Engineering

### Full Stack Development - 21CS62

#### Lab Manual

#### 6<sup>th</sup> Semester

#### Prepared By:

Mr. Praveen Kumar K C  
Assistant Professor  
Dept. of ISE  
CIT, Gubbi

<b>FULLSTACK DEVELOPMENT</b>			
Course Code	21CS62	CIE Marks	50
Teaching Hours/Week (L:T:P: S)	3:0:2:0	SEE Marks	50
Total Hours of Pedagogy	40 T + 20 P	Total Marks	100
Credits	04	Exam Hours	03
<b>Course Learning Objectives:</b>			
CLO 1.Explain the use of learning full stack web development.			
CLO 2.Make use of rapid application development in the design of responsive web pages.			
CLO 3.Illustrate Models, Views and Templates with their connectivity in Django for full stack web development.			
CLO 4.Demonstrate the use of state management and admin interfaces automation in Django.			
CLO 5.Design and implement Django apps containing dynamic pages with SQL databases.			
<b>Teaching-Learning Process (General Instructions)</b>			
These are sample Strategies, which teachers can use to accelerate the attainment of the various course outcomes.			
<ol style="list-style-type: none"> <li>1. Lecturer method (L) does not mean only traditional lecture method, but different type of teaching methods may be adopted to develop the outcomes.</li> <li>2. Show Video/animation films to explain functioning of various concepts.</li> <li>3. Encourage collaborative (Group Learning) Learning in the class.</li> <li>4. Ask at least three HOT (Higher order Thinking) questions in the class, which promotes critical thinking.</li> <li>5. Adopt Problem Based Learning (PBL), which fosters students' Analytical skills, develop thinking skills such as the ability to evaluate, generalize, and analyze information rather than simply recall it.</li> <li>6. Topics will be introduced in a multiple representation.</li> <li>7. Show the different ways to solve the same problem and encourage the students to come up with their own creative ways to solve them.</li> <li>8. Discuss how every concept can be applied to the real world - and when that's possible, it helps improve the students' understanding.</li> </ol>			
<b>Module-1: MVC based Web Designing</b>			
Web framework, MVC Design Pattern, Django Evolution, Views, Mapping URL to Views, Working of Django URL Confs and Loose Coupling, Errors in Django, Wild Card patterns in URLS.			
<b>Textbook 1: Chapter 1 and Chapter 3</b>			
<b>Laboratory Component:</b>			
<ol style="list-style-type: none"> <li>1. Installation of Python, Django and Visual Studio code editors can be demonstrated.</li> <li>2. Creation of virtual environment, Django project and App should be demonstrated</li> <li>3. Develop a Django app that displays current date and time in server</li> <li>4. Develop a Django app that displays date and time four hours ahead and four hours before as an offset of current date and time in server.</li> </ol>			
<b>Teaching-Learning Process</b>	<ol style="list-style-type: none"> <li>1. Demonstration using Visual Studio Code</li> <li>2. PPT/Prezi Presentation for Architecture and Design Patterns</li> <li>3. Live coding of all concepts with simple examples</li> </ol>		
<b>Module-2: Django Templates and Models</b>			
Template System Basics, Using Django Template System, Basic Template Tags and Filters, MVT Development Pattern, Template Loading, Template Inheritance, MVT Development Pattern.			

Configuring Databases, Defining and Implementing Models, Basic Data Access, Adding Model String Representations, Inserting/Updating data, Selecting and deleting objects, Schema Evolution

**Textbook 1: Chapter 4 and Chapter 5**

**Laboratory Component:**

1. Develop a simple Django app that displays an unordered list of fruits and ordered list of selected students for an event
2. Develop a layout.html with a suitable header (containing navigation menu) and footer with copyright and developer information. Inherit this layout.html and create 3 additional pages: contact us, About Us and Home page of any website.
3. Develop a Django app that performs student registration to a course. It should also display list of students registered for any selected course. Create students and course as models with enrolment as ManyToMany field.

**Teaching-Learning Process**

1. Demonstration using Visual Studio Code
2. PPT/Prezi Presentation for Architecture and Design Patterns
3. Live coding of all concepts with simple examples
4. Case Study: Apply concepts learnt for an Online Ticket Booking System

**Module-3: Django Admin Interfaces and Model Forms**

Activating Admin Interfaces, Using Admin Interfaces, Customizing Admin Interfaces, Reasons to use Admin Interfaces.

Form Processing, Creating Feedback forms, Form submissions, custom validation, creating Model Forms, URLConf Ticks, Including Other URLConfs.

**Textbook 1: Chapters 6, 7 and 8**

**Laboratory Component:**

1. For student and course models created in Lab experiment for Module2, register admin interfaces, perform migrations and illustrate data entry through admin forms.
2. Develop a Model form for student that contains his topic chosen for project, languages used and duration with a model called project.

**Teaching-Learning Process**

1. Demonstration using Visual Studio Code
2. PPT/Prezi Presentation for Architecture and Design Patterns
3. Live coding of all concepts with simple examples

**Module-4: Generic Views and Django State Persistence**

Using Generic Views, Generic Views of Objects, Extending Generic Views of objects, Extending Generic Views.

MIME Types, Generating Non-HTML contents like CSV and PDF, Syndication Feed Framework, Sitemap framework, Cookies, Sessions, Users and Authentication.

**Textbook 1: Chapters 9, 11 and 12**

**Laboratory Component:**

1. For students enrolment developed in Module 2, create a generic class view which displays list of students and detailview that displays student details for any selected student in the list.
2. Develop example Django app that performs CSV and PDF generation for any models created in previous laboratory component.

**Teaching-Learning Process**

1. Demonstration using Visual Studio Code
2. PPT/Prezi Presentation for Architecture and Design Patterns

	<ol style="list-style-type: none"> <li>3. Live coding of all concepts with simple examples</li> <li>4. Project Work: Implement all concepts learnt for Student Admission Management.</li> </ol>
<b>Module-5: jQuery and AJAX Integration in Django</b>	
Ajax Solution, Java Script, XMLHttpRequest and Response, HTML, CSS, JSON, iFrames, Settings of Java Script in Django, jQuery and Basic AJAX, jQuery AJAX Facilities, Using jQuery UI Autocomplete in Django	
<b>Textbook 2: Chapters 1, 2 and 7.</b>	
<b>Laboratory Component:</b>	
<ol style="list-style-type: none"> <li>1. Develop a registration page for student enrolment as done in Module 2 but without page refresh using AJAX.</li> <li>2. Develop a search application in Django using AJAX that displays courses enrolled by a student being searched.</li> </ol>	
<b>Teaching-Learning Process</b>	<ol style="list-style-type: none"> <li>1. Demonstration using Visual Studio Code</li> <li>2. PPT/Prezi Presentation for Architecture and Design Patterns</li> <li>3. Live coding of all concepts with simple examples</li> <li>4. Case Study: Apply the use of AJAX and jQuery for development of EMI calculator.</li> </ol>
<b>Course outcome (Course Skill Set)</b>	
At the end of the course the student will be able to:	
CO 1. Understand the working of MVT based full stack web development with Django.	
CO 2. Designing of Models and Forms for rapid development of web pages.	
CO 3. Analyze the role of Template Inheritance and Generic views for developing full stack web applications.	
CO 4. Apply the Django framework libraries to render nonHTML contents like CSV and PDF.	
CO 5. Perform jQuery based AJAX integration to Django Apps to build responsive full stack web applications,	
<b>Assessment Details (both CIE and SEE)</b>	
The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 40% of the maximum marks (20 marks). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/course if the student secures not less than 35% (18 Marks out of 50) in the semester-end examination (SEE), and a minimum of 40% (40 marks out of 100) in the sum total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together	
<b>Continuous Internal Evaluation:</b>	
Three Unit Tests each of <b>20 Marks (duration 01 hour)</b>	
<ol style="list-style-type: none"> <li>1. First test at the end of 5<sup>th</sup> week of the semester</li> <li>2. Second test at the end of the 10<sup>th</sup> week of the semester</li> <li>3. Third test at the end of the 15<sup>th</sup> week of the semester</li> </ol>	
Two assignments each of <b>10 Marks</b>	
<ol style="list-style-type: none"> <li>4. First assignment at the end of 4<sup>th</sup> week of the semester</li> <li>5. Second assignment at the end of 9<sup>th</sup> week of the semester</li> </ol>	

Practical Sessions need to be assessed by appropriate rubrics and viva-voce method. This will contribute to **20 marks**.

- Rubrics for each Experiment taken average for all Lab components – 15 Marks.
- Viva-Voce– 5 Marks (more emphasized on demonstration topics)

The sum of three tests, two assignments, and practical sessions will be out of 100 marks and will be **scaled down to 50 marks**

(to have a less stressed CIE, the portion of the syllabus should not be common /repeated for any of the methods of the CIE. Each method of CIE should have a different syllabus portion of the course).

**CIE methods /question paper has to be designed to attain the different levels of Bloom’s taxonomy as per the outcome defined for the course.**

#### **Semester End Examination:**

Theory SEE will be conducted by University as per the scheduled timetable, with common question papers for the subject (**duration 03 hours**)

1. The question paper will have ten questions. Each question is set for 20 marks. Marks scored shall be proportionally reduced to 50 marks
2. There will be 2 questions from each module. Each of the two questions under a module (with a maximum of 3 sub-questions), **should have a mix of topics** under that module.

The students have to answer 5 full questions, selecting one full question from each module

#### **Suggested Learning Resources:**

##### **Textbooks**

1. Adrian Holovaty, Jacob Kaplan Moss, The Definitive Guide to Django: Web Development Done Right, Second Edition, Springer-Verlag Berlin and Heidelberg GmbH & Co. KG Publishers, 2009
2. Jonathan Hayward, Django Java Script Integration: AJAX and jQuery, First Edition, Pack Publishing, 2011

##### **Reference Books**

1. Aidas Bendroraitis, Jake Kronika, Django 3 Web Development Cookbook, Fourth Edition, Packt Publishing, 2020
2. William Vincent, Django for Beginners: Build websites with Python and Django, First Edition, Amazon Digital Services, 2018
3. Antonio Mele, Django3 by Example, 3<sup>rd</sup> Edition, Pack Publishers, 2020
4. Arun Ravindran, Django Design Patterns and Best Practices, 2<sup>nd</sup> Edition, Pack Publishers, 2020.
5. Julia Elman, Mark Lavin, Light weight Django, David A. Bell, 1<sup>st</sup> Edition, Oreily Publications, 2014

##### **Weblinks and Video Lectures (e-Resources):**

1. MVT architecture with Django: <https://freevideolectures.com/course/3700/django-tutorials>
2. Using Python in Django: <https://www.youtube.com/watch?v=2BqoLiMT3Ao>
3. Model Forms with Django: <https://www.youtube.com/watch?v=gMM1rtTwKxE>
4. Real time Interactions in Django: <https://www.youtube.com/watch?v=3gHmfoeZ45k>
5. AJAX with Django for beginners: <https://www.youtube.com/watch?v=3VaKNyjlxAU>

##### **Activity Based Learning (Suggested Activities in Class)/ Practical Based learning**

1. Real world problem solving - applying the Django framework concepts and its integration with AJAX to develop any shopping website with admin and user dashboards.

***Short Preamble on Full Stack Web Development:***

*Website development is a way to make people aware of the services and/or products they are offering, understand why the products are relevant and even necessary for them to buy or use, and highlight the striking qualities that set it apart from competitors. Other than commercial reasons, a website is also needed for quick and dynamic information delivery for any domain. Development of a well-designed, informative, responsive and dynamic website is need of the hour from any computer science and related engineering graduates. Hence, they need to be augmented with skills to use technology and framework which can help them to develop elegant websites. Full Stack developers are in need by many companies, who knows and can develop all pieces of web application (Front End, Back End and business logic). MVT based development with Django is the cutting-edge framework for Full Stack Web Development. Python has become an easier language to use for many applications. Django based framework in Python helps a web developer to utilize framework and develop rapidly responsive and secure web applications.*

**CONTENTS**

<b>MODULE</b>	<b>SUB - COMPONENT</b>	<b>PAGE NO.</b>
<b>Laboratory Component 1</b>	<b>1.1</b>	<b>2</b>
	<b>1.2</b>	<b>3</b>
	<b>1.3</b>	<b>8</b>
	<b>1.4</b>	<b>8</b>
<b>Laboratory Component 2</b>	<b>2.1</b>	<b>13</b>
	<b>2.2</b>	<b>16</b>
	<b>2.3</b>	<b>22</b>
<b>Laboratory Component 3</b>	<b>3.1</b>	<b>36</b>
	<b>3.2</b>	<b>39</b>
<b>Laboratory Component 4</b>	<b>4.1</b>	<b>45</b>
	<b>4.2</b>	<b>50</b>
<b>Laboratory Component 5</b>	<b>5.1</b>	<b>53</b>
	<b>5.2</b>	<b>57</b>

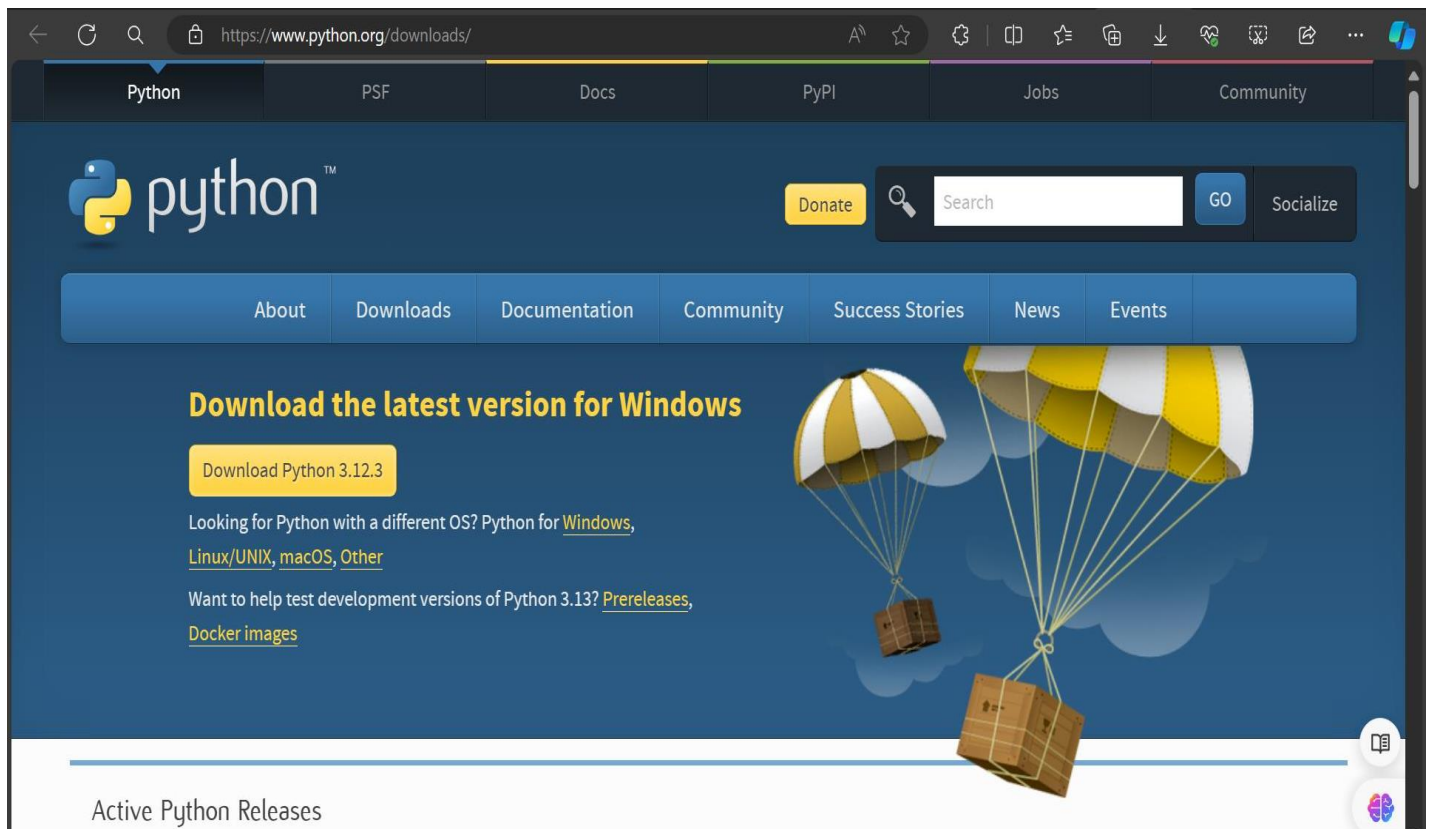
## Laboratory Component - 1:

1. Installation of Python, Django and Visual Studio code editors can be demonstrated.
2. Creation of virtual environment, Django project and App should be demonstrated
3. Develop a Django app that displays current date and time in server
4. Develop a Django app that displays date and time four hours ahead and four hours before as an offset of current date and time in server.

### 1.1 Installation

#### a) Python:

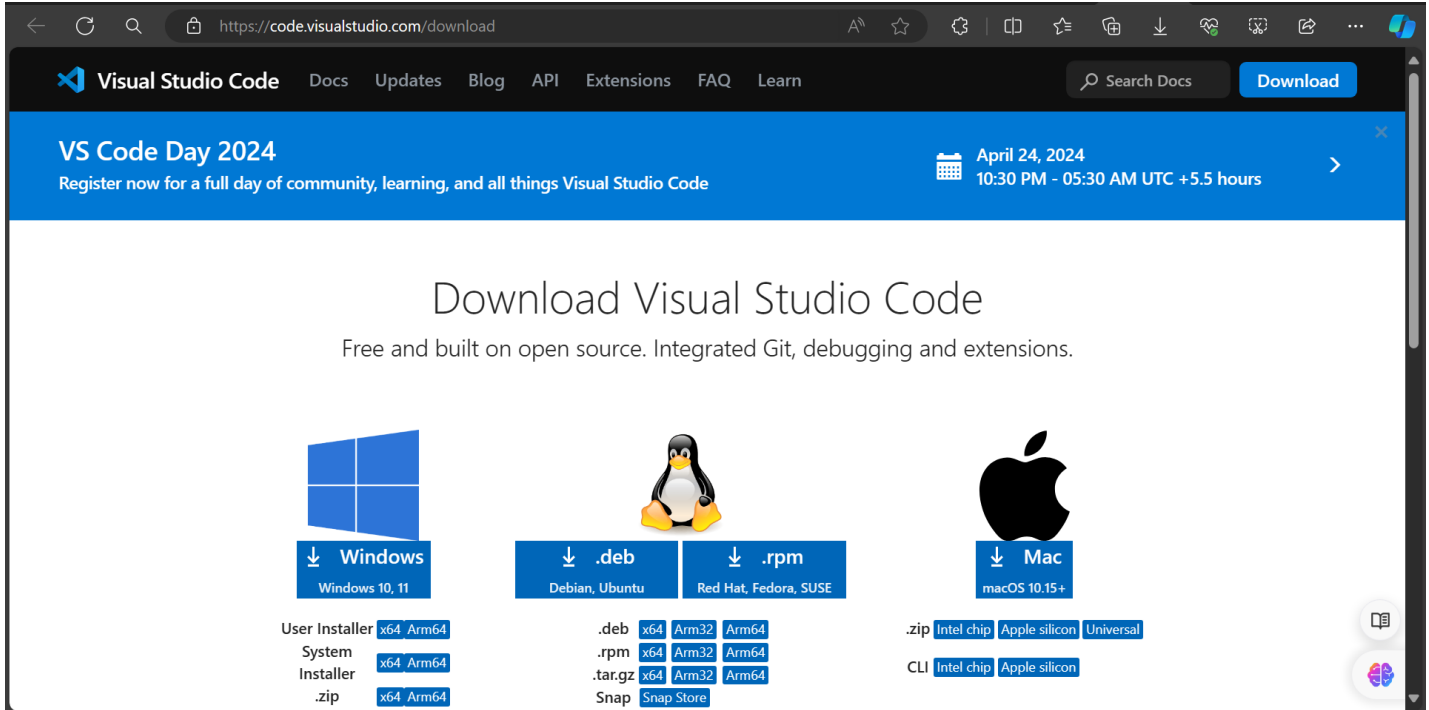
1. Download the latest Python installer from <https://www.python.org/downloads/>. (Python 3.11.5 preferred).
2. Run the installer and follow the on-screen instructions. Ensure "**Add Python to PATH**" is checked for easy access from the command line.
3. Open a command prompt or terminal and type **python --version** to verify installation.





**b) Visual Studio Code:**

1. Download and install Visual Studio Code from <https://code.visualstudio.com/download>.
2. Install the Python extension for code completion and debugging within VS Code.



**1.2 Virtual Environment and Project Setup:**

1. On your file system, create a folder, such as django\_lab.
2. In that folder, use the following command (as appropriate to your computer) to create a virtual environment named myenv based on your current interpreter.

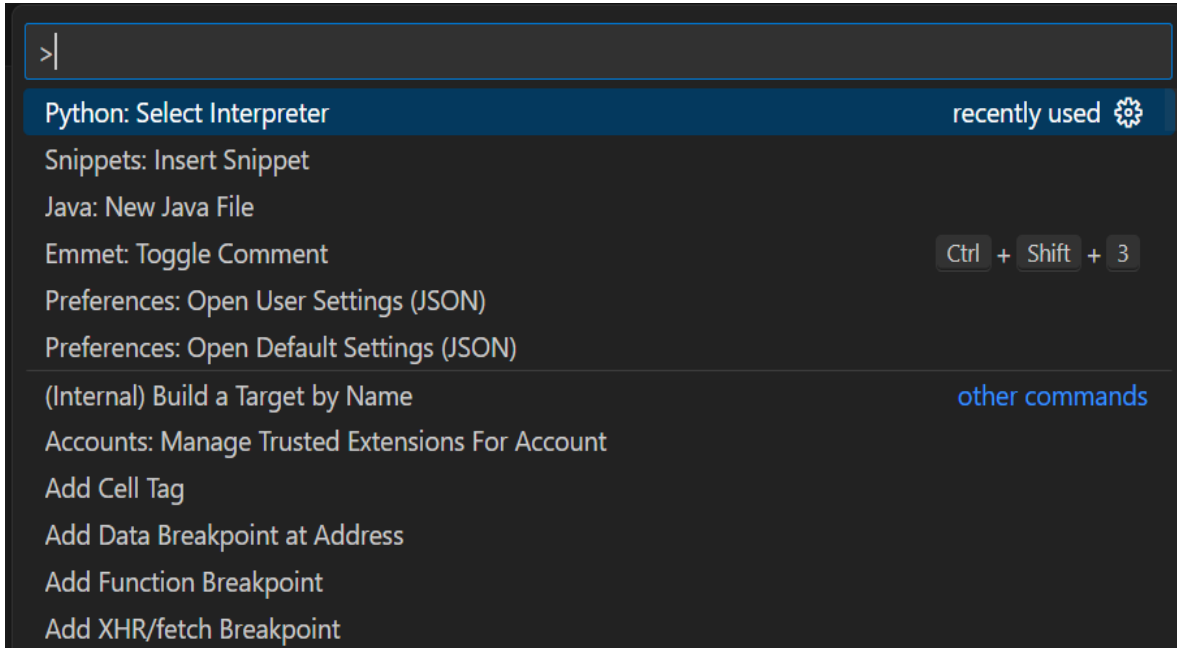
```
# Linux
sudo apt-get install python3-venv # If needed
python3 -m venv myenv
source myenv/bin/activate

# macOS
python3 -m venv myenv
source myenv/bin/activate

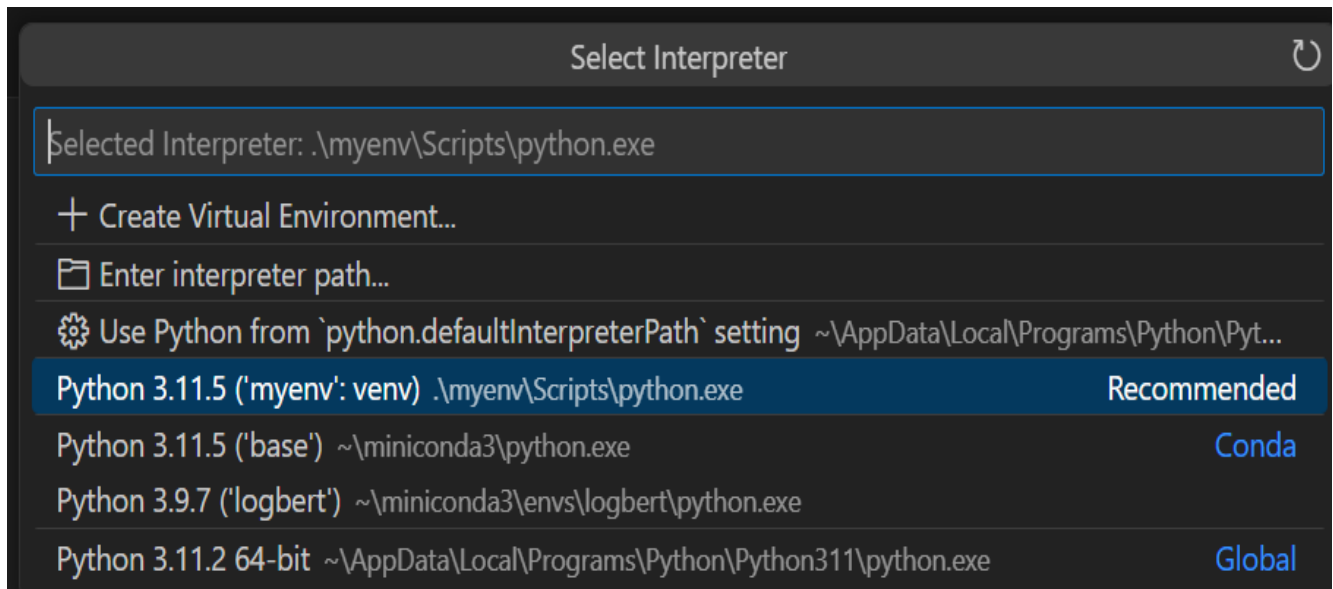
# Windows
py -3 -m venv myenv
myenv\scripts\activate
```

3. Open the project folder in VS Code by running `code .`, or by running VS Code and using the File > Open Folder command.

- In VS Code, open the Command Palette (View > Command Palette or (Ctrl+Shift+P)). Then select the Python: Select Interpreter command:

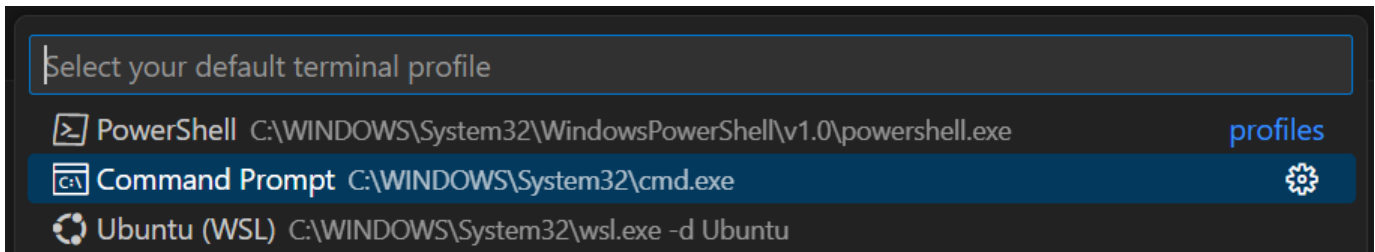
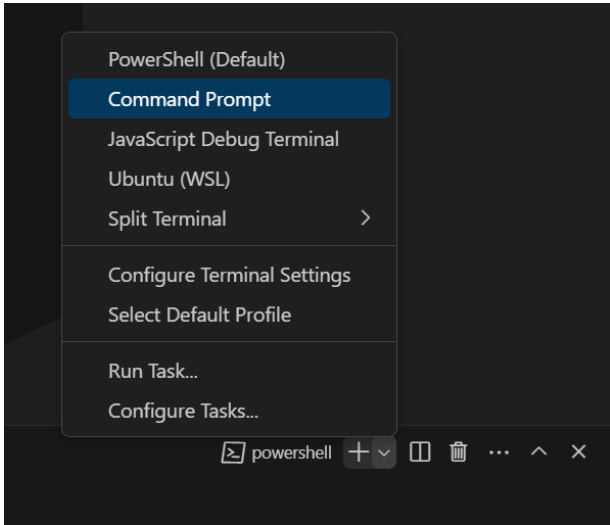


- The command presents a list of available interpreters that VS Code can locate automatically (your list will vary; if you don't see the desired interpreter, see Configuring Python environments). From the list, select the virtual environment in your project folder that starts with myenv:



- Run Terminal: Create New Terminal (Ctrl+Shift+`) from the Command Palette, which creates a terminal and automatically activates the virtual environment by running its activation script.

*Note: On Windows, if your default terminal type is PowerShell, you may see an error that it cannot run activate.ps1 because running scripts is disabled on the system. The error provides a link for information on how to allow scripts. Otherwise, use Terminal: Select Default Profile to set "Command Prompt" or "Git Bash" as your default instead.*



- Update pip in the virtual environment by running the following command in the VS Code Terminal:  
**python -m pip install --upgrade pip**
- Install Django in the virtual environment by running the following command in the VS Code Terminal:  
**python -m pip install django**

*You now have a self-contained environment ready for writing Django code. VS Code activates the environment automatically when you use Terminal: Create New Terminal (Ctrl+Shift+`). If you open a separate command prompt or terminal, activate the environment by running source myenv/bin/activate (Linux/macOS) or myenv\Scripts\Activate.ps1 (Windows). You know the environment is activated when the command prompt shows (myenv) at the beginning.*

## Create and run a minimal Django app

In Django terminology, a "Django project" is composed of several site-level configuration files, along with one or more "apps" that you deploy to a web host to create a full web application. A Django project can contain multiple apps, each of which typically has an independent function in the project, and the same app can be in multiple Django projects. An app, for its part, is just a Python package that follows certain conventions that Django expects.

To create a minimal Django app, then, it's necessary to first create the Django project to serve as the container for the app, then create the app itself. For both purposes, you use the Django administrative utility, `django-admin`, which is installed when you install the Django package.

### Create the Django project

1. In the VS Code Terminal where your virtual environment is activated, run the following command:

```
django-admin startproject myproject .
```

This `startproject` command assumes (by use of `.` at the end) that the current folder is your project folder, and creates the following within it:

- `manage.py`: The Django command-line administrative utility for the project. You run administrative commands for the project using `python manage.py <command> [options]`.
  - A subfolder named `myproject`, which contains the following files:
    - `__init__.py`: an empty file that tells Python that this folder is a Python package.
    - `asgi.py`: an entry point for [ASGI-compatible](#) web servers to serve your project. You typically leave this file as-is as it provides the hooks for production web servers.
    - `settings.py`: contains settings for Django project, which you modify in the course of developing a web app.
    - `urls.py`: contains a table of contents for the Django project, which you also modify in the course of development.
    - `wsgi.py`: an entry point for WSGI-compatible web servers to serve your project. You typically leave this file as-is as it provides the hooks for production web servers.
2. Create an empty development database by running the following command:

```
python manage.py migrate
```

When you run the server the first time, it creates a default SQLite database in the file `db.sqlite3` that is intended for development purposes, but can be used in production for low-volume web apps.

To verify the Django project, make sure your virtual environment is activated, then start Django's development server using the command `python manage.py runserver`. The server runs on the default port 8000, and you see output like the following output in the terminal window:

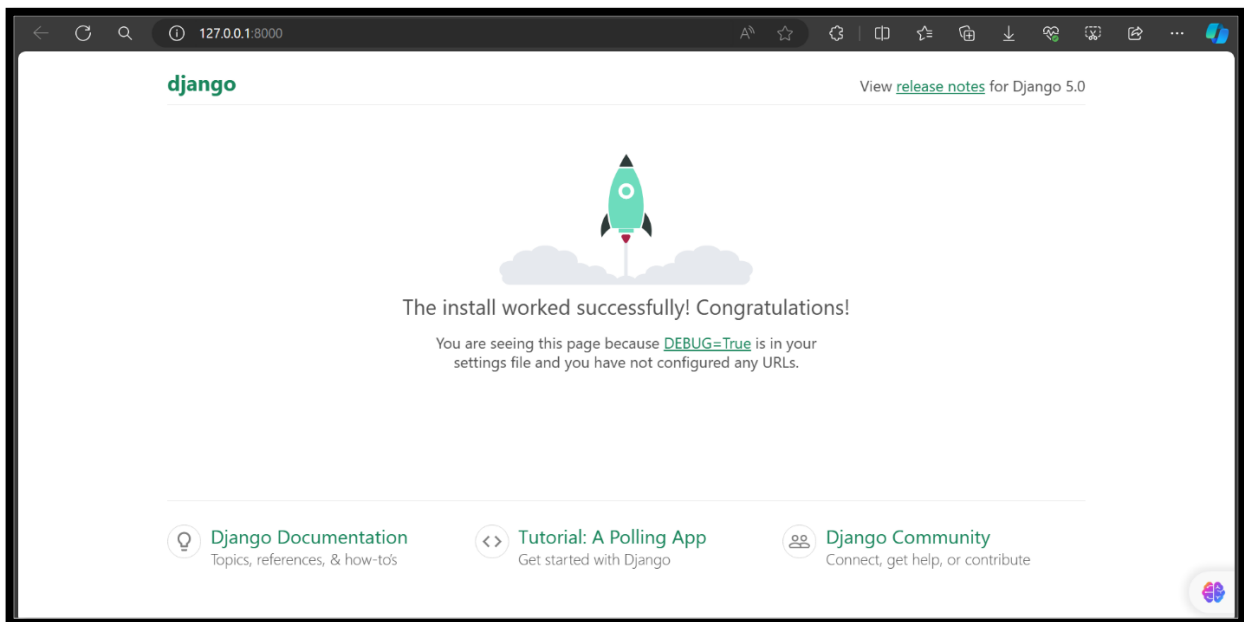
```
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
April 20, 2024 - 13:59:07
Django version 5.0.3, using settings 'myproject.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Django's built-in web server is intended *only* for local development purposes. When you deploy to a web host, however, Django uses the host's web server instead. The `wsgi.py` and `asgi.py` modules in the Django project take care of hooking into the production servers.

If you want to use a different port than the default 8000, specify the port number on the command line, such as `python manage.py runserver 5000`.

3. **Ctrl+click** the `http://127.0.0.1:8000/` URL in the terminal output window to open your default browser to that address. If Django is installed correctly and the project is valid, you see the default page shown below. The VS Code terminal output window also shows the server log.



4. When you're done, close the browser window and stop the server in VS Code using **Ctrl+C** as indicated in the terminal output window.

### 1.3 - 1.4 Create a Django app

1. In the VS Code Terminal with your virtual environment activated, run the administrative utility's `startapp` command in your project folder (where `manage.py` resides):

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

The command creates a folder called `myapp` that contains a number of code files and one subfolder. Of these, you frequently work with `views.py` (that contains the functions that define pages in your web app) and `models.py` (that contains classes defining your data objects). The `migrations` folder is used by Django's administrative utility to manage database versions as discussed later in this tutorial. There are also the files `apps.py` (app configuration), `admin.py` (for creating an administrative interface), and `tests.py` (for creating tests), which are not covered here.

2. Modify `myapp/views.py` to match the following code

#### #views.py

```
import pytz
```

```
from datetime import datetime, timedelta
```

```
from django.shortcuts import render
```

```
#Using pytz library
```

```
def current_datetime(request):
```

```
    # Get the standard UTC time
```

```
    utc = pytz.utc
```

```
    # Get the time zone of the specified location (IST - Indian Standard Time)
```

```
    ist = pytz.timezone('Asia/Kolkata')
```

```
    # Get the current time in UTC and IST
```

```
    datetime_utc = datetime.now(utc)
```

```
    datetime_ist = datetime.now(ist)
```

```
    # Format the date and time
```

```
    formatted_utc = datetime_utc.strftime('%Y-%m-%d %H:%M:%S %Z %z')
```

```
    formatted_ist = datetime_ist.strftime('%Y-%m-%d %H:%M:%S %Z %z')
```

```
    # Pass the formatted date and time to the template
```

```
    context = {
```

```
        'utc_time': formatted_utc,
```

```
        'ist_time': formatted_ist
```

```
    }
```

```
    # Render the template with the context
```

```
    return render(request, 'myapp/current_datetime.html', context)
```

```
#Using pytz library
def date_time_offset(request):
    # Get the current date and time on the server
    current_datetime = datetime.now()

    # Calculate the date and time four hours ahead and four hours before
    datetime_ahead = current_datetime + timedelta(hours=4)
    datetime_before = current_datetime - timedelta(hours=4)

    # Format the date and time strings
    formatted_current_datetime = current_datetime.strftime('%Y-%m-%d %H:%M:%S')
    formatted_datetime_ahead = datetime_ahead.strftime('%Y-%m-%d %H:%M:%S')
    formatted_datetime_before = datetime_before.strftime('%Y-%m-%d %H:%M:%S')

    # Pass the formatted date and time strings to the template
    context = {
        'current_datetime': formatted_current_datetime,
        'datetime_ahead': formatted_datetime_ahead,
        'datetime_before': formatted_datetime_before
    }

    # Render the template with the context
    return render(request, 'myapp/date_time_offset.html', context)
```

3. You need to install the pytz library.

```
pip install pytz
```

4. Create a file, myapp/urls.py, with the contents below. The urls.py file is where you specify patterns to route different URLs to their appropriate views.

```
#urls.py (myapp/urls.py)
from django.urls import path
from myapp import views

urlpatterns = [
    path("current_datetime/", views.current_datetime, name="current_datetime"),
    path("date_time_offset/", views.date_time_offset, name="date_time_offset"),
]
```

5. The myproject folder also contains a urls.py file, which is where URL routing is actually handled. Open myproject/urls.py and modify it to match the following code (you can retain the instructive comments if you like). This code pulls in the app's myapp/urls.py using **django.urls.include**, which keeps the app's routes contained within the app. This separation is helpful when a project contains multiple apps.

#### #urls.py (myproject/urls.py)

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

urlpatterns = [
    path("", include("myapp.urls")),
    path("admin/", admin.site.urls),
]
```

6. In the `myproject/settings.py` file, locate the `INSTALLED_APPS` list and add the following entry, which makes sure the project knows about the app so it can handle templating:

```
'myapp',
```

7. Inside the myapp folder, create a folder named templates, and then another subfolder named myapp to match the app name (this two-tiered folder structure is typical Django convention).

In the templates/myapp folder, create a file named `current_datetime.html` with the contents below.

#### #current\_datetime.html (myapp/templates/myapp/current\_datetime.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Current Time</title>
</head>
<body>
  <h1>Current Time</h1>
  <p>UTC Time: {{ utc_time }}</p>
  <p>Indian Standard Time (IST): {{ ist_time }}</p>
</body>
</html>
```



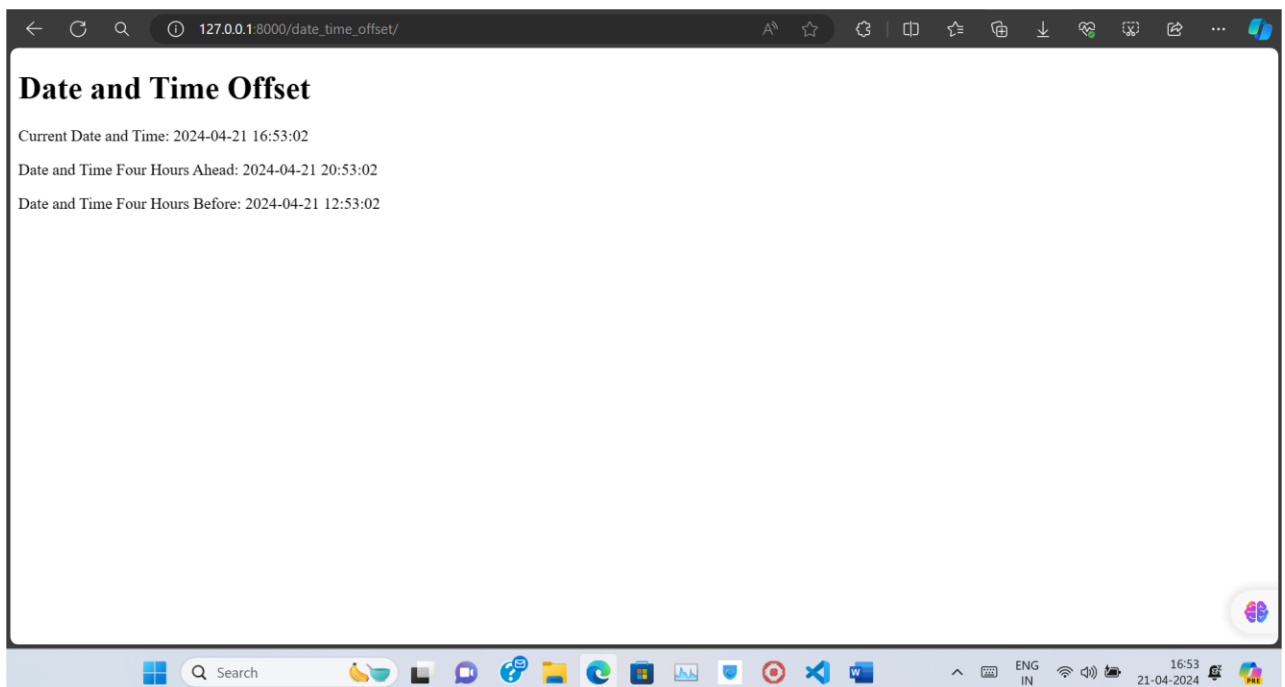
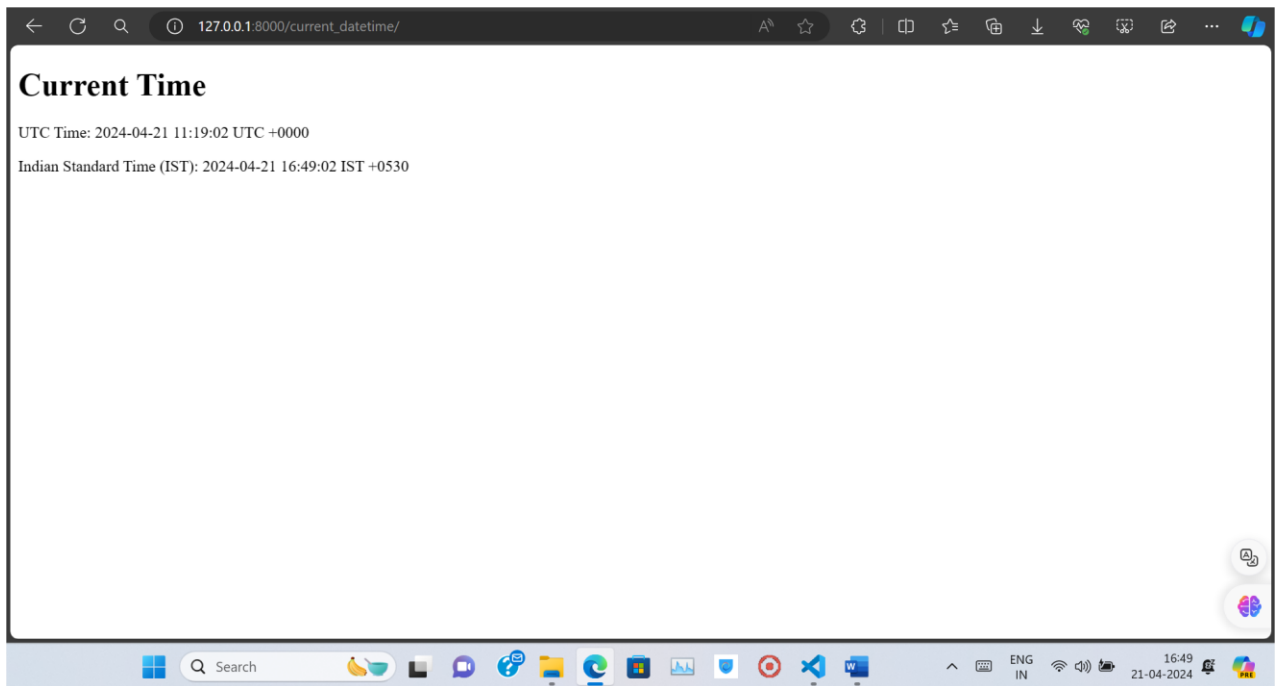
In the templates/myapp folder, create a file named date\_time\_offset.html with the contents below.

**#date\_time\_offset.html (myapp/templates/myapp/ date\_time\_offset.html)**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Date and Time Offset</title>
</head>
<body>
  <h1>Date and Time Offset</h1>
  <p>Current Date and Time: {{ current_datetime }}</p>
  <p>Date and Time Four Hours Ahead: {{ datetime_ahead }}</p>
  <p>Date and Time Four Hours Before: {{ datetime_before }}</p>
</body>
</html>
```

8. Save all modified files.
9. In the VS Code Terminal, again with the virtual environment activated, run the development server with **python manage.py runserver** and open a browser to **http://127.0.0.1:8000/**
10. In the url box of the browser, navigate to **http://127.0.0.1:8000/current\_datetime** to view the current date time. Similarly, navigate to **http://127.0.0.1:8000/date\_time\_offset** to view the offset time.

## OUTPUT:



## Laboratory Component - 2:

1. Develop a simple Django app that displays an unordered list of fruits and ordered list of selected students for an event
2. Develop a layout.html with a suitable header (containing navigation menu) and footer with copyright and developer information. Inherit this layout.html and create 3 additional pages: contact us, About Us and Home page of any website.
3. Develop a Django app that performs student registration to a course. It should also display list of students registered for any selected course. Create students and course as models with enrolment as ManyToMany field.

### 2.1 Create a Django app

1. In the VS Code Terminal with your virtual environment activated, run the administrative utility's `startapp` command in your myproject folder (where `manage.py` resides):

```
python manage.py startapp fruits_and_students
```

2. Modify `fruits_and_students/views.py` to match the following code.

```
#views.py
from django.shortcuts import render

def fruits_and_students(request):
    print(request.build_absolute_uri())
    fruits = ['Apple', 'Banana', 'Orange', 'Grapes']
    students = ['Alice', 'Bob', 'Charlie', 'David']
    return render(request, 'fruits_and_students/fruits_and_students.html', {'fruits': fruits, 'students':
students})
```

3. Create a file, `fruits_and_students/urls.py`, with the contents below. The `urls.py` file is where you specify patterns to route different URLs to their appropriate views.

```
#urls.py (fruits_and_students/urls.py)
from django.urls import path
from .views import fruits_and_students

urlpatterns = [
    path("", fruits_and_students, name='fruits_and_students'),
]
```

4. The myproject folder also contains a urls.py file, which is where URL routing is actually handled. Keep in mind the myproject/urls.py will be used to handle all of the laboratory component apps' that will be built. Just add the path url routing line of code to the urlpatterns list every time in the already existing code.

#### #urls.py (myproject/urls.py)

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

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

5. In the `myproject/settings.py` file, locate the `INSTALLED_APPS` list and add the following entry, which makes sure the project knows about the app so it can handle templating:

```
'fruits_and_students',
```

6. Inside the fruits\_and\_students folder, create a folder named templates, and then another subfolder named fruits\_and\_students to match the app name (this two-tiered folder structure is typical Django convention).

In the templates/fruits\_and\_students folder, create a file named fruits\_and\_students.html with the contents below.

#### #fruits\_and\_students.html (fruits\_and\_students/templates/fruits\_and\_students/fruits\_and\_students.html)

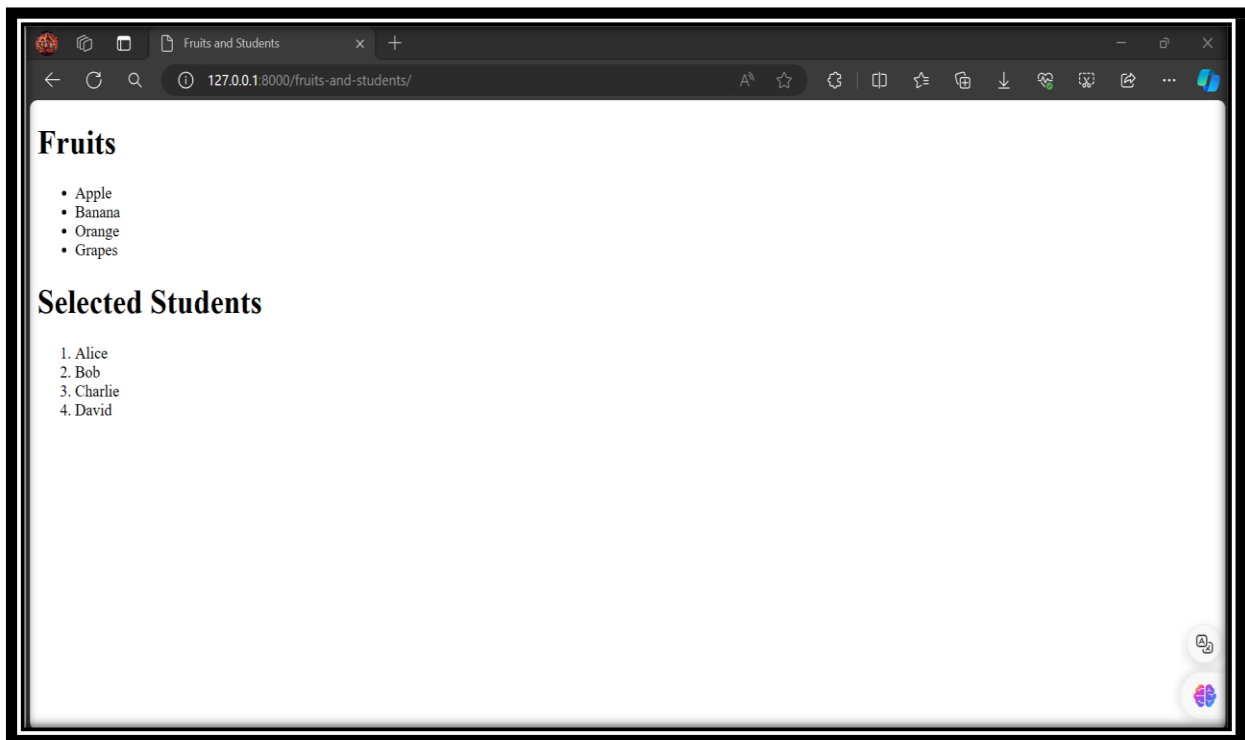
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Fruits and Students</title>
</head>
<body>
  <h1>Fruits</h1>
  <ul>
    {% for fruit in fruits %}
```

```
<li>{{ fruit }}</li>
{% endfor %}
</ul>

<h1>Selected Students</h1>
<ol>
  {% for student in students %}
  <li>{{ student }}</li>
  {% endfor %}
</ol>
</body>
</html>
```

7. Save all modified files.
8. In the VS Code Terminal, again with the virtual environment activated, run the development server with **python manage.py runserver** and open a browser to **http://127.0.0.1:8000/**
9. In the url box of the browser, navigate to **http://127.0.0.1:8000/fruits\_and\_students** to view the output.

### OUTPUT:



## 2.2 Create a Django app

1. In the VS Code Terminal with your virtual environment activated, run the administrative utility's `startapp` command in your myproject folder (where `manage.py` resides):

```
python manage.py startapp website_pages
```

2. Modify `website_pages/views.py` to match the following code.

```
#views.py
from django.shortcuts import render
def home(request):
    return render(request, 'website_pages/home.html')
def about_us(request):
    return render(request, 'website_pages/about_us.html')
def contact_us(request):
    return render(request, 'website_pages/contact_us.html')
```

3. Create a file, `website_pages/urls.py`, with the contents below. The `urls.py` file is where you specify patterns to route different URLs to their appropriate views.

```
#urls.py (website_pages /urls.py)
from django.urls import path
from website_pages import views
urlpatterns = [
    path('home/', views.home, name='home'),
    path('about_us/', views.about_us, name='about_us'),
    path('contact_us/', views.contact_us, name='contact_us'),
]
```

4. The myproject folder also contains a `urls.py` file, which is where URL routing is actually handled.

```
#urls.py (myproject/urls.py)

from django.contrib import admin
from django.urls import path, include
urlpatterns = [
    path("", include("myapp.urls")),
    path("admin/", admin.site.urls),
    path('fruits_and_students/', include('fruits_and_students.urls')),
    path("", include('website_pages.urls')),
]
```

5. In the `myproject/settings.py` file, locate the `INSTALLED_APPS` list and add the following entry, which makes sure the project knows about the app so it can handle templating:

```
'website_pages',
```

6. Inside the `website_pages` folder, create a folder named `templates`. Inside the `templates` folder, create a file named `layout.html`.

### #templates/layout.html (The CSS is optional)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>{% block title %}My Website{% endblock %}</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 0;
      padding: 0;
    }
    header {
      background-color: #333;
      color: #fff;
      padding: 10px;
    }
    nav ul {
      list-style-type: none;
      padding: 0;
    }
    nav ul li {
      display: inline;
      margin-right: 20px;
    }
    nav ul li a {
      text-decoration: none;
      color: #fff;
    }
    main {
      padding: 20px;
    }
  </style>
</head>
<body>
  <header>
    <h1>My Website</h1>
  </header>
  <nav>
    <ul>
      <li><a href="#">Home</a></li>
      <li><a href="#">About</a></li>
      <li><a href="#">Contact</a></li>
    </ul>
  </nav>
  <main>
    <h2>Welcome to My Website</h2>
  </main>
</body>
</html>
```

```
        footer {
            background-color: #333;
            color: #fff;
            text-align: center;
            padding: 10px;
            position: fixed;
            bottom: 0;
            width: 100%;
        }
    </style>
</head>

<body>

    <header>
        <nav>
            <ul>
                <li><a href="{% url 'home' %}">Home</a></li>
                <li><a href="{% url 'about_us' %}">About Us</a></li>
                <li><a href="{% url 'contact_us' %}">Contact Us</a></li>
            </ul>
        </nav>
    </header>

    <main>
        {% block content %}
        {% endblock %}
    </main>

    <footer>
        <p>&copy; 2024 My Website. All rights reserved. </p>
        <p>Developed by CIT</p>
    </footer>

</body>
</html>
```

7. Inside the templates folder, create another subfolder named `website_pages` to match the app name (this two-tiered folder structure is typical Django convention).

In the `templates/website_pages` folder, create files named `home.html`, `about_us.html`, and `contact_us.html` with the contents below.



**#home.html (website\_pages/templates/website\_pages/home.html)**

```
{% extends 'layout.html' % }
{% block title % }Home{% endblock % }
{% block content % }
    <h1>Welcome to Cambridge Institute of Technology</h1>
    <p>Empowering students with a blend of knowledge and innovation.</p>
    <p>Nestled in the bustling city of Bengaluru, our campus is a hub of academic excellence and cutting-edge research.</p>

    <h2>Discover Your Potential</h2>
    <ul>
        <li><strong>Undergraduate Programs:</strong> Dive into our diverse range of engineering courses designed to fuel your passion and drive innovation.</li>
        <li><strong>Postgraduate Programs:</strong> Advance your expertise with our specialized master's programs and embrace leadership in technology.</li>
    </ul>

    <p>Join our vibrant community where ideas flourish and inventions come to life in our state-of-the-art labs and research centers.</p>

    <p>Benefit from our strong industry ties and placement programs that open doors to exciting career opportunities.</p>
{% endblock % }
```

**#about\_us.html (website\_pages/templates/website\_pages/about\_us.html)**

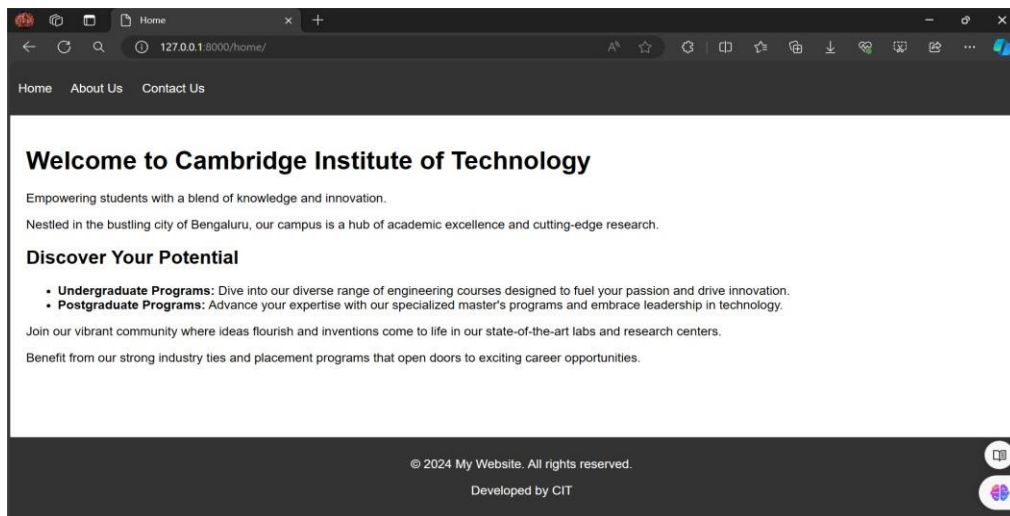
```
{% extends 'layout.html' % }
{% block title % }About Us{% endblock % }
{% block content % }
    <h1>Our Legacy</h1>
    <p>Founded on the principles of quality education and societal contribution, we've been at the forefront of technological education for over four decades.</p>
    <h1>Vision and Mission</h1>
    <p>Our vision is to be a beacon of knowledge that lights the way for aspiring minds, and our mission is to nurture innovative thinkers who will shape the future of technology.</p>
    <h1>Campus Life</h1>
    <p>Experience a dynamic campus life enriched with cultural activities, technical clubs, and community service initiatives that foster holistic development.</p>
{% endblock % }
```

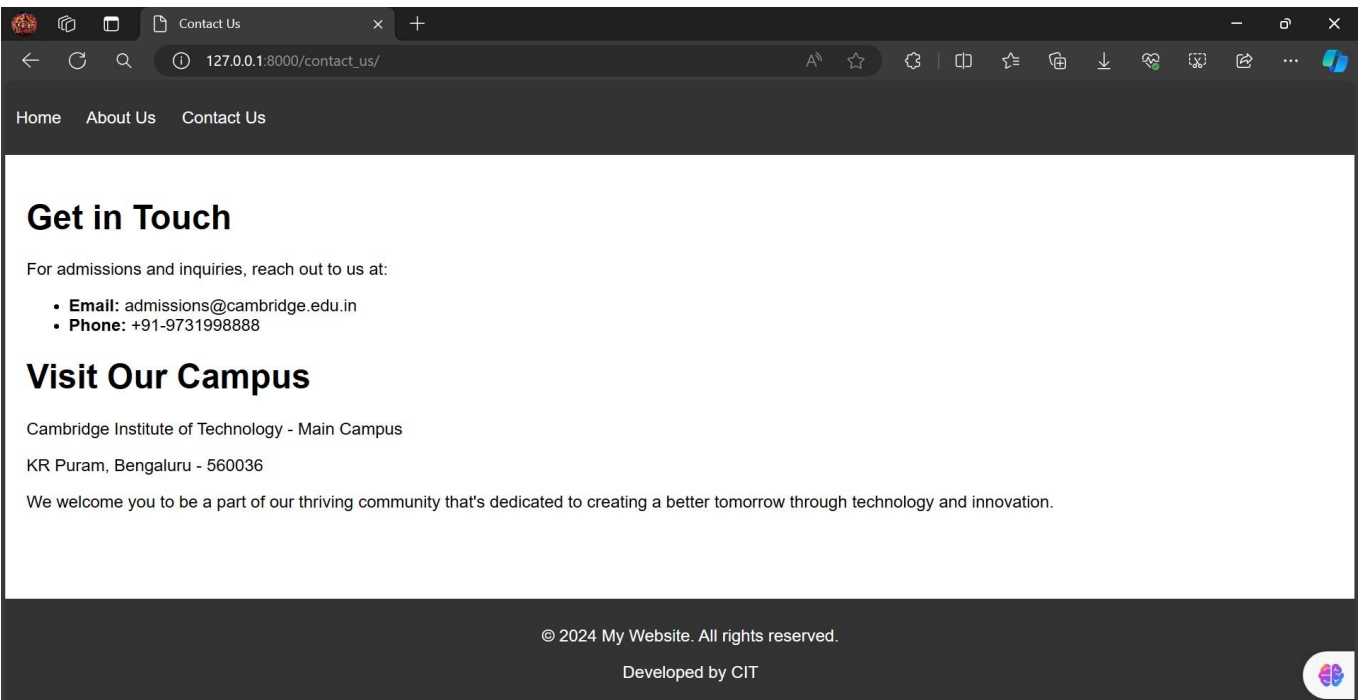
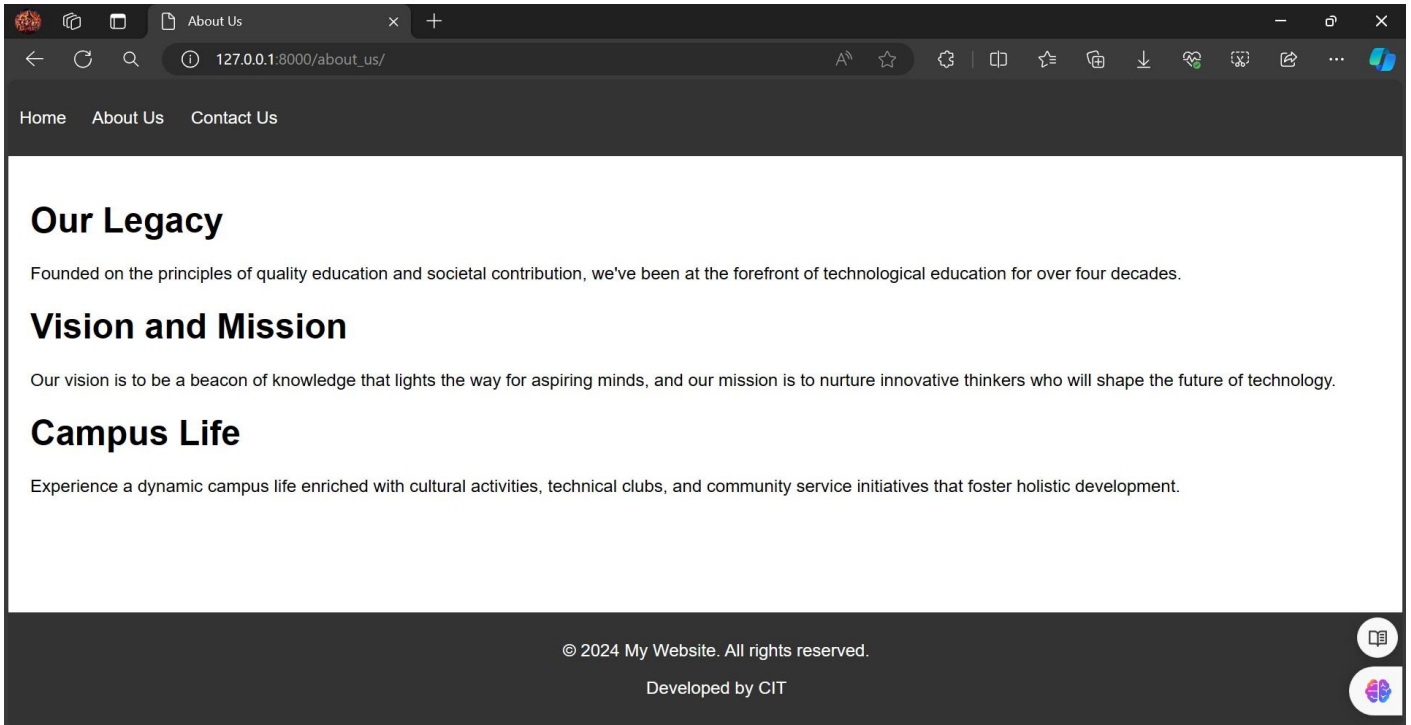
#contact\_us.html (website\_pages/templates/website\_pages/contact\_us.html)

```
{% extends 'layout.html' % }
{% block title % }Contact Us{% endblock % }
{% block content % }
    <h1>Get in Touch</h1>
    <p>For admissions and inquiries, reach out to us at:</p>
    <ul>
        <li><strong>Email:</strong> admissions@cambridge.edu.in</li>
        <li><strong>Phone:</strong> +91-9731998888</li>
    </ul>
    <h1>Visit Our Campus</h1>
    <p>Cambridge Institute of Technology - Main Campus</p>
    <p>KR Puram, Bengaluru - 560036</p>
    <p>We welcome you to be a part of our thriving community that's dedicated to creating a better
tomorrow through technology and innovation.</p>
{% endblock % }
```

8. Save all modified files.
9. In the VS Code Terminal, again with the virtual environment activated, run the development server with **python manage.py runserver** and open a browser to **http://127.0.0.1:8000/**
10. In the url box of the browser, navigate to **http://127.0.0.1:8000/home** to view the output. Similarly, you can navigate through by clicking on the navigation menu. Also check out **http://127.0.0.1:8000/about\_us** and **http://127.0.0.1:8000/contact\_us**

**OUTPUT:**





### 2.3 Create a Django app

1. In the VS Code Terminal with your virtual environment activated, run the administrative utility's `startapp` command in your myproject folder (where `manage.py` resides):

```
python manage.py startapp course_registration
```

2. Modify `course_registration/views.py` to match the following code.

#### **#views.py**

```
from .forms import StudentForm, CourseForm
from .models import Student, Course
from django.shortcuts import render, redirect, get_object_or_404

def add_student(request):
    if request.method == 'POST':
        form = StudentForm(request.POST)
        if form.is_valid():
            form.save()
            # Redirect to a view that lists all students
            return redirect('student_list')
    else:
        form = StudentForm()
        return render(request, 'course_registration/add_student.html', {'form': form})

def add_course(request):
    if request.method == 'POST':
        form = CourseForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('course_registration')
    else:
        form = CourseForm()
        return render(request, 'course_registration/add_course.html', {'form': form})

def register_student(request):
    if request.method == 'POST':
        student_name = request.POST.get('student_name')
        course_id = request.POST.get('course_id')
        # Validate that both student_name and course_id are provided
        if not student_name or not course_id:
```

```

    return render(request, 'course_registration/register_student.html', {'courses': Course.objects.all(),
'error_message': 'Please provide both student name and select a course.})
    try:
        # Retrieve the course based on course_id or return 404 if not found
        course = get_object_or_404(Course, pk=course_id)
        # Check if the student already exists in the database
        student = Student.objects.filter(name=student_name).first()
        if not student:
            # If the student does not exist, return an error message
            return render(request, 'course_registration/register_student.html', {'courses':
Course.objects.all(), 'error_message': 'Student does not exist in the database.})
        # Add the student to the course
        course.students.add(student)
        return redirect('course_registration')
    except Course.DoesNotExist:
        return render(request, 'course_registration/register_student.html', {'courses': Course.objects.all(),
'error_message': 'Invalid course ID. Please select a valid course.})

    # If not a POST request, render the registration form with all courses
    return render(request, 'course_registration/register_student.html', {'courses': Course.objects.all()})

def course_registration(request):
    courses = Course.objects.all()
    return render(request, 'course_registration/course_registration.html', {'courses': courses})

def students_list(request, course_id):
    # Retrieve the course based on course_id or return 404 if not found
    course = get_object_or_404(Course, course_id=course_id)
    # Retrieve the students associated with the course
    students = course.students.all()
    return render(request, 'course_registration/students_list.html', {'course': course, 'students': students})

```

3. Create a file, `course_registration/urls.py`, with the contents below. The `urls.py` file is where you specify patterns to route different URLs to their appropriate views.

```

#urls.py (course_registration/urls.py)
from django.urls import path
from . import views

urlpatterns = [
    path('add_student/', views.add_student, name='add_student'),
    path('add_course/', views.add_course, name='add_course'),

```

```
path('register/', views.register_student, name='register_student'),
path('courses/', views.course_registration, name='course_registration'),
path('students_list/<int:course_id>/',
     views.students_list, name='students_list'),
]
```

4. The myproject folder also contains a urls.py file, which is where URL routing is actually handled.  
**#urls.py (myproject/urls.py)**

```
from django.contrib import admin
from django.urls import path, include
urlpatterns = [
    path("", include("myapp.urls")),
    path("admin/", admin.site.urls),
    path('fruits_and_students/', include('fruits_and_students.urls')),
    path("", include('website_pages.urls')),
    path('registration/', include('course_registration.urls')),
]
```

5. Modify `course_registration/models.py` to match the following code.

```
#models.py
from django.db import models

class Student(models.Model):
    name = models.CharField(max_length=100,unique=True)
    date_of_birth = models.DateField(
        default='1900-01-01', blank=False, null=False) # Set a default date
    email = models.EmailField(
        default='example@example.com', blank=False, null=False) # Set a default email
    def __str__(self):
        return self.name

class Course(models.Model):
    name = models.CharField(max_length=100,unique=True)
    students = models.ManyToManyField(Student, related_name='courses')
    course_id = models.IntegerField(default=0,unique=True)
    def __str__(self):
        return self.name
```

6. Modify `course_registration/forms.py` to match the following code.

**#forms.py**

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

```
class CourseForm(forms.ModelForm):
    class Meta:
        model = Course
        fields = ['name', 'course_id']
```

```
class StudentForm(forms.ModelForm):
    class Meta:
        model = Student
        fields = ['name', 'date_of_birth', 'email']
```

7. In the `myproject/settings.py` file, locate the `INSTALLED_APPS` list and add the following entry, which makes sure the project knows about the app so it can handle templating:

```
'course_registration',
```

8. Create a templates folder, create inside templates folder, a subfolder named `course_registration` to match the app name (this two-tiered folder structure is typical Django convention). In the `templates/course_registration` folder, create files named `add_student.html`, `add_course.html`, `register_student.html`, `course_registration.html`, and `students_list.html` with the contents below.

**#templates/course\_registration/add\_student.html (The CSS is optional)**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Add Student</title>
</head>
<body>
  <h1>Add Student</h1>
  <form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Submit</button>
  </form>
</body>
</html>
```

#templates/course\_registration/add\_course.html (The CSS is optional)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Add Course</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background-color: #f7f7f7;
      margin: 0;
      padding: 20px;
    }
    h1 {
      color: #333;
    }
    form {
      background-color: #fff;
      padding: 20px;
      border-radius: 5px;
      box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    }
    label {
      display: block;
      margin-bottom: 10px;
      font-weight: bold;
    }
    input[type="text"], input[type="number"] {
      width: 100%;
      padding: 10px;
      margin-bottom: 20px;
      border: 1px solid #ccc;
      border-radius: 5px;
      box-sizing: border-box;
      font-size: 16px;
    }
    button[type="submit"] {
      background-color: #007bff;
      color: #fff;
      padding: 10px 20px;
      border: none;
```



```
        border-radius: 5px;
        cursor: pointer;
        font-size: 16px;
    }
    button[type="submit"]:hover {
        background-color: #0056b3;
    }
</style>
</head>
<body>
    <h1>Add Course</h1>
    <form method="POST">
        {% csrf_token %}
        <label for="course_name">Course Name:</label>
        <input type="text" id="course_name" name="name" required>
        <label for="course_id">Course ID:</label>
        <input type="number" id="course_id" name="course_id" required>
        <button type="submit">Add Course</button>
    </form>
</body>
</html>
```

### #templates/course\_registration/register\_student.html (The CSS is optional)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Register Student</title>
<style>
    body {
        font-family: Arial, sans-serif;
        background-color: #f8f9fa;
        margin: 0;
        padding: 0;
    }
    .container {
        max-width: 600px;
        margin: 50px auto;
        background-color: #fff;
        padding: 20px;
```

```
border-radius: 5px;
box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}
.form-group {
margin-bottom: 20px;
}
.form-control {
width: 100%;
padding: 10px;
border: 1px solid #ccc;
border-radius: 5px;
box-sizing: border-box;
}
.btn {
padding: 10px 20px;
background-color: #007bff;
color: #fff;
border: none;
border-radius: 5px;
cursor: pointer;
}
.btn-primary {
background-color: #007bff;
}
</style>
</head>
<body>
<div class="container">
<h1>Register Student to Course</h1>
<form method="POST" class="form">
{% csrf_token %}
<div class="form-group">
<label for="student_name">Student Name:</label>
<input type="text" id="student_name" name="student_name" class="form-control" required>
</div>
<div class="form-group">
<label for="course_id">Select Course:</label>
<select name="course_id" id="course_id" class="form-control">
{% for course in courses %}
<option value="{{ course.id }}">{{ course.name }}</option>
{% endfor %}
</select>
```

```
        </div>
        <button type="submit" class="btn btn-primary">Register</button>
    </form>
</div>
</body>
</html>
```

### #templates/course\_registration/course\_registration.html (The CSS is optional)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Course Registration</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #f8f9fa;
            margin: 0;
            padding: 0;
        }
        .container {
            max-width: 600px;
            margin: 50px auto;
            background-color: #fff;
            padding: 20px;
            border-radius: 5px;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
        }
        .list-group {
            list-style-type: none;
            padding: 0;
        }
        .list-group-item {
            margin-bottom: 10px;
        }
        .list-group-item a {
            text-decoration: none;
            color: #333;
        }
    </style>
```

```
</head>
<body>
  <div class="container">
    <h1>Course Registration</h1>
    <ul class="list-group">
      {% for course in courses %}
      <li class="list-group-item">
        <a href="{% url 'students_list' course.course_id %}">{{ course.name }} (ID: {{
course.course_id }})</a>
      </li>
      {% endfor %}
    </ul>
  </div>
</body>
</html>
```

### #templates/course\_registration/students\_list.html (The CSS is optional)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Students List</title>

  <style>
    body {
      font-family: Arial, sans-serif;
      background-color: #f8f9fa;
      margin: 0;
      padding: 0;
    }
    .container {
      max-width: 600px;
      margin: 50px auto;
      background-color: #fff;
      padding: 20px;
      border-radius: 5px;
      box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    }
    .list-group {
      list-style-type: none;
      padding: 0;
```

```
    }
    .list-group-item {
        margin-bottom: 10px;
    }
</style>
</head>

<body>

    <div class="container">
        <h1>Students Registered for {{ course.name }}</h1>
        <ul class="list-group">
            {% for student in students %}
            <li class="list-group-item">{{ student.name }}</li>
            {% empty %}
            <li class="list-group-item">No students registered for this course.</li>
            {% endfor %}
        </ul>
    </div>
</body>
</html>
```

9. Save all modified files.

10. In the VS Code Terminal, again with the virtual environment activated, run the below commands to migrate changes.

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

11. In order to view the database and its tables, you can use SQLite DB Browser.

**<https://sqlitebrowser.org/dl/>**

12. In the VS Code Terminal, again with the virtual environment activated, run the development server with **python manage.py runserver** and open a browser to **http://127.0.0.1:8000/**

13. In the url box of the browser, navigate to **http://127.0.0.1:8000/registration/add\_student** to add the students. Similarly, check out **http://127.0.0.1:8000/registration/add\_course** to add courses, **http://127.0.0.1:8000/registration/register**, **http://127.0.0.1:8000/registration/courses**, and **http://127.0.0.1:8000/registration/students\_list/<course\_id>**

OUTPUT:

127.0.0.1:8000/registration/add\_student/

### Add Student

Name:

Date of birth:

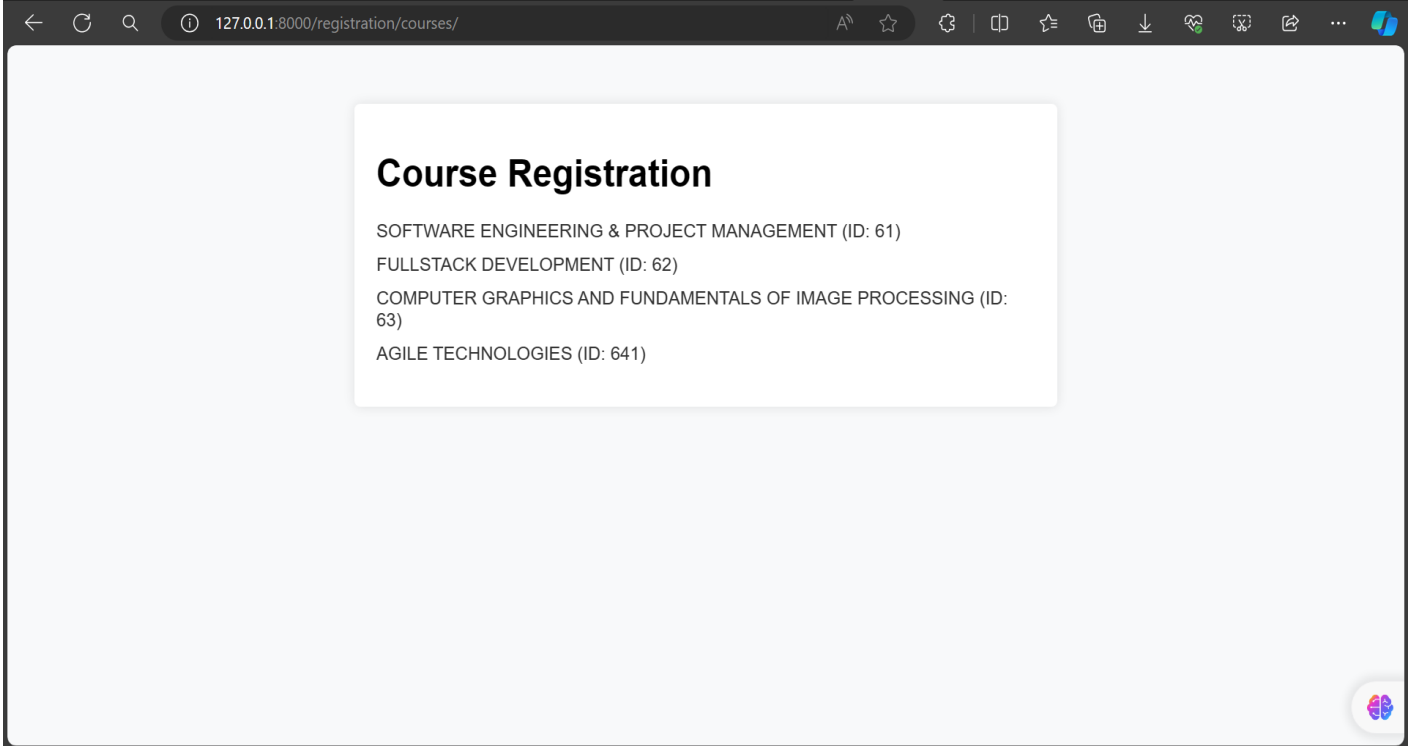
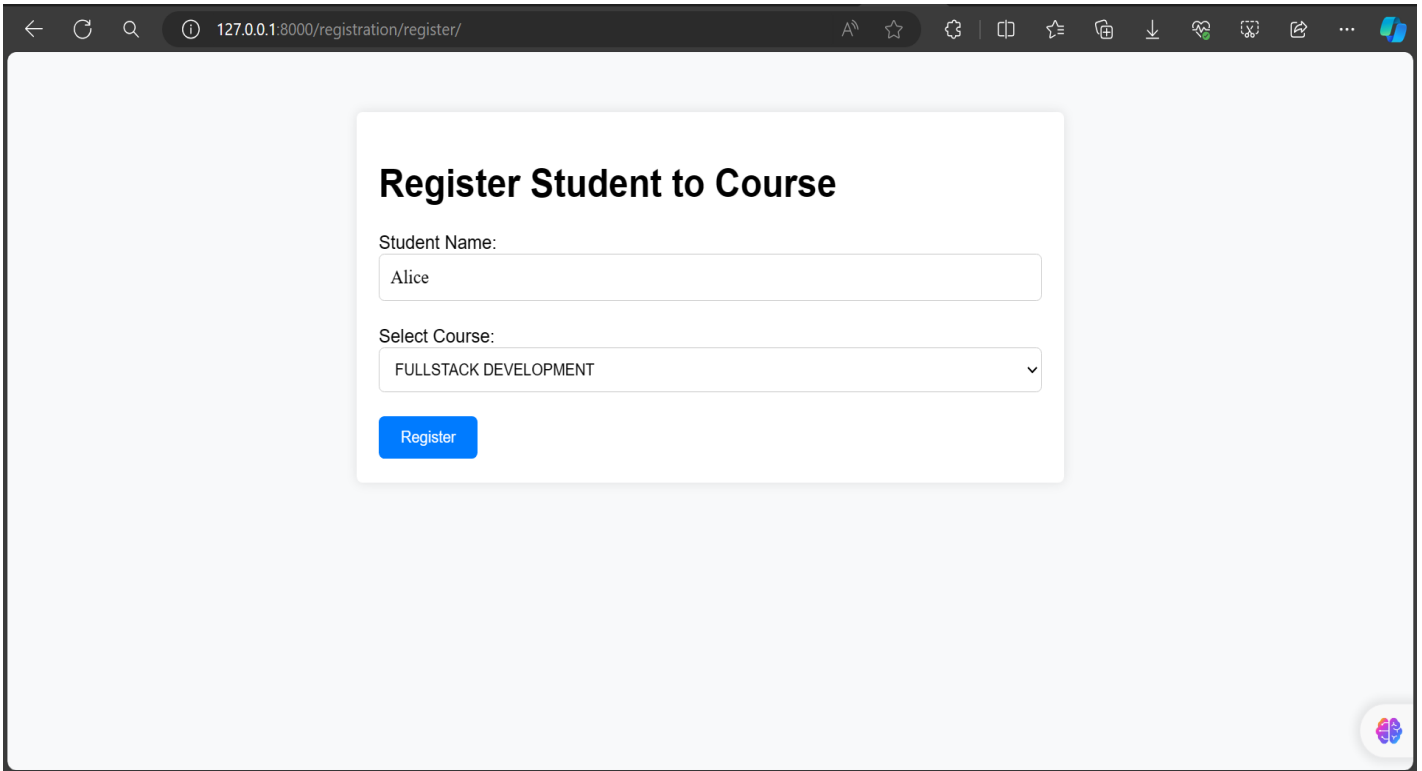
Email:

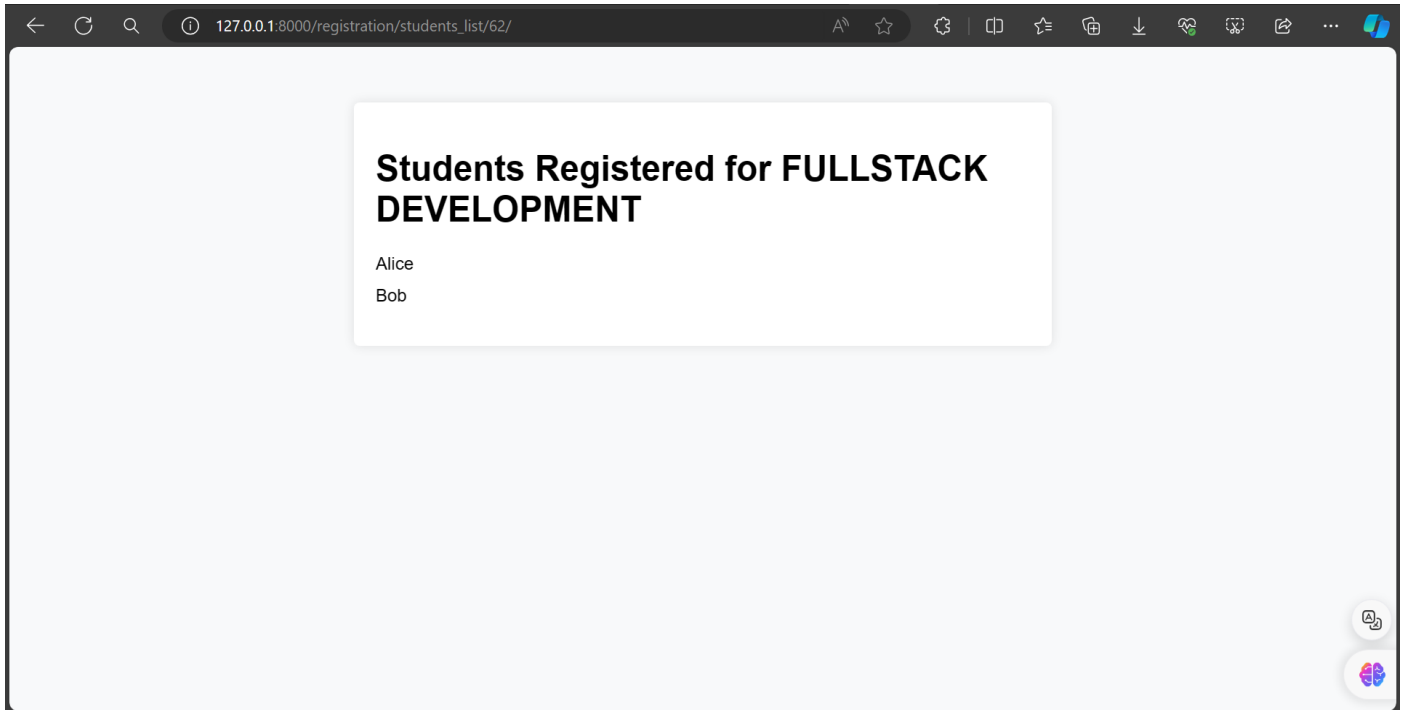
127.0.0.1:8000/registration/add\_course/

### Add Course

Course Name:

Course ID:



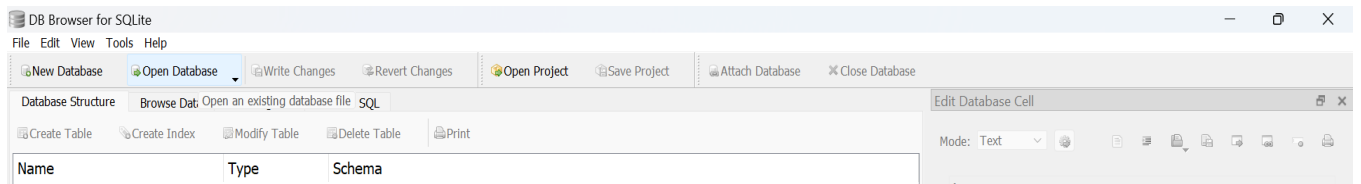


**ManyToMany** field application can be checked through the fact that **multiple students** can be enrolled to **multiple courses**.

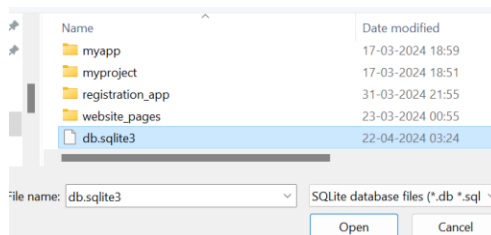
**Try it out!!!!**

Use the SQLite DB Browser to view, insert, update, and delete the records in the tables created in the db.sqlite3 database.

1. Open SQLite DB Browser.
2. Click on **Open Database**.

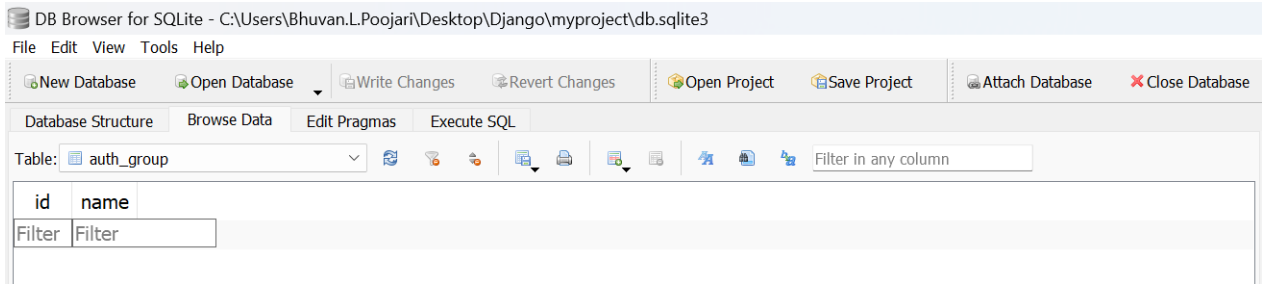


3. Select **db.sqlite3** database.

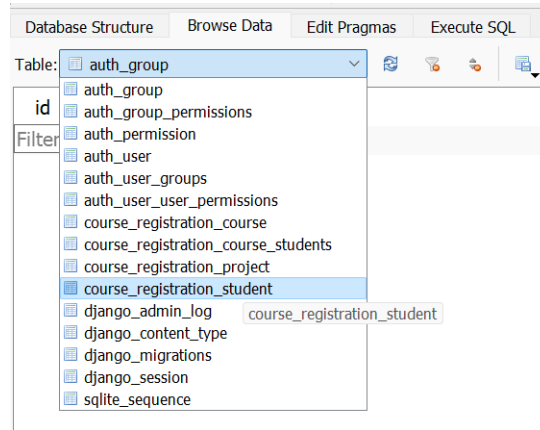




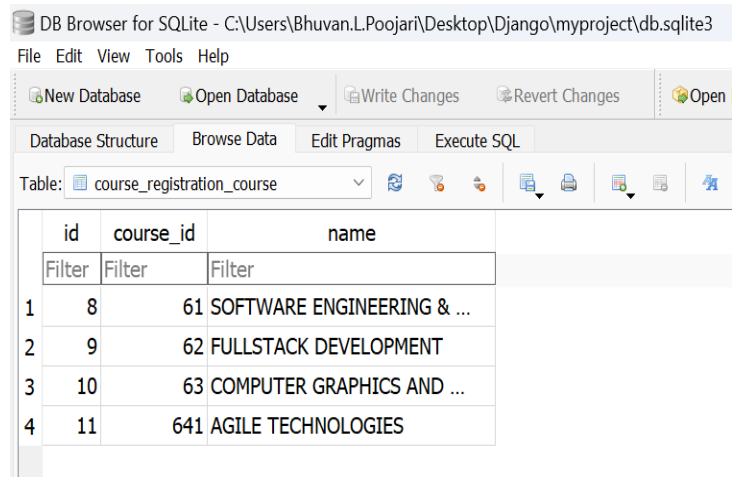
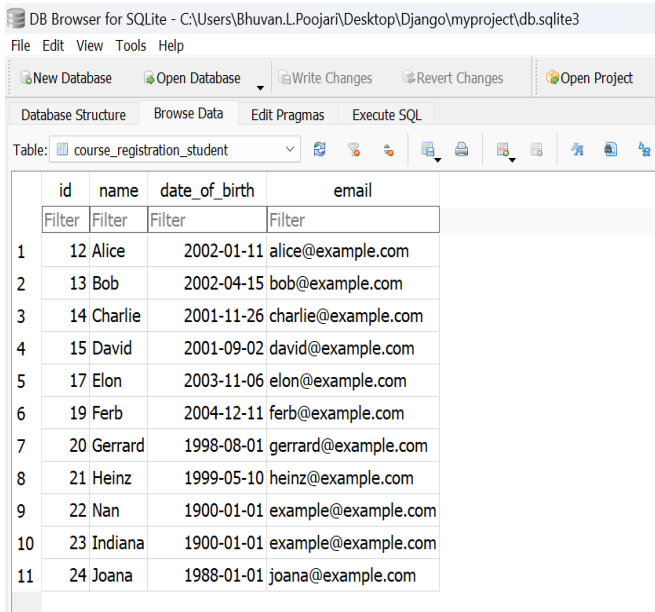
4. Click on **Browse Data**.



5. Select the required table.



6. You can now access your tables.



### Laboratory Component - 3:

1. For student and course models created in Lab experiment for Module2, register admin interfaces, perform migrations and illustrate data entry through admin forms.
2. Develop a Model form for student that contains his topic chosen for project, languages used and duration with a model called project.

#### 3.1 Admin Interface.

1. Modify the `course_registration/admin.py`.

```
# admin.py
from django.contrib import admin
from .models import Student, Course

class CourseAdmin(admin.ModelAdmin):
    list_display = ['name', 'course_id']

admin.site.register(Student)
admin.site.register(Course, CourseAdmin)
```

2. Perform migrations using the commands.

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

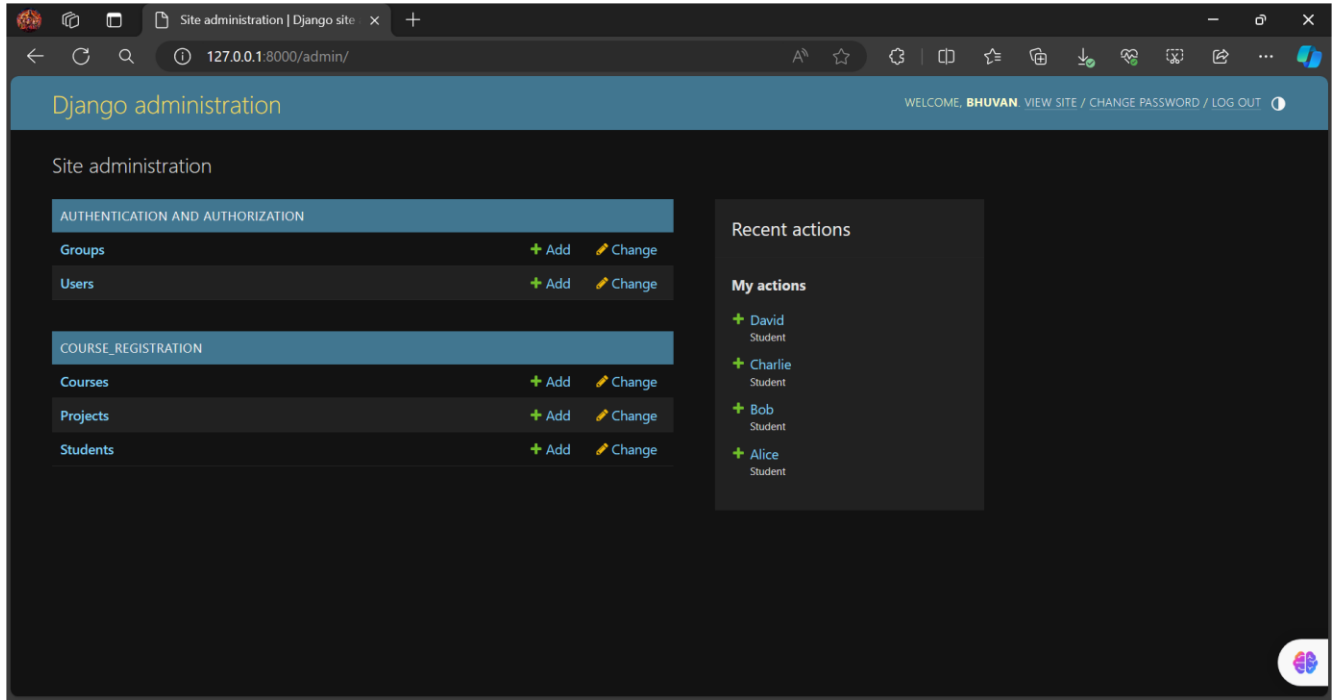
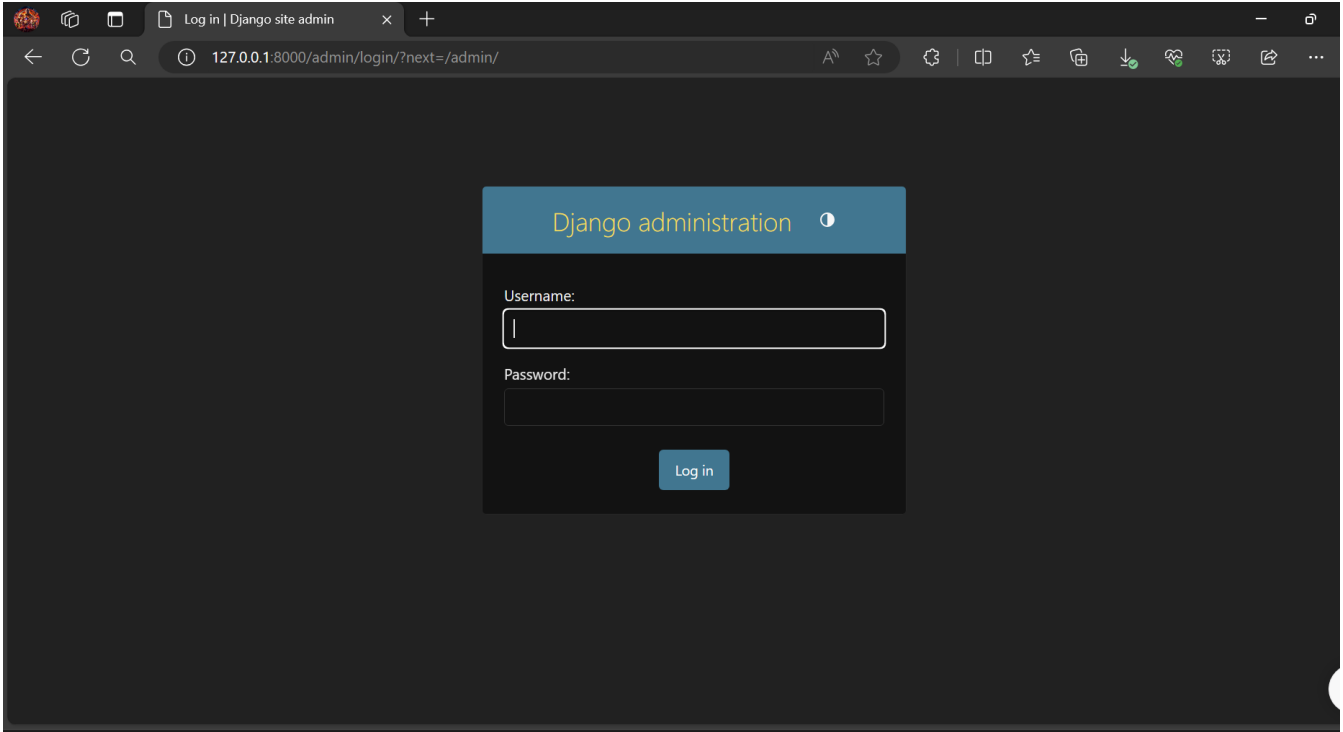
3. Now to enter data through admin interfaces, we need to first create a super user to get access to the admin dashboard. In the VS Code terminal run the below command,

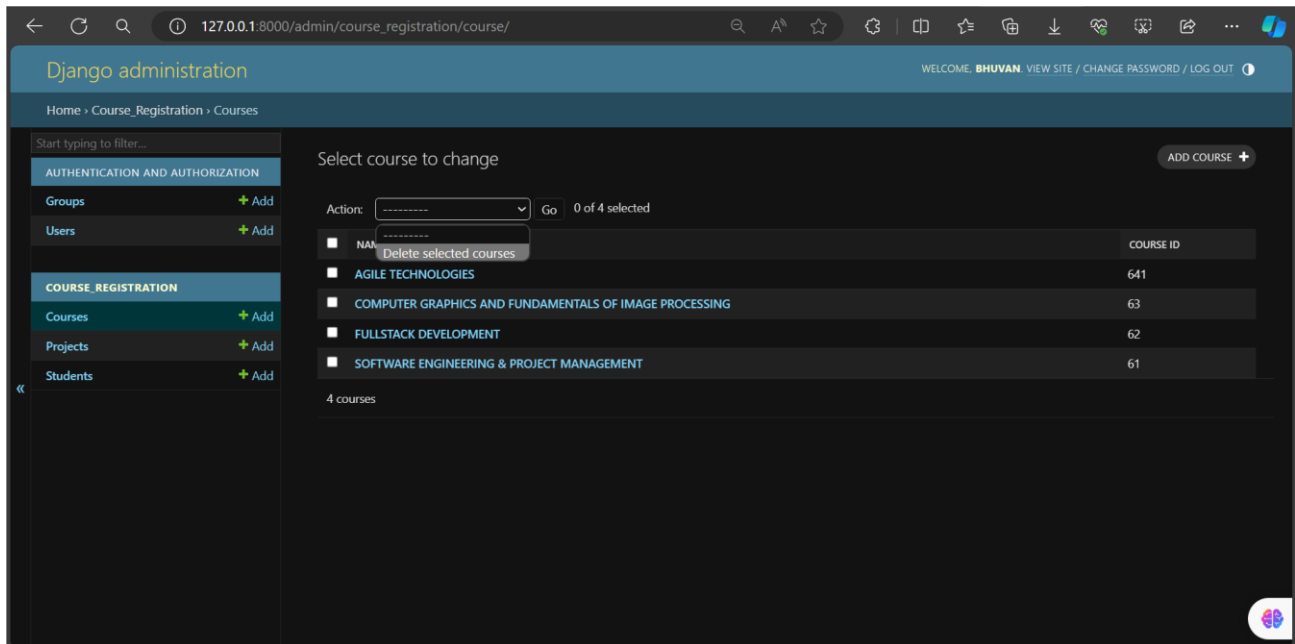
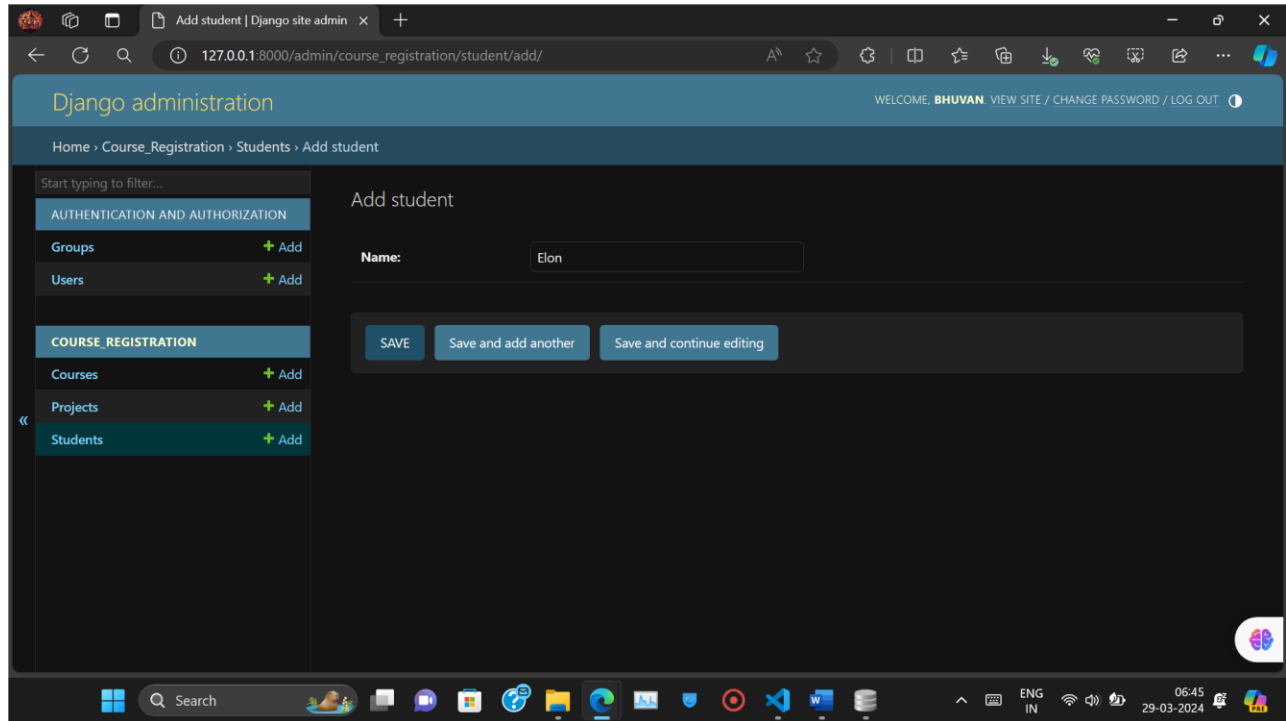
```
python manage.py createsuperuser
```

4. Once the superuser is successfully created. Run the server in the VS Code terminal.

```
python manage.py runserver
```

5. In the url box of the browser, navigate to path <http://127.0.0.1:8000/admin>
6. Enter the username and password given.





### 3.2 Modify the previous app files.

1. Add the below code to `course_registration/views.py` existing code.

```
#views.py
from .models import Project
from .forms import ProjectForm

def project_list(request):
    projects = Project.objects.all()
    return render(request, 'course_registration/project_list.html', {'projects': projects})

def add_project(request):
    if request.method == 'POST':
        form = ProjectForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('project_list')
    else:
        form = ProjectForm()
    return render(request, 'course_registration/add_project.html', {'form': form})
```

2. In the `course_registration/urls.py`, include the new paths to existing urlpatterns list.

```
#urls.py (course_registration/urls.py)
... (to indicate rest of code)
path('project_list/', views.project_list, name='project_list'),
path('add_project/', views.add_project, name='add_project'),
...
```

3. Add below lines of code to existing `course_registration/models.py`

```
#models.py
...
class Project(models.Model):
    topic = models.CharField(max_length=100)
    languages_used = models.CharField(max_length=100)
    duration = models.CharField(max_length=50)
    def __str__(self):
        return self.topic
...
```

4. Add below lines of code to existing `course_registration/forms.py`

```
#forms.py
from .models import Project
...
class ProjectForm(forms.ModelForm):
    class Meta:
        model = Project
        fields = ['topic', 'languages_used', 'duration']
...
```

5. In the `templates/course_registration` folder, create files named `add_project.html`, and `project_list.html` with the contents below.

**#templates/course\_registration/add\_project.html (The CSS is optional)**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Add Project</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 0;
      padding: 20px;
    }
    h1 {
      color: #333;
    }
    form {
      background-color: #f9f9f9;
      padding: 20px;
      border-radius: 5px;
    }
    label {
      display: block;
      margin-bottom: 5px;
    }
    input[type="text"] {
      width: 100%;
      padding: 8px;
      margin-bottom: 10px;
    }
  </style>

```

```
        border-radius: 5px;
        border: 1px solid #ccc;
    }
    button {
        background-color: #007bff;
        color: #fff;
        padding: 10px 20px;
        border: none;
        border-radius: 5px;
        cursor: pointer;
    }
</style>
</head>
<body>
    <h1>Add Project</h1>
    <form method="post">
        {% csrf_token %}
        <label for="id_topic">Topic:</label>
        {{ form.topic }}
        <label for="id_languages_used">Languages used:</label>
        {{ form.languages_used }}
        <label for="id_duration">Duration:</label>
        {{ form.duration }}
        <button type="submit">Submit</button>
    </form>
</body>
</html>
```

### #templates/course\_registration/project\_list.html (The CSS is optional)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Project List</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 20px;
        }
        h1 {
```

```

        color: #333;
    }
    ul {
        list-style-type: none;
        padding: 0;
    }
    li {
        margin-bottom: 10px;
        background-color: #f9f9f9;
        padding: 10px;
        border-radius: 5px;
    }
    a {
        text-decoration: none;
        color: #007bff;
    }
</style>
</head>
<body>
<h1>Project List</h1>
<ul>
{% for project in projects %}
    <li>
        <strong>{{ project.topic }}</strong><br>
        <em>Languages used:</em> {{ project.languages_used }}<br>
        <em>Duration:</em> {{ project.duration }}
    </li>
{% empty %}
    <li>No projects available</li>
{% endfor %}
</ul>
<a href="{% url 'add_project' %}">Add Project</a>
</body>
</html>

```

6. In the `admin.py` register the new Project model.

**#admin.py**

```

...
from .models import Student, Course, Project
...
admin.site.register(Project)
...

```

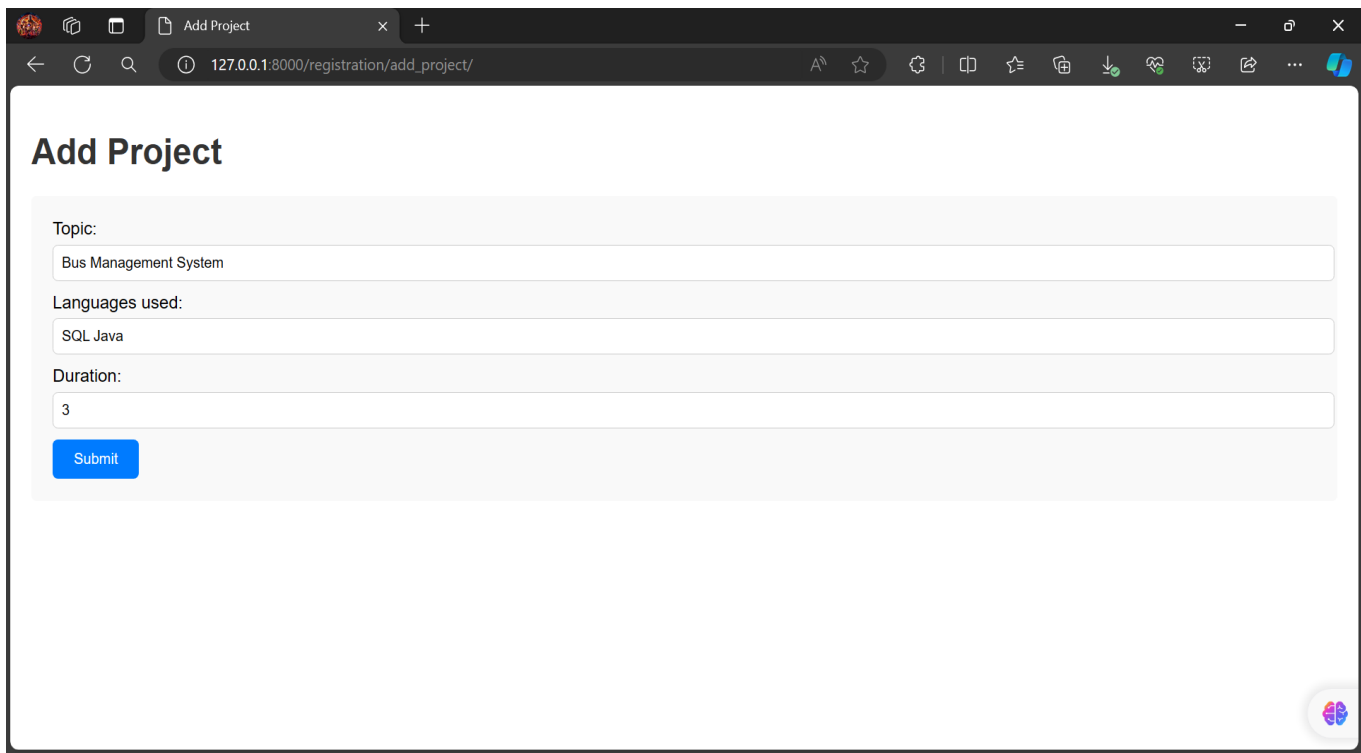


7. Save all modified files.
8. In the VS Code Terminal, again with the virtual environment activated, run the below commands to migrate changes.

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

9. In the VS Code Terminal, again with the virtual environment activated, run the development server with **python manage.py runserver** and open a browser to **http://127.0.0.1:8000/**
10. In the url box of the browser, navigate to **http://127.0.0.1:8000/registration/add\_project** to add the projects into database. Similarly, check out **http://127.0.0.1:8000/registration/project\_list** to view projects.

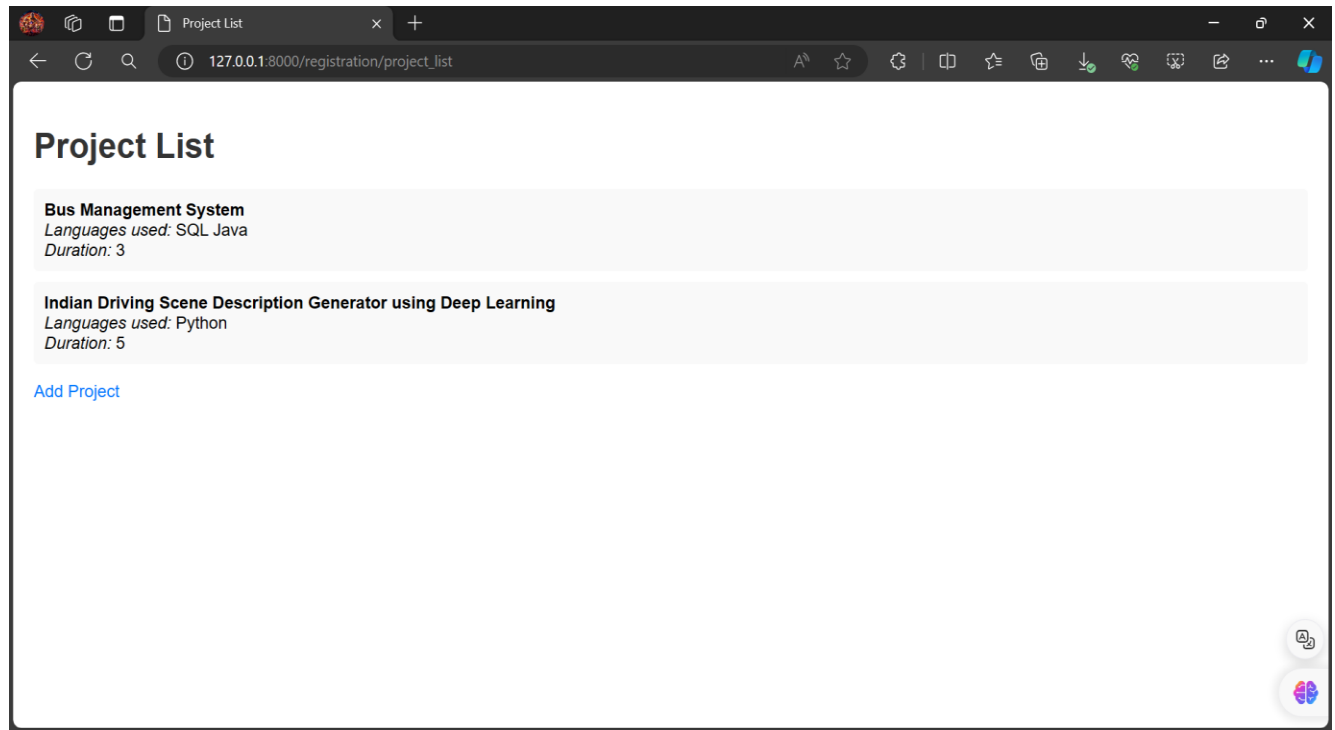
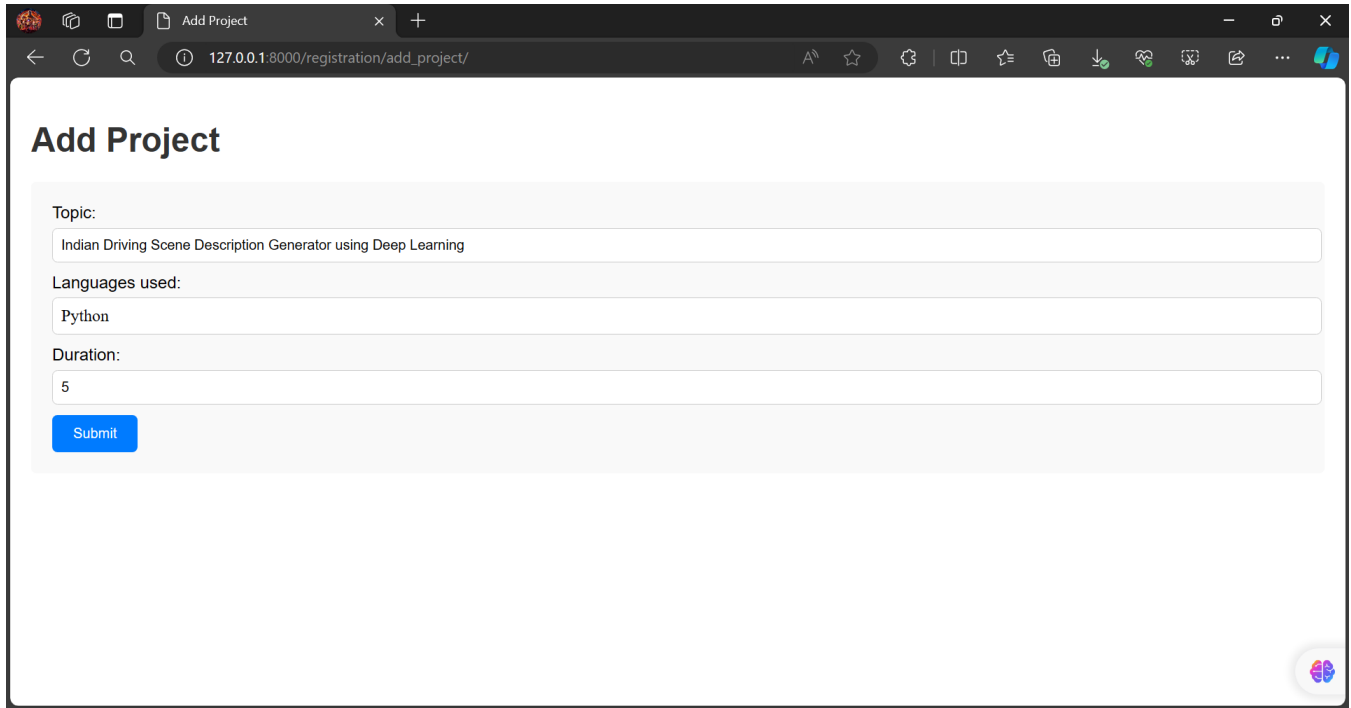
## OUTPUT:



The screenshot shows a web browser window with the address bar displaying `127.0.0.1:8000/registration/add_project/`. The page title is "Add Project". The form contains the following fields:

- Topic:
- Languages used:
- Duration:

A blue "Submit" button is located below the "Duration" field.



### Laboratory Component - 4:

1. For students' enrolment developed in Module 2, create a generic class view which displays list of students and detailview that displays student details for any selected student in the list.
2. Develop example Django app that performs CSV and PDF generation for any models created in previous laboratory component.

#### 4. 1. Modify the previous app files.

1. Add the below code to `course_registration/views.py` existing code.

```
#views.py
from .models import Student
from django.views.generic import ListView, DetailView

class StudentListView(ListView):
    model = Student
    template_name = 'course_registration/student_list.html'
    context_object_name = 'students'

class StudentDetailView(DetailView):
    model = Student
    template_name = 'course_registration/student_detail.html'
    context_object_name = 'student'

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        student = self.object # Get the student object
        context['date_of_birth'] = student.date_of_birth
        context['email'] = student.email
        # Add more fields as needed
        return context
```

2. In the `course_registration/urls.py`, include the new paths to existing urlpatterns list.

```
#urls.py (course_registration/urls.py)
... (to indicate rest of code)
path('students/', views.StudentListView.as_view(), name='student_list'),
path('student/<int:pk>/', views.StudentDetailView.as_view(),
     name='student_detail'),
...
```

3. In the templates/course\_registration folder, create files named student\_list.html, and student\_detail.html with the contents below.

**#templates/course\_registration/student\_list.html (The CSS is optional)**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Student List</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 0;
      padding: 20px;
    }
    h1 {
      color: #333;
    }
    ul {
      list-style-type: none;
      padding: 0;
    }
    li {
      margin-bottom: 10px;
    }
    a {
      text-decoration: none;
      color: #007bff;
      font-weight: bold;
    }
    a:hover {
      color: #0056b3;
    }
  </style>
</head>
<body>
  <h1>Student List</h1>
  <ul>
    {% for student in object_list %}
      <li><a href="{% url 'student_detail' student.pk %}">{{ student.name }}</a></li>
    {% empty %}

```

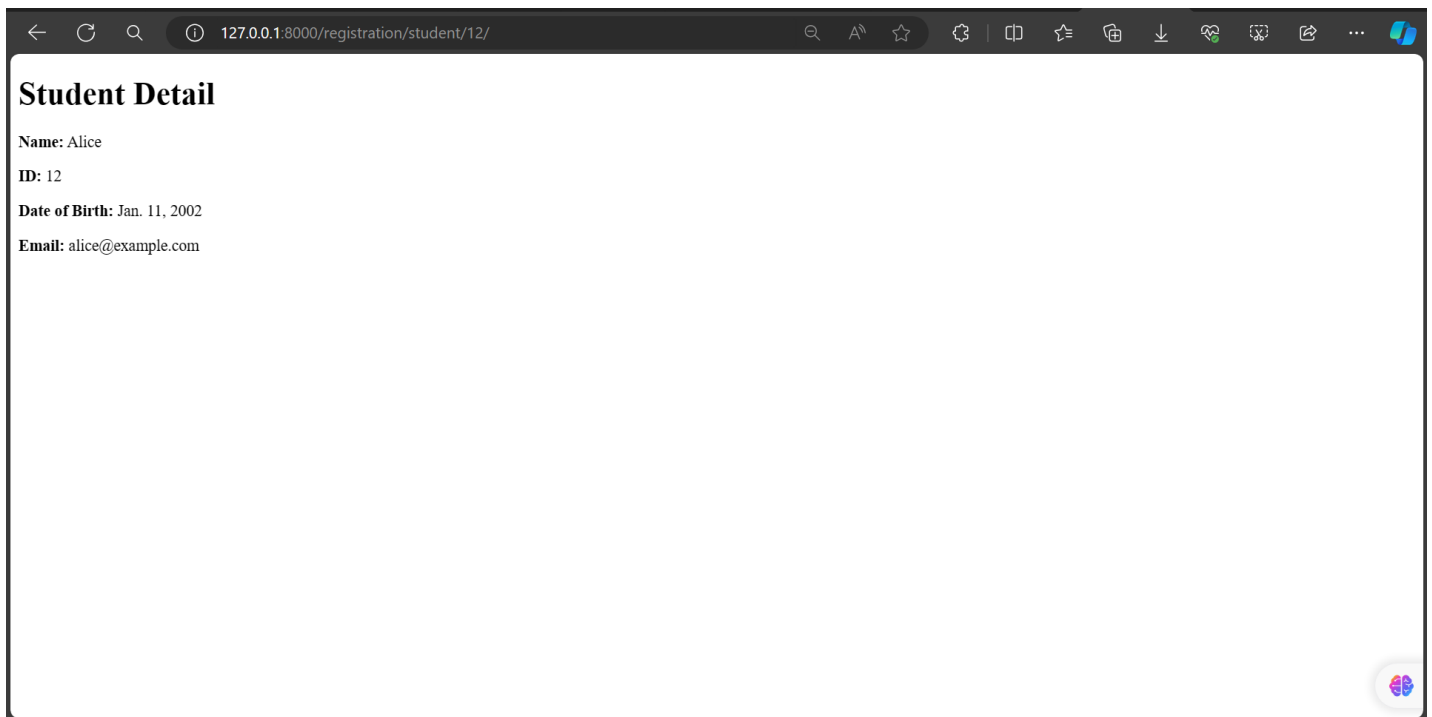
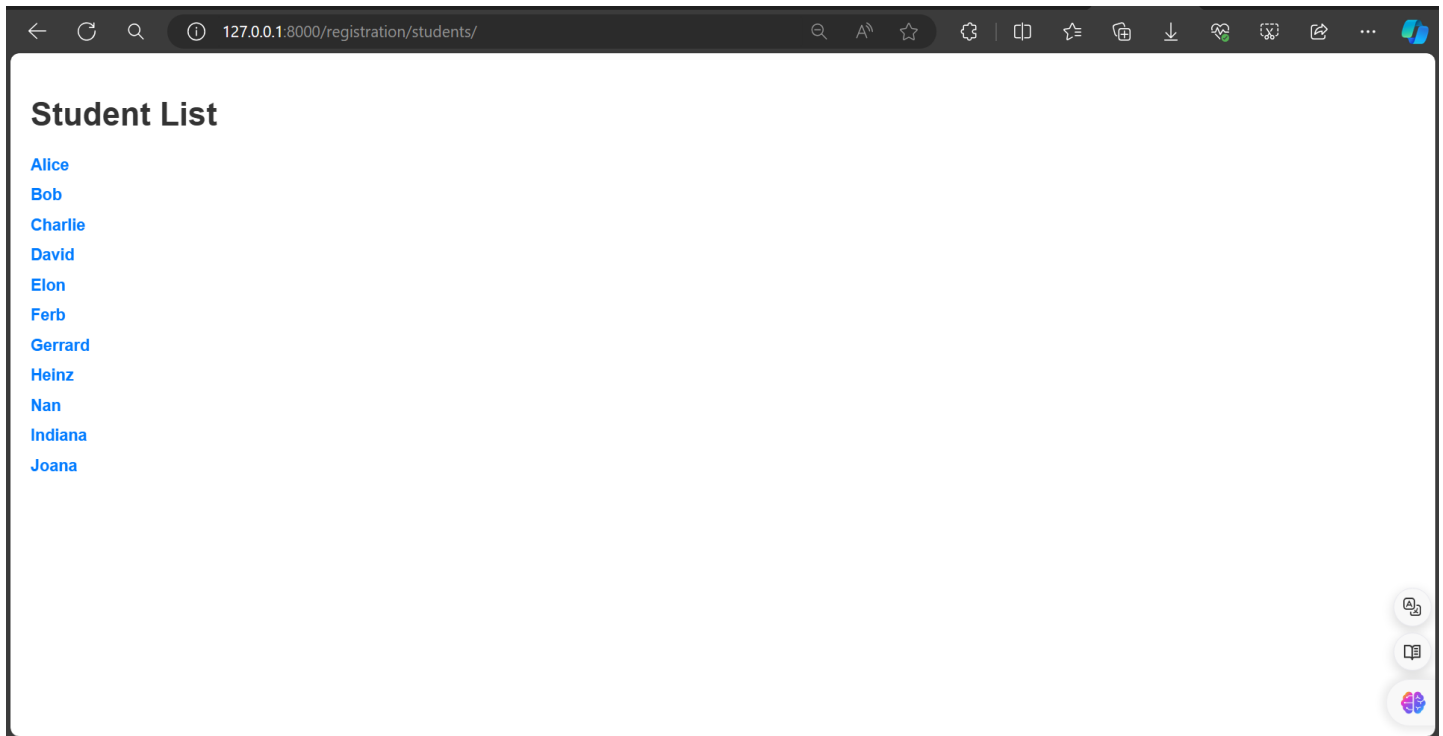
```
        <li>No students available</li>
    {% endfor %}
</ul>
</body>
</html>
```

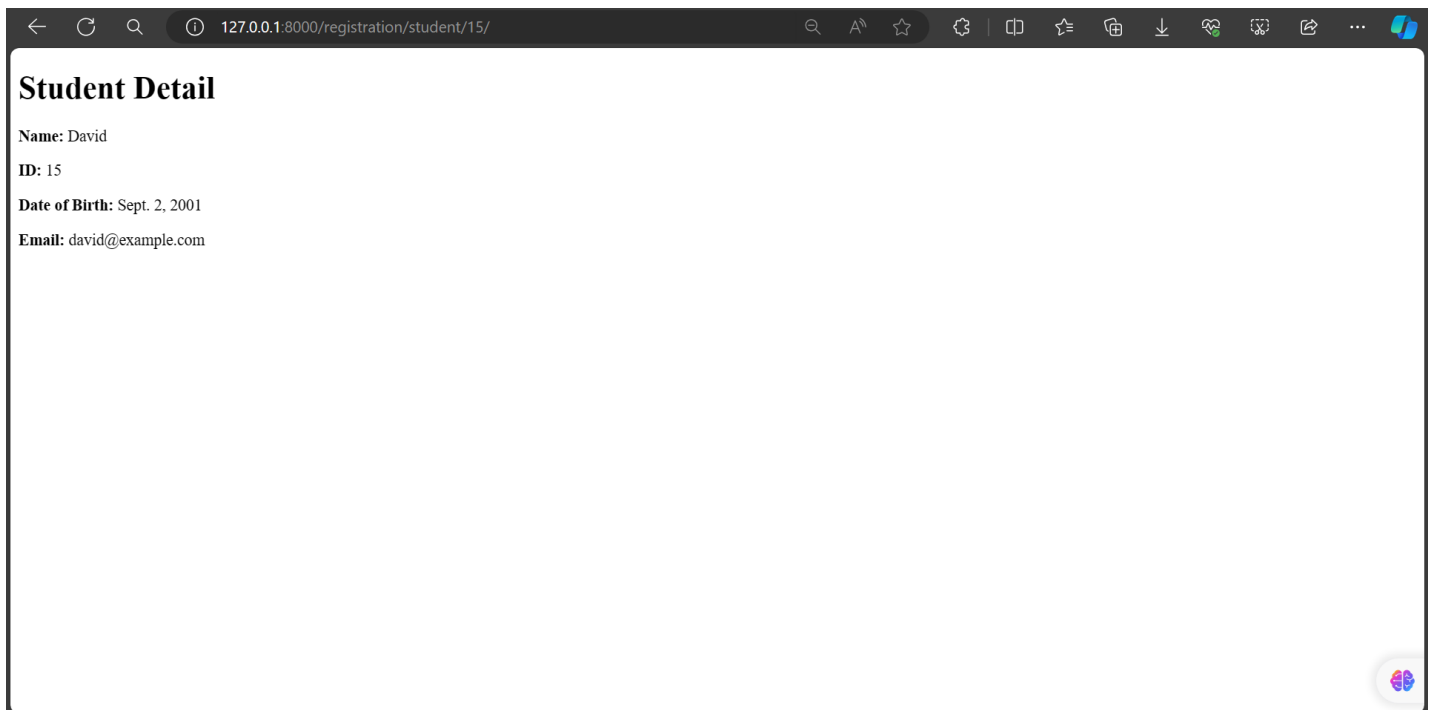
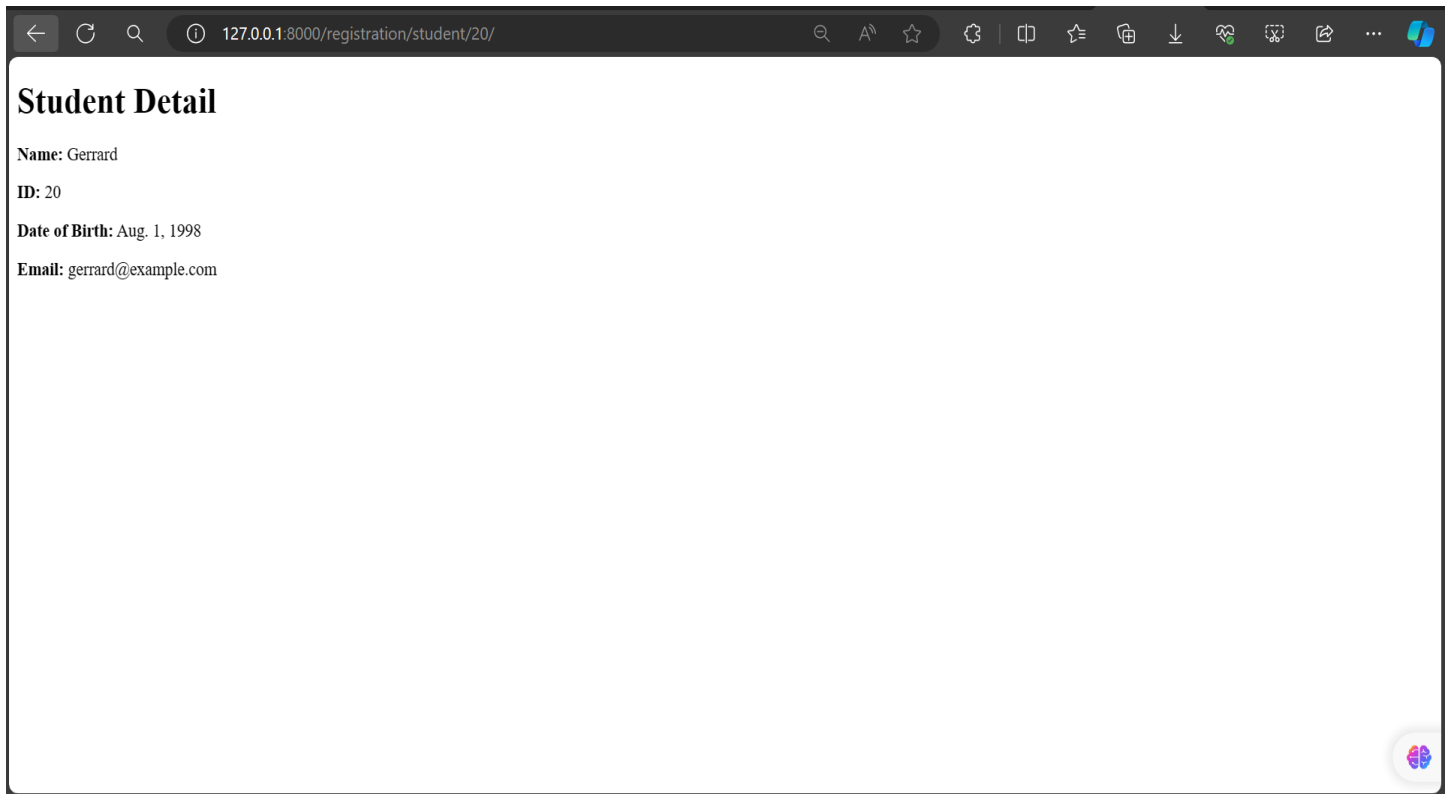
### #templates/course\_registration/student\_list.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Student Detail</title>
</head>
<body>
    <h1>Student Detail</h1>
    <p><strong>Name:</strong> {{ student.name }}</p>
    <p><strong>ID:</strong> {{ student.id }}</p>
    <p><strong>Date of Birth:</strong> {{ date_of_birth }}</p>
    <p><strong>Email:</strong> {{ email }}</p>
    <!-- Add more details as needed -->
</body>
</html>
```

4. Save all modified files.
5. In the VS Code Terminal, again with the virtual environment activated, run the development server with **python manage.py runserver** and open a browser to **http://127.0.0.1:8000/**
6. In the url box of the browser, navigate to **http://127.0.0.1:8000/registration/students**. Similarly, check out **http://127.0.0.1:8000/registration/student/<id or pk>**.

OUTPUT:





## 4.2. Create new app for generating pdf and csv.

1. In the VS Code Terminal with your virtual environment activated, run the administrative utility's `startapp` command in your myproject folder (where `manage.py` resides):

```
python manage.py startapp generate
```

1. Add the below code to `generate/views.py`.

### **#views.py**

```
from course_registration.models import Student
from django.http import HttpResponse
import csv
from io import BytesIO
from reportlab.lib.pagesizes import letter
from reportlab.pdfgen import canvas
```

```
def generate_csv(request):
```

```
    # Retrieve all student objects from the database
    students = Student.objects.all()
    # Create an HTTP response with content type 'text/csv'
    response = HttpResponse(content_type='text/csv')
    # Set the content disposition header to specify the filename for download
    response['Content-Disposition'] = 'attachment; filename="students.csv"'
    # Create a CSV writer object
    writer = csv.writer(response)
    # Write the header row
    writer.writerow(['ID', 'Name', 'Date of Birth', 'Email'])
    # Write each student's information to a row in the CSV file
    for student in students:
        writer.writerow([student.id, student.name,
                        student.date_of_birth, student.email])
    # Return the HTTP response containing the CSV file
    return response
```

```
def generate_pdf(request):
```

```
    # Retrieve all student objects from the database
    students = Student.objects.all()
    # Create a BytesIO buffer to store the PDF content
    buffer = BytesIO()
    # Create a canvas object with letter size (8.5x11 inches)
```



```

p = canvas.Canvas(buffer, pagesize=letter)
# Set the title for the PDF document
p.setFont("Helvetica-Bold", 16)
p.drawString(100, 750, "Student List")
# Set the starting y-coordinate for student information
y = 700
# Iterate over each student and add their information to the PDF
for student in students:
    # Set font size and add student information to the PDF
    p.setFont("Helvetica", 12)
    p.drawString(
        100, y, f"ID: {student.id}, Name: {student.name}, DoB: {student.date_of_birth}, Email:
{student.email}")
    # Move to the next line
    y -= 20
# Save the PDF document
p.showPage()
p.save()
# Move the buffer's cursor to the beginning
buffer.seek(0)
# Create an HTTP response with content type 'application/pdf'
response = HttpResponse(buffer.getvalue(), content_type='application/pdf')
# Set the content disposition header to specify the filename for download
response['Content-Disposition'] = 'attachment; filename="students.pdf"'
# Return the HTTP response containing the PDF file
return response

```

2. In the generate/urls.py, include the new paths.

```
#urls.py (generate/urls.py)
```

```

from django.urls import path
from . import views

```

```

urlpatterns = [
    path('generate_csv/', views.generate_csv, name='generate_csv'),
    path('generate_pdf/', views.generate_pdf, name='generate_pdf'),
]

```

3. The myproject folder also contains a urls.py file, which is where URL routing is actually handled.

#urls.py (myproject/urls.py)

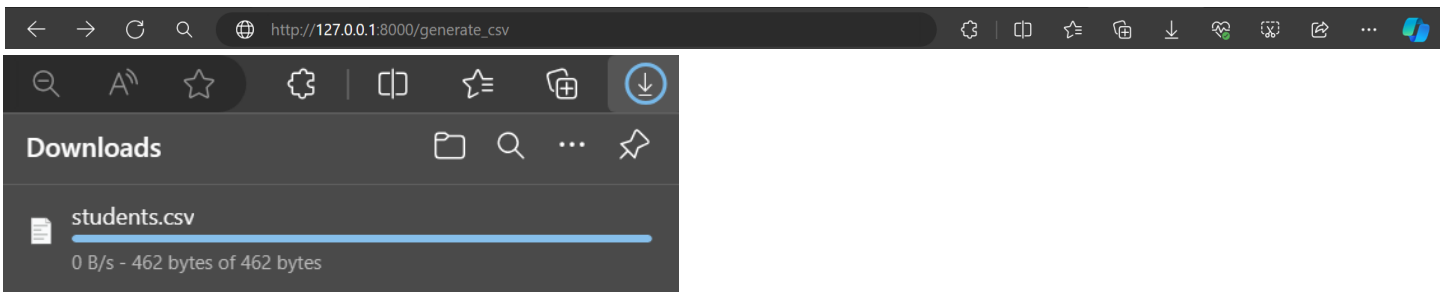
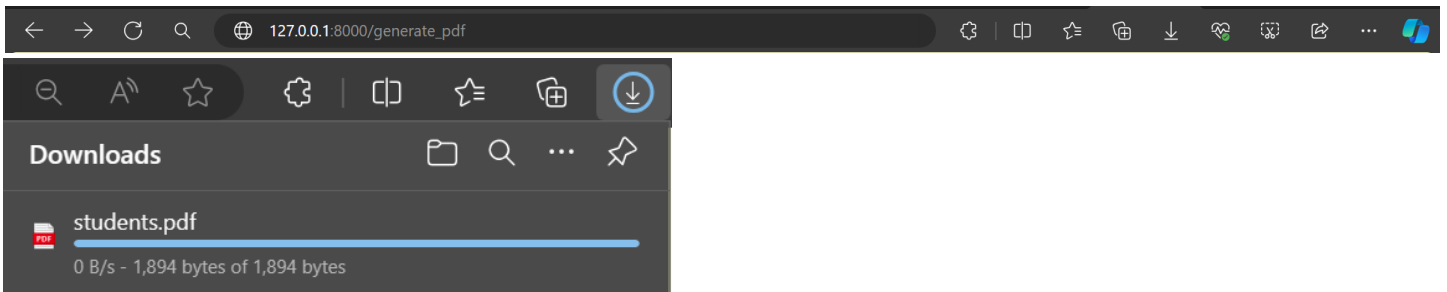
```
from django.contrib import admin
from django.urls import path, include
urlpatterns = [
    path("", include("myapp.urls")),
    path("admin/", admin.site.urls),
    path("", include('website_pages.urls')),
    path('fruits_and_students/', include('fruits_and_students.urls')),
    path('registration/', include('course_registration.urls')),
    path("", include('generate.urls')),
]
```

4. In the myproject/settings.py file, locate the INSTALLED\_APPS list and add the following entry, which makes sure the project knows about the app so it can handle templating:

‘generate’,

5. Save all modified files.
6. In the VS Code Terminal, again with the virtual environment activated, run the development server with **python manage.py runserver** and open a browser to **http://127.0.0.1:8000/**
7. In the url box of the browse, navigate to **http://127.0.0.1:8000/generate\_pdf**. Similarly, check out **http://127.0.0.1:8000/generate\_csv**.

OUTPUT:



### Laboratory Component - 5:

1. Develop a registration page for student enrolment as done in Module 2 but without page refresh using AJAX.
2. Develop a search application in Django using AJAX that displays courses enrolled by a student being searched.

#### 5.1 Create a Django app

1. In the VS Code Terminal with your virtual environment activated, run the administrative utility's `startapp` command in your myproject folder (where `manage.py` resides):

```
python manage.py startapp enrollment
```

2. Modify `enrollment/views.py` to match the following code.

##### **#views.py**

```
from django.shortcuts import render
from django.http import JsonResponse
from .forms import StudentRegistrationForm
```

```
def register_student(request):
    if request.method == 'POST':
        form = StudentRegistrationForm(request.POST)
        if form.is_valid():
            # Here you would typically save the data to a database or perform other actions
            return JsonResponse({"success": True, "message": "Student registered successfully!"})
        else:
            return JsonResponse({"success": False, "errors": form.errors})
    else:
        form = StudentRegistrationForm()
        return render(request, 'enrollment/register.html', {'form': form})
```

3. Create a file, `enrollment/urls.py`, with the contents below. The `urls.py` file is where you specify patterns to route different URLs to their appropriate views.

##### **#urls.py (enrollment/urls.py)**

```
from django.urls import path
from .views import register_student
urlpatterns = [
    path('register/', register_student, name='register_student'),
]
```

- The myproject folder also contains a urls.py file, which is where URL routing is actually handled.  
**#urls.py (myproject/urls.py)**

```
...
path("", include('enrollment.urls')),
...
```

- In the `myproject/settings.py` file, locate the `INSTALLED_APPS` list and add the following entry, which makes sure the project knows about the app so it can handle templating:

```
'enrollment',
```

- Create forms.py and add the following code.

```
#forms.py
from django import forms

class StudentRegistrationForm(forms.Form):
    name = forms.CharField(label='Full Name', max_length=100)
    email = forms.EmailField(label='Email')
    course = forms.CharField(label='Course', max_length=100)
```

- Inside the enrollment folder, create a folder named templates, and then another subfolder named enrollment to match the app name (this two-tiered folder structure is typical Django convention).

In the templates/enrollment folder, create a file named register.html with the contents below.

```
#register.html (enrollment/templates/enrollment/register.html)
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Student Registration</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
</head>
<body>
  <form id="registrationForm">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Register</button>
  </form>
```

```
<div id="message"></div>

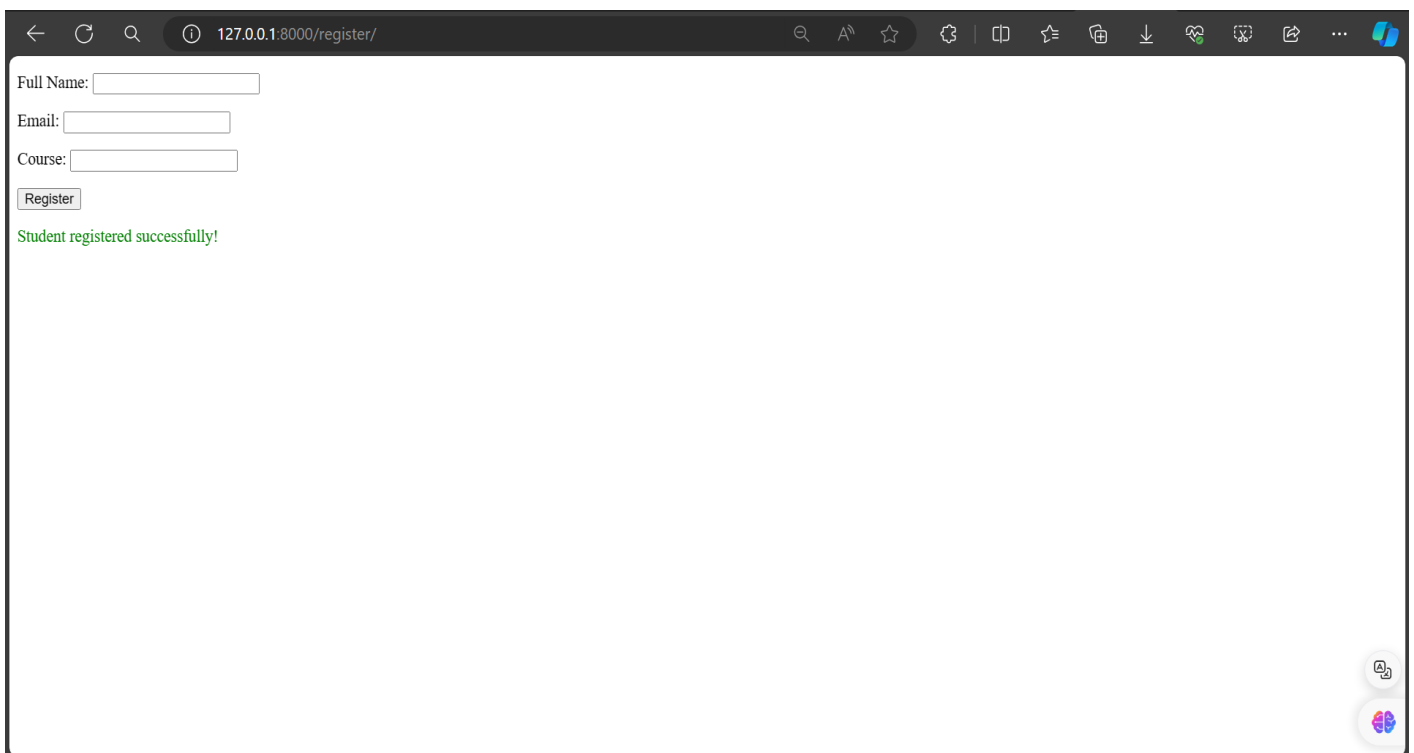
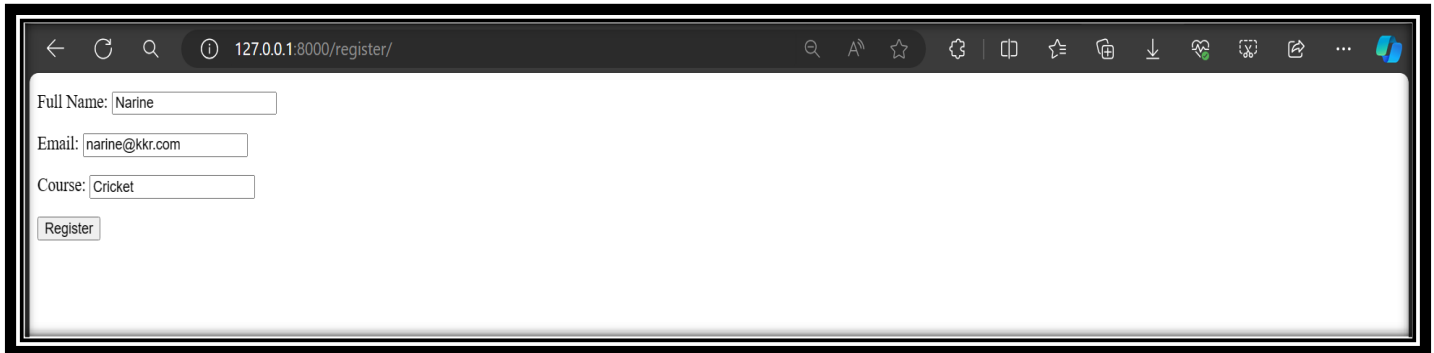
<script>
  function getCookie(name) {
    let cookieValue = null;
    if (document.cookie && document.cookie !== "") {
      const cookies = document.cookie.split(';');
      for (let i = 0; i < cookies.length; i++) {
        const cookie = jQuery.trim(cookies[i]);
        if (cookie.substring(0, name.length + 1) === (name + '=')) {
          cookieValue = decodeURIComponent(cookie.substring(name.length + 1));
          break;
        }
      }
    }
    return cookieValue;
  }

  $(document).ready(function () {
    $('#registrationForm').submit(function (e) {
      e.preventDefault();
      $.ajax({
        type: 'POST',
        url: '{% url "register_student" %}',
        data: $(this).serialize(),
        success: function (response) {
          if (response.success) {
            $('#message').html('<p style="color: green;">' + response.message + '</p>');
            $('#registrationForm').trigger('reset'); // Reset form if needed
          } else {
            $('#message').html('<p style="color: red;">' + JSON.stringify(response.errors) +
'</p>');
          }
        }
      });
    });
  });
</script>
</body>
</html>
```

7. Save all the modified files.

8. In the VS Code Terminal, again with the virtual environment activated, run the development server with **python manage.py runserver** and open a browser to **http://127.0.0.1:8000/**
9. In the url box of the browse, navigate to **http://127.0.0.1:8000/register**. Uses AJAX to submit data without refreshing the page. Just click on the Register button once details are entered.

**OUTPUT:**



## 5.2 Create a Django app

1. In the VS Code Terminal with your virtual environment activated, run the administrative utility's `startapp` command in your myproject folder (where `manage.py` resides):

```
python manage.py startapp course_search
```

2. Modify `course_search/views.py` to match the following code.

```
#views.py
from django.http import JsonResponse
from course_registration.models import Student, Course
from django.shortcuts import render

def search_courses(request):
    if request.headers.get('x-requested-with') == 'XMLHttpRequest':
        query = request.GET.get('query', None)
        if query:
            # Perform the search based on the query parameter
            courses = Course.objects.filter(students__name__icontains=query)
            # Create a list of course names
            course_names = [{'name': course.name,
                             'course_id': course.course_id} for course in courses]
            # Return the list of course names as JSON response
            return JsonResponse({'courses': course_names})
        else:
            return JsonResponse({'error': 'No query parameter provided'})
    else:
        # Optionally, handle non-AJAX requests here
        return render(request, 'course_search/search.html')
```

3. Create a file, `course_search/urls.py`, with the contents below. The `urls.py` file is where you specify patterns to route different URLs to their appropriate views.

```
#urls.py (course_search/urls.py)
from django.urls import path
from .views import search_courses

urlpatterns = [
    path('search/', search_courses, name='search_courses'),
]
```

4. The myproject folder also contains a urls.py file, which is where URL routing is actually handled.  
**#urls.py (myproject/urls.py)**

```
...
path('', include('course_search.urls')),
...
```

5. In the `myproject/settings.py` file, locate the `INSTALLED_APPS` list and add the following entry, which makes sure the project knows about the app so it can handle templating:

```
'course_search',
```

6. Inside the `course_search` folder, create a folder named `templates`, and then another subfolder named `course_search` to match the app name (this two-tiered folder structure is typical Django convention).

In the `templates/course_search` folder, create a file named `search.html` with the contents below.

**#search.html (course\_search/templates/course\_search/search.html)**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Search Courses</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script>
    $(document).ready(function() {
      $('#search-form').submit(function(event) {
        event.preventDefault();
        var formData = $(this).serialize();
        $.ajax({
          url: '/search/',
          type: 'get',
          data: formData,
          dataType: 'json',
          success: function(response) {
            if (response.courses) {
              var coursesHtml = "";
              response.courses.forEach(function(course) {
                coursesHtml += '<li>' + course.name + ' (' + course.course_id + ')</li>';
              });
            }
          }
        });
      });
    });
  </script>
</head>
</html>
```



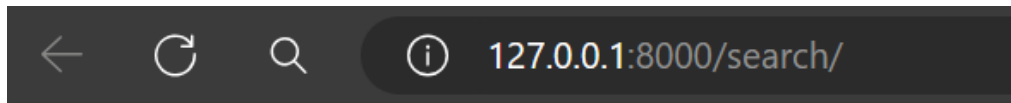
```

        $('#courses-list').html(coursesHtml);
    } else {
        $('#courses-list').html('<li>No courses found</li>');
    }
},
error: function(xhr, status, error) {
    console.error('Error:', error);
}
});
});
});
</script>
</head>
<body>
    <h1>Search Courses by Student</h1>
    <form id="search-form" method="get">
        <input type="text" name="query" placeholder="Enter student name">
        <button type="submit">Search</button>
    </form>
    <ul id="courses-list"></ul>
</body>
</html>

```

7. Save all the modified files.
8. In the VS Code Terminal, again with the virtual environment activated, run the development server with **python manage.py runserver** and open a browser to **http://127.0.0.1:8000/**
9. In the url box of the browse, navigate to **http://127.0.0.1:8000/search**. Uses AJAX to retrieve data without refreshing the page. Just click on the Search button once details are entered.

**OUTPUT:**



# Search Courses by Student



←

127.0.0.1:8000/search/

## Search Courses by Student

Alice

- SOFTWARE ENGINEERING & PROJECT MANAGEMENT (61)
- AGILE TECHNOLOGIES (641)
- FULLSTACK DEVELOPMENT (62)



127.0.0.1:8000/search/

## Search Courses by Student

Bob

- SOFTWARE ENGINEERING & PROJECT MANAGEMENT (61)
- FULLSTACK DEVELOPMENT (62)

