# RNS INSTITUTE OFTECHNOLOGY

*(AICTE Approved, VTU Affiliated and NAAC 'A' Accredited)*

*UG Programs - CSE, ECE, ISE, EIE and EEE have been Accredited by NBA up to 30.06.2025*

**DR. VISHNUVARDHAN ROAD, CHANNASANDRA, RR NAGAR POST, BENGALURU – 560 098**


ESTD : 2001

*An Institute with a Difference*

# FULLSTACK DEVELOPMENT

## For Sixth Semester B.E
## [VTU/NEP, 2021 syllabus]

### Subject Code – 21CS62

**NAME :** _____

**BRANCH :** _____

**SECTION :** _____

**USN :** _____

# VISION AND MISSION OF INSTITUTION

## Vision

## Building RNSIT into a World Class Institution

## Mission

**To impart high quality education in Engineering, Technology and Management with a Difference, Enabling Students to Excel in their Career by**

1. Attracting quality Students and preparing them with a strong foundation in fundamentals soas to achieve distinctions in various walks of life leading to outstanding contributions
2. Imparting value based, need based, choice based and skill based professional education to theaspiring youth and carving them into disciplined, World class Professionals with social responsibility
3. Promoting excellence in Teaching, Research and Consultancy that galvanizes academic consciousness among Faculty and Students
4. Exposing Students to emerging frontiers of knowledge in various domains and make them suitable for Industry, Entrepreneurship, Higher studies, and Research & Development
5. Providing freedom of action and choice for all the Stake holders with better visibility

# VISION AND MISSION OF DEPARTMENT

## Vision

**Preparing Better Computer Professionals for a Real World**

## Mission

**The Department of CSE will make every effort to promote an intellectual and ethical environment by**

1. Imparting solid foundations and applied aspects in both Computer Science Theory and Programming practices
2. Providing training and encouraging R&D and Consultancy Services in frontier areas ofComputer Science and Engineering with a Global outlook
3. Fostering the highest ideals of ethics, values and creating awareness of the role of Computing in Global Environment
4. Educating and preparing the graduates, highly sought after, productive, and well-respected for their work culture.
5. Supporting and inducing lifelong learning

# *FULLSTACK DEVELOPMENT LABORATORY (21CS62)*

## *INTERTANAL EVALUATION SHEET*

| *EVALUATION (MAX MARKS 20)* | | | |
|---|---|---|---|
| **TEST A** | **REGULAR EVALUATION B** | **RECORD** | **TOTAL MARKS A+B+C** |
| **20** | **10** | **20** | **50** |

| *R1: REGULAR LAB EVALUATION WRITE UP RUBRIC (MAX MARKS 05)* | | | | |
|---|---|---|---|---|
| **Sl. No.** | **Parameters** | **Good** | **Average** | **Needs improvement** |
| **a.** | **Understanding of problem (2M)** | Clear understanding of problem statement while designing and implementing (2) | Problem statement is understood clearly but few mistakes while designing and implementing () | Problem statement is not clearly understood while designing (1) |
| **b.** | **Write-up (1M)** | Write the program neatly, demonstrating the concepts of Data Structure and algorithms using C Language (4) | Moderate understanding of algorithms (3) | Problem statement is not clearly understood while writing a program (1) |
| **c.** | **Execution and Result (2M)** | Program handles all possible conditions (2) | Demonstrates some of the conditions defined (1) | Documentation does not take care all conditions (1) |

| *R2: REGULAR LAB EVALUATION VIVA RUBRIC (MAX MARKS 05)* | | | | | |
|---|---|---|---|---|---|
| **Sl. No.** | **Parameter** | **Excellent** | **Good** | **Average** | **Needs Improvement** |
| **a.** | **Conceptual understanding (05 marks)** | Answers 80% of the viva questions asked (5) | Answers 60% of the viva questions asked (4) | Answers 30% of the viva questions asked (3) | Unable to relate the concepts (2) |

| *R3: REGULAR LAB PROGRAM EXECUTION RUBRIC (MAX MARKS 05)* | | | | |
|---|---|---|---|---|
| **Sl. No.** | **Parameters** | **Excellent** | **Good** | **Needs Improvement** |
| **a.** | **Design, implementation and demonstration (5M)** | Program follows syntax and semantics of C programming language. Demonstrate the complete knowledge of the program written (5) | Program has few logical errors, moderately demonstrates all possible concepts implemented in Program (03) | Documentation on does not take care the given conditions (2) |
| **b.** | **Result & documentation (5M)** | All test cases are successful, all errors are debugged with own practical knowledge and clear documentation according to the guidelines (5) | Moderately debugs the experiment, few test case are unsuccessful and Partial documentation (4) | Test cases are not taken care, unable to debug the errors And no proper documentation (1) |

| *R4: RECORD EVALUATION RUBRIC (MAX MARKS 05)* | | | | | |
|---|---|---|---|---|---|
| **Sl. No.** | **Parameter** | **Excellent** | **Good** | **Average** | **Needs Improvement** |
| **a.** | **Documentation (05 M)** | Meticulous record writing required Complete Program with comments and test cases asper the guidelines mentioned (05) | Write up contains Program and test cases with no comments (04) | Write up contains only program (03) | Program written with few mistakes. (02) |

## A. TEST/LAB INTERNALSMARKS(MAXMARKS10M)

| TEST# | Write up 10M | Execution 30M | Viva 10M | Teacher Signature | Total 50M | Average 50M | Final 05 |
|---|---|---|---|---|---|---|---|
| TEST-1 | | | | | | 50M | 05M |
| TEST-2 | | | | | | | |

## B. REGULAR LAB EVALUATION (MAX MARKS 5)

| Sl. # | Date of Execution | Additional programs | Write up (05) | Execn (05) | Viva (05) | Total 15 | Teacher Signature |
|---|---|---|---|---|---|---|---|
| 1 | | | | | | | |
| 2 | | | | | | | |
| 3 | | | | | | | |
| 4 | | | | | | | |
| 5 | | | | | | | |
| 6 | | | | | | | |
| 7 | | | | | | | |
| 8 | | | | | | | |
| 9 | | | | | | | |
| 10 | | | | | | | |
| 11 | | | | | | | |
| 12 | | | | | | | |
| 13 | | | | | | | |
| Total Marks | | — | 15 | | | 15 | |

| | | |
|---|---|---|
| Final Marks obtained from<br>A. Test (05) +<br>B. Regular Evaluation (10) +<br>C. Record (05) | 20 | Lab in charge:<br>HOD: |

# PREFACE

We have developed this comprehensive laboratory manual on **FULL STACK DEVELOPMENT 21CS62** with the primary objectives: Familiarize the students with computer hardware, concept of naming the program files, storing, compilation, execution and debugging.

This material provides the students an exposure to problem solving approaches and solution to number of problems using C programming Language. The problems discussed in this manual comprises of an algorithm, programming solution, alternative logic and extensive test cases. Viva questions, frequently appeared examination questions and practicing programming problems constitute an indispensable part of this material.

Our profound and sincere efforts will be fruitful only when students acquire the extensive knowledge by reading this manual and apply the concepts learnt apart from the requirements specified in Microcontroller and Embedded Systems Laboratory as prescribed by VTU, Belagavi.

**Departments of CSE**

## ACKNOWLEDGMENT

A material of this scope would not have been possible without the contribution of many people. We express our sincere gratitude to Dr. R N Shetty, Chairman, RNS Group of Companies for his magnanimous support in all our endeavors.

We are grateful to Dr. Ramesh Babu H.S, Principal, RNSIT, Dr. P Kiran, HOD, CSE for extending their constant encouragement and support.

Our heartfelt thanks to Ms. Vidya Y, Ms. Anupama and Ms. Deepa Shetty for their unparalleled contribution throughout the preparation of this comprehensive manual. We also acknowledge our colleagues for their timely suggestions and unconditional support.

**Departments of CSE**

# TABLE OF CONTENTS

# SYALLABUS

## SEMESTER – VI

## FULLSTACK DEVELOPMENT LABORATORY

| | |
|---|---|
| **Course Code – 21CS62** | **CIE Marks - 50** |
| **Number of Contact Hours/Week -3:0:2:0** | **SEE Marks - 50** |
| **Total Number of Lab Contact Hours – 40t+20P** | **Exam Hours - 03** |

Course outcome (Course Skill Set)
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

**Assessment Details (both CIE and SEE)**

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.

**Continuous Internal Evaluation:**

Three Unit Tests each of **20 Marks (duration 01 hour)**

1. First test at the end of 5th week of the semester
2. Second test at the end of the 10th week of the semester
3. Third test at the end of the 15th week of the semester

Two assignments each of **10 Marks**

4. First assignment at the end of 4th week of the semester
5. Second assignment at the end of 9th week of the semester

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)

## Laboratory Components

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.
5. Develop a simple Django app that displays an unordered list of fruits and ordered list of selected students for an event.
6. 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.
7. 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.
8. For student and course models created in Lab experiment for Module2, register admin interfaces, perform migrations and illustrate data entry through admin forms.
9. Develop a model form for student that contains his topic chosen for project, languages used and duration with a model called project.
10. For students' enrolment developed in Module 2, create a generic class view which displays list of students and detail view that displays student details for any selected student in the list.
11. Develop example Django app that performs CSV and PDF generation for any models created in previous laboratory component.
12. Develop a registration page for student enrolment as done in Module 2 but without page refresh using AJAX.
13. Develop a search application in Django using AJAX that displays courses enrolled by a student being searched.

# Experiment-01

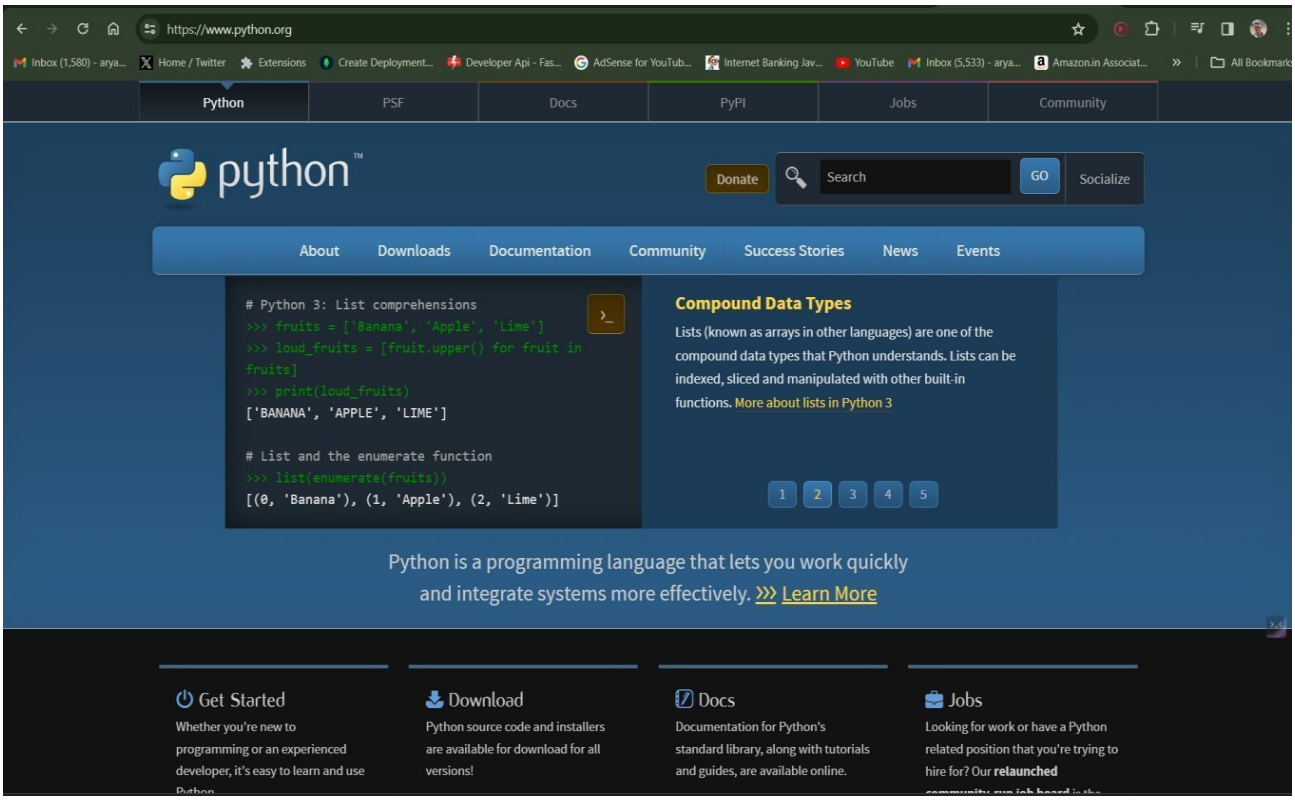
Installation of Python, Django and Visual Studio code editors can be demonstrated.

## Python

- Python is a popular high-level, general-use programming language.
- Python is a programming language that enables rapid development as well as more effective system integration.
- Python has two main different versions: Python 2 and Python 3. Both are really different.

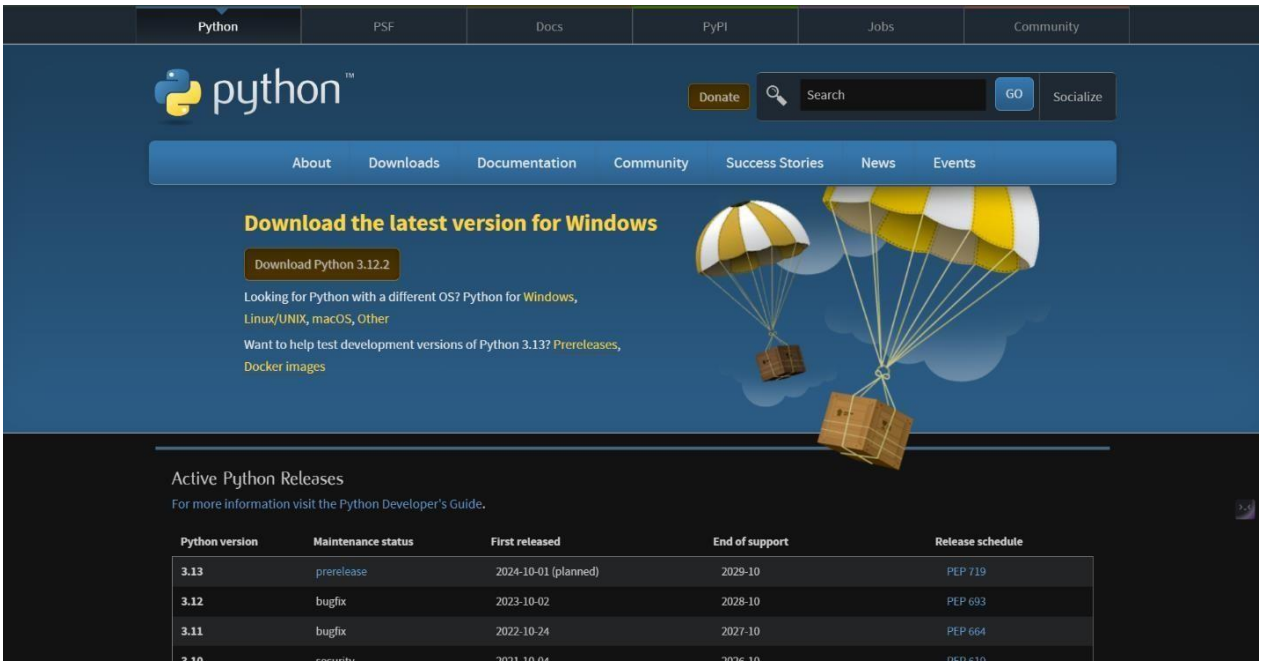
### Here are the steps you can follow to download Python: Steps to Download & Install Python

Visit the link ***https://www.python.org*** to download the latest release of Python.

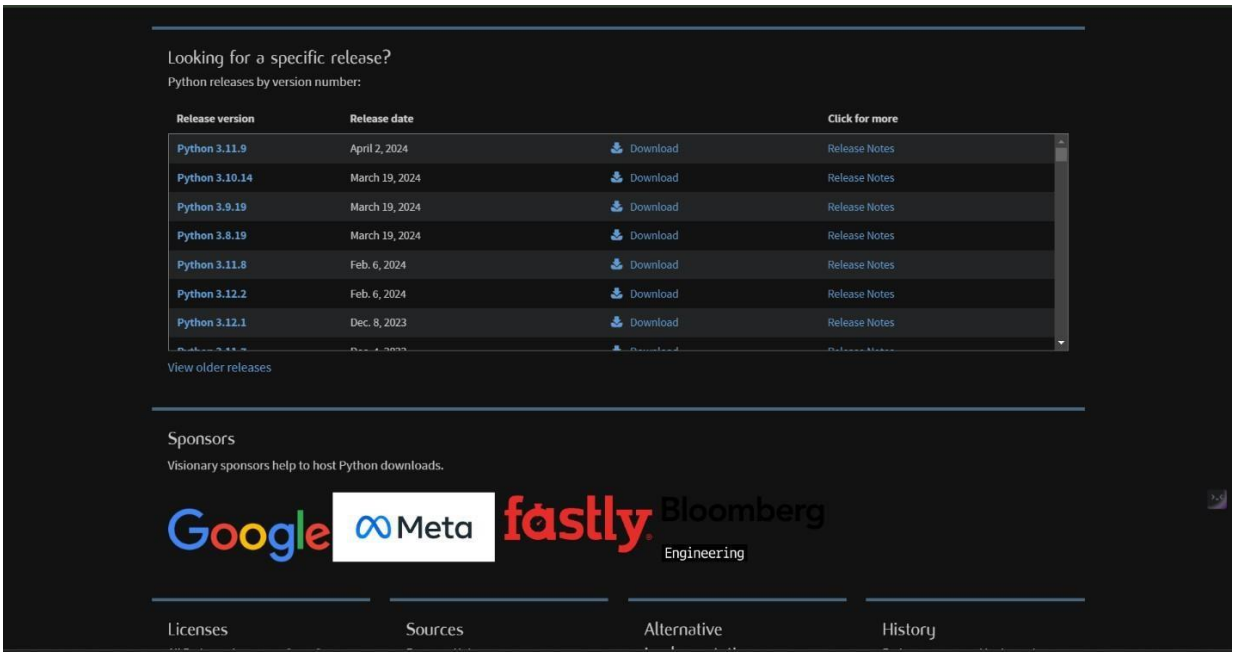


**Step - 1: Select the Python's version to download.**

Click on the download button to download the exe file of Python.

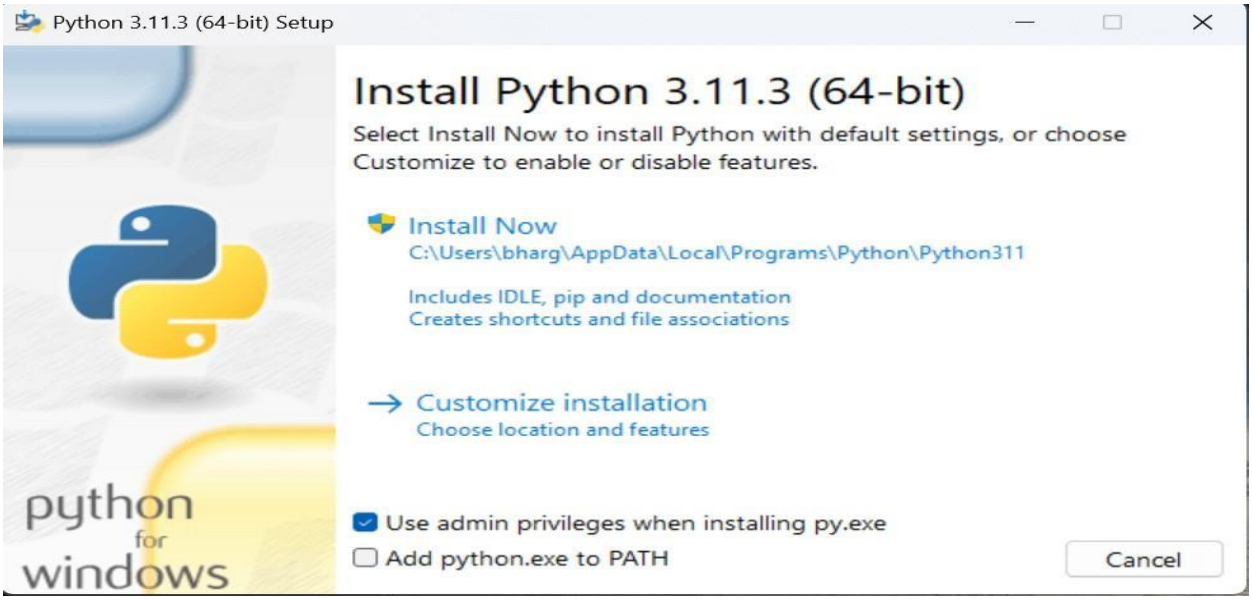



If in case you want to download the specific version of Python. Then, you can scroll down further below to see different versions from 2 and 3 respectively. Click on download button right next to the version number you want to download.



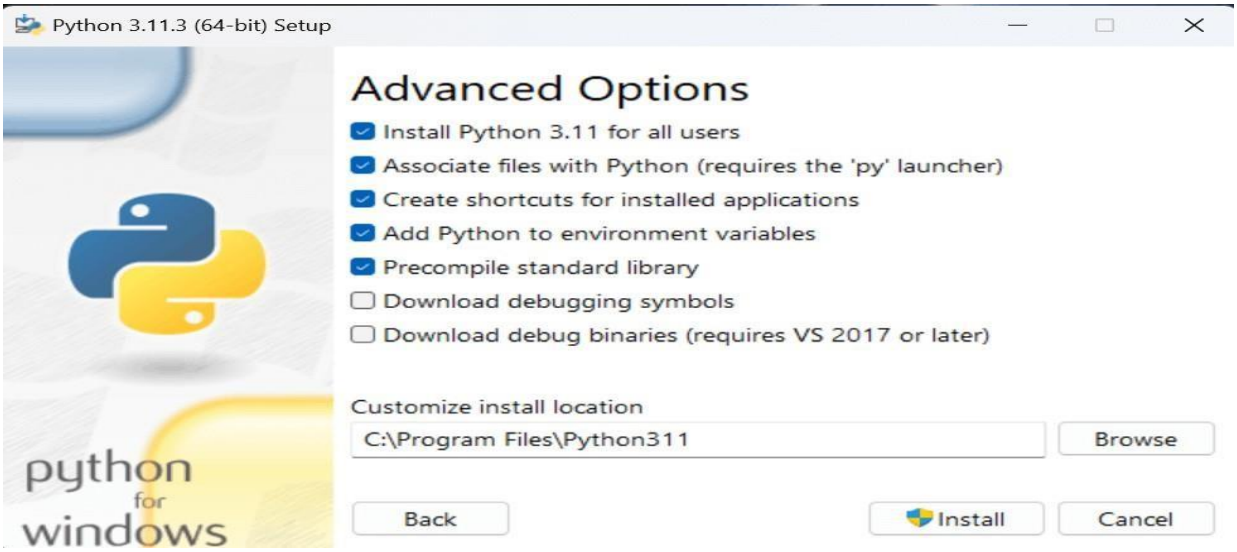
**Step - 2: Click on the Install Now**

Double-click the executable file, which is downloaded.

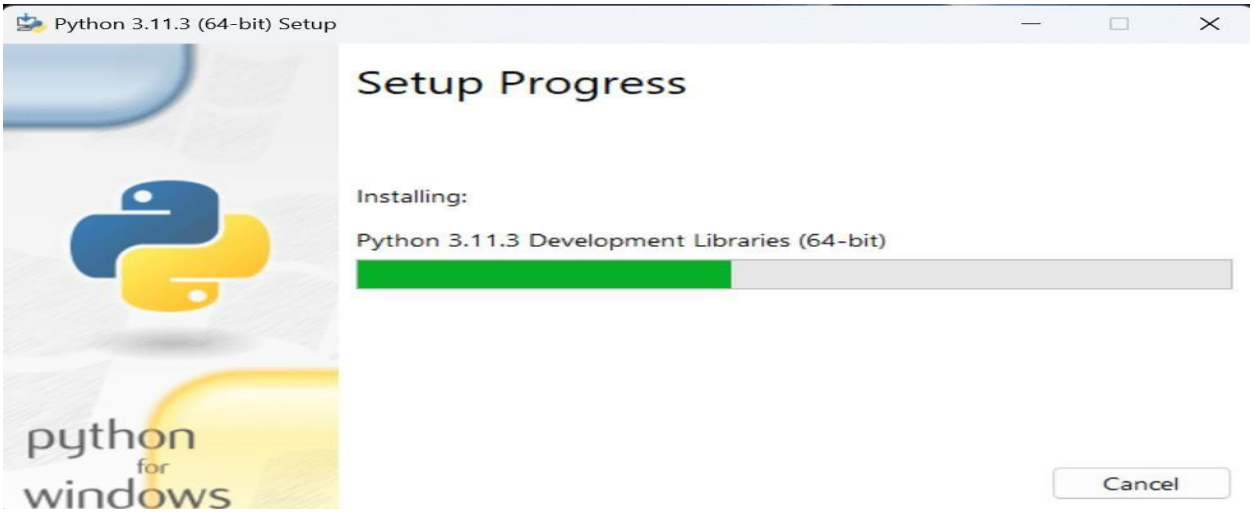



The following window will open. Click on the Add Path check box, it will set the Python path automatically.

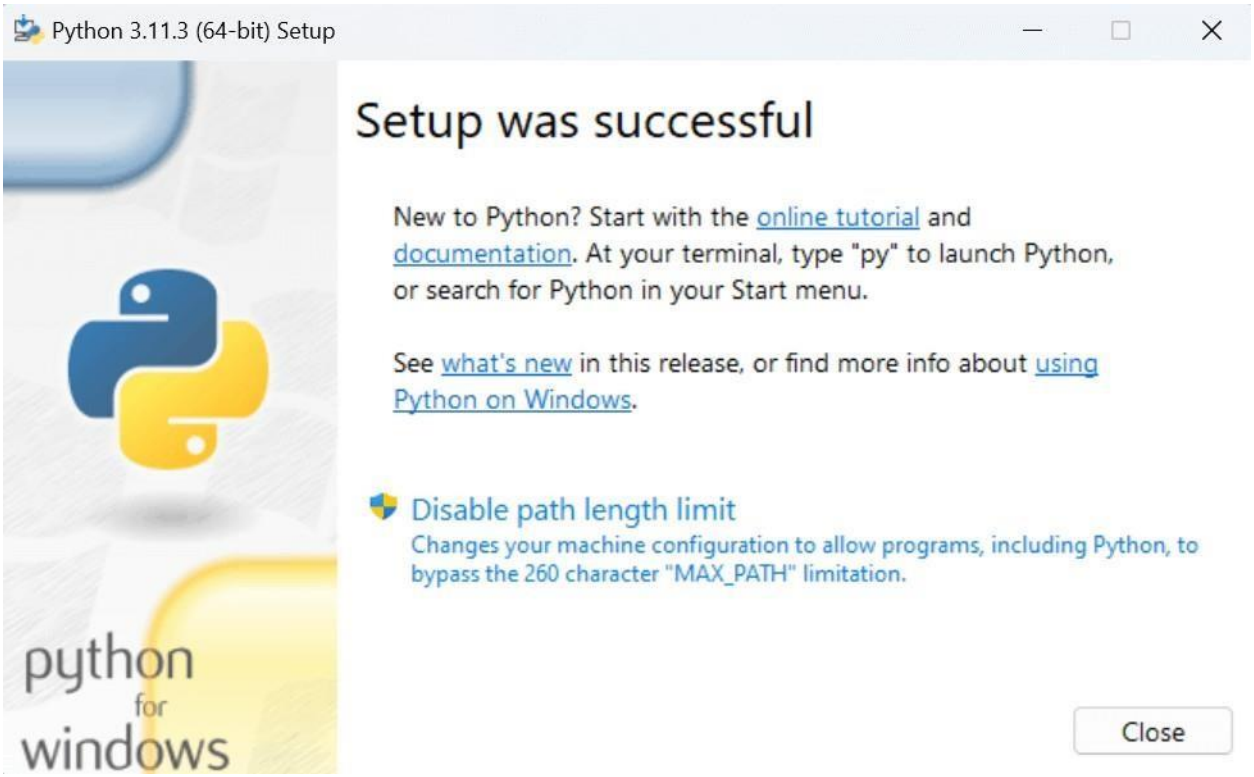
Now, Select Customize installation and proceed. We can also click on the customize installation to choose desired location and features. Other important thing is installing launcher for the all user must be checked.



**Step - 3 Installation in Process**



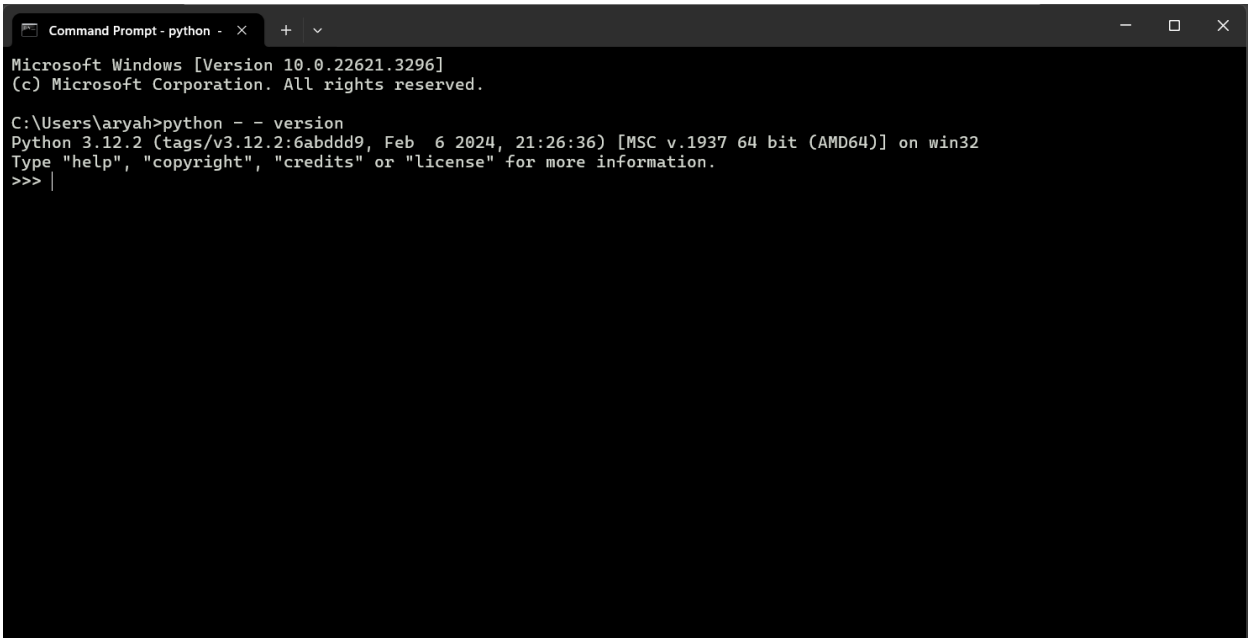

The set up is in progress. All the python libraries, packages, and other python default files will be installed in our system. Once the installation is successful, the following page will appear saying " Setup was successful ".



**Step - 4: Verifying the Python Installation**

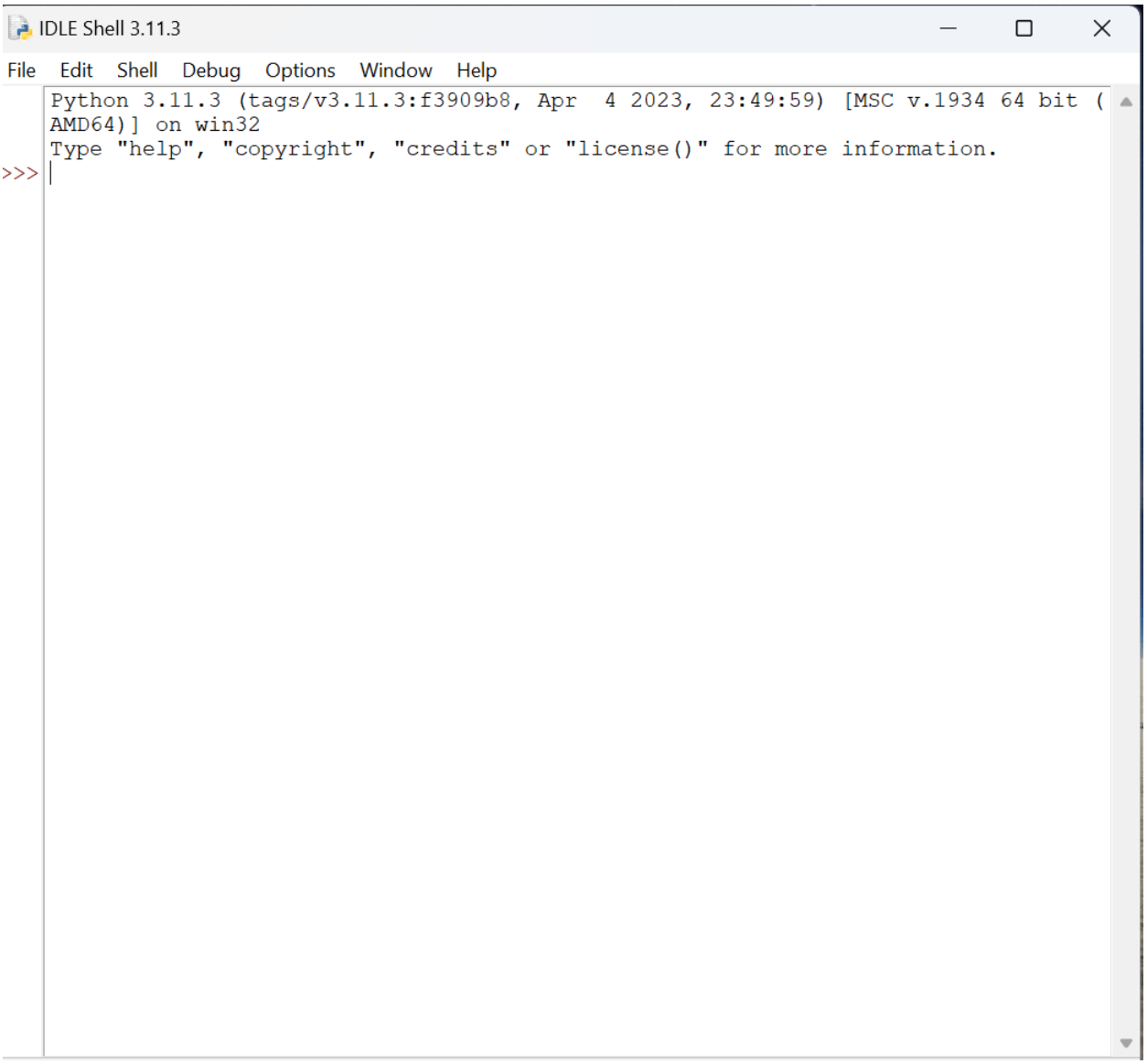
To verify whether the python is installed or not in our system, we have to do the following.

- Go to "Start" button, and search " cmd ".
- Then type, " python - - version ".
- If python is successfully installed, then we can see the version of the python installed.
- If not installed, then it will print the error as " 'python' is not recognized as an internal or external command, operable program or batch file. ".




We are ready to work with the Python.

**Step - 5: Opening idle**

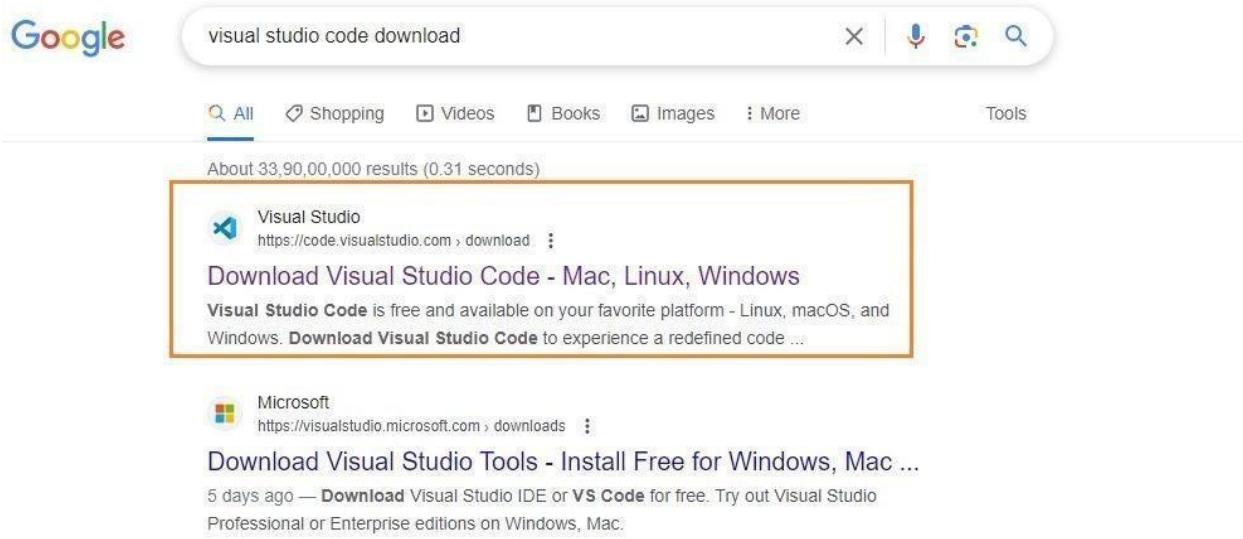



Now, to work on our first python program, we will go the interactive interpreter prompt(idle). To open this, go to "Start" and type idle. Then, click on open to start working on idle.

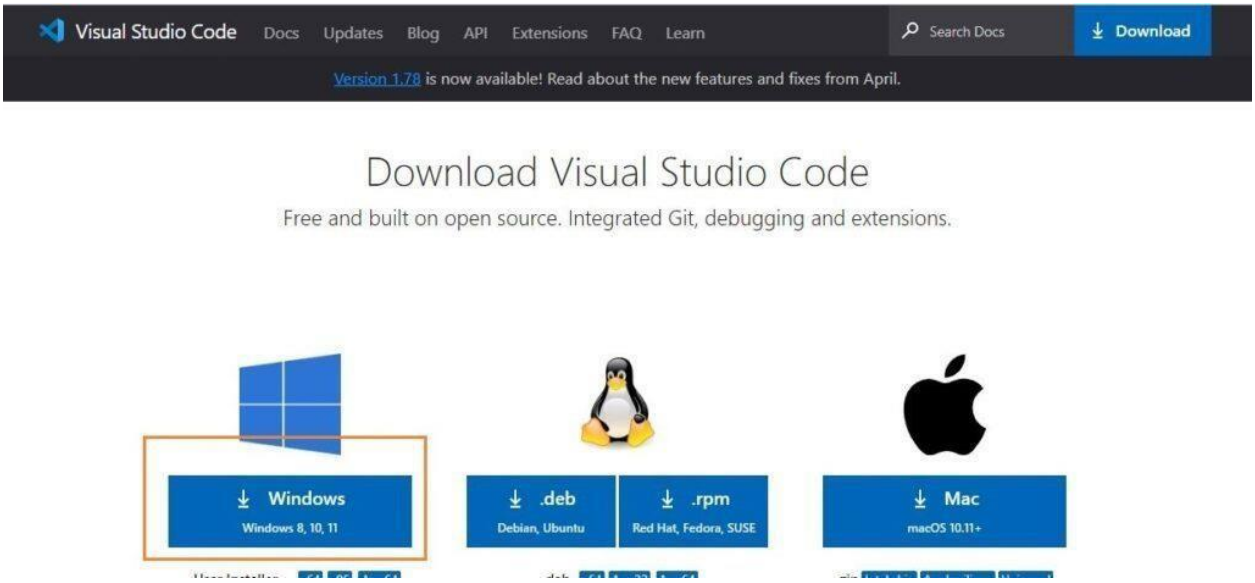
## Here are the steps you can follow: Steps to Download & Install VS Code

**Step – 1: Open Google and type Visual Studio Code download in the search bar.**


**Step – 2: Click on the below highlighted link for any OS.**

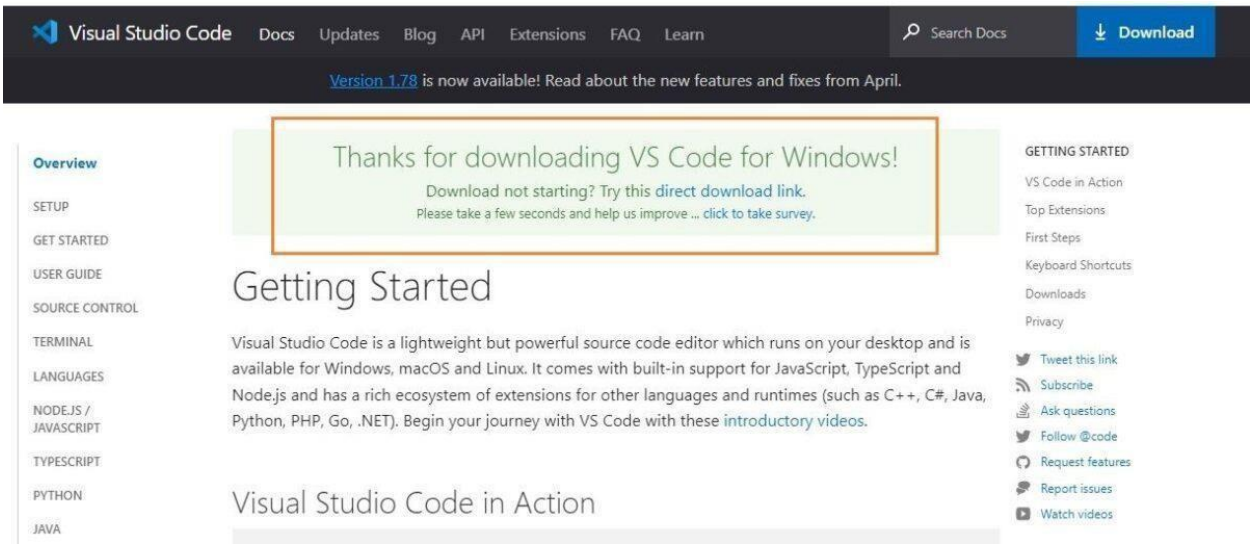


**Step – 3: Now, select the respective OS. In this case we are selecting Windows.**

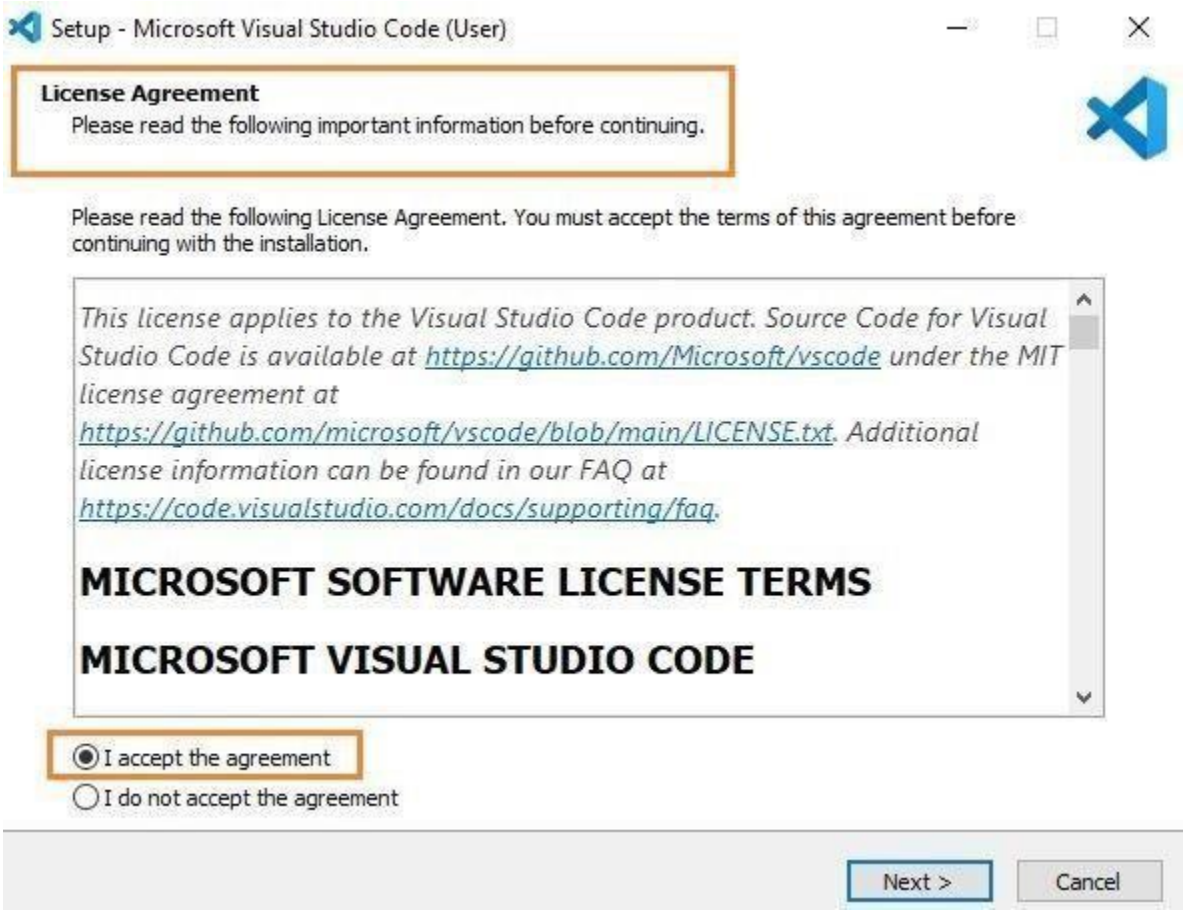



**Step – 4: The file will be downloaded onto your system. Open the file and then click on Install.**

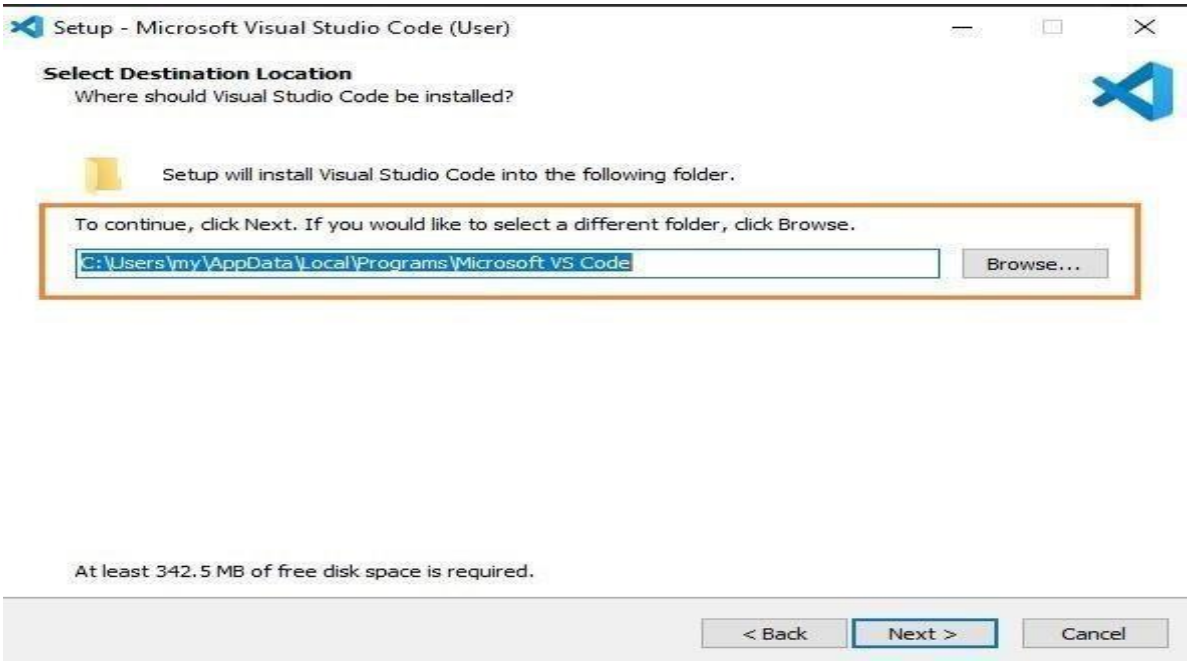
After downloading the VS Code file, the official site will display a Thanks message for downloading the file.



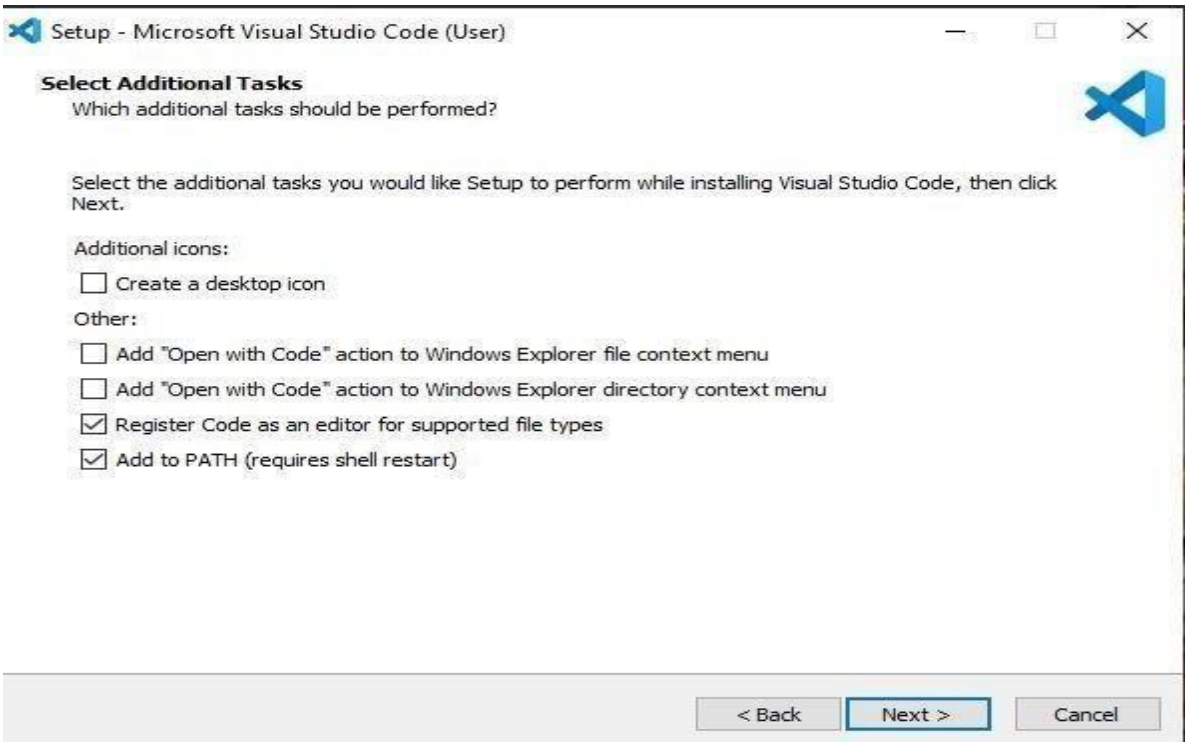
**Step – 5: Now accept the license agreement.**



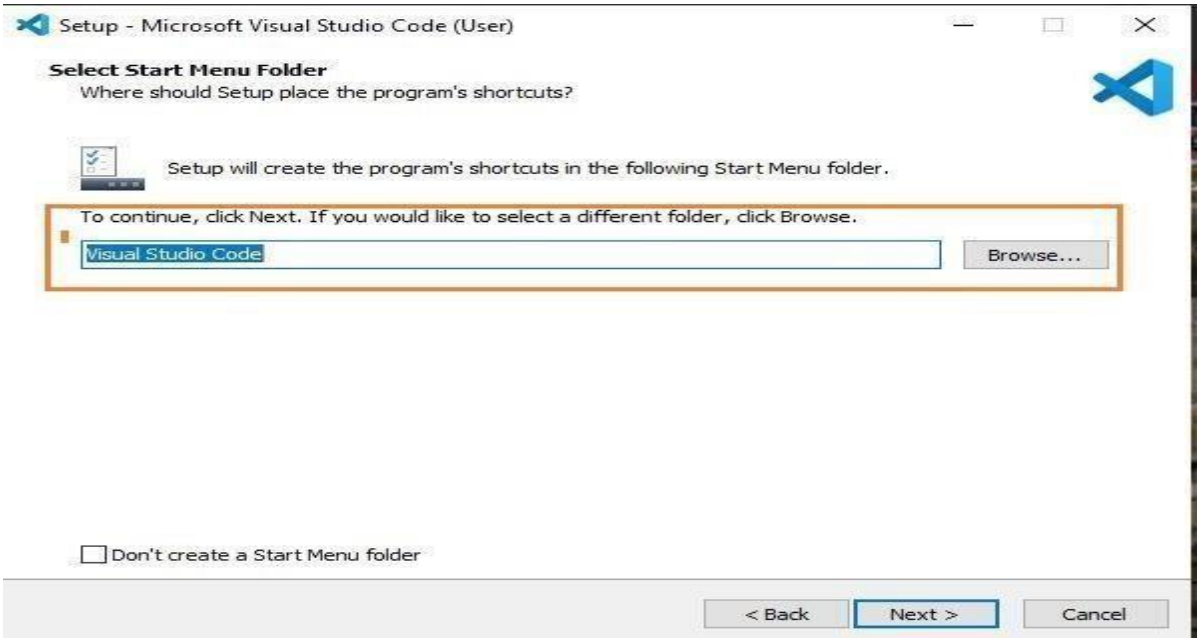
**Step – 6: Then it prompts for the file location, where you want to save the VS Code file.** Browse the location and then click on Next.



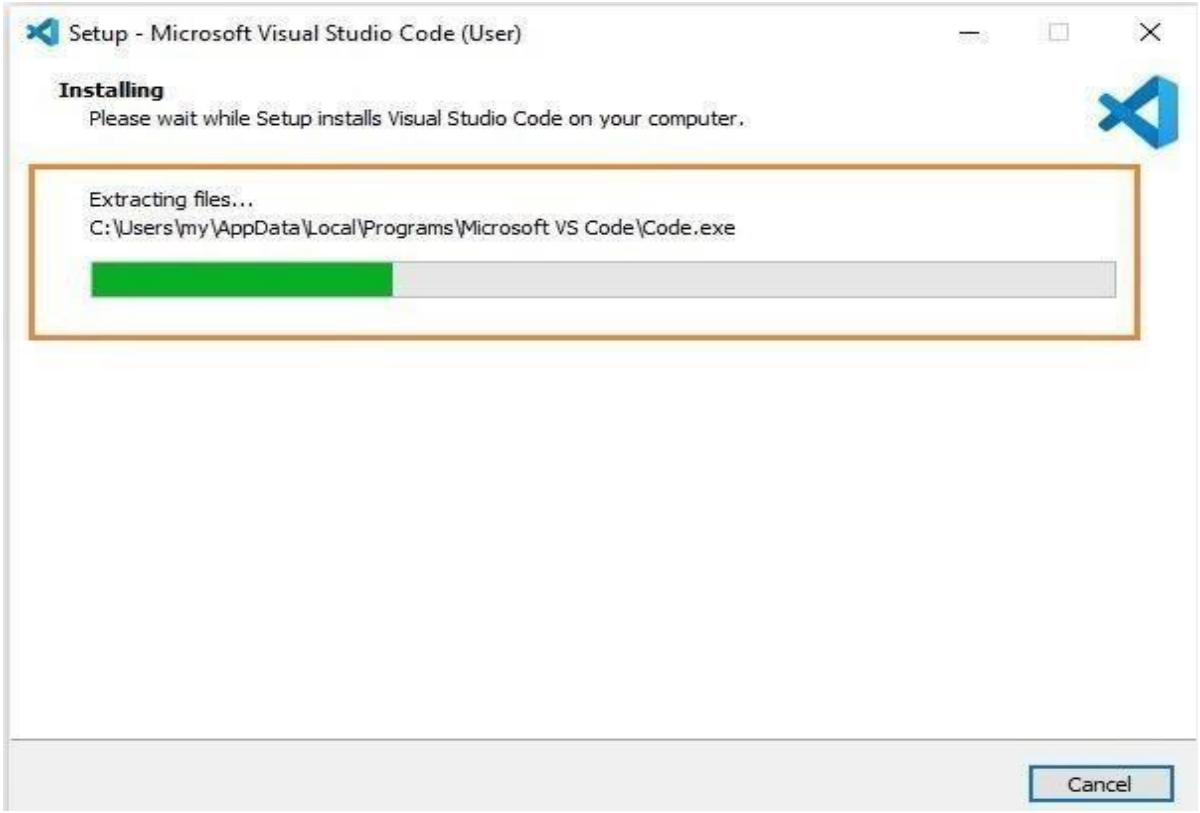

**Step – 7: Next, you see the prompt for the additional task which we want the VS Code to perform.** At this step, choose the default settings and then click on next.



**Step – 8: The next prompt is how you want the VS Code on your startup. Change according to your convenience and click on Next.**

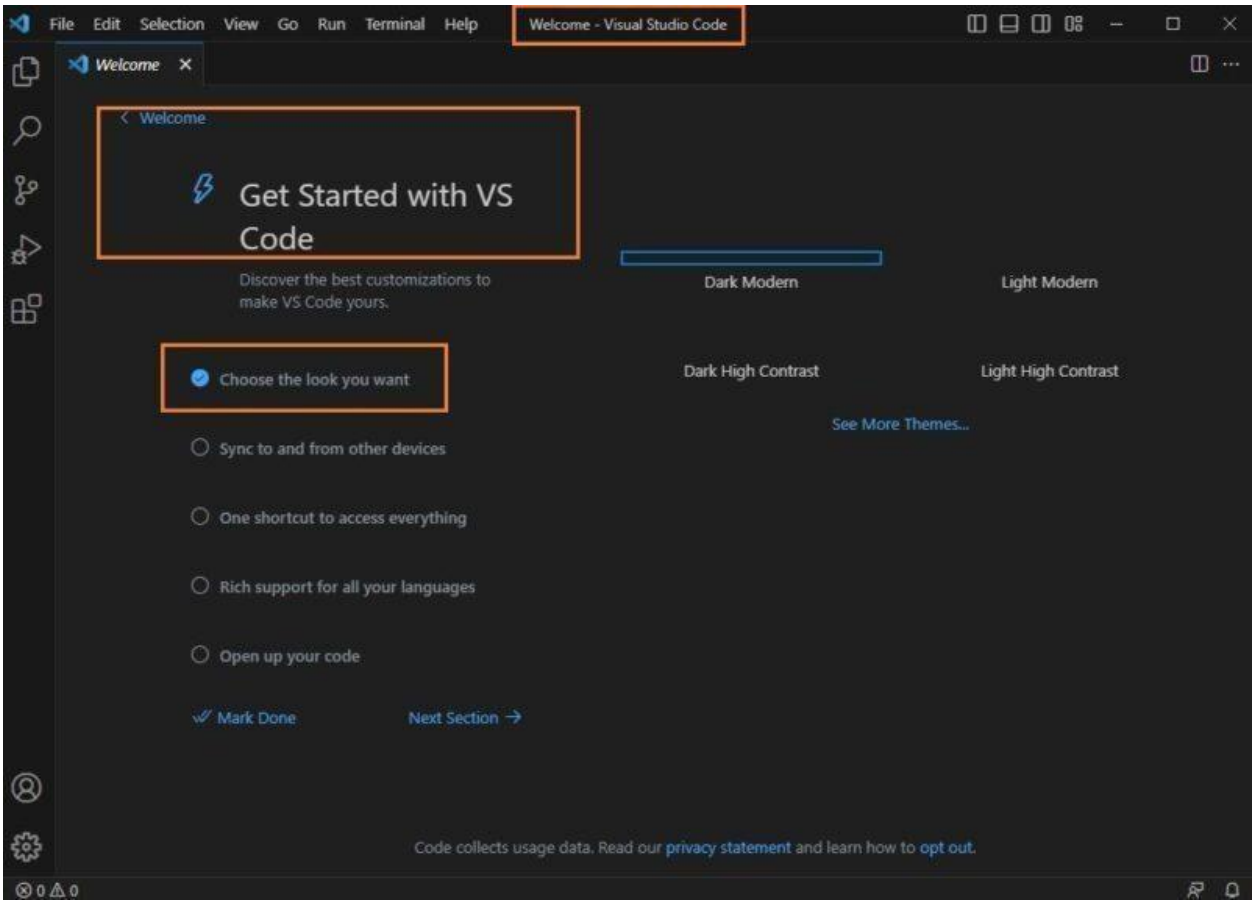



**Step – 9: The installation of VS Code will now begin.**



**Step – 10: At this step, we have completed installing VS Code, click on Finish.**

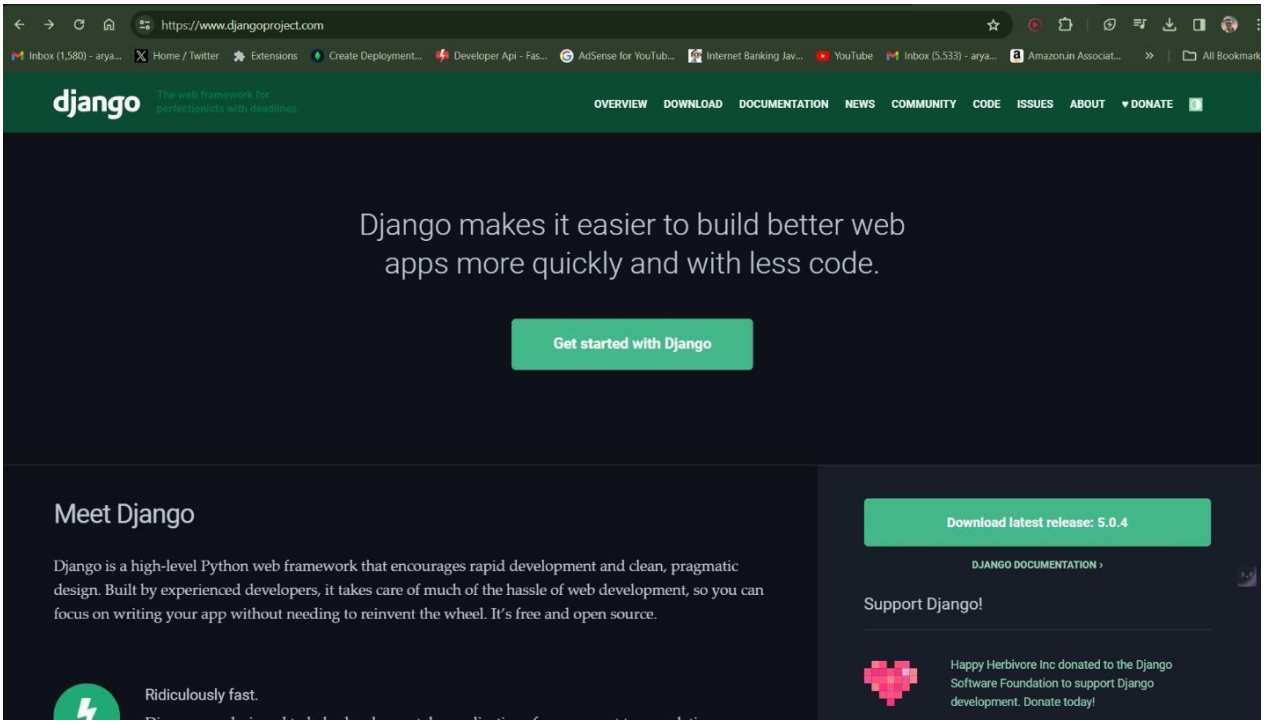
**Step – 11: Now that VS Code installation is successful, the page appears as below:**

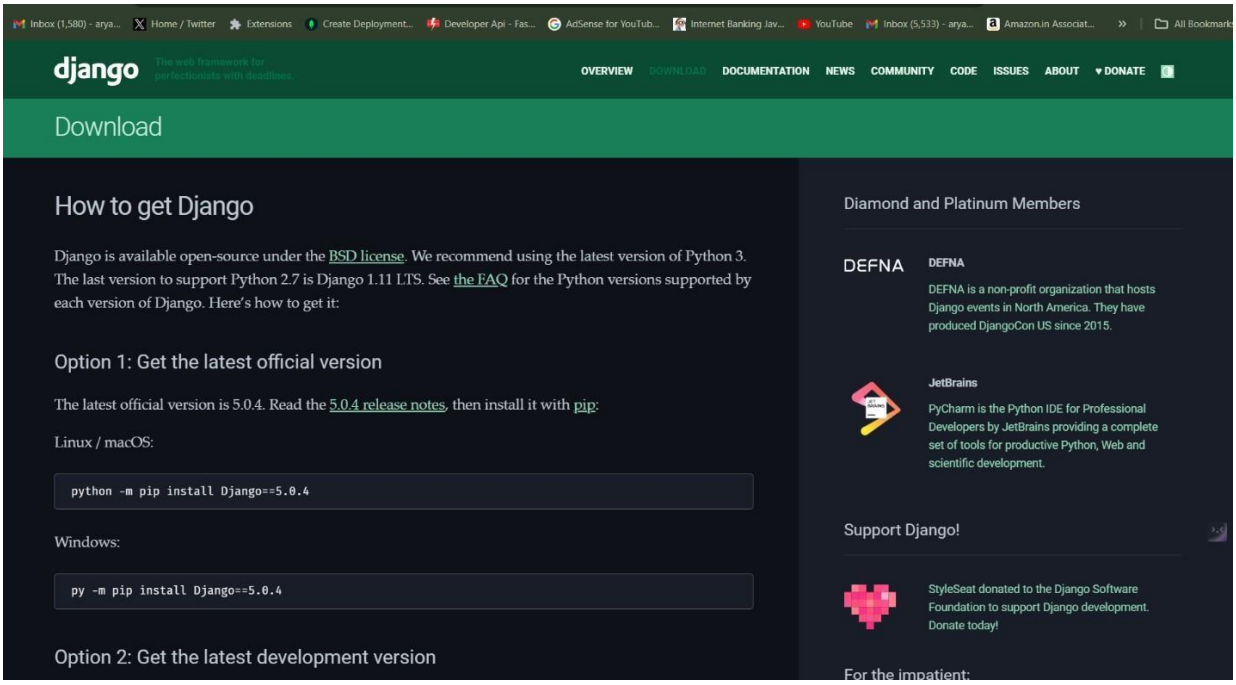



We can change the look as per our choice and continue working on it.

## Here are the steps you can follow: Steps to Install Django

To install Django, first visit to **Django official site (https://www.djangoproject.com)** and download Django by clicking on the download section.



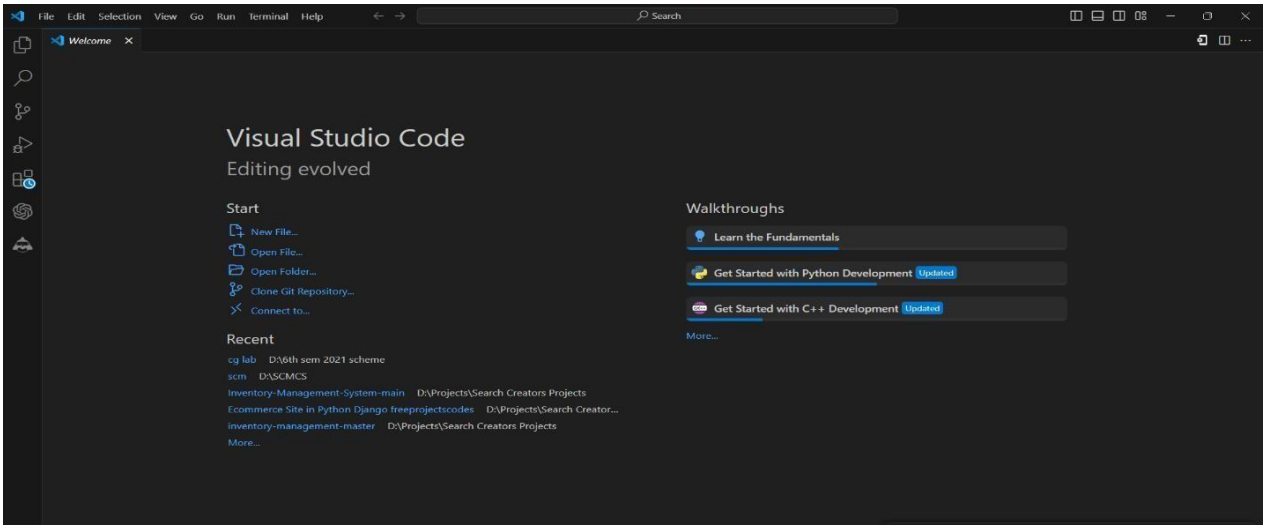



# Experiment-02

Creation of virtual environment, Django project and App should be demonstrated.

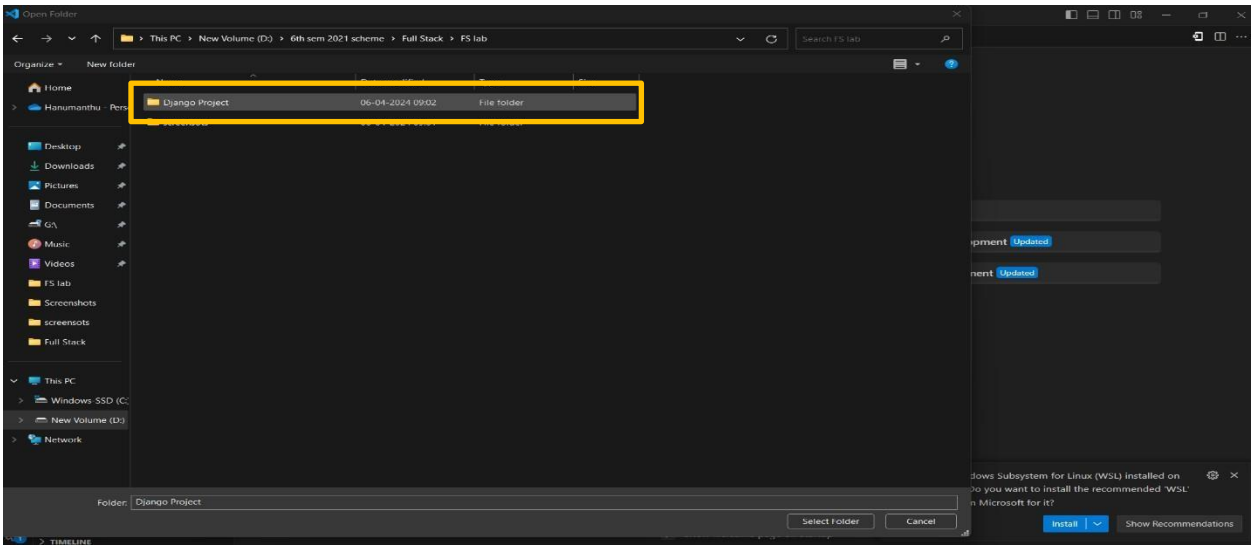
**Here are the steps you can follow to create virtual environment: Steps to Create Virtual Environment, Django Project and App**

### Step-01: Open Visual Studio Code



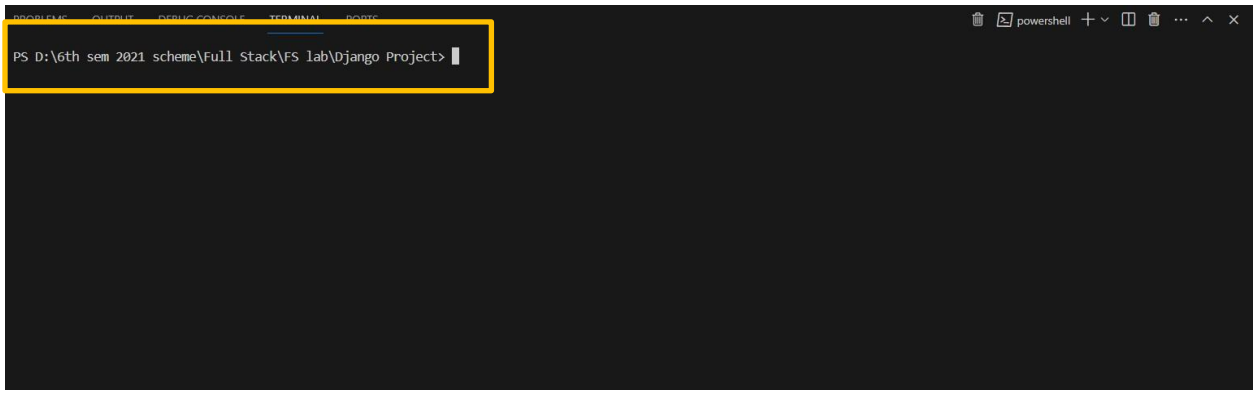

### Step-02: Create a new folder for your project

In VS Code, go to **File > Open...** and create a new folder or select an existing folder where you want to create your Django project.



**Step-03: Open the integrated terminal**

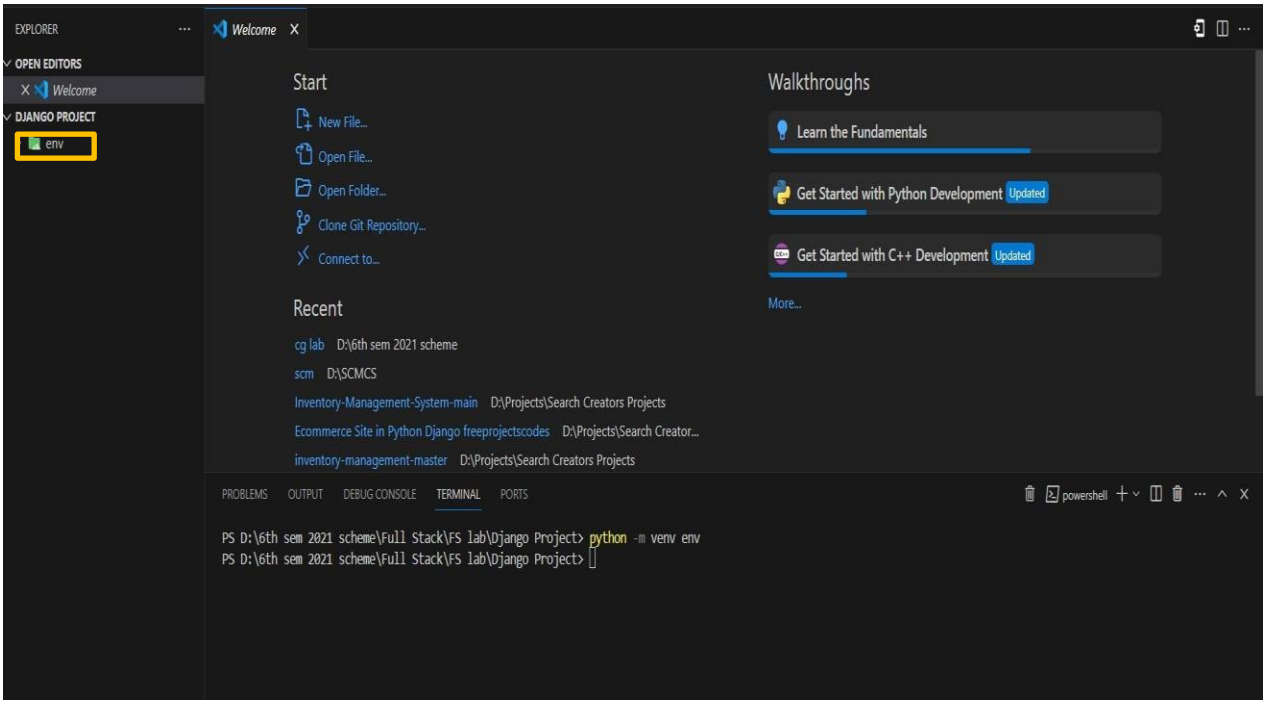
In VS Code, go to View > Terminal or use the keyboard shortcut Ctrl+`` (Windows/Linux) or Cmd+`' (macOS) to open the integrated terminal.



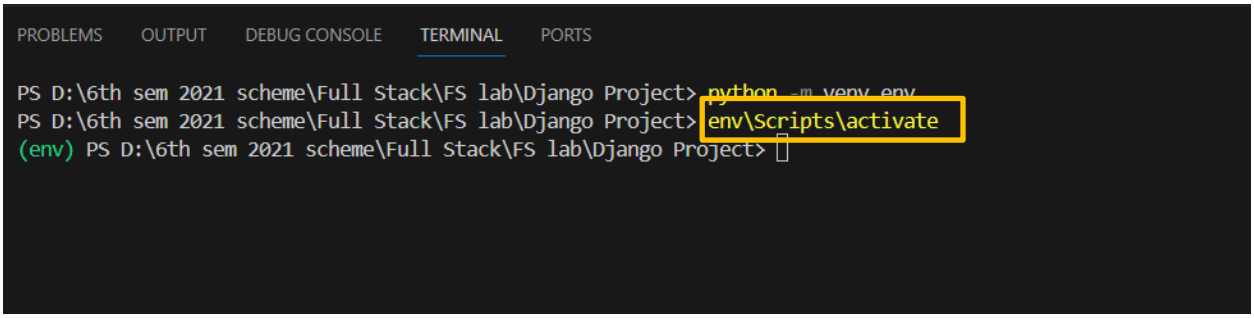

**Step-03: Create a virtual environment**

In the terminal, run the following command to create a new virtual environment:

**python -m venv env**

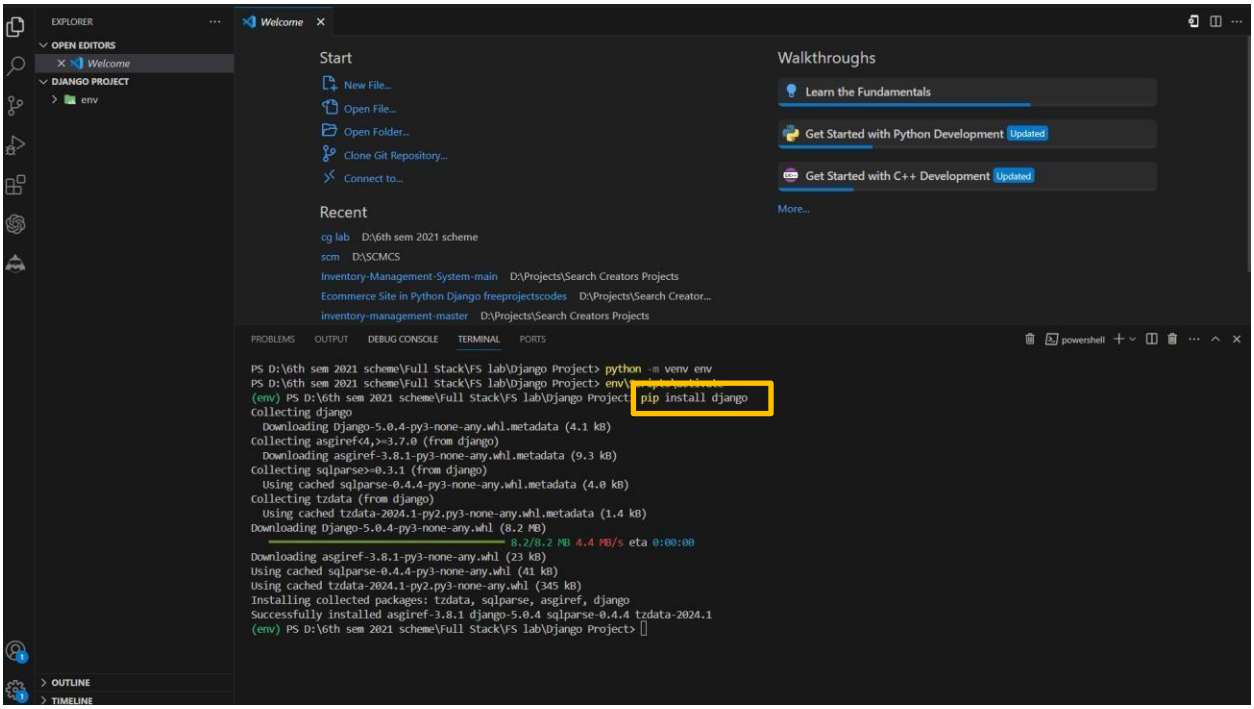


### Step-04: Activate the virtual environment




### Step-05: Install Django

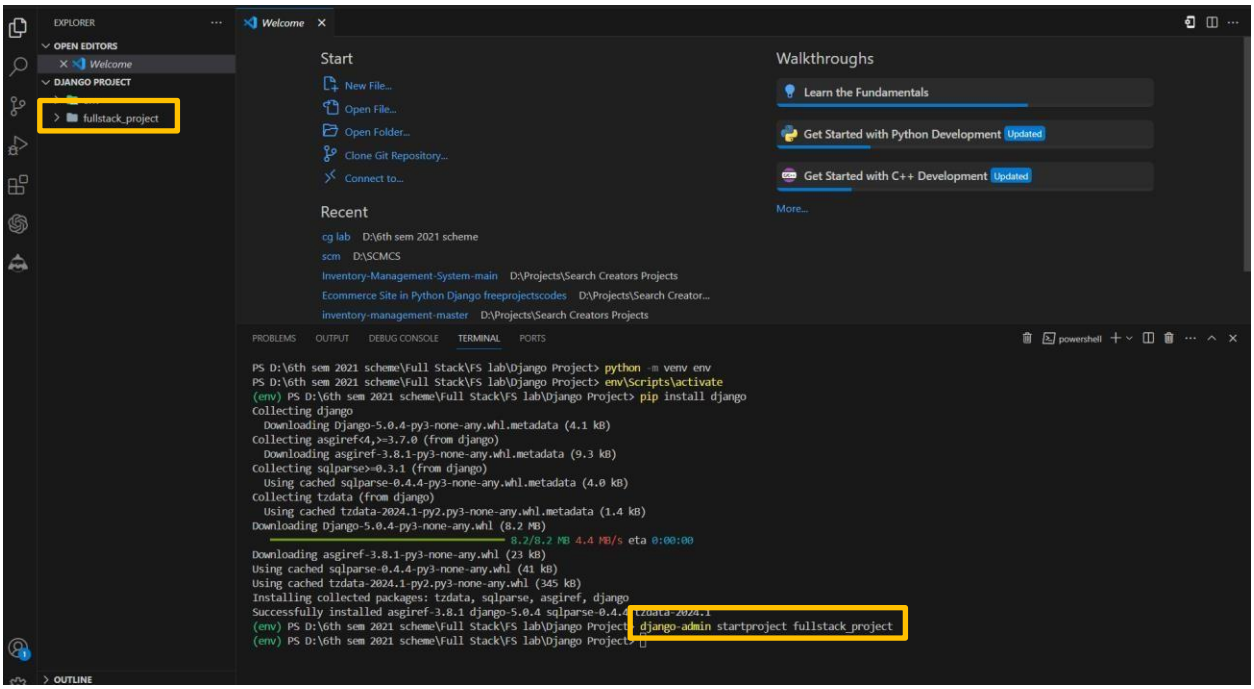
With the virtual environment active, install Django by running the following command in the terminal: **pip install django**



**Step-06: Create a new Django project**

Run the following command to create a new Django project:

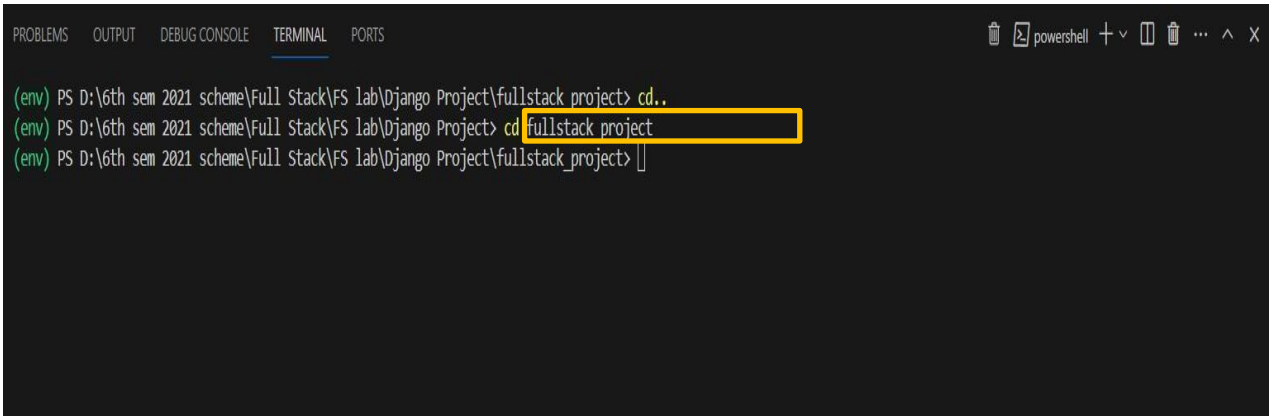
**django-admin startproject myproject [change Project name as You Desired]**




**Step-07: Create a new Django app**

In the VS Code terminal, navigate to your project directory:

**cd myproject [change Project name as You Desired]**



Create a new Django app by running the following command

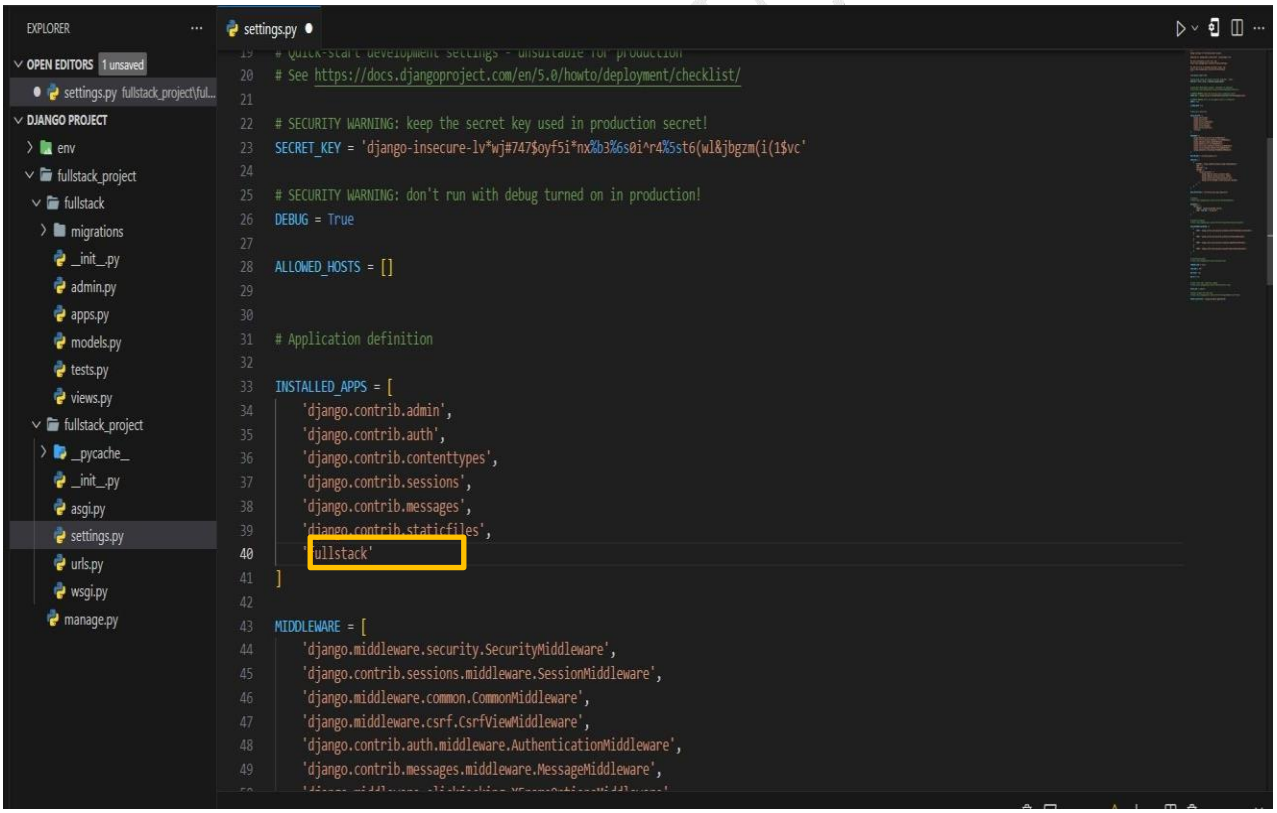
**python manage.py startapp myapp**

[Replace myapp with the name you want to give to your app.]

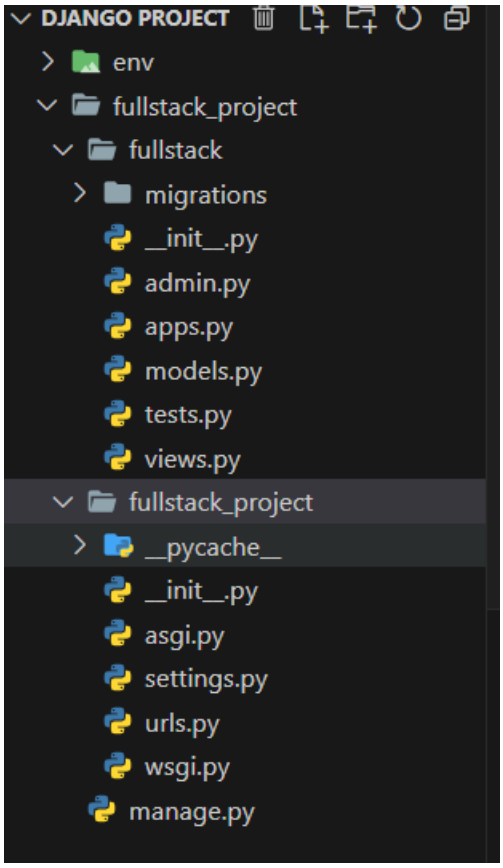

**Step-08: Add the app to the INSTALLED_APPS list**

In the VS Code file explorer, locate the settings.py file (**usually located in the myproject directory**) and open it.

Locate the **INSTALLED_APPS** list and add the name of your new app to the list



**Here is the Django project structure that you wanted to create**




after creating your Django app, the next step is to **create database migrations** for your app's models (if you have any defined). Here's how you can do that using the **python manage.py makemigrations** command in Visual Studio Code (VS Code)

```
(env) PS D:\6th sem 2021 scheme\Full Stack\FS lab\Django Project\fullstack_project> python manage.py makemigrations
No changes detected
```

**Step-08**: **Apply the migrations**

Once you've reviewed the migration files and are ready to apply the changes to your database, run the following command in the terminal:

```
(env) PS D:\6th sem 2021 scheme\Full Stack\FS lab\Django Project\fullstack_project> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
```

**Step-09**: **Run Your Project**

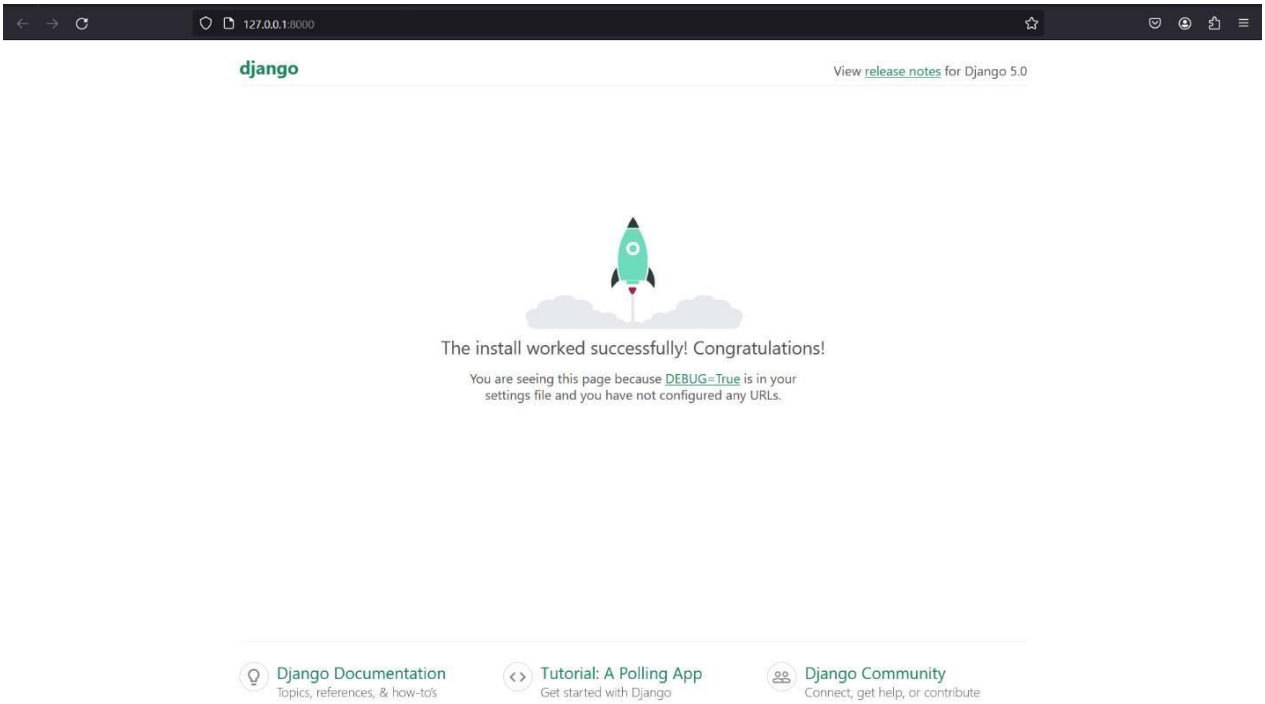
Here's how you can run the Django development server using the **python manage.py runserver** command in Visual Studio Code (VS Code)

```
(env) PS D:\6th sem 2021 scheme\Full Stack\FS lab\Django Project\fullstack_project> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
April 06, 2024 - 11:23:06
Django version 5.0.4, using settings 'fullstack_project.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

**Check the server output**

The terminal will display the URL where the development server is running, typically **http://127.0.0.1:8000/.** It will also show any startup logs or warnings, if any. Open the development server in your browser Copy the URL from the terminal output (e.g., http://127.0.0.1:8000/) and paste it into your web browser's address bar.

Here Displays the Development Server After Running Server



# Experiment-03

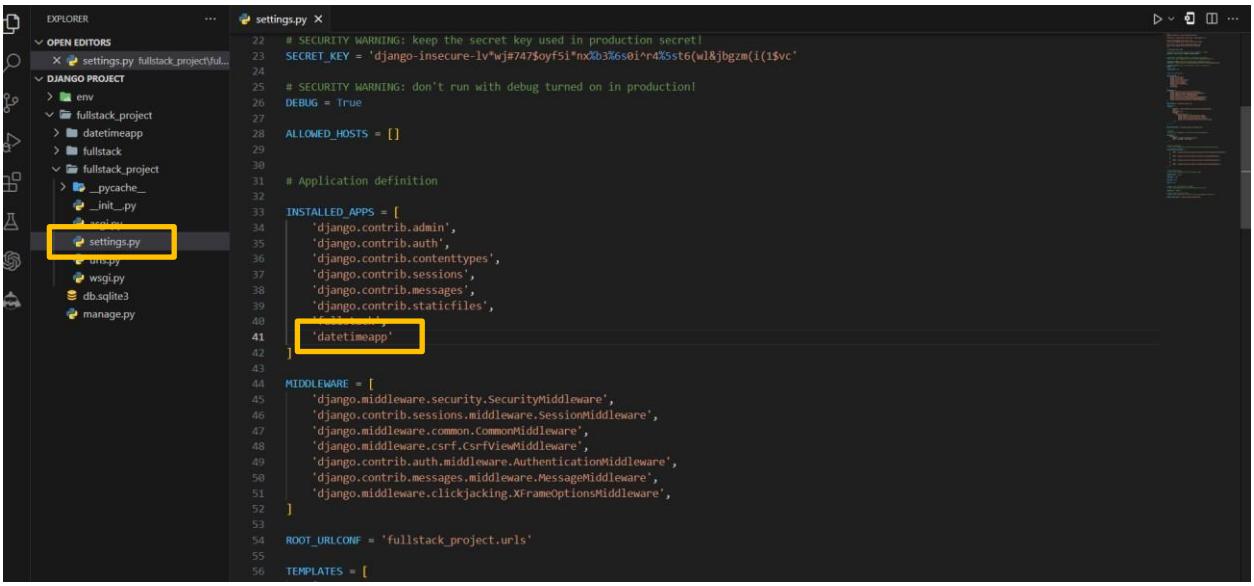
Develop a Django app that displays current date and time in server.

```
(env) PS D:\6th sem 2021 scheme\Full Stack\FS lab\Django Project\fullstack_project> python manage.py startapp datetimeapp
(env) PS D:\6th sem 2021 scheme\Full Stack\FS lab\Django Project\fullstack_project>
```

**Step-02: Add the app to INSTALLED_APPS**

Open the settings.py file in your project's directory (e.g., myproject/settings.py).

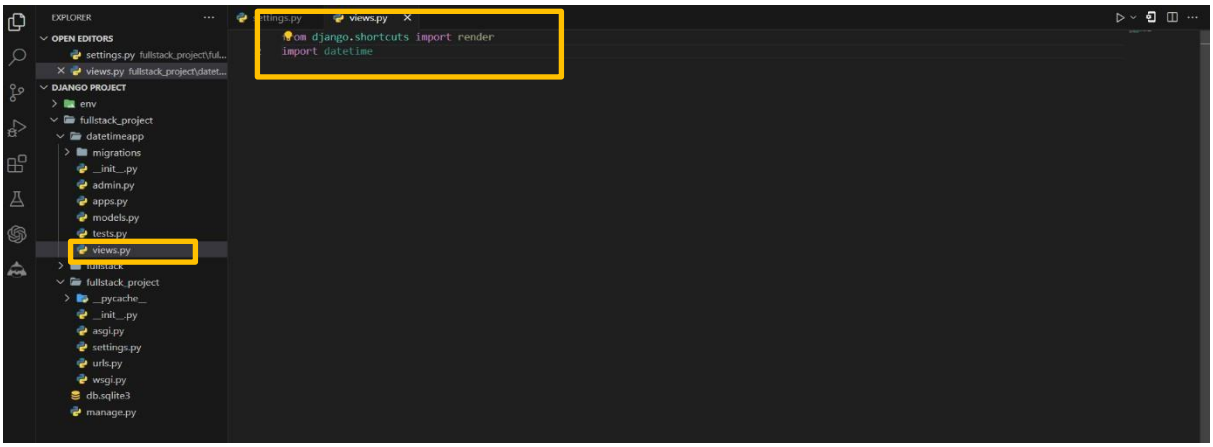
Locatethe **INSTALLED_APPS** list and add the name of your new app to the list:




**Step-01**: **This app will be created in the Django project we made earlier.**

**Step-03: Create a view function**

- Open the views.py file in your Django app's directory (e.g., **datetimeapp/views.py**).
- Import the necessary modules at the top of the file



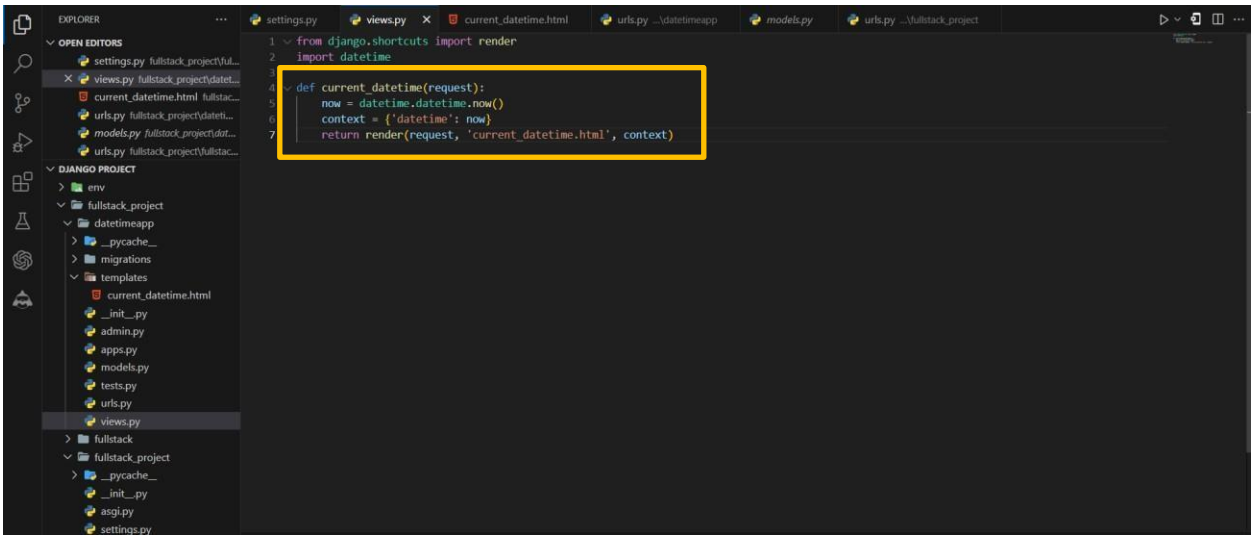

- Create a new **view function** that will handle the request and render the date and time:

**def current_datetime(request):**

**now = datetime.datetime.now ()**

**context = {'datetime': now}**

**return render (request, 'current_datetime.html', context)**



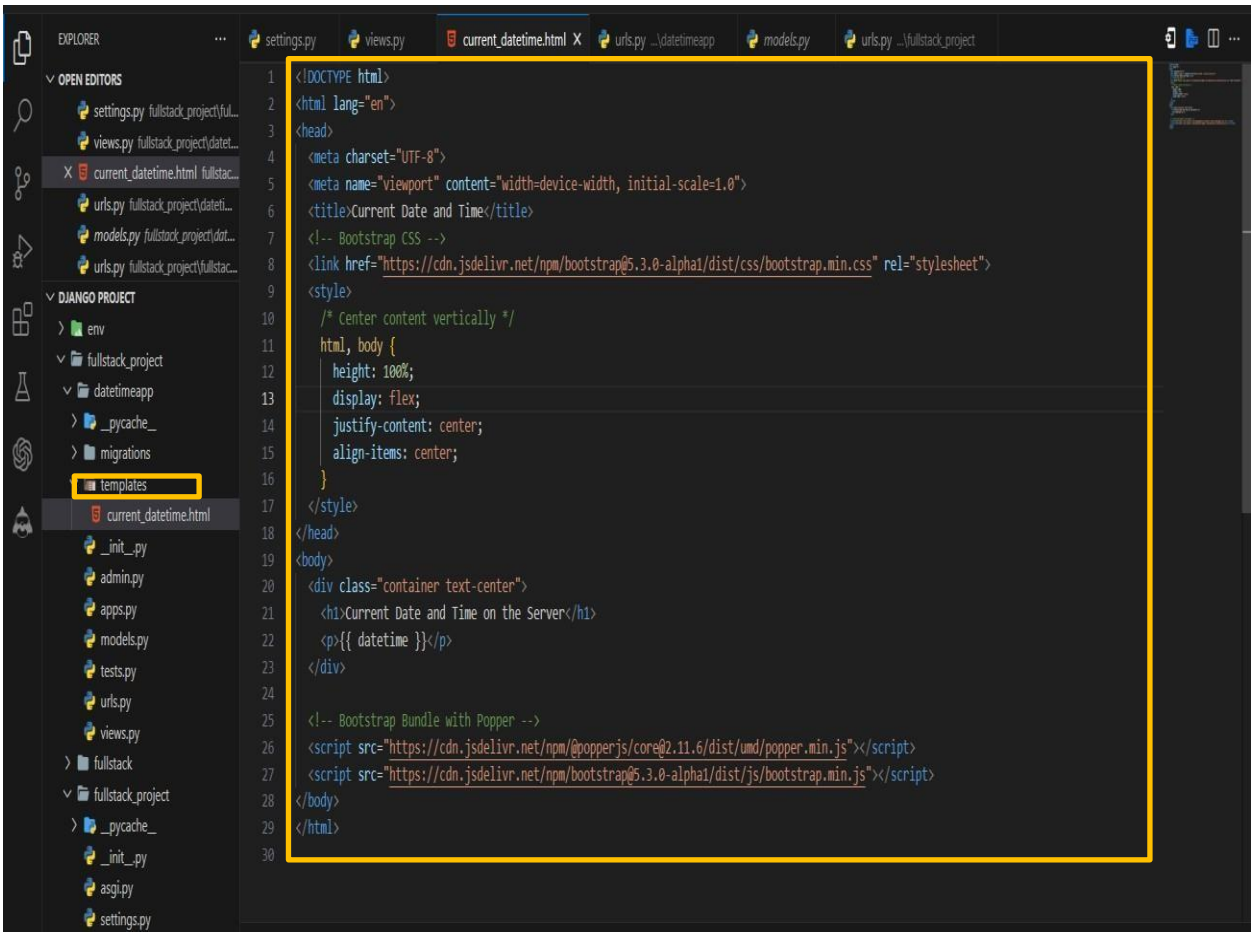
**Step-04: Create a template**

- In your app's directory **(e.g., datetimeapp),** create a new folder named templates.
- Inside the templates folder, create another folder with the same name as your app (e.g., myapp).

- Inside the **datetimeapp folder**, create a new file named **current_datetime.html.**
- Open **current_datetime.html** and add the following code to display the current date and time:

```
<!DOCTYPE html>
<html lang="en">
<head><meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Current Date and Time</title>
  <!-- Bootstrap CSS -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css" rel="stylesheet">
  <style>
    /* Center content vertically */
    html, body {
      height: 100%;
      display: flex;
      justify-content: center;
    align-items: center;
   }
  </style>
</head>
```

```
<body>
  <div class="container text-center">
    <h1>Current Date and Time on the Server</h1>
    <p>{{ datetime }}</p>
  </div>
  <!-- Bootstrap Bundle with Popper -->
  <script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd/popper.min.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-
alpha1/dist/js/bootstrap.min.js"></script>
</body>
</html>
```



**Step-05: Map the view function to a URL**

- Open the **urls.py** file in your Django app's directory (e.g., datetimeapp/urls.py).
- Import the view function at the top of the file
- Add a new URL pattern to the urlpatterns list

**from django. urls import path**

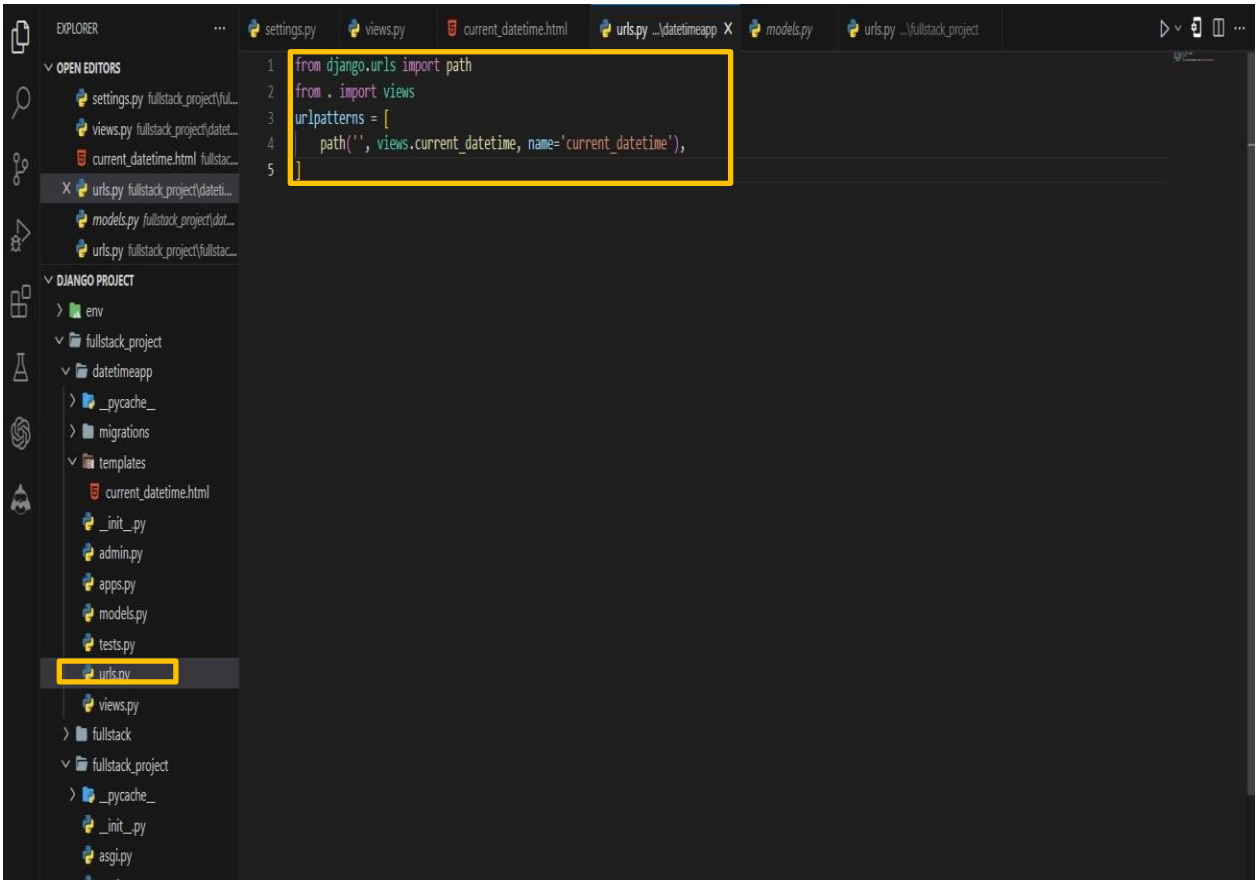
**from. import views**

**urlpatterns = [**

**path ('', views. current_datetime, name='current_datetime'),**

**]**

This maps the **current_datetime** view function to the root URL (/).



**Step-06: Include the app's URLs in the project's URL patterns**

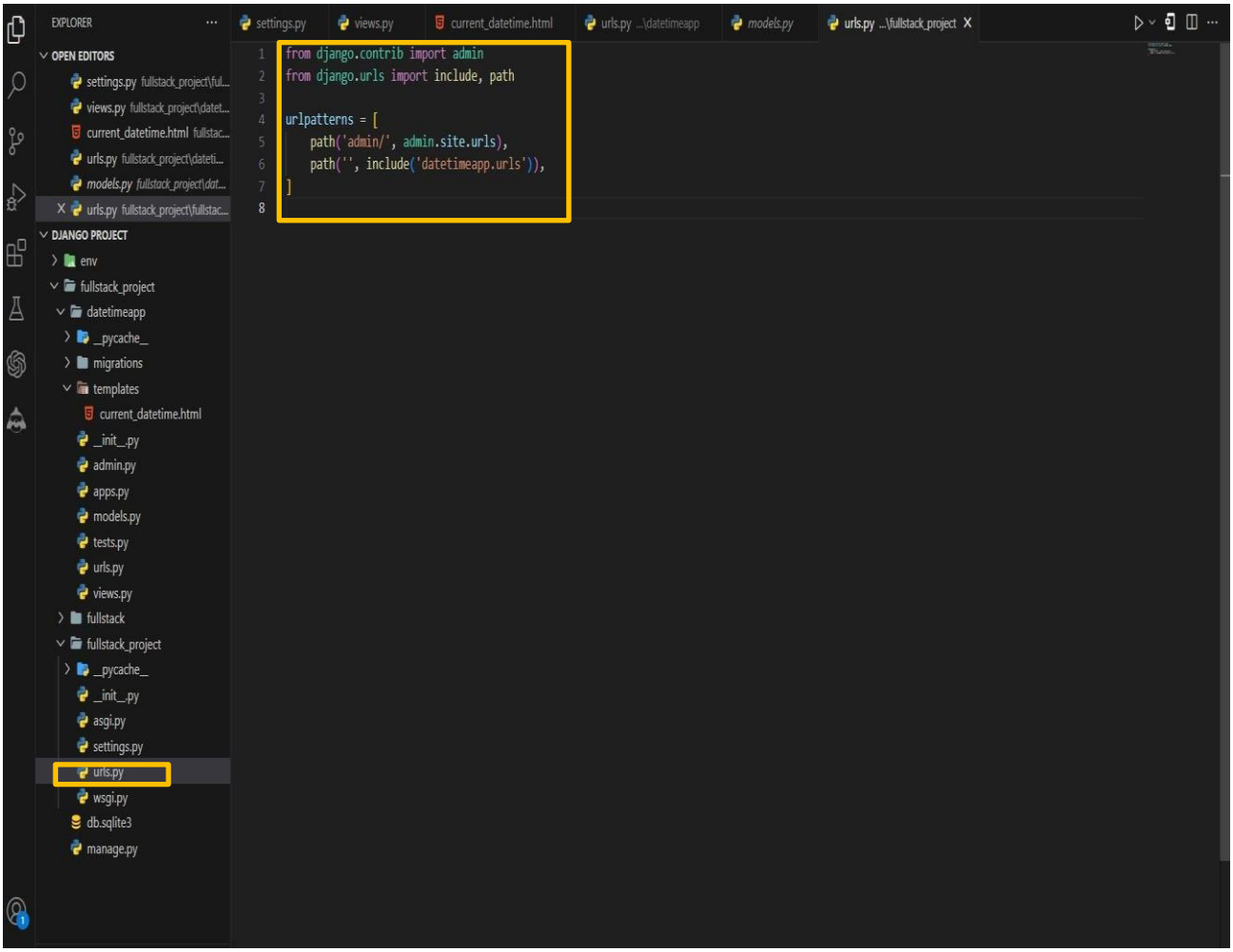
- Open the **urls.py** file in your project's directory **(e.g., fullstack_project/urls.py).**
- Import the include function from **django. urls** and the path function from **django. urls**:

**from django.urls import include, path**

- Add a new URL pattern to the urlpatterns list

**path ('', include ('datetimeapp. urls')),**

- This includes the URL patterns from your app's urls.py file.

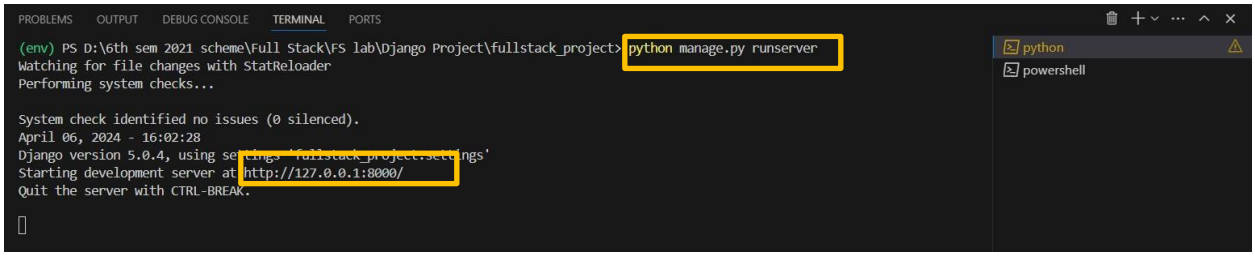


**Step-07: Run the development server**

- In the VS Code terminal, navigate to your Django project's directory (the one containing manage.py).
- Run the development server

**python manage.py runserver**

- Open your web browser and visit **http://127.0.0.1:8000/.**




**Final Output of the Date and Time App**


Current Date and Time on the Server

April 6, 2024, 4:03 p.m.

# Experiment-04

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.

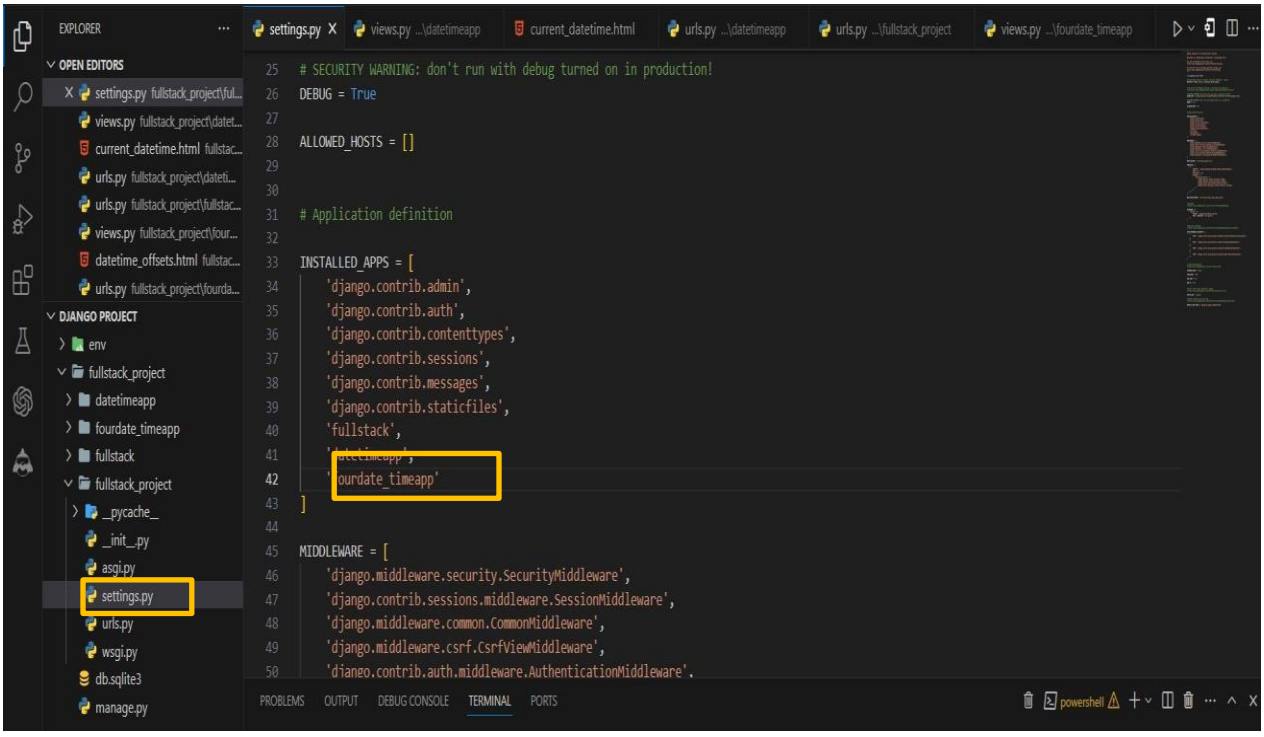
**Step-01**: **This app will be created in the Django project we made earlier.**

```
PS D:\6th sem 2021 scheme\Full Stack\FS lab\Django Project> env\Scripts\activate
(env) PS D:\6th sem 2021 scheme\Full Stack\FS lab\Django Project> python manage.py startapp fourdate_timeapp
```

**Step-02: Add the app to INSTALLED_APPS**

Open the settings.py file in your project's directory (e.g., myproject/settings.py).

Locate the **INSTALLED_APPS** list and add the name of your new app to the list:

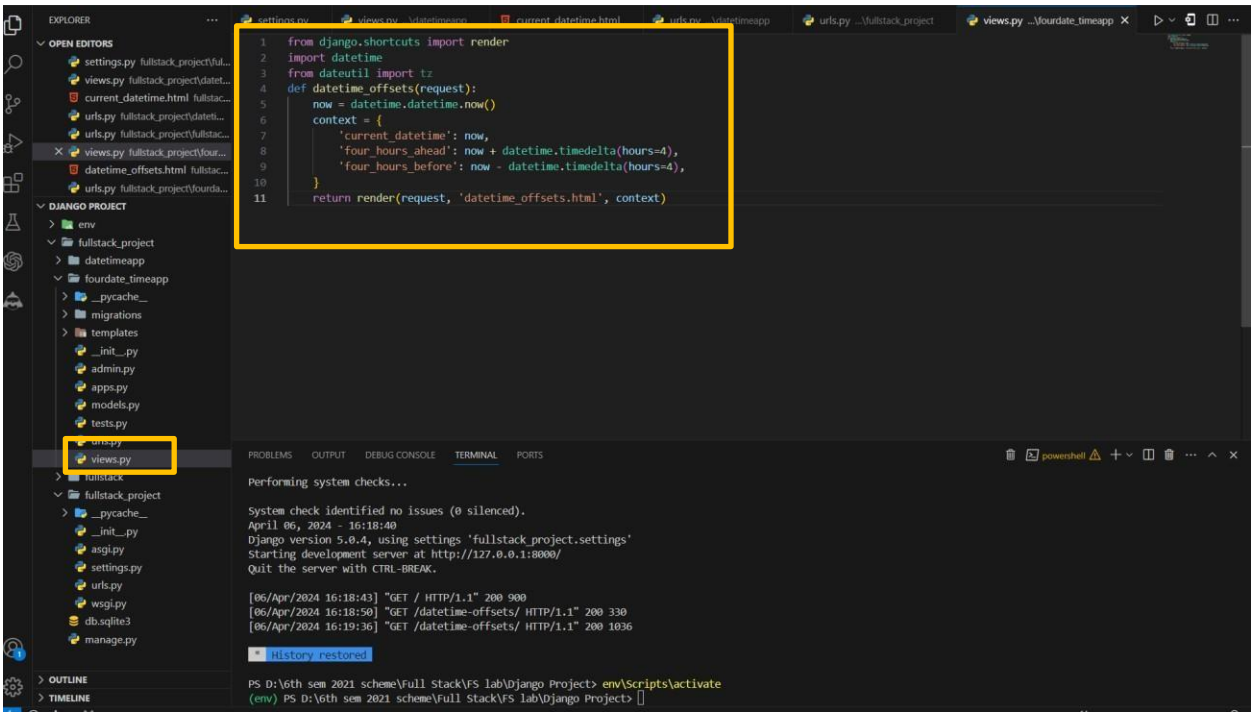


**Step-03: Create a view function**

- Open the views.py file in your Django app's directory (e.g., **fourdate_timeapp/views.py**).
- Import the necessary modules at the top of the file
- Create a new **view function** that will handle the request and render the date and time:

```
from django.shortcuts import render
import datetime
from dateutil import tz
def datetime_offsets(request):
    now = datetime.datetime.now()
    context = {
        'current_datetime': now,
        'four_hours_ahead': now + datetime.timedelta(hours=4),
        'four_hours_before': now - datetime.timedelta(hours=4),
    }
    return render(request, 'datetime_offsets.html', context)
```

- This view function gets the current date and time using datetime.datetime.now(), calculates the date and time four hours ahead and four hours before using datetime.timedelta, and then passes all three values to the template as context variables.




**Step-04: Create a new template**

- In your app's directory **(e.g., fourdate_timeapp),** create a new folder named templates (if it doesn't already exist).
- Inside the templates folder, create another folder with the same name as your app (e.g., fourdate_timeapp).
- Inside the fourdate_timeapp folder, create a new file named **datetime_offsets.html.**
- Open **datetime_offsets.html** and add the following code to display the current date and time, along with the offsets:

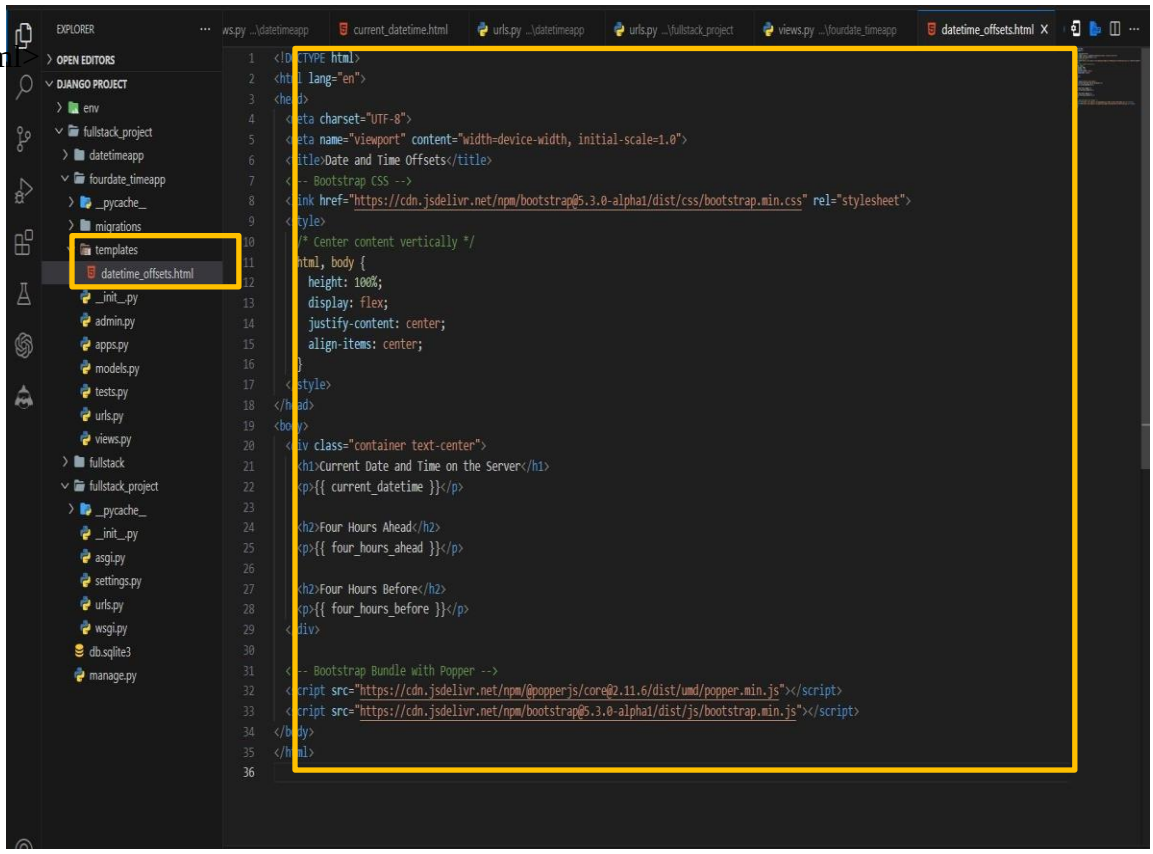
```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
  <title>Date and Time Offsets</title>
  <!-- Bootstrap CSS -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css"
rel="stylesheet">
<style>
  /* Center content vertically */ tml,
  body {
    height: 100%;
    display: flex;
    justify-content: center;
    lign-items: center;
  }
</style>
</head>
<body>
<div class="container text-center">
  <h1>Current Date and Time on the Server</h1>
  <p>{{ current_datetime }}</p>
  <h2>Four Hours Ahead</h2>
    <p>{{ four_hours_ahead }}</p>
    <h2>Four Hours Before</h2>
```

```
    <p>{{ four_hours_before }}</p>

  </div>

  <!-- Bootstrap Bundle with Popper -->
```

- This template displays the **current_datetime, four_hours_ahead, and four_hours_before** variables passed from the view function.

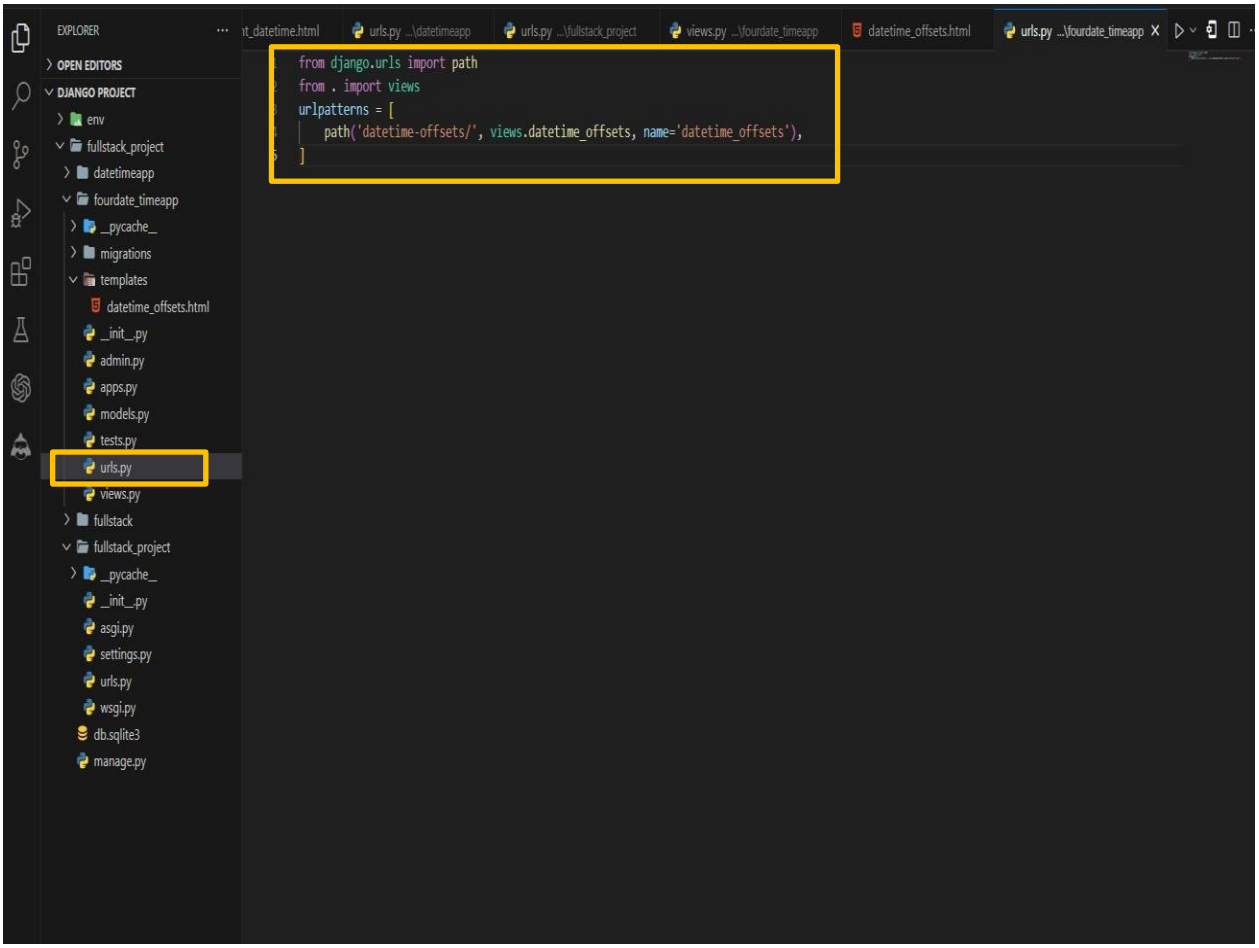
```
  <script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd/popper.min.js"></script>

  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-
alpha1/dist/js/bootstrap.min.js"></script>

</body></html>
```



**Step-05: Map the view function to a URL**

- Open the **urls.py** file in your Django app's directory (e.g., fourdate_timeapp/urls.py).
- Import the view function at the top of the file
- Add a new URL pattern to the urlpatterns list

```
from django.urls import path

from . import views

urlpatterns = [

    path('datetime-offsets/', views.datetime_offsets, name='datetime_offsets'),

]
```



**Step-06: Include the app's URLs in the project's URL patterns**

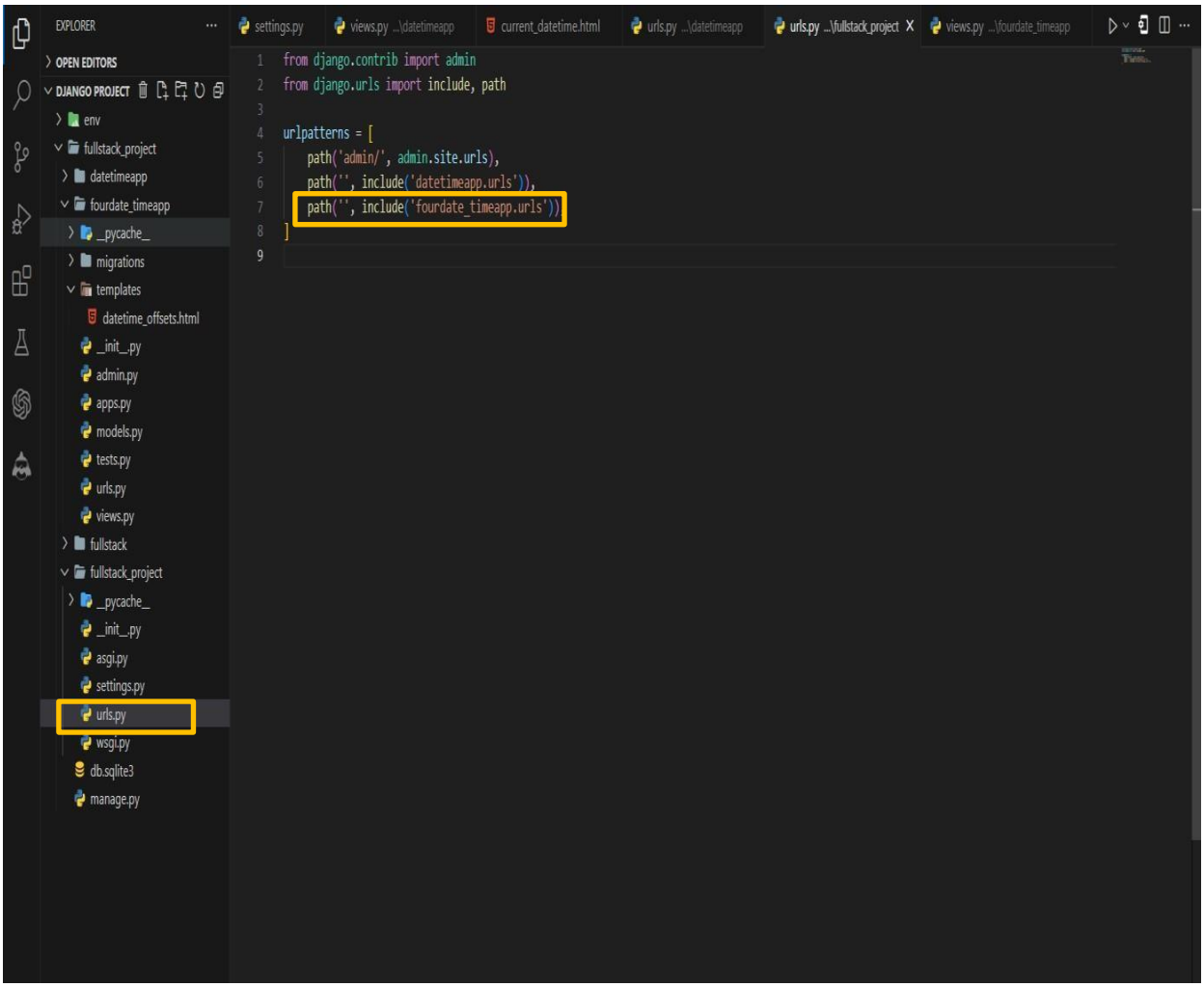
- Open the **urls.py** file in your project's directory **(e.g., fullstack_project/urls.py).**
- Import the include function from **django. urls** and the path function from **django. urls**:

**from django.urls import include, path**

- Add a new URL pattern to the urlpatterns list

**path('', include('fourdate_timeapp.urls')),**

- This includes the URL patterns from your app's **urls.py** file.

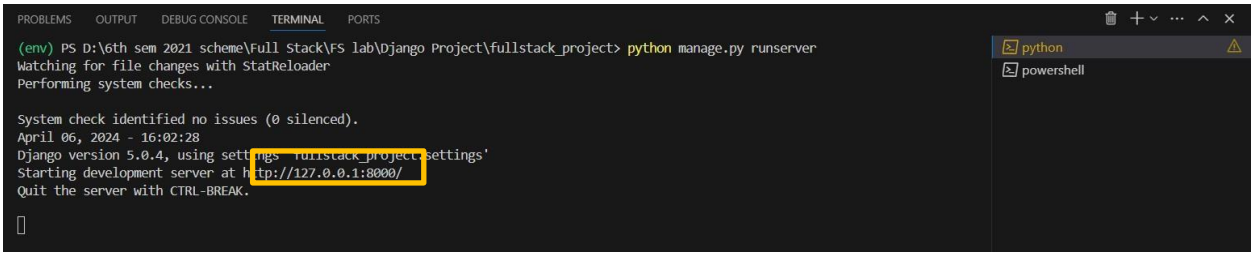


**Step-07: Run the development server**

- In the VS Code terminal, navigate to your Django project's directory (the one containing manage.py).
- Run the development server

**python manage.py runserver**

- Open your web browser and visit **http://127.0.0.1:8000/.**
- **Type or copy this http://127.0.0.1:8000/datetime-offsets/**




**Final Output of the Date and Time App**


Current Date and Time on the Server

April 6, 2024, 10:15 p.m.

Four Hours Ahead

April 7, 2024, 2:15 a.m.

Four Hours Before

April 6, 2024, 6:15 p.m.

# Experiment-05

Develop a simple Django app that displays an unordered list of fruits and ordered list of selected students for an event.

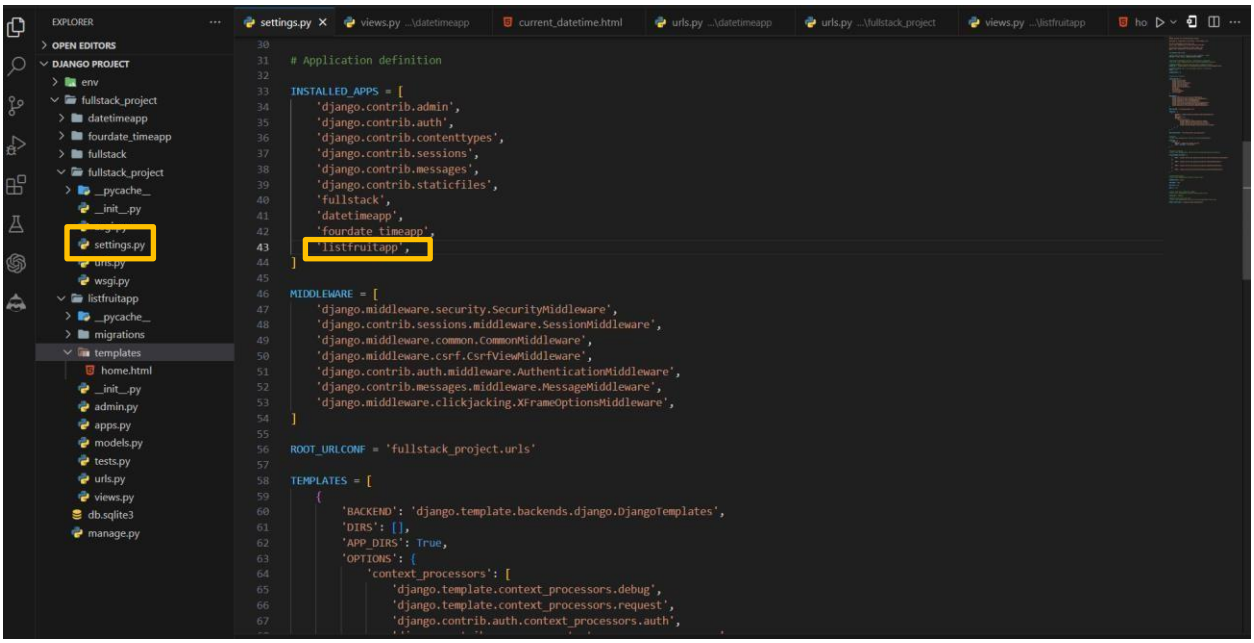
**Step-01**: **This app will be created in the Django project we made earlier.**

```
PS D:\6th sem 2021 scheme\Full Stack\FS lab\Django Project> cd fullstack_project
PS D:\6th sem 2021 scheme\Full Stack\FS lab\Django Project\fullstack_project> python manage.py startapp listfruitapp
```

**Step-02: Add the app to INSTALLED_APPS**

Open the settings.py file in your project's directory (e.g., myproject/settings.py).

Locate the **INSTALLED_APPS** list and add the name of your new app to the list:



**Step-03: Create a view function**

- Open the views.py file in your Django app's directory (e.g., **listfruitapp/views.py**).
- Create a new **view function** that will handle the request and render the date and time:

**from django.shortcuts import render**

**def home(request):**

**fruits = ['Apple', 'Banana', 'Orange', 'Mango', 'Pineapple']**

**students = ['John', 'Jane', 'Mike', 'Sarah', 'Tom']**

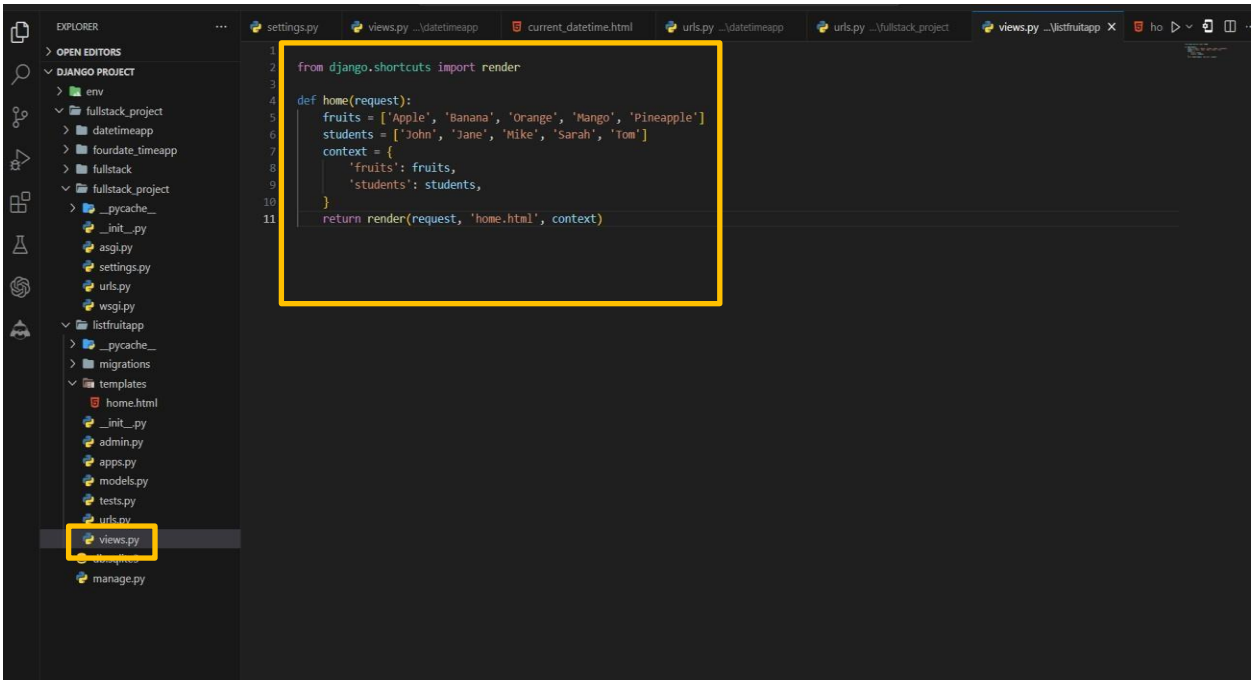
**context = {**

**'fruits': fruits,**

**'students': students,**

**}**

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




- Here, we define a view function home that creates two lists: fruits and students. These lists are then passed to the template as a context dictionary.

**Step-04: Create a new template**

- In your app's directory **(e.g., listfruitapp),** create a new folder named templates (if it doesn't already exist).
- Inside the templates folder, create another folder with the same name as your app (e.g.,

  display: flex;

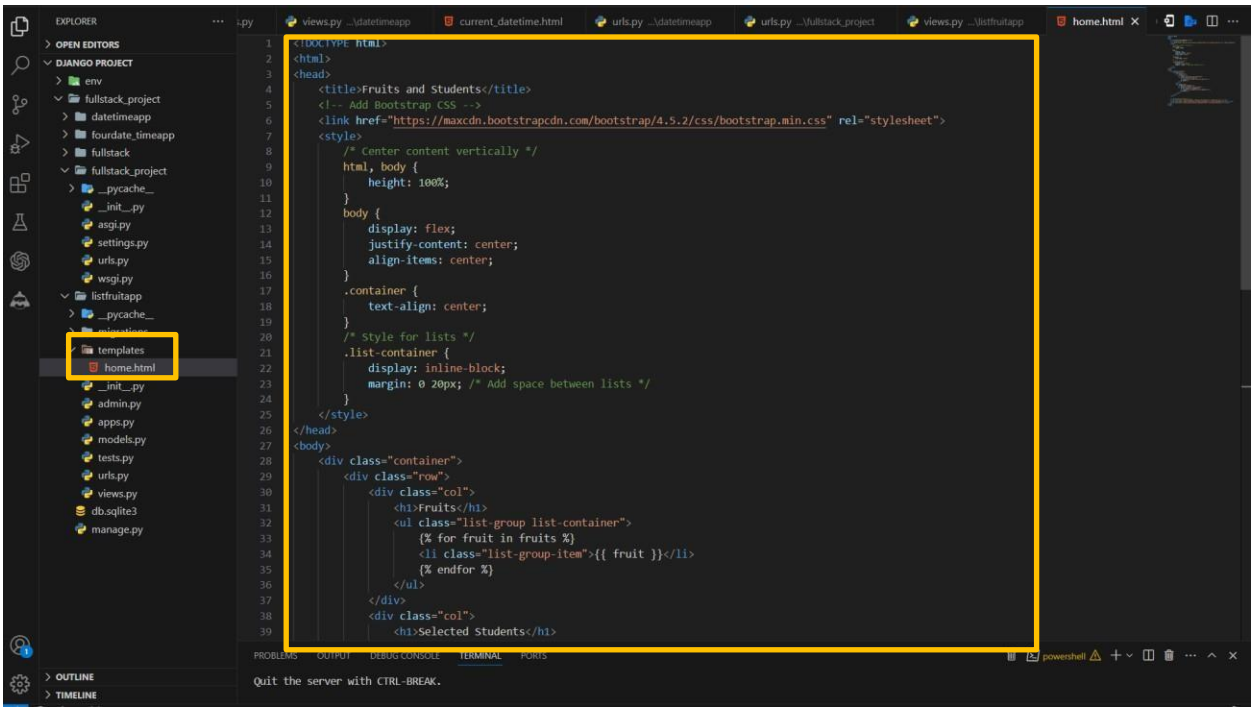
  justify-content: center;

  **listfruitapp**).
- Inside the fourdate_timeapp folder, create a new file named **home.html.**
- Open **home.html** and add the following code.

```
<!DOCTYPE html>

<html>

<head>
   <title>Fruits and Students</title>

   <!-- Add Bootstrap CSS -->

   <link href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"
rel="stylesheet">

   <style>

      /* Center content vertically */

      html, body {

         height: 100%;

      }

      body {

         align-items: center;
```

```
        }
        .container {
            text-align: center;
                    <li class="list-group-item">{{ fruit }}</li>
                    {% endfor %}
                </ul>
            }
            /* Style for lists */
            .list-container  { isplay:
                inline-block;
                margin: 0 20px; /* Add space between lists */
            }
    </style>
</head>
<body>
    <div class="container">
        <div class="row">
            <div class="col">
                <h1>Fruits</h1>
                <ul class="list-group list-container">
```

```
            </div>
            <div class="col">
                <h1>Selected Students</h1>
                <ol class="list-group list-container">
                    {% for student in students %}
                    <li class="list-group-item">{{ student }}</li>
                    {% endfor %}
                </ol>
            </div>
        </div>
    </div>

    <!-- Bootstrap JS (optional) -->
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</body>
</html>
```

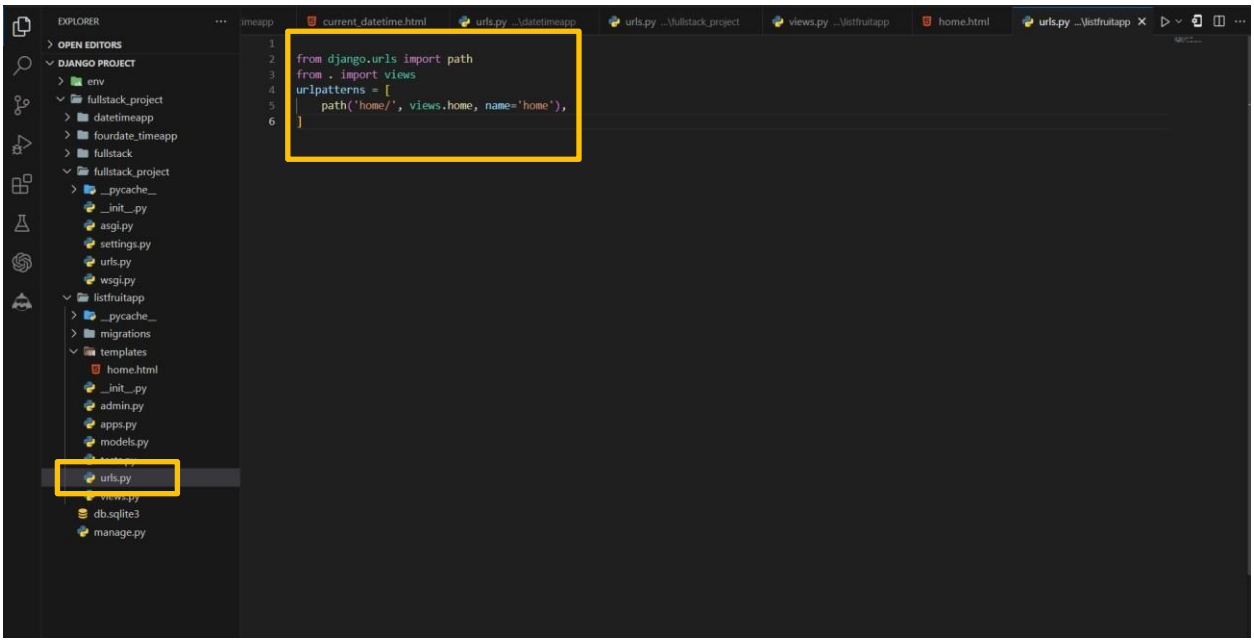



- In this template, we use Django's template tags to loop through the fruits and students lists and render them as an unordered list and an ordered list, respectively.

**Step-05: Map the view function to a URL**

- Open the **urls.py** file in your Django app's directory (e.g., listfruitapp/urls.py).
- Import the view function at the top of the file
- Add a new URL pattern to the urlpatterns list

```
from django.urls import path

from . import views

urlpatterns = [

    path('home/', views.home, name='home'),

]
```




**Step-06: Include the app's URLs in the project's URL patterns**

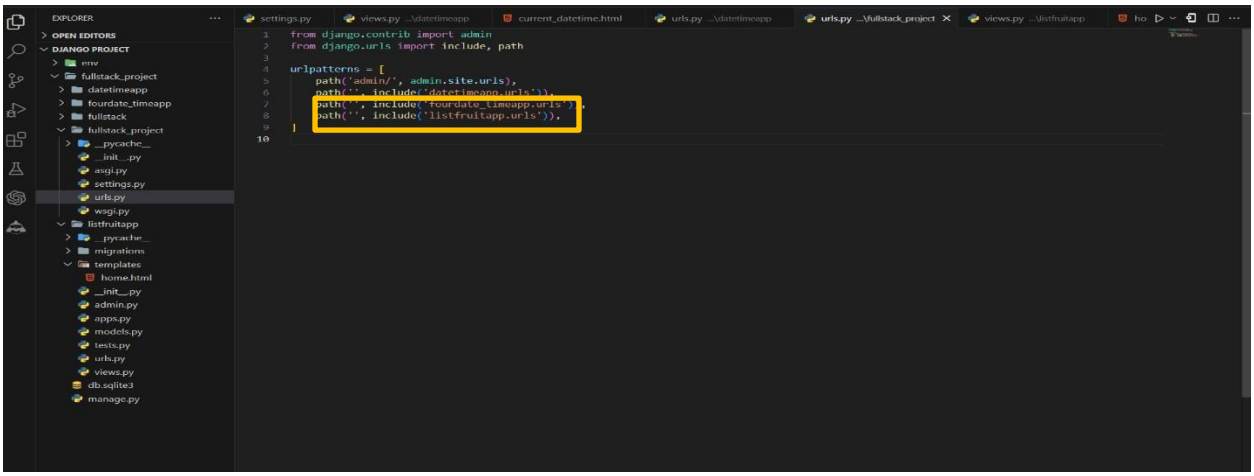
- Open the **urls.py** file in your project's directory **(e.g., fullstack_project/urls.py).**
- Import the include function from **django. urls** and the path function from **django. urls**:

**from django.urls import include, path**

- Add a new URL pattern to the urlpatterns list

**path('', include(listfruitapp.urls')),**

- This includes the URL patterns from your app's **urls.py** file.

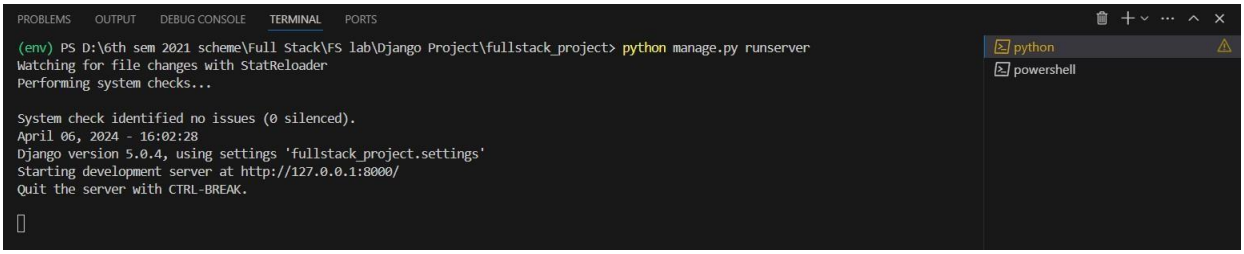


**Step-07: Run the development server**

- In the VS Code terminal, navigate to your Django project's directory (the one containing manage.py).
- Run the development server

**python manage.py runserver**

- Open your web browser and visit **http://127.0.0.1:8000/.**




- **Type or copy this http://127.0.0.1:8000/home/**

**Final Output of the Unorder list of Fruits and Students**


Fruits

- Apple
- Banana
- Orange
- Mango
- Pineapple

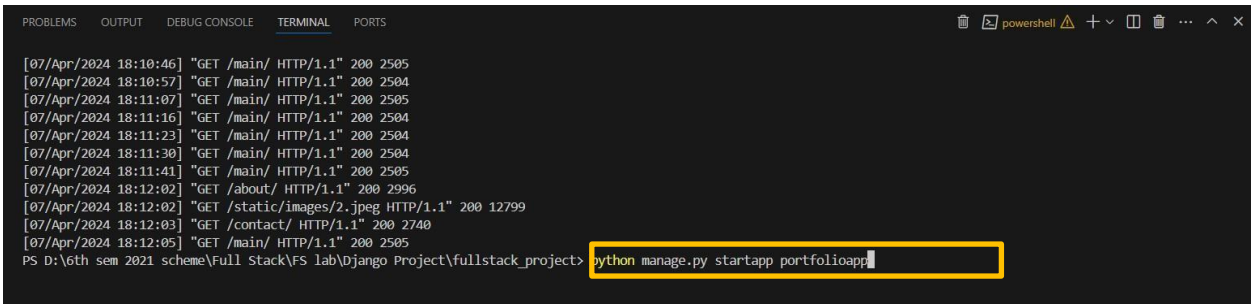
Selected Students

- John
- Jane
- Mike
- Sarah
- Tom

# Experiment-06

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.

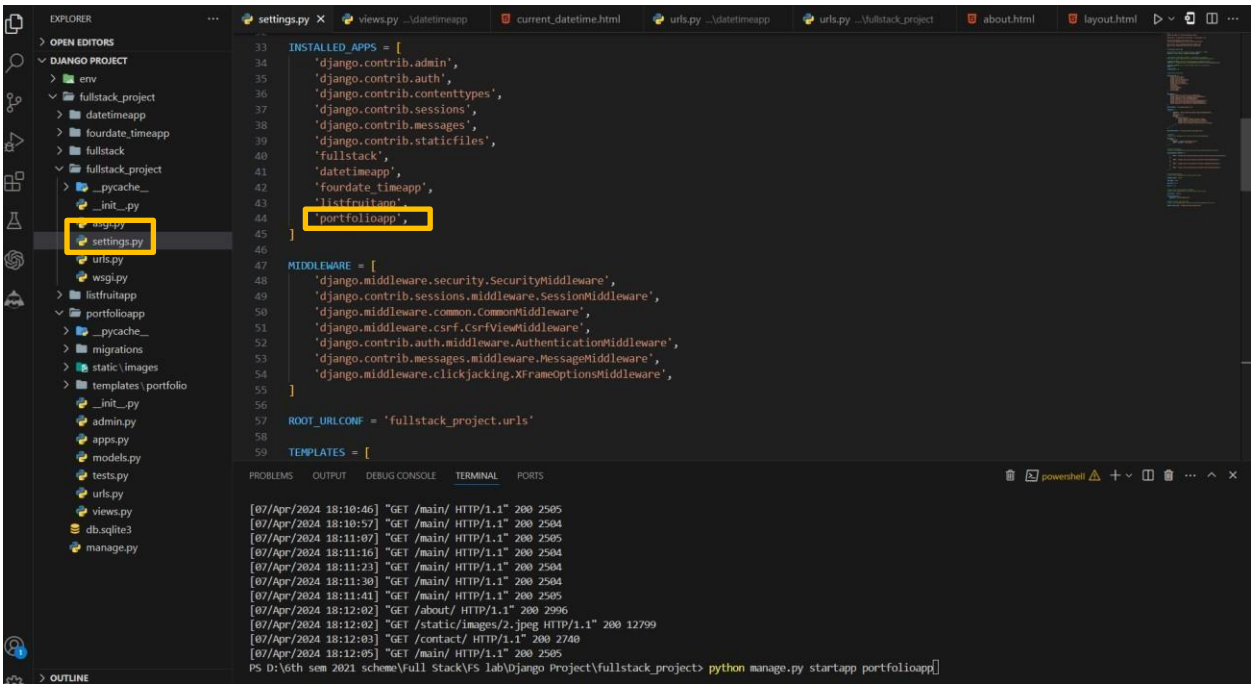
**Step-01**: **This app will be created in the Django project we made earlier.**




**Step-02: Add the app to INSTALLED_APPS**

Open the settings.py file in your project's directory (e.g., **fullstack_project/settings.py**).

Locate the **INSTALLED_APPS** list and add the name of your new app to the list:



**Step-03: Create a new template**

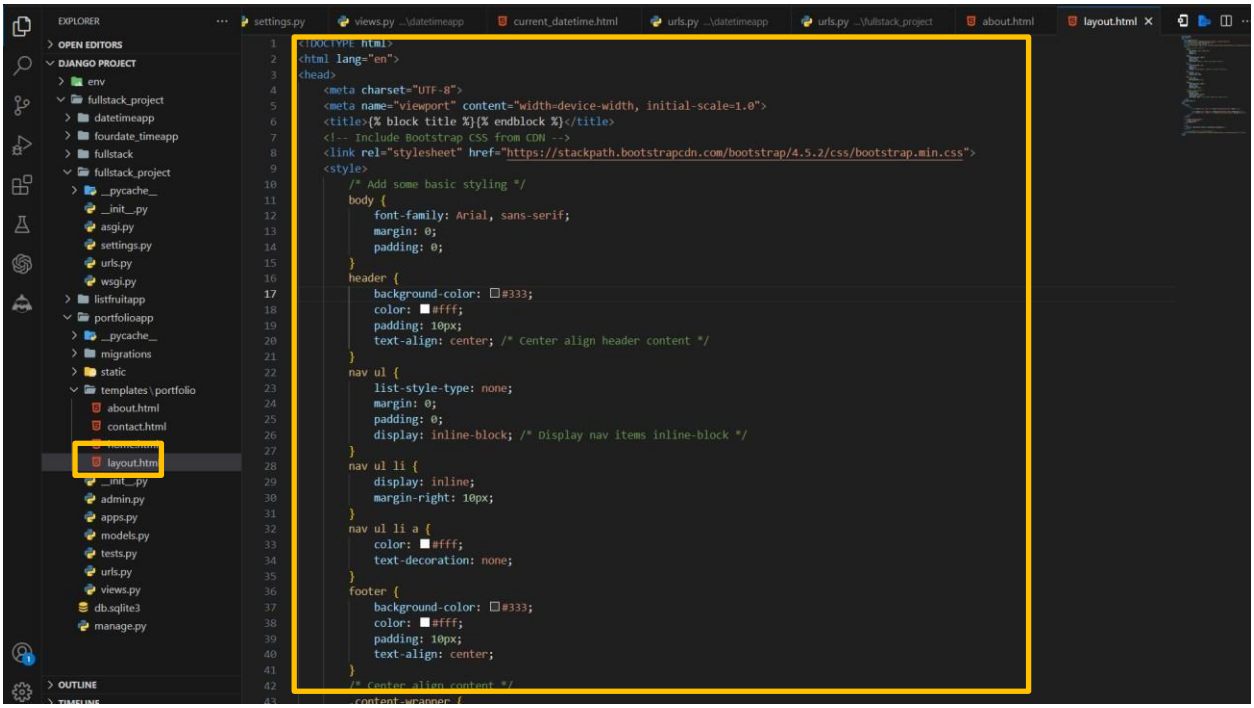
- Inside the portfolioapp/templates/ portfolio directory, create the following files: **layout.html**

```
<!DOCTYPE html>
  <html lang="en">

  <head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>{% block title %}{% endblock %}</title>

    <!-- Include Bootstrap CSS from CDN -->

    <link rel="stylesheet"
  href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">

    <style>

      /* Add some basic styling */

      body {

        font-family: Arial, sans-serif;

        margin: 0;

        padding: 0;

      }


     header {

        background-color: #333;

        color: #fff;
```

```
    padding: 10px;
    text-align: center; /* Center align header content */
}
nav ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
    display: inline-block; /* Display nav items inline-block */
}
nav ul li {
    display: inline;
    margin-right: 10px;
}
nav ul li a {
    color: #fff;
    text-decoration: none;
}
footer {

    background-color: #333;
    color: #fff;
    padding: 10px;
```

```
            text-align: center;
        }
        /* Center align content */
            <li><a href="{% url 'about' %}" style="text-decoration:none;">About Us</a></li>
            <li><a href="{% url 'contact' %}" style="text-decoration:none;">Contact Us</a></li>
    .content-wrapper {
        display: flex;
        justify-content: center;
        align-items: center;
        min-height: 80vh; /* Set minimum height for content area */
    }
</style>
{% load static %}
</head>
<body>
<header>
    <nav>
        <ul>
            <li><a href="{% url 'main' %}" style="text-decoration:none;">Home</a></li>
            </ul>
```

```
        </nav>
    </header>

    <div class="content-wrapper">
        {% block content %}
        {% endblock %}
    </div>

    <footer>
        <p>&copy; 2024 Search Creators. Developed by Hanumanthu.</p>
    </footer>

    <!-- Include Bootstrap JS from CDN (Optional) -->
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</body>
</html>
```




**home.html**

{% extends 'portfolio/layout.html' %}

{% block title %}Home{% endblock %}

{% block extra_css %}

<!-- Include Bootstrap CSS from CDN -->

<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">

<style>

/* Custom CSS to center align content */

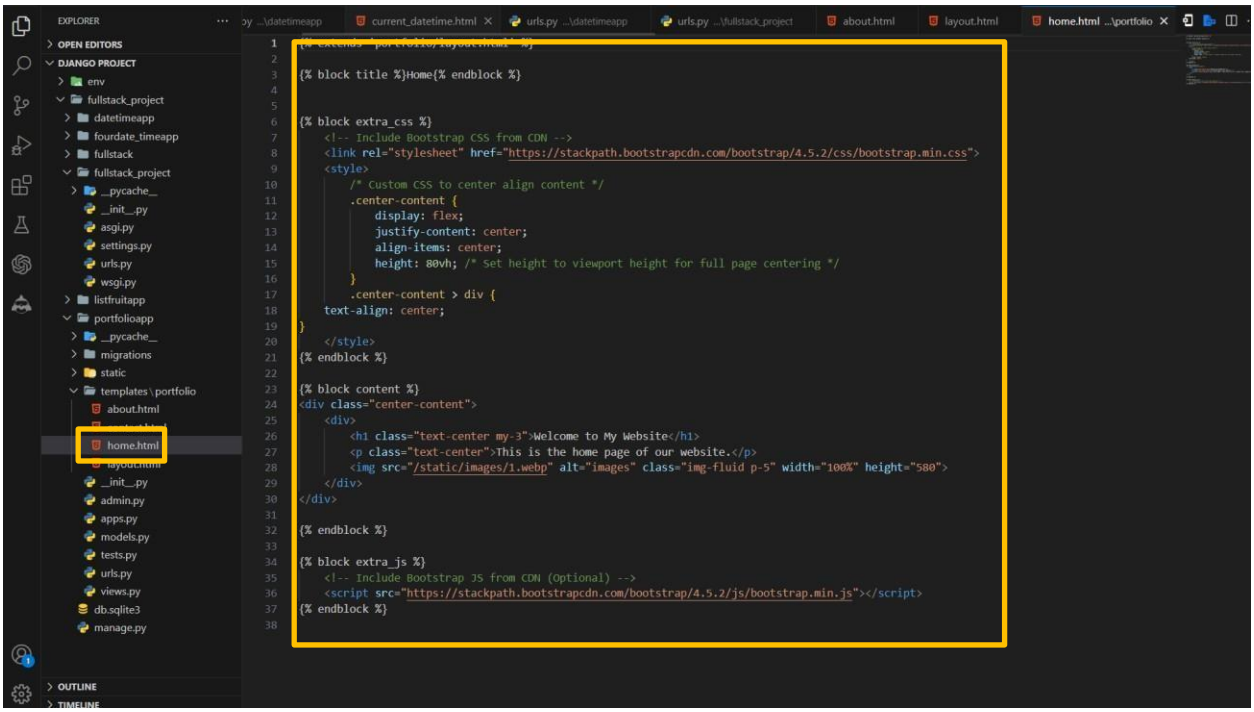
.center-content {

display: flex;

justify-content: center;

```
            align-items: center;
            height: 80vh; /* Set height to viewport height for full page centering */
        .center-
    content > div {
    text-align:
    center;
}
    </style>
{% endblock %}
{% block content %}
<div class="center-content">
    <div>
        <h1 class="text-center my-3">Welcome to My Website</h1>
        <p class="text-center">This is the home page of our website.</p>
        <img src="/static/images/1.webp" alt="images" class="img-fluid p-5"
width="100%" height="580">
    </div>
</div>
          }

     <!-- Include Bootstrap JS from CDN (Optional) -->
     <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
```

{% endblock %}




**about.html**

{% extends 'portfolio/layout.html' %}

{% block title %}About Us{% endblock %}

{% block extra_css %}

<!-- Include Bootstrap CSS from CDN -->

<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">

{% endblock %}

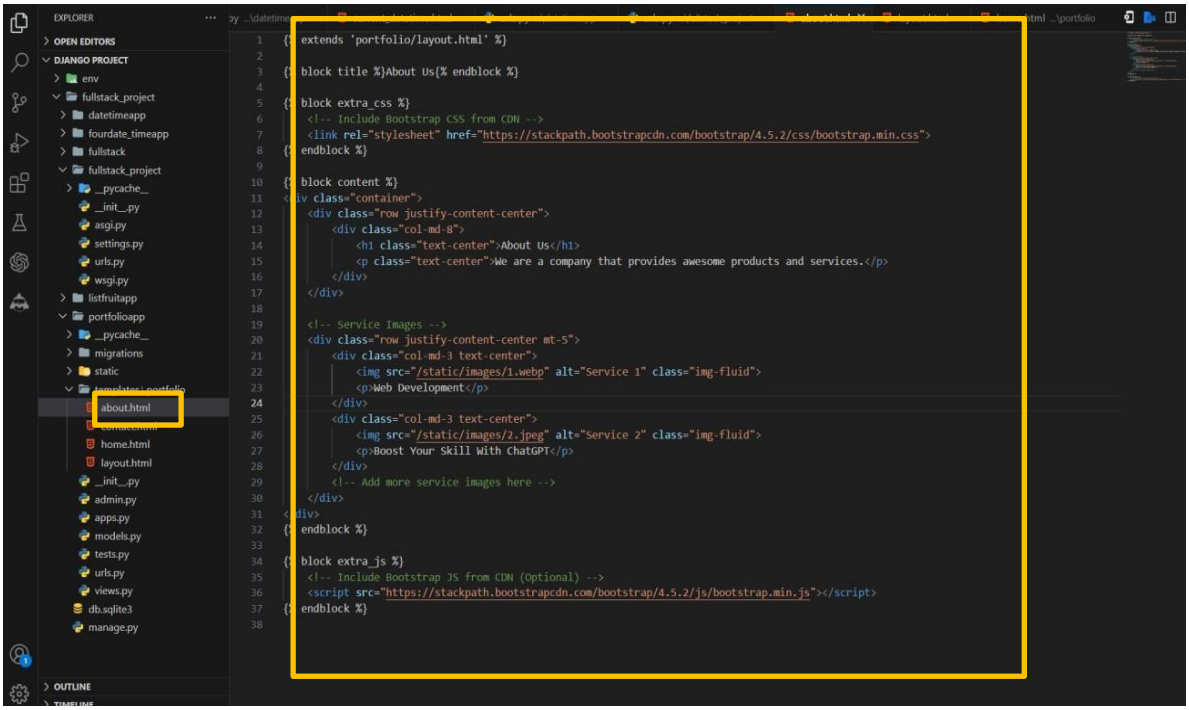
{% block content %}

```
        <div class="col-md-8">
            <h1 class="text-center">About Us</h1>
            <p class="text-center">We are a company that provides awesome products and
 services.</p>

    </div>
</div>
<!-- Service Images -->
<div class="row justify-content-center mt-5">
    <div class="col-md-3 text-center">
        <img src="/static/images/1.webp" alt="Service 1" class="img-fluid">
        <p>Web Development</p>
    </div>
    <div class="col-md-3 text-center">
        <img src="/static/images/2.jpeg" alt="Service 2" class="img-fluid">
        <p>Boost Your Skill With ChatGPT</p>
    </div>
    <!-- Add more service images here -->
</div>
</div>
{% endblock %}
 {% block extra_js %}
```

<!-- Include Bootstrap JS from CDN (Optional) -->

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

{% endblock %}




**contact.html**

{% extends 'portfolio/layout.html' %}

{% block title %}Contact Us{% endblock %}

{% block extra_css %}

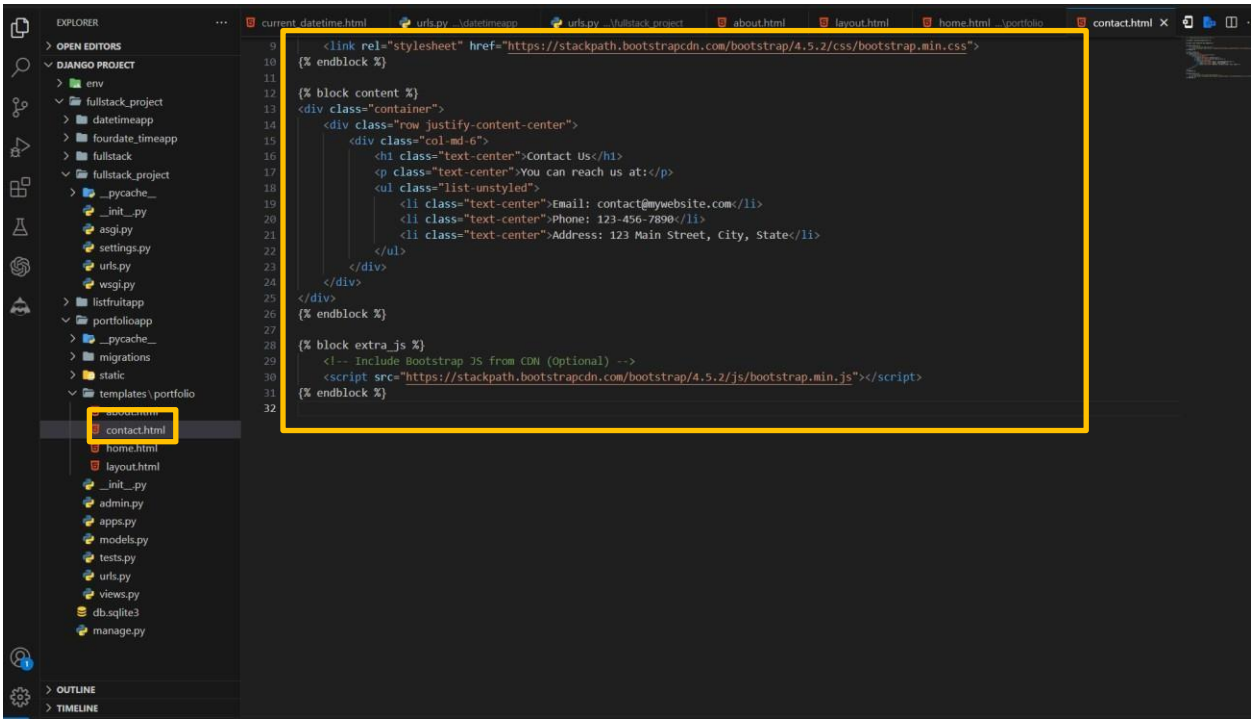
<!-- Include Bootstrap CSS from CDN -->

<link rel"stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">

{% endblock %}

{% block content %}

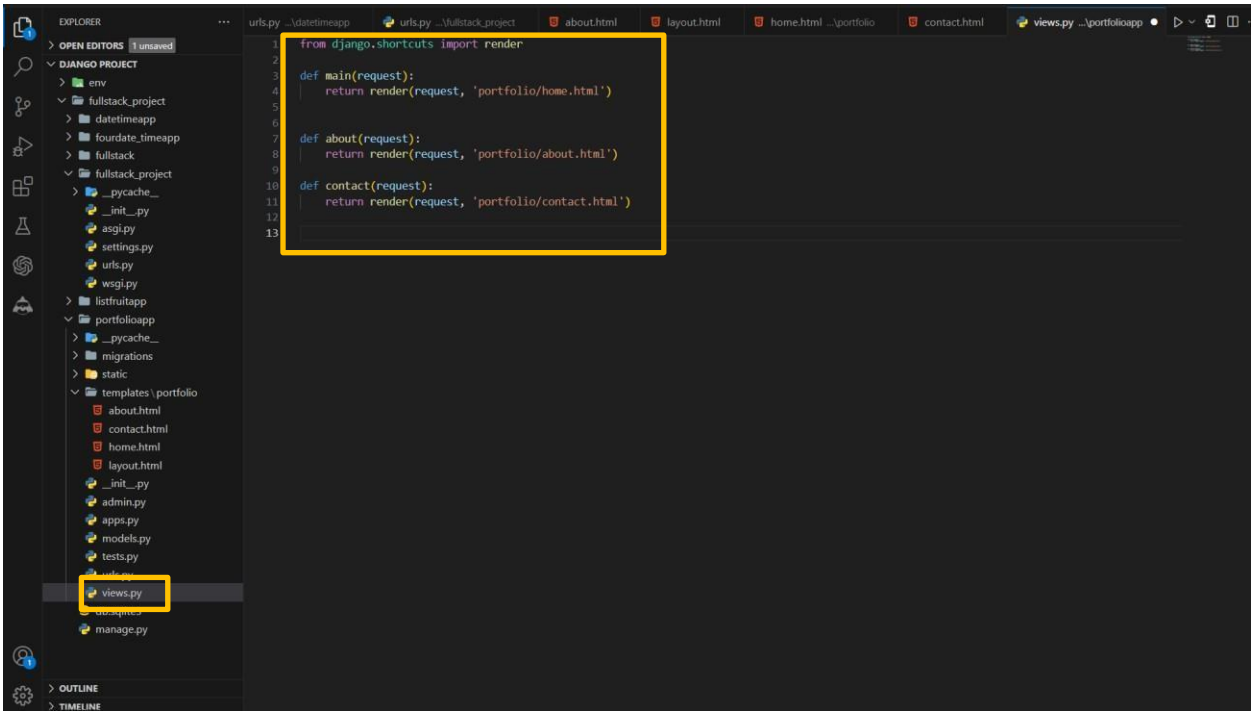
```
 <div class="container">
    <div class="row justify-content-center">
       <div class="col-md-6">

         <h1 class="text-center">Contact Us</h1>
         <p class="text-center">You can reach us at:</p>
         <ul class="list-unstyled">
            <li class="text-center">Email: contact@mywebsite.com</li>
            <li class="text-center">Phone: 123-456-7890</li>
            <li class="text-center">Address: 123 Main Street, City, State</li>
         </ul>
      </div>
   </div>
</div>
{% endblock %}
{% block extra_js %}
   <!-- Include Bootstrap JS from CDN (Optional) -->
   <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
{% endblock %}
```




**Step-04: Create a view function**

- Open the views.py file in your Django app's directory (e.g., **portfolioapp/views.py**).

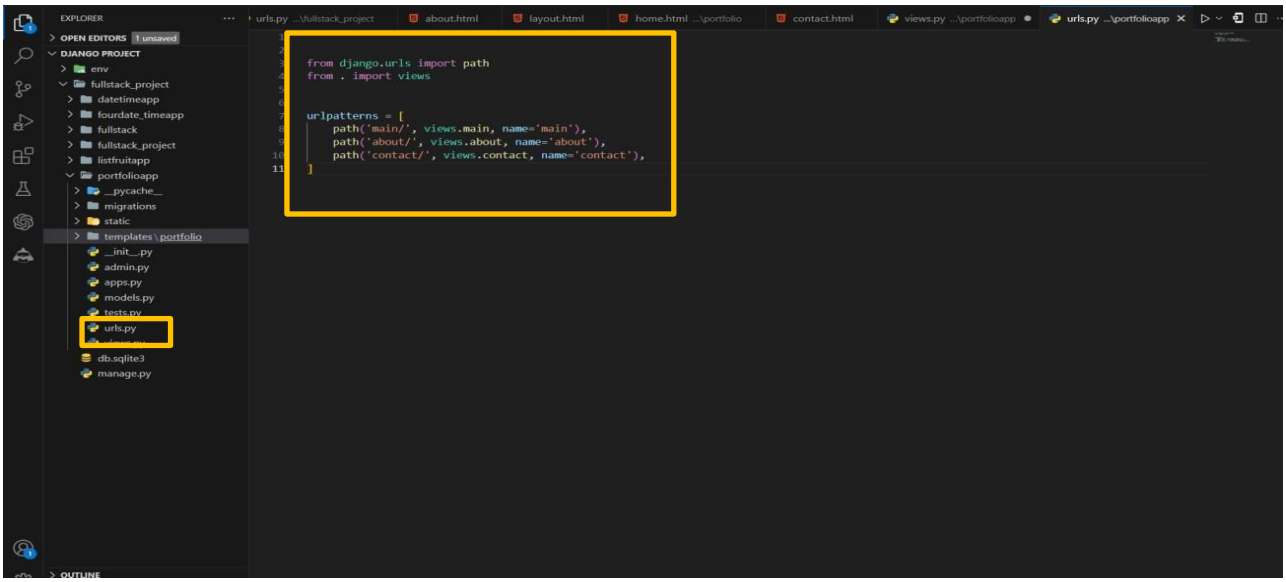
```
from django.shortcuts import render

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

def about(request):
    return render(request, 'portfolio/about.html')

def contact(request):
    return render(request, 'portfolio/contact.html')
```




**Step-05: Map the view function to a URL**

- Open the **urls.py** file in your Django app's directory (e.g., listfruitapp/urls.py).
- Import the view function at the top of the file
- Add a new URL pattern to the urlpatterns list

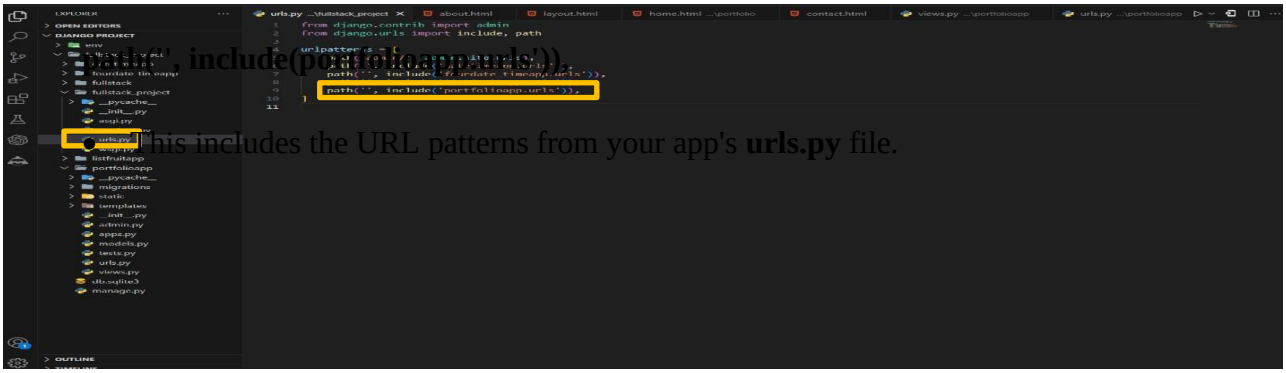
```
from django.urls import path

from . import views

urlpatterns = [

    path('main/', views.main, name='main'),

    path('about/', views.about, name='about'),

    path('contact/', views.contact, name='contact'),

]
```




**Step-06: Include the app's URLs in the project's URL patterns**

- Open the **urls.py** file in your project's directory **(e.g., fullstack_project/urls.py).**
- Import the include function from **django. urls** and the path function from **django. urls**:
- Add a new URL pattern to the urlpatterns list

**from django.urls import include, path [if does not exits]**



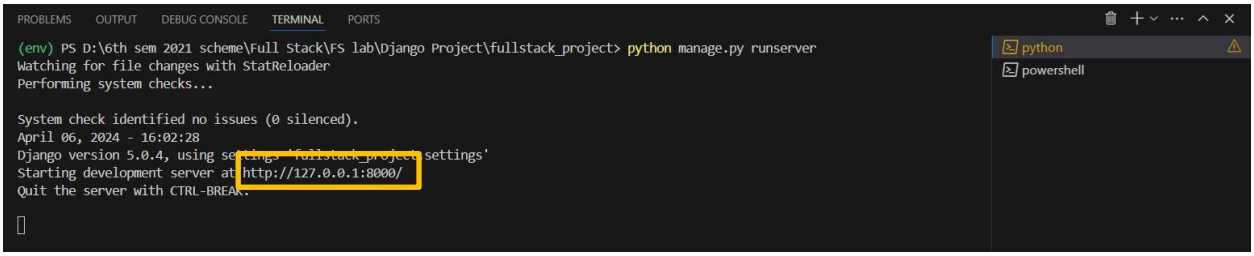

- This includes the URL patterns from your app's **urls.py** file.

**Step-07: Run the development server**

- In the VS Code terminal, navigate to your Django project's directory (the one containing manage.py).
- Run the development server

**python manage.py runserver**

- Open your web browser and visit **http://127.0.0.1:8000/.**




- **Type or copy this http://127.0.0.1:8000/main/**

**Final Output By Clicking Above Navbar Home, About Us, Contact Us Buttons**


**Fig: Home Page Screen**

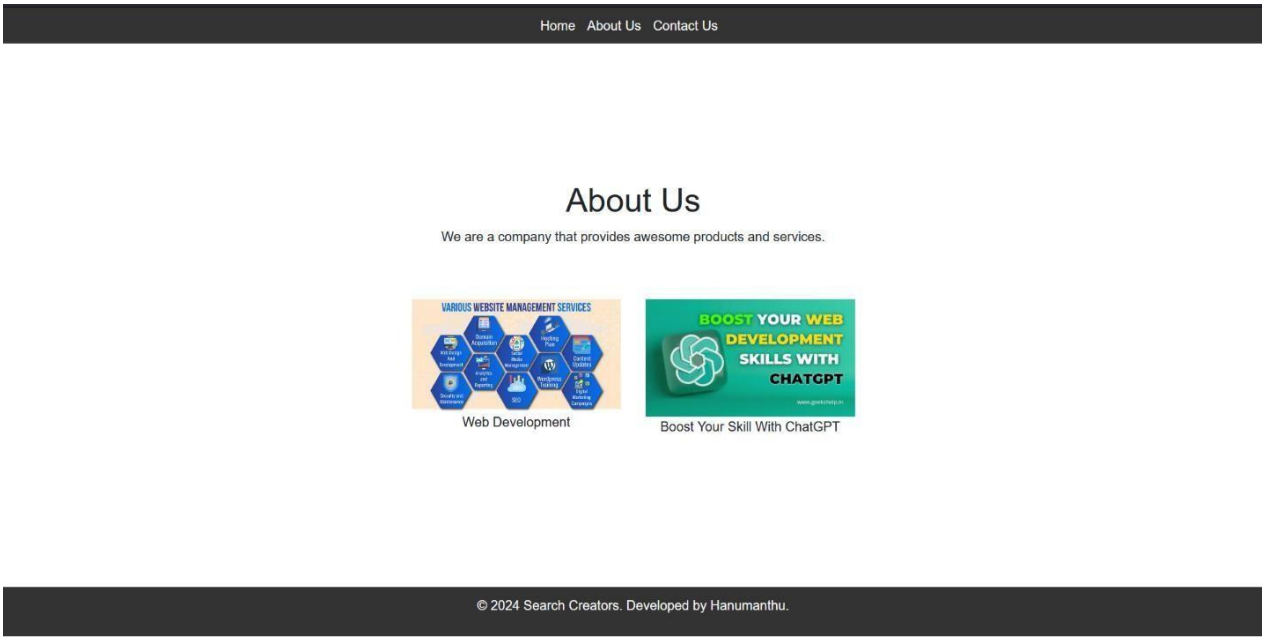



**Fig: About Us Screen**

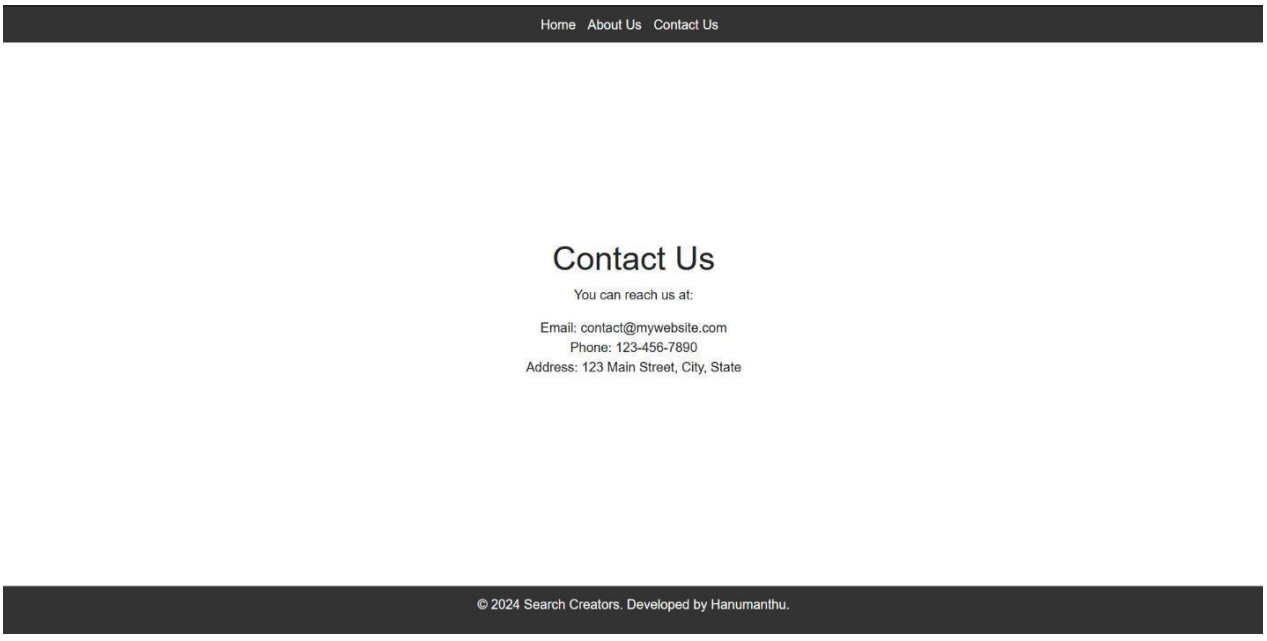



**Fig: About Us Screen**

# Experiment-07

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.

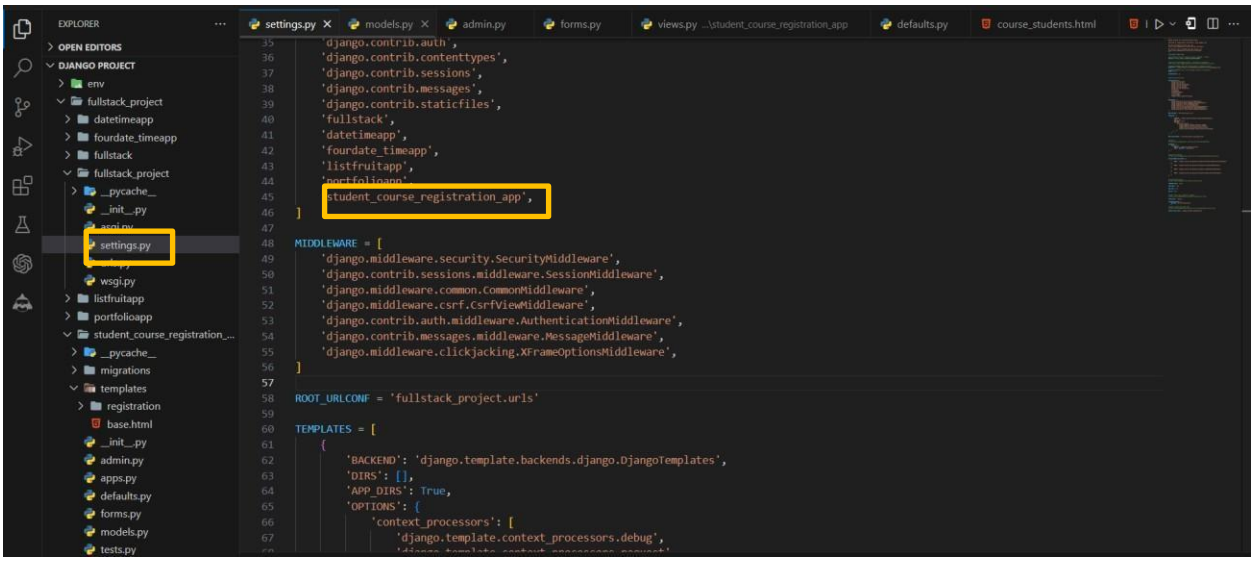
**Step-01**: **This app will be created in the Django project we made earlier.**

```
[08/Apr/2024 00:21:00] "GET /student-list/1/ HTTP/1.1" 200 1333
[08/Apr/2024 00:21:02] "GET /student-list/2/ HTTP/1.1" 200 1326
PS D:\6th sem 2021 scheme\Full Stack\FS lab\Django Project\fullstack_project> python manage.py makemigrations student_course_registration_app
```

**Step-02: Add the app to INSTALLED_APPS**

Open the settings.py file in your project's directory (e.g., **fullstack_project/settings.py**).

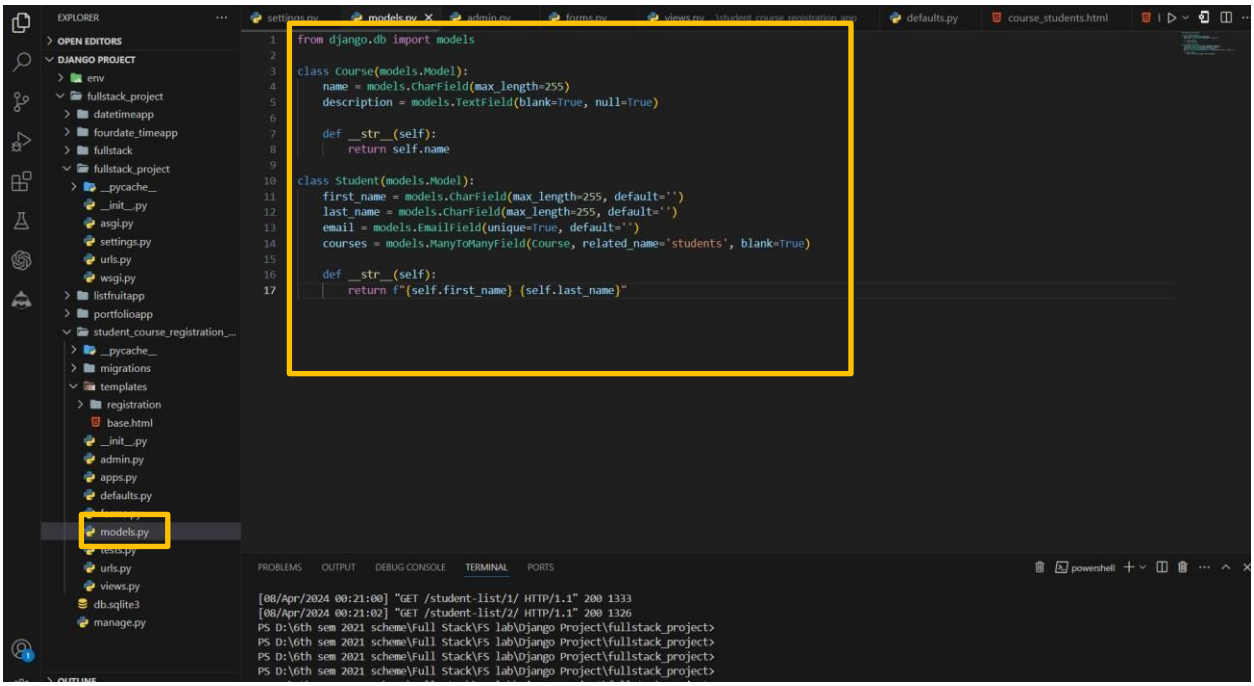
Locate the **INSTALLED_APPS** list and add the name of your new app to the list:



**Step-03: Create models**

- Open the **models.py** file inside the student_course_registration_app and define your models:

```
from django.db import models


class Course(models.Model):
    name = models.CharField(max_length=255)
    description = models.TextField(blank=True, null=True)
    def__str_(self):
        return self.name
class Student(models.Model):
    first_name = models.CharField(max_length=255, default='')
    last_name = models.CharField(max_length=255, default='')
    email = models.EmailField(unique=True, default='')
    courses = models.ManyToManyField(Course, related_name='students', blank=True)
    def str (self):
        return f"{self.first_name} {self.last_name}"
```




**Step-04: Register models in the admin site**

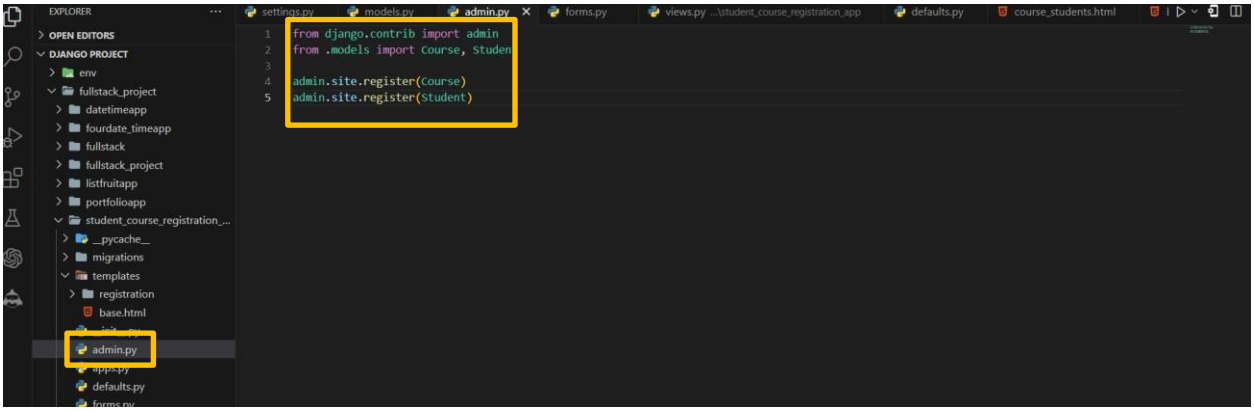
- Open the **admin.py** file inside the student_course_registration_app and register your models:

**from django.contrib import admin**

**from .models import Course, Student**

**admin.site.register(Course)**

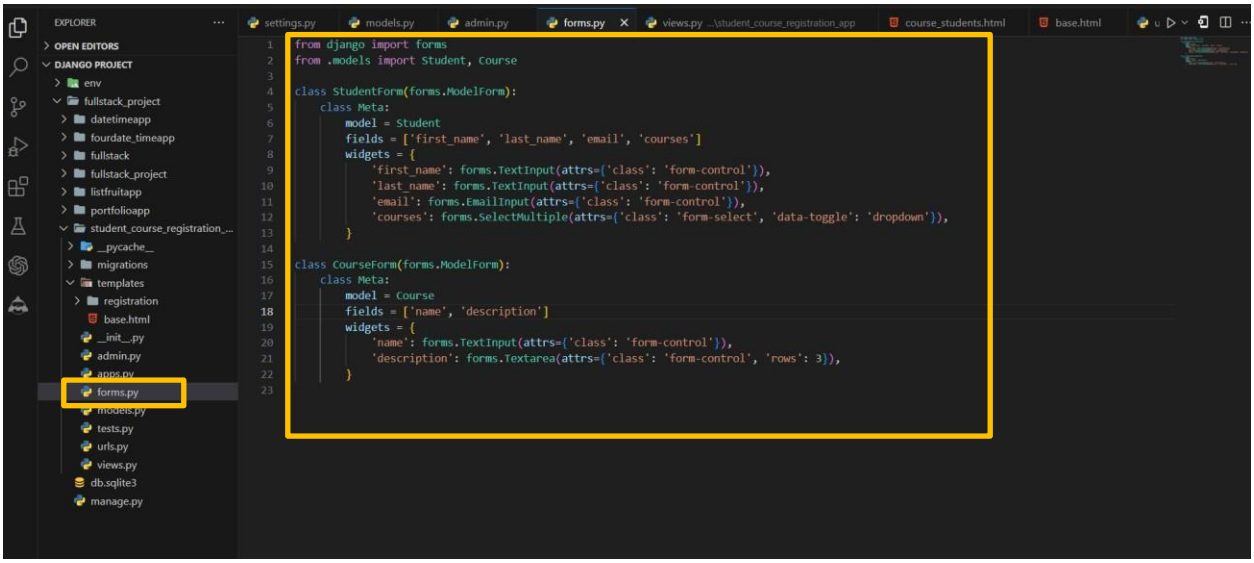
**admin.site.register(Student)**



**Step-05: Create forms**

- Create a new file called **forms.py** inside the student_course_registration_app and define your forms:

```
from django import forms
from .models import Student, Course
class StudentForm(forms.ModelForm):
    class Meta:
        model = Student
        fields = ['first_name', 'last_name', 'email', 'courses']
        widgets = {
            'first_name': forms.TextInput(attrs={'class': 'form-control'}),
            'last_name': forms.TextInput(attrs={'class': 'form-control'}),
            'email': forms.EmailInput(attrs={'class': 'form-control'}),
            'courses': forms.SelectMultiple(attrs={'class': 'form-select', 'data-toggle': 'dropdown'}),
        }
class CourseForm(forms.ModelForm):
    class Meta:
        model = Course
        fields = ['name', 'description']
        widgets = {
            'name': forms.TextInput(attrs={'class': 'form-control'}),
            'description': forms.Textarea(attrs={'class': 'form-control', 'rows': 3}),
        }
```

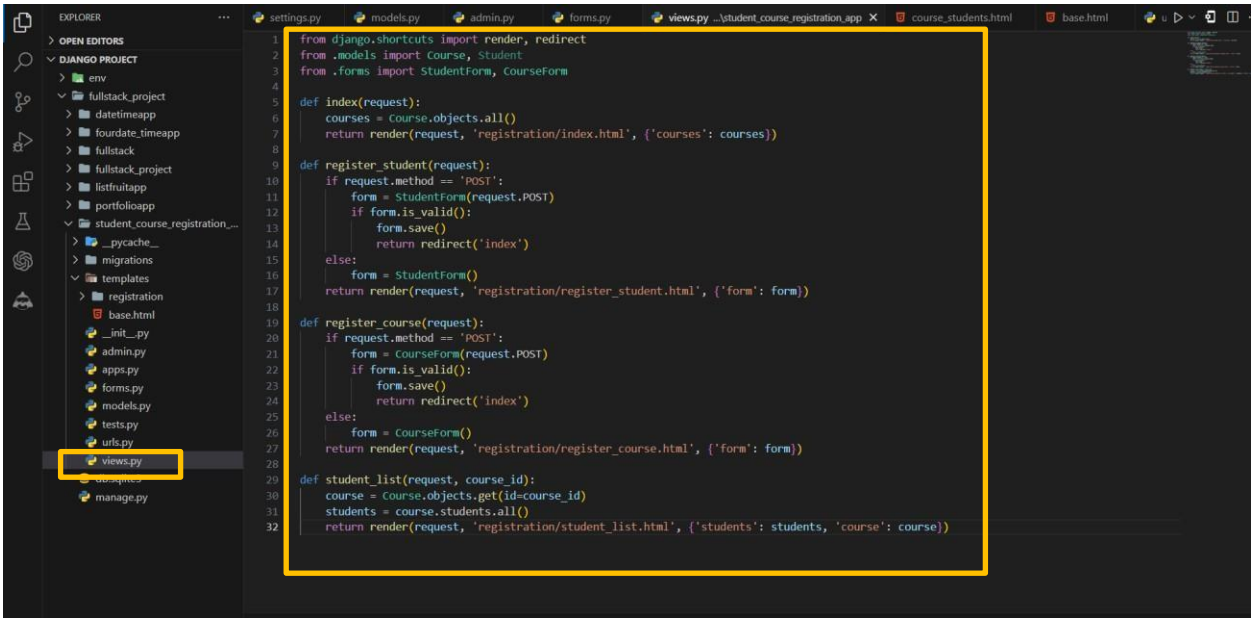



**Step-06: Create views**

Open the **views.py** file inside the student_course_registration_app and define your views:

```
from django.shortcuts import render, redirect
from .models import Course, Student
from .forms import StudentForm, CourseForm
def index(request):
    courses = Course.objects.all()
    return render(request, 'registration/index.html', {'courses': courses})
def register_student(request):
    if request.method == 'POST':
        form = StudentForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('index')
    else:
```

```
        form = StudentForm()
    return render(request, 'registration/register_student.html', {'form': form})
def register_course(request):
    if request.method == 'POST':
        form = CourseForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('index')
    else:
        form = CourseForm()
    return render(request, 'registration/register_course.html', {'form': form})
def student_list(request, course_id):
    course = Course.objects.get(id=course_id)
    students = course.students.all()
    return render(request, 'registration/student_list.html', {'students': students, 'course': course})
```

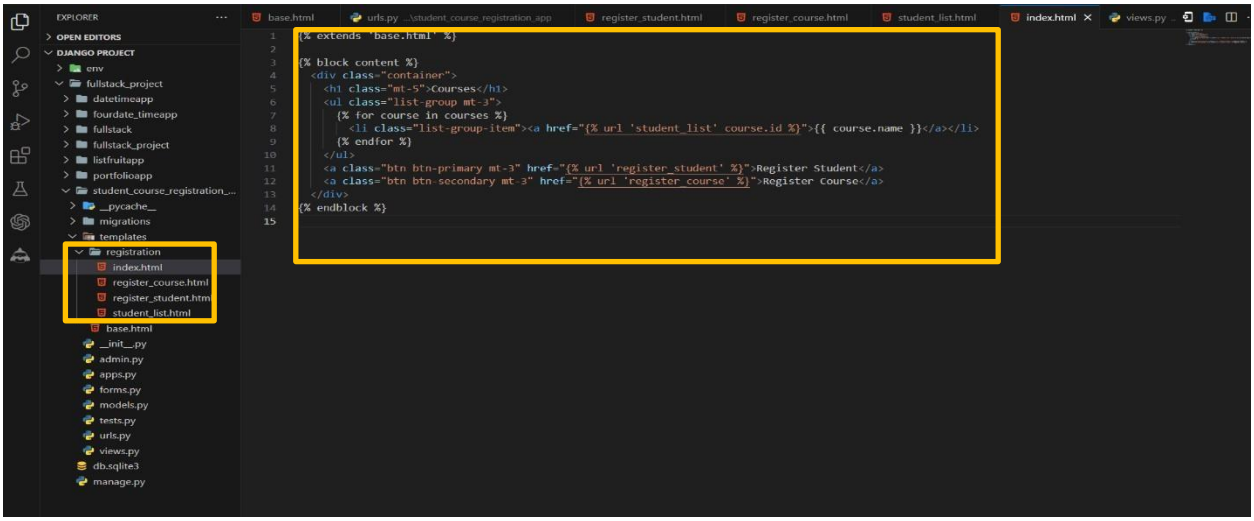


**Step-07: Create templates**

- Create a new directory called templates inside your student_course_registration_app, and create the following template files:

**index.html**

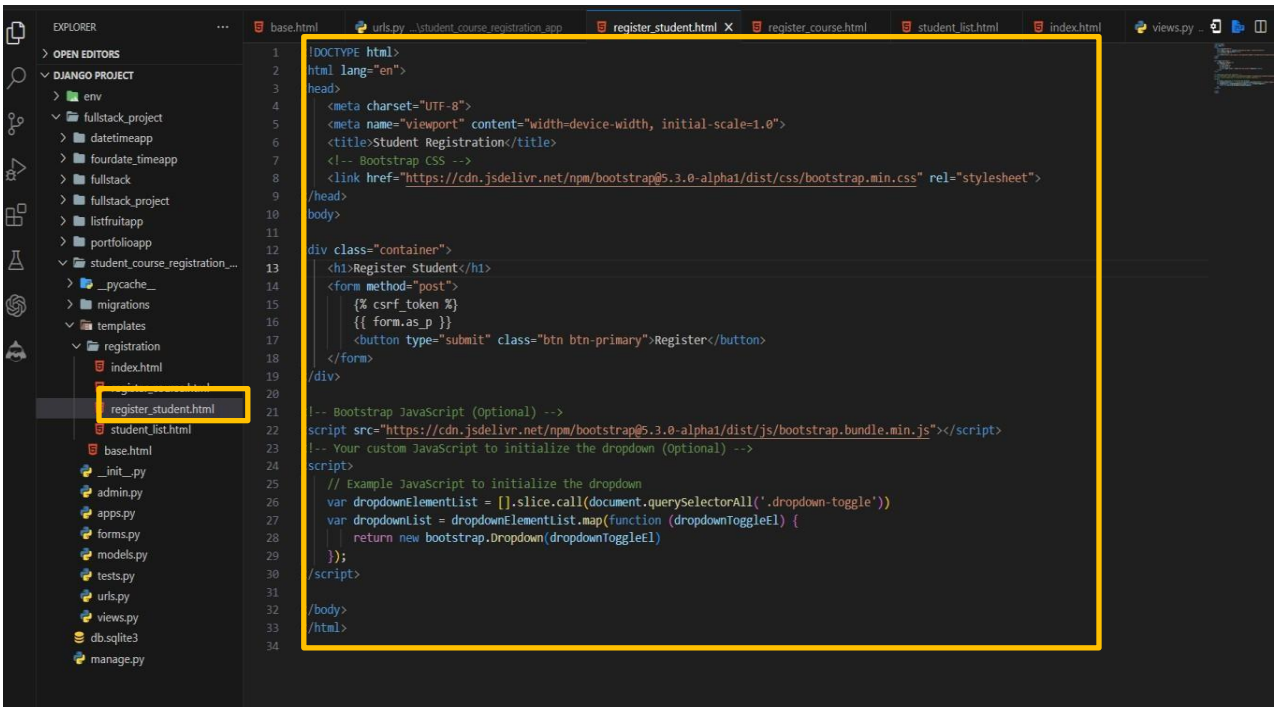
```
{% extends 'base.html' %}

{% block content %}
  <div class="container">
    <h1 class="mt-5">Courses</h1>
    <ul class="list-group mt-3">
      {% for course in courses %}
        <li class="list-group-item"><a href="{% url 'student_list' course.id %}">{{ course.name }}</a></li>
      {% endfor %}
    </ul>
    <a class="btn btn-primary mt-3" href="{% url 'register_student' %}">Register Student</a>
    <a class="btn btn-secondary mt-3" href="{% url 'register_course' %}">Register Course</a>
  </div>
{% endblock %}
```



**register_student.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Student Registration</title>
<!-- Bootstrap CSS -->
    <link      href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css"
rel="stylesheet">
</head>
<body>
<div class="container">
    <h1>Register Student</h1>
    <form method="post">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit" class="btn btn-primary">Register</button>
    </form>
</div>
<!-- Bootstrap JavaScript (Optional) -->
<script                                          src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-
alpha1/dist/js/bootstrap.bundle.min.js"></script>
<!-- Your custom JavaScript to initialize the dropdown (Optional) -->
<script>
    // Example JavaScript to initialize the dropdown
```

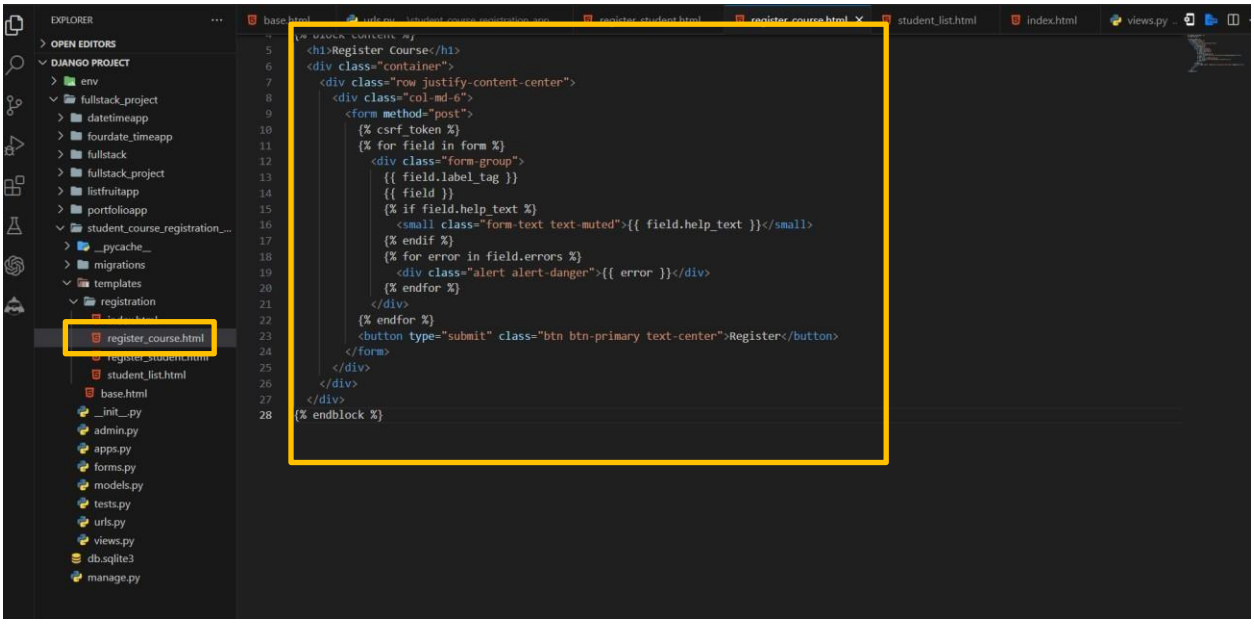
```
    var dropdownElementList = [].slice.call(document.querySelectorAll('.dropdown-toggle'))
    var dropdownList = dropdownElementList.map(function (dropdownToggleEl) {
        return new bootstrap.Dropdown(dropdownToggleEl)
    });
</script>
</body>
</html>
```




**register_course.html**

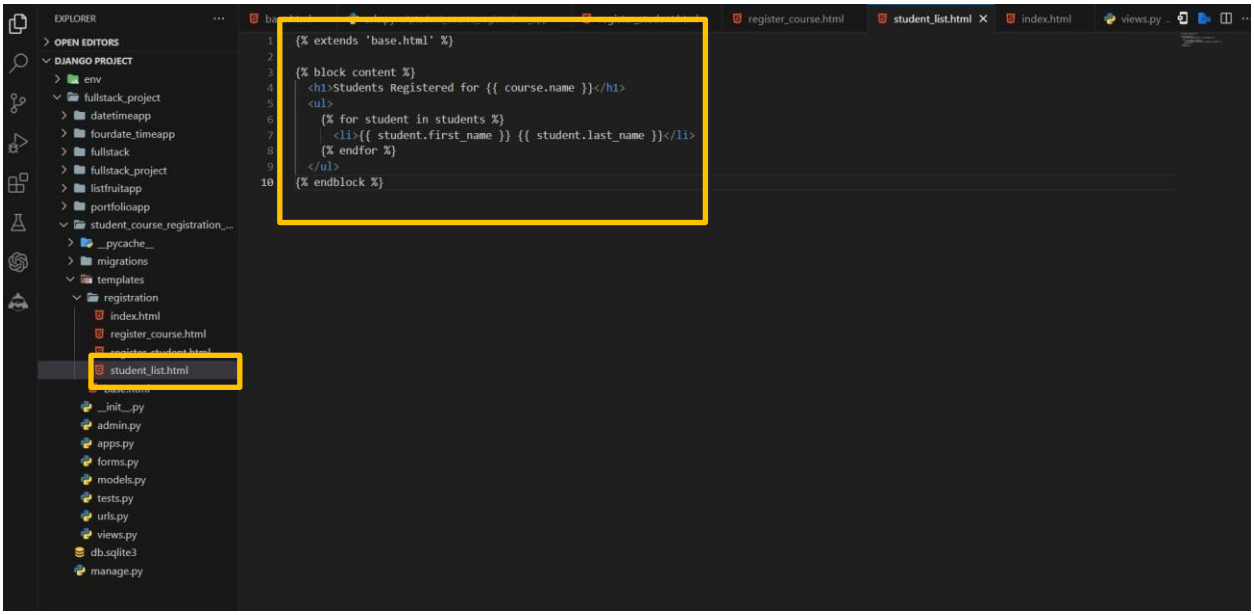
```
{% extends 'base.html' %}
{% load static %}
{% block content %}
  <h1>Register Course</h1>
  <div class="container">
    <div class="row justify-content-center">
```

```
      <div class="col-md-6">
        <form method="post">
          {% csrf_token %}
          {% for field in form %}

           <div class="form-group">
             {{ field.label_tag }}
             {{ field }}
             {% if field.help_text %}
               <small class="form-text text-muted">{{ field.help_text }}</small>
             {% endif %}
             {% for error in field.errors %}
               <div class="alert alert-danger">{{ error }}</div>
             {% endfor %}
           </div>
         {% endfor %}
         <button type="submit" class="btn btn-primary text-center">Register</button>
       </form>
     </div>
   </div>
 </div>
{% endblock %}
```




**student_list.html**

```
{% extends 'base.html' %}

{% block content %}

  <h1>Students Registered for {{ course.name }}</h1>

  <ul>

    {% for student in students %}

      <li>{{ student.first_name }} {{ student.last_name }}</li>

    {% endfor %}

  </ul>

{% endblock %}
```

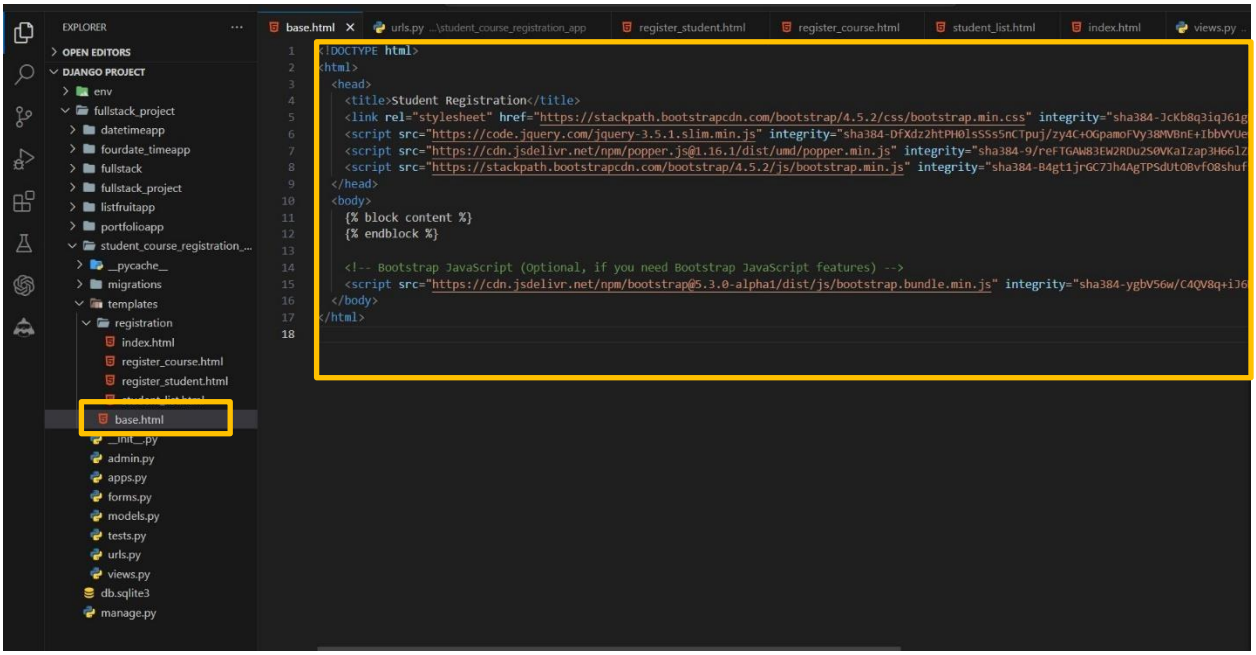



**Step-08: Create a base template**

- Create a **base.html** file inside the templates directory with the following content:

**base.html**

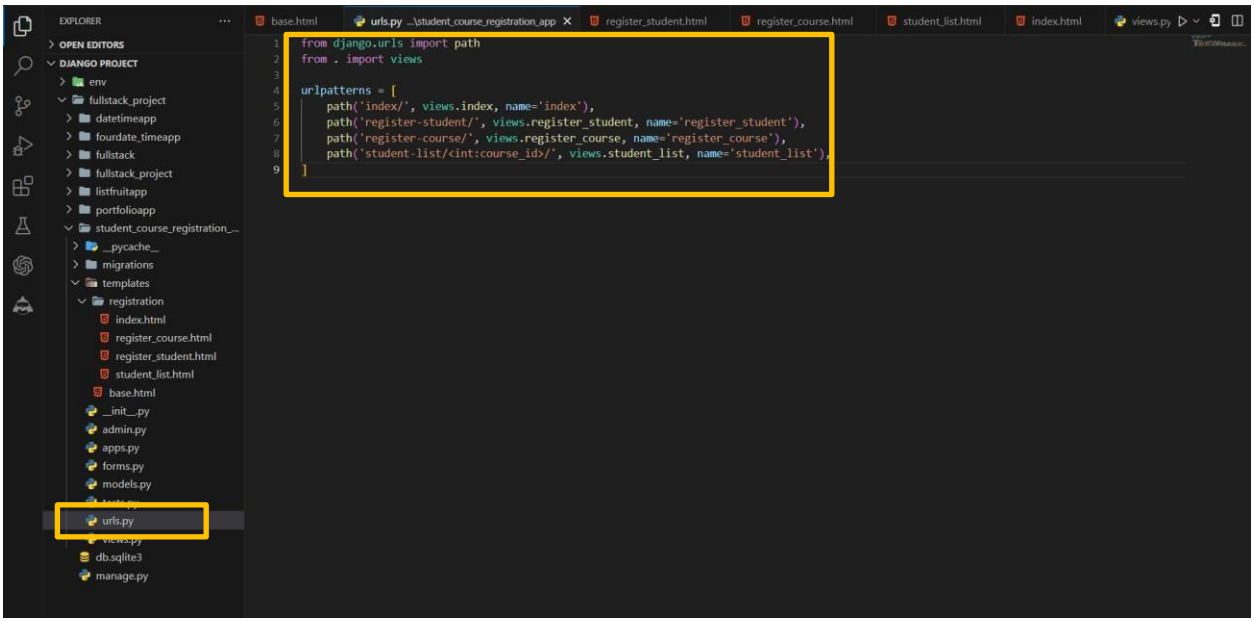
```
<!DOCTYPE html>
<html>
  <head>
    <title>Student Registration</title>
  </head>
  <body>
    {% block content %}
    {% endblock %}
  </body>
</html>
```




**Step-09: Configure URLs**

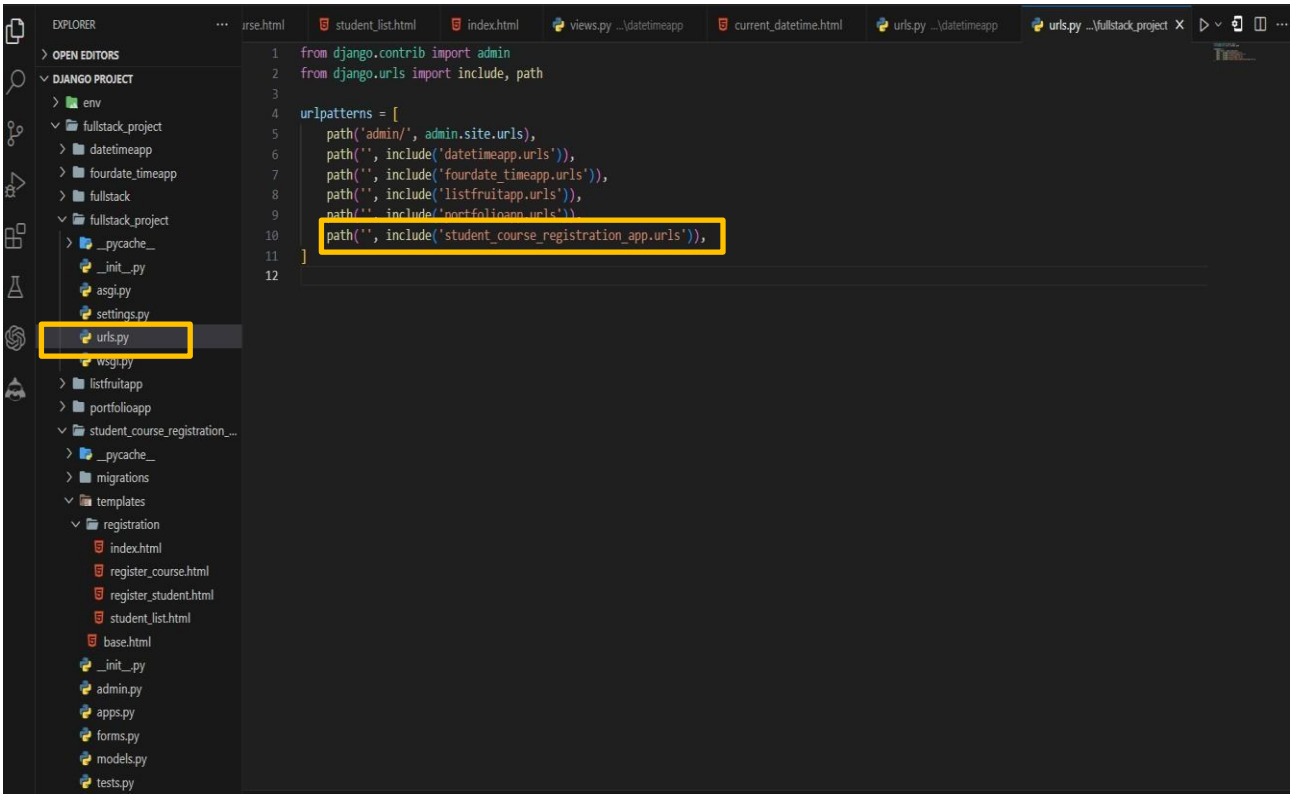
- Open the urls.py file inside your student_course_registration_app and define your URLs:

```
from django.urls import path

from . import views

urlpatterns = [

    path('index/', views.index, name='index'),

    path('register-student/', views.register_student, name='register_student'),

    path('register-course/', views.register_course, name='register_course'),

    path('student-list/<int:course_id>/', views.student_list, name='student_list'),

]
```




## Step-09: Update project URLs

- Open the urls.py file inside your project and include the URLs from the student_course_registration_app:



### Step-10: Make Migration for check Models saved or not into the database

```
PS D:\6th sem 2021 scheme\Full Stack\FS lab\Django Project\fullstack_project> python manage.py makemigrations
Migrations for 'student_course_registration_app':
  student_course_registration_app\migrations\0002_remove_student_name_student_courses_and_more.py
    - Remove field name from student
    - Add field courses to student
    - Add field first_name to student
    - Add field last_name to student
    - Alter field description on course
    - Alter field name on course
    - Alter field email on student
    - Delete model Enrollment
```

### Step-10: Migrate To Save the Models into database

```
PS D:\6th sem 2021 scheme\Full Stack\FS lab\Django Project\fullstack_project> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions, student_course_registration_app
Running migrations:
  Applying student_course_registration_app.0002_remove_student_name_student_courses_and_more... OK
```

### Step-07: Run the development server

- In the VS Code terminal, navigate to your Django project's directory (the one containing manage.py).
- Run the development server

**python manage.py runserver**

- Open your web browser and visit **http://127.0.0.1:8000/.**

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
(env) PS D:\6th sem 2021 scheme\Full Stack\FS lab\Django Project\fullstack_project> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
April 06, 2024 - 16:02:28
Django version 5.0.4, using settings 'fullstack_project.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

- **Type or copy this http://127.0.0.1:8000/index/**

### Final Output By Clicking Register Student and Register Course

**Fig: Index Screen**

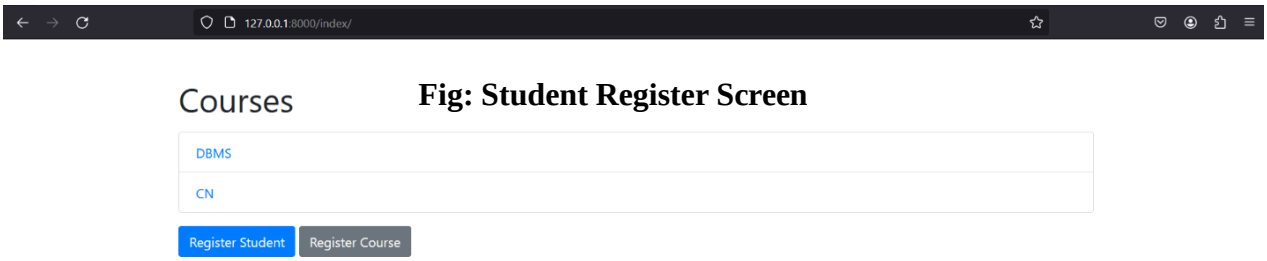



**Fig: Student Register Screen**

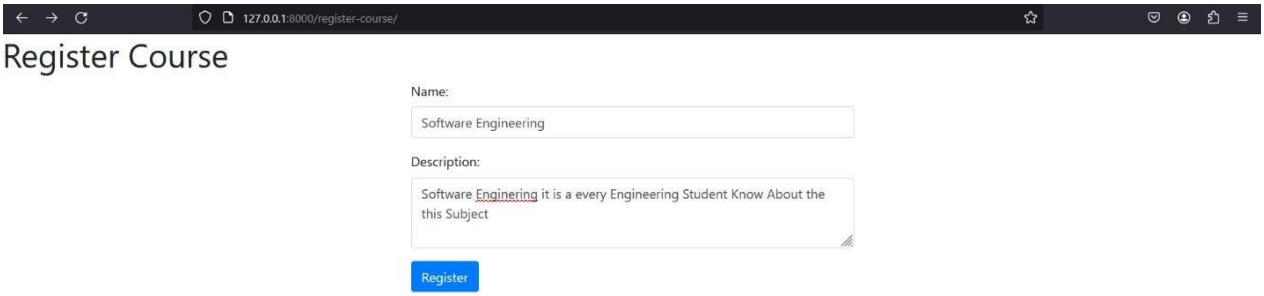



**Fig: Course Register Screen**

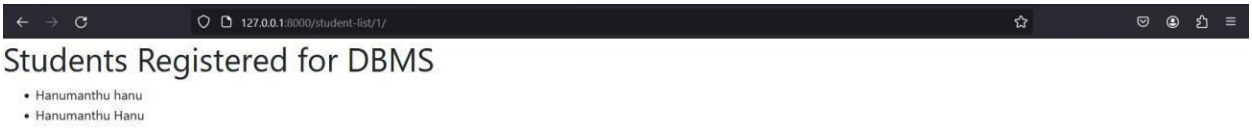



**Fig: Students Register Particular Courses Screen [You can Check as You Register Courses]**

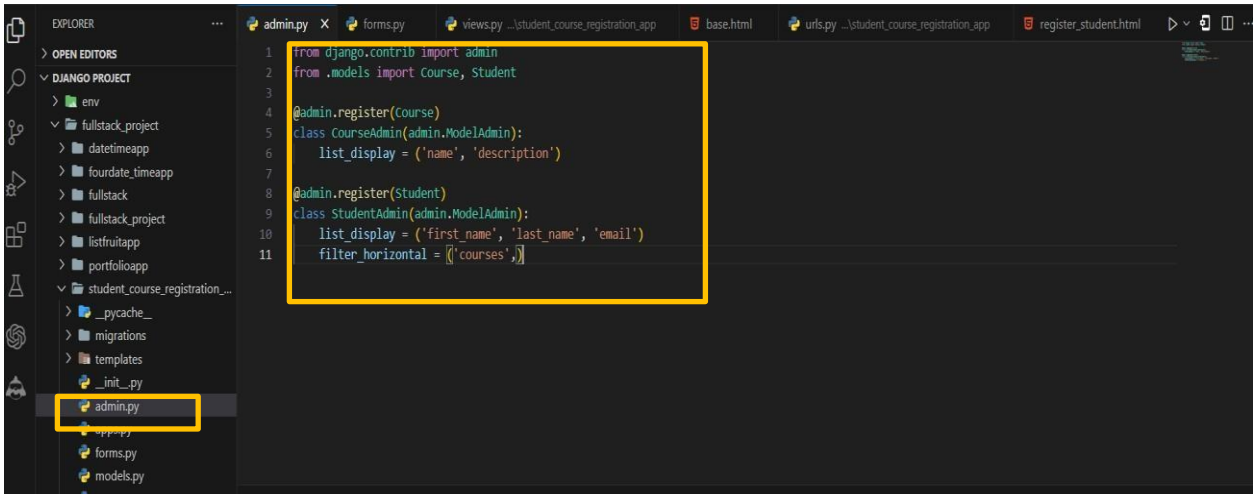
## Experiment-08

For student and course models created in Lab experiment for Module2, register admin interfaces, perform migrations and illustrate data entry through admin forms.

- In this code, we've registered the Course and Student models with the Django admin site.

**Step-01: Register models in the admin site**

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

@admin.register(Course)
class CourseAdmin(admin.ModelAdmin):
    list_display = ('name', 'description')

@admin.register(Student)
class StudentAdmin(admin.ModelAdmin):
    list_display = ('first_name', 'last_name', 'email')
    filter_horizontal = ('courses',)
```



- We've also specified the fields to be displayed in the list view for each model using the list_display attribute.
- For the StudentAdmin class, we've added the filter_horizontal attribute to display the courses field (which is a many-to-many relationship) as a horizontal filter in the admin interface.

**Step-02: Create and apply migrations**

**python manage.py makemigrations**

```
No such file or directory
(env) PS D:\6th sem 2021 scheme\Full Stack\FS lab\Django Project> cd fullstack_project
(env) PS D:\6th sem 2021 scheme\Full Stack\FS lab\Django Project\fullstack_project> python manage.py makemigrations
No changes detected
```

**python manage.py migrate**

```
(env) PS D:\6th sem 2021 scheme\Full Stack\FS lab\Django Project\fullstack_project> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions, student_course_registration_app
Running migrations:
  No migrations to apply.
```

**Step-03: Create a superuser**

- If you haven't already created a superuser for your project, you'll need to do so to access the admin interface.
- In your terminal or command prompt, run the following command

**python manage.py createsuperuser**

```
(env) PS D:\6th sem 2021 scheme\Full Stack\FS lab\Django Project\fullstack_project> python manage.py createsuperuser
Username (leave blank to use 'aryah'): admin
Email address:
Password:
Password (again):
This password is too short. It must contain at least 8 characters.
This password is too common.
This password is entirely numeric.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
```

**Type Username: admin**

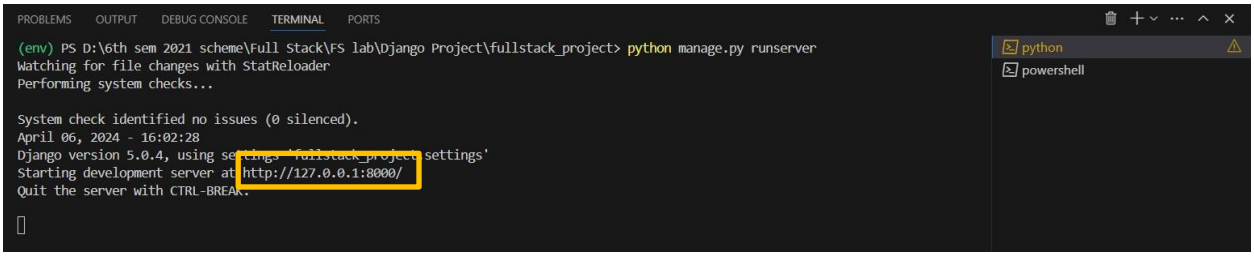
**Password:1234 [For Simply but your wish to create your own username and password]**

**Step-03: Access the admin interface**

- Start the Django development server by running the following command:
- In the VS Code terminal, navigate to your Django project's directory (the one containing manage.py).
- Run the development server

**python manage.py runserver**

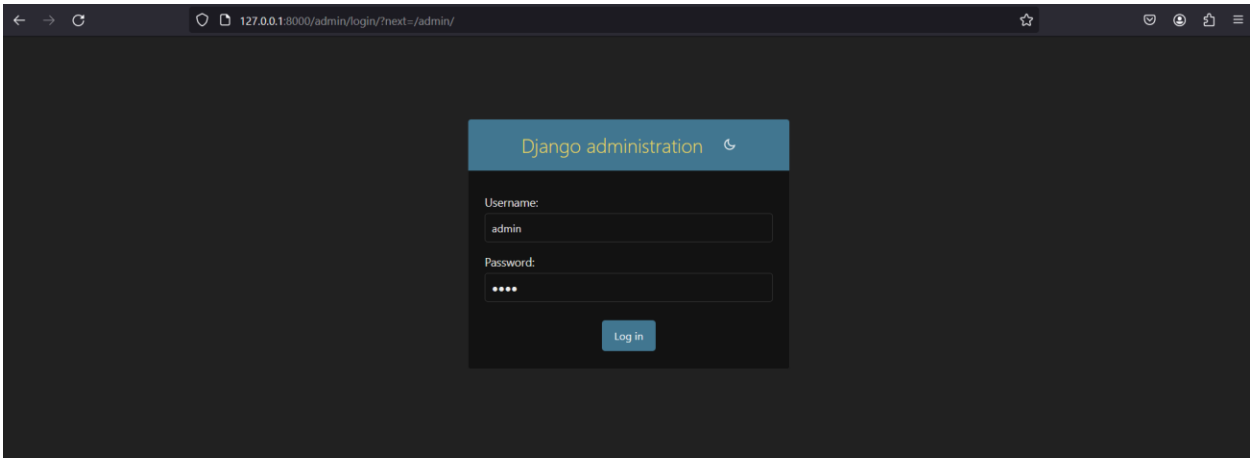
- Open your web browser and visit **http://127.0.0.1:8000/.**




- **Type or copy this http://127.0.0.1:8000/admin/**

**Step-04: Log in to the admin interface**

- Back in your web browser, enter the superuser credentials you just created and log in to the admin interface.



**Step-05: Add data through admin forms**

Once you're logged in to the admin interface, you'll see the "Courses" and "Students" sections in the left sidebar. Click on "Courses" to add a new course.

- Click on the "Add course" button in the top-right corner.

**Fig: Admin Main Screen**

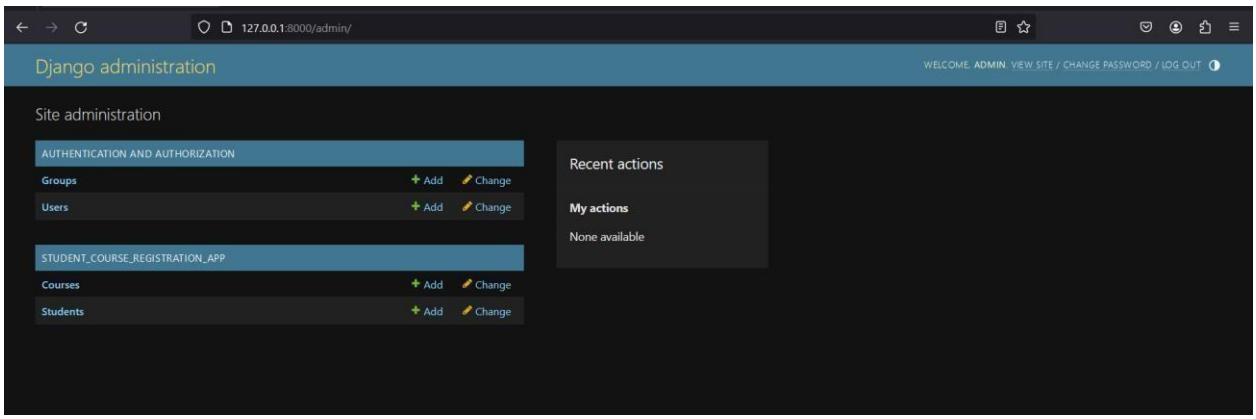
- Fill in the "Name" and "Description" fields for the new course.
- Click the "Save" button to create the new course.

Next, click on "Students" in the left sidebar to add a new student.

- Click on the "Add student" button in the top-right corner.
- Fill in the "First name", "Last name", and "Email" fields for the new student.
- In the "Courses" section, select the courses you want to enroll the student in by checking the appropriate boxes.
- Click the "Save" button to create the new student.

You can repeat these steps to add more courses and students through the admin interface. Additionally, you can view, edit, and delete existing courses and students from the admin interface.

**Final Output**



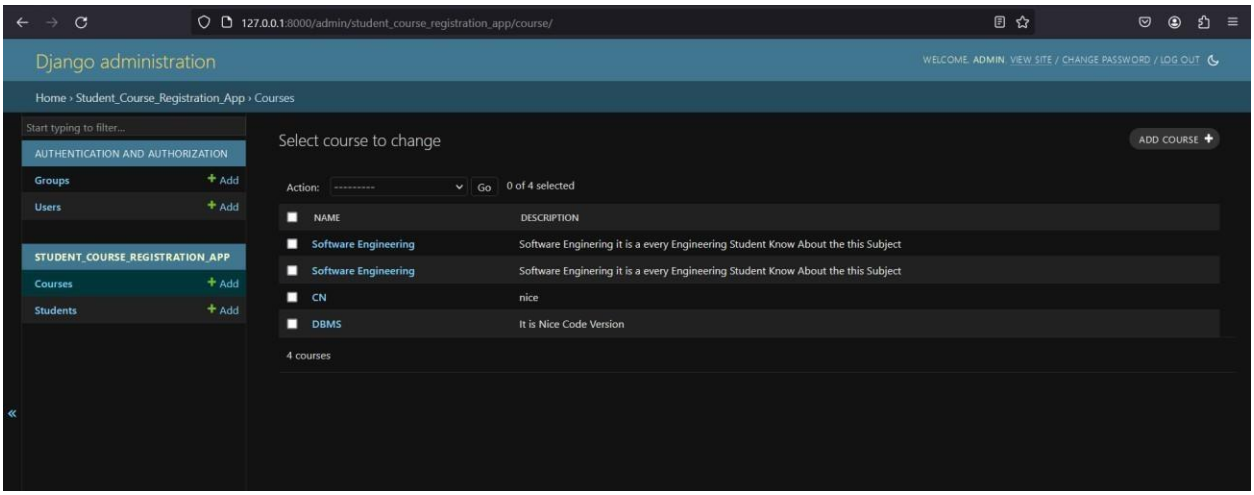



**Fig: Admin Course Screen**

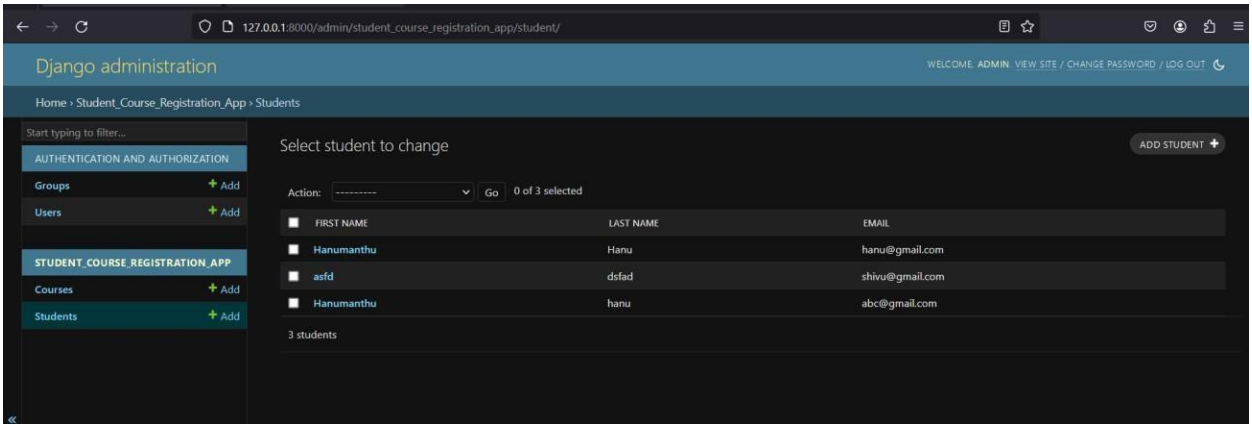



**Fig: Admin Student Screen**

# Experiment-09

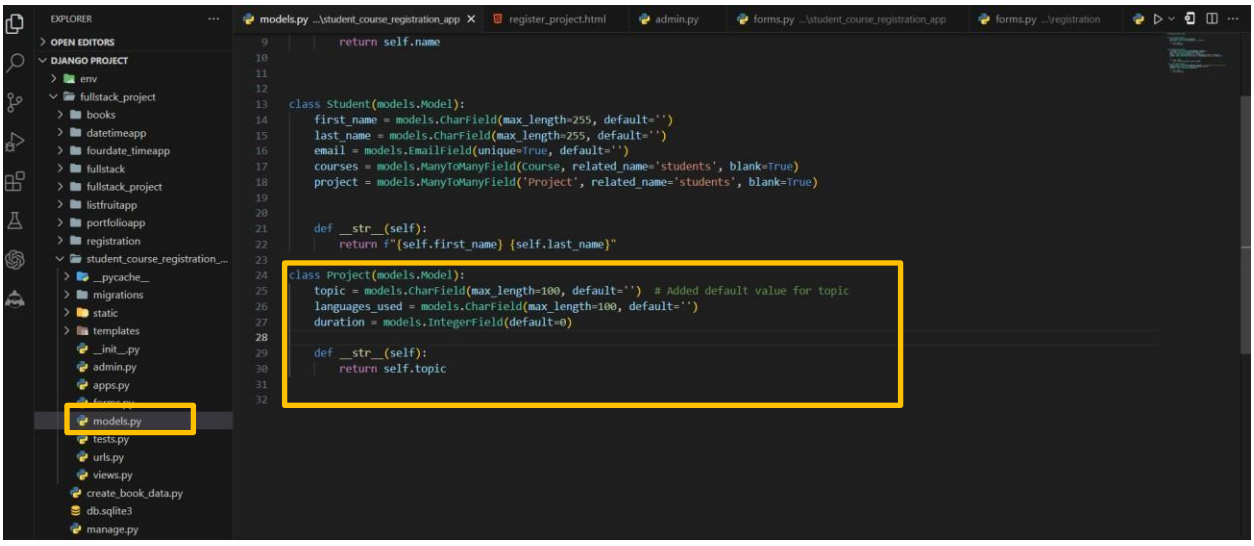
Develop a Model form for student that contains his topic chosen for project, languages used and duration with a model called project.

**Step 1: Use an existing app i.e. student_course_registration_app**

```
[08/Apr/2024 00:21:00] "GET /student-list/1/ HTTP/1.1" 200 1333
[08/Apr/2024 00:21:02] "GET /student-list/2/ HTTP/1.1" 200 1326
PS D:\6th sem 2021 scheme\Full Stack\FS lab\Django Project\fullstack_project> python manage.py makemigrations student_course_registration_app
```

**Step 2: Open the models.py file in the projects app and define the Project model.**

```
class Project(models.Model):
    topic = models.CharField(max_length=100, default='')  # Added default value for topic
    languages_used = models.CharField(max_length=100, default='')
    duration = models.IntegerField(default=0)
    def__str_(self):
        return self.topic
```



**Step 3: Create and apply migrations to create the necessary database tables.**

**python manage.py makemigrations**

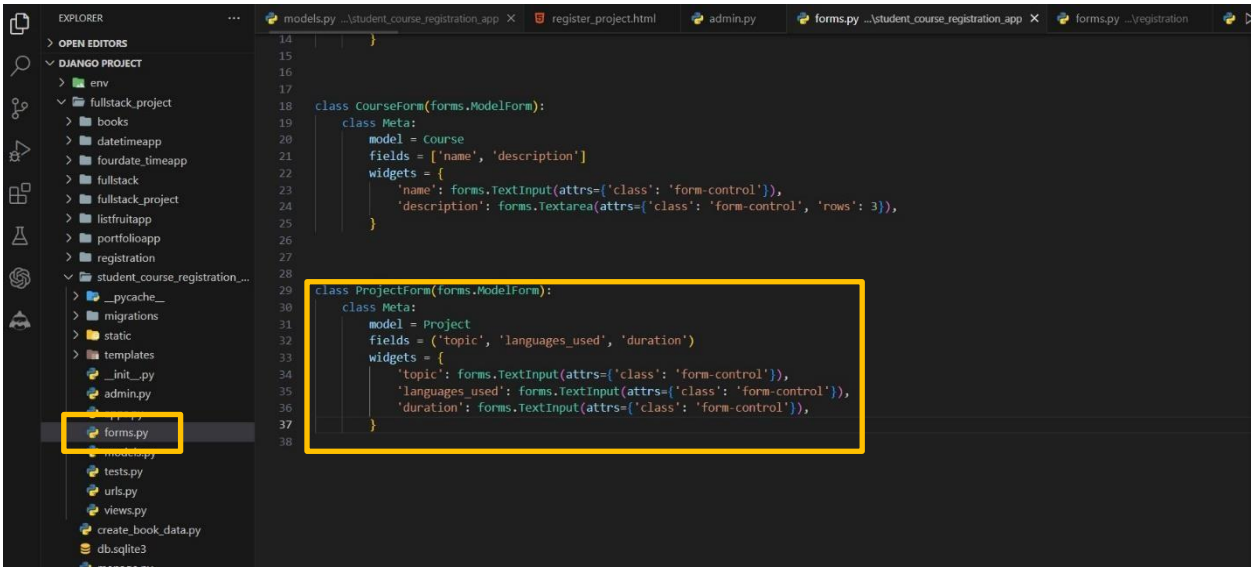
```
No such file or directory
(env) PS D:\6th sem 2021 scheme\Full Stack\FS lab\Django Project> cd fullstack_project
(env) PS D:\6th sem 2021 scheme\Full Stack\FS lab\Django Project\fullstack_project> python manage.py makemigrations
No changes detected
```

**python manage.py migrate**

```
(env) PS D:\6th sem 2021 scheme\Full Stack\FS lab\Django Project\fullstack_project> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions, student_course_registration_app
Running migrations:
  No migrations to apply.
```

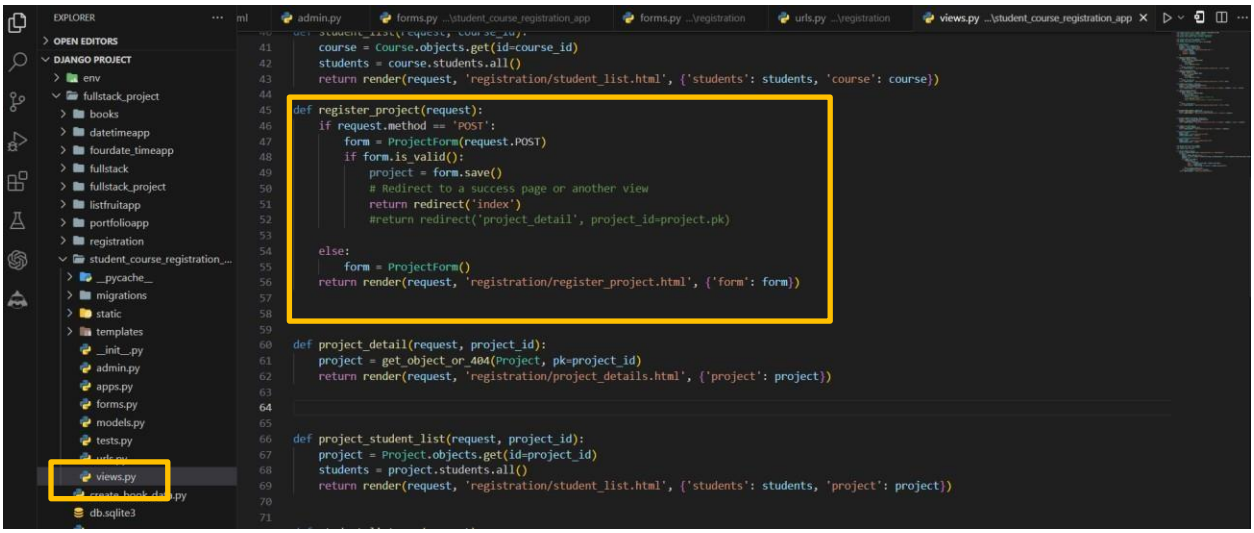
**Step 4: Create a forms.py file in the projects app and define a ProjectForm based on the Project model.**

```
class ProjectForm(forms.ModelForm):
    class Meta:
        model = Project
        fields = ('topic', 'languages_used', 'duration')
        widgets = {
            'topic': forms.TextInput(attrs={'class': 'form-control'}),
            'languages_used': forms.TextInput(attrs={'class': 'form-control'}),
            'duration': forms.TextInput(attrs={'class': 'form-control'}),
        }
```



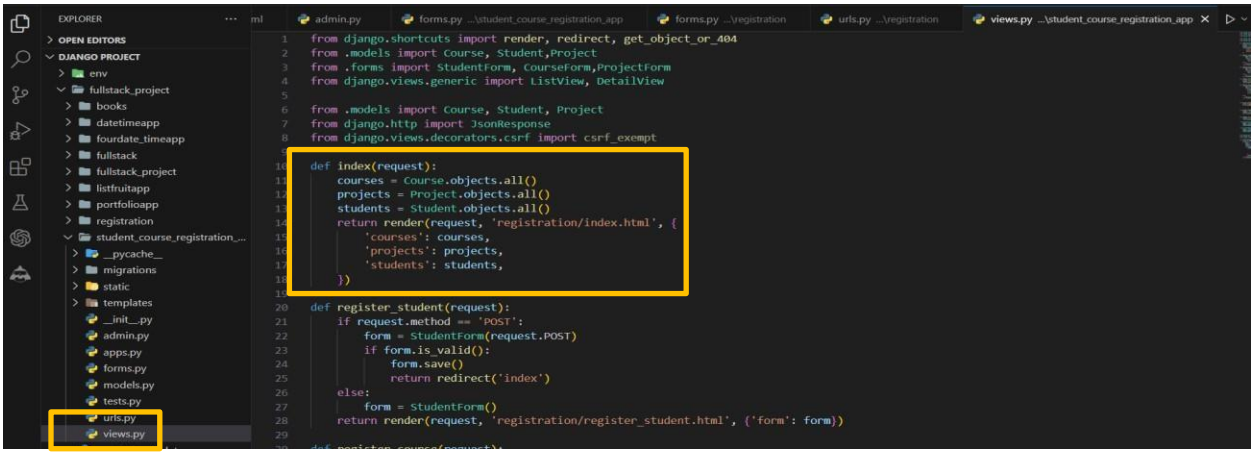

**Step 5: In views.py view function, import the ProjectForm and handle the form submission.**

```
from .forms import ProjectForm

def register_project(request):
    if request.method == 'POST':
        form = ProjectForm(request.POST)
        if form.is_valid():
            project = form.save()
            # Redirect to a success page or another view
            return redirect('index')
            #return redirect('project_detail', project_id=project.pk)
    else:
        form = ProjectForm()
    return render(request, 'registration/register_project.html', {'form': form})
```




**Step 6: In views.py index function, add the projects Objects for Displaying the Register Projects .**

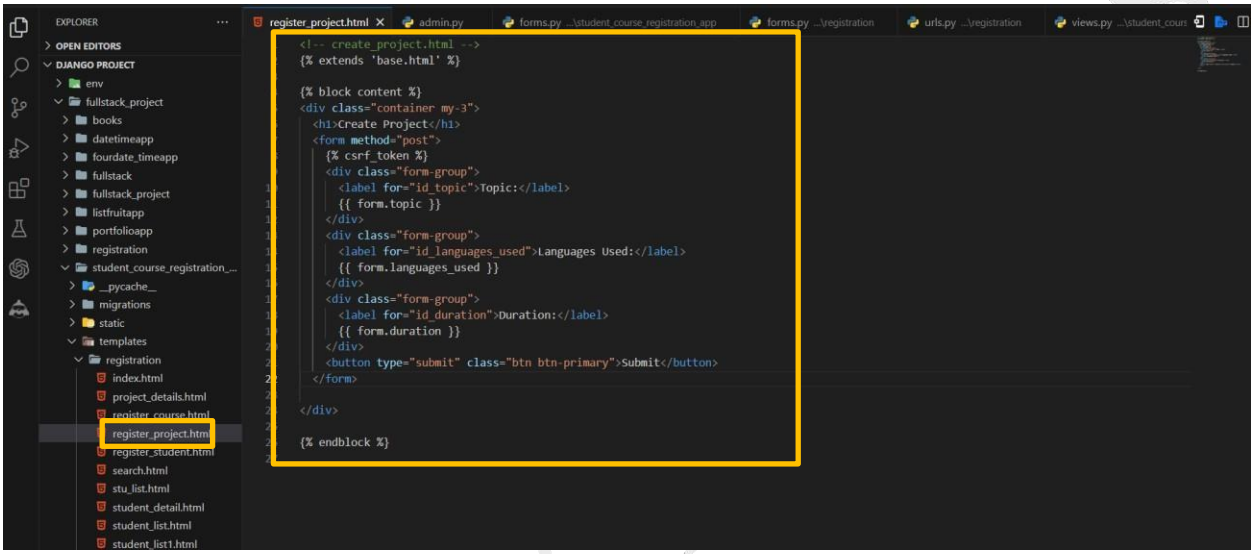
```
def index(request):
    courses = Course.objects.all()
    projects = Project.objects.all()
    return render(request, 'registration/index.html', {
        'courses': courses,
        'projects': projects,
    })
```



**Step 7: Create an HTML template file register_project.html in the templates/registration/ directory and render the form.**

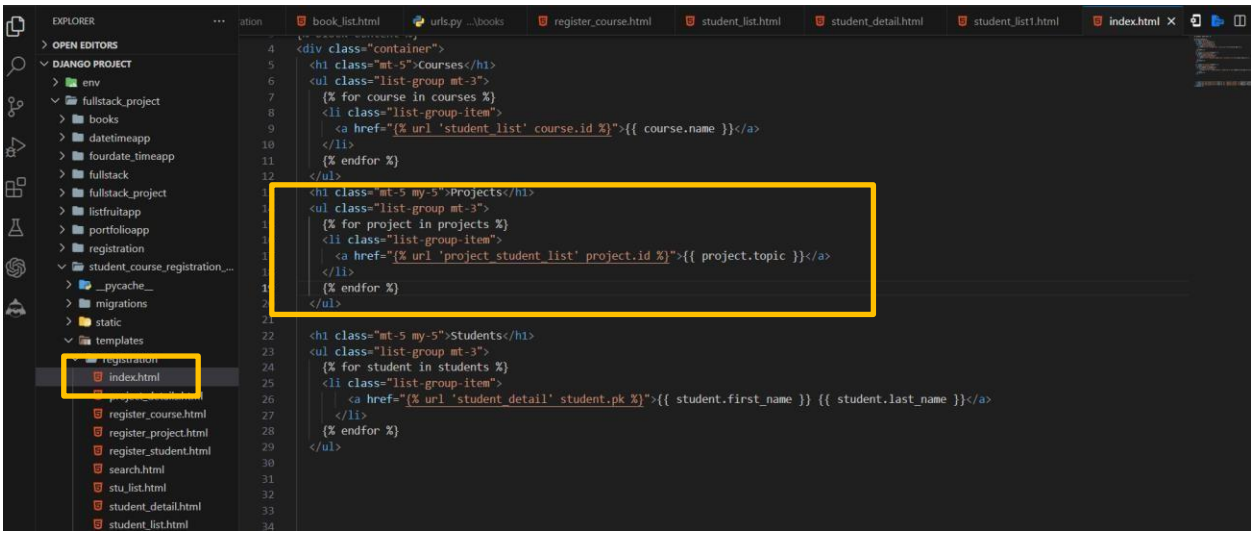
```
<!-- create_project.html -->
{% extends 'base.html' %}
    {{ form.duration }}

{% block content %}
<div class="container my-3">
  <h1>Create Project</h1>
  <form method="post">
    {% csrf_token %}
    <div class="form-group">
      <label for="id_topic">Topic:</label>
      {{ form.topic }}
    </div>
    <div class="form-group">
      <label for="id_languages_used">Languages Used:</label>
      {{ form.languages_used }}
    </div>
    <div class="form-group">
      <label for="id_duration">Duration:</label>
      </div>
```

```
    <button type="submit" class="btn btn-primary">Submit</button>
  </form>
</div>
{% endblock %}
```



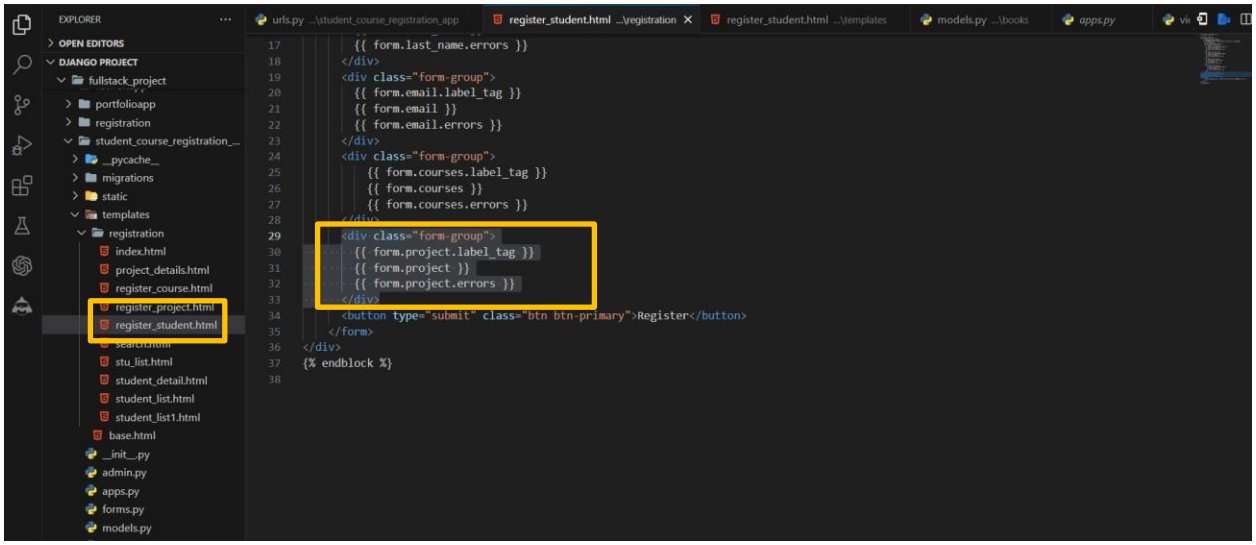

**Step 8: add the index.html to show the registered projects**

```
<h1 class="mt-5 my-5">Projects</h1>
  <ul class="list-group mt-3">
    {% for project in projects %}
    <li class="list-group-item">
      <a href="{% url 'project_student_list' project.id %}">{{ project.topic }}</a>
    </li>
    {% endfor %}
  </ul>
```



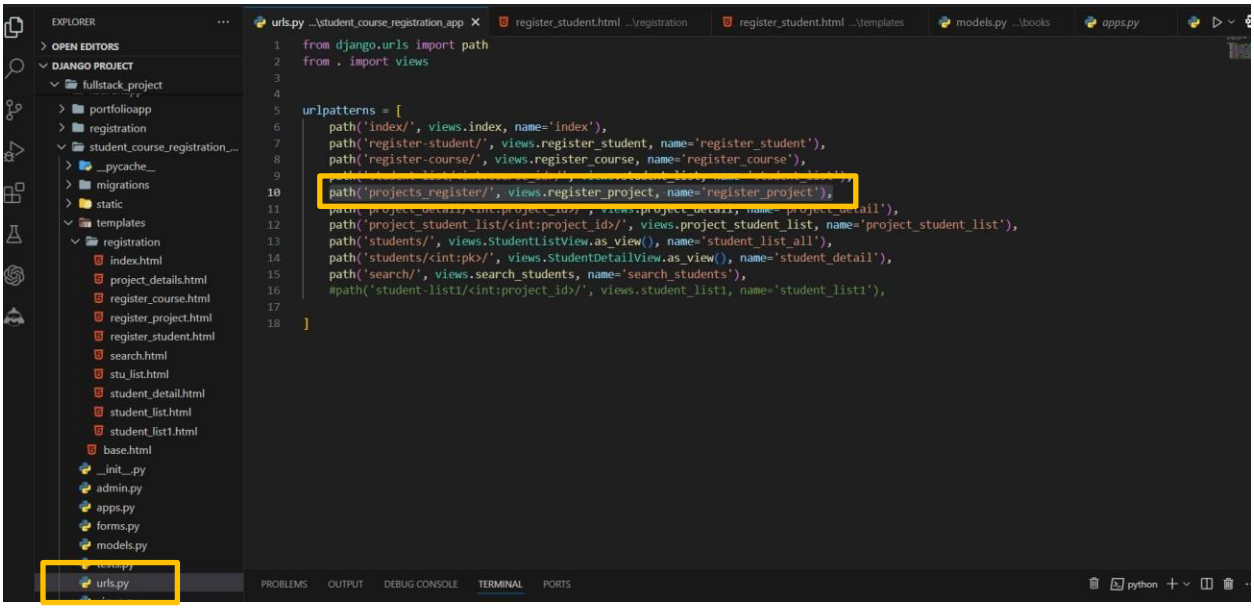

- Add the **register_student.html** to this is Student can Register the Created Project

```
<div class="form-group">
    {{ form.project.label_tag }}
    {{ form.project }}
    {{ form.project.errors }}
</div>
```



**Step 9: Add a URL pattern for the register_project in your urls.py file**

**path('projects_register/', views.register_project, name='register_project'),**

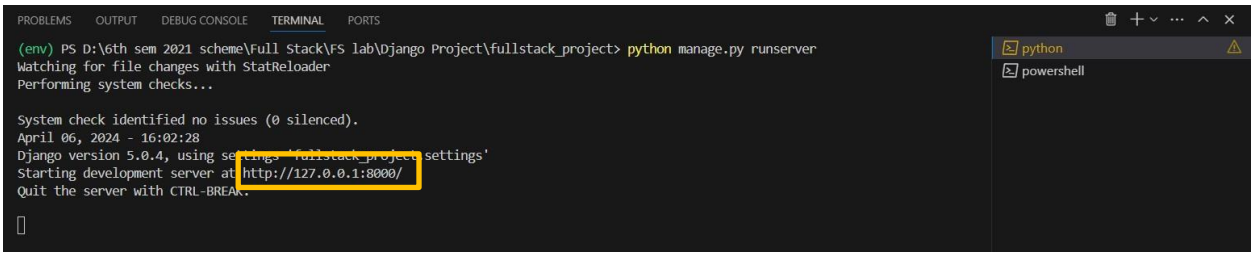



**Step-10: Run the development server**

- In the VS Code terminal, navigate to your Django project's directory (the one containing manage.py).
- Run the development server

**python manage.py runserver**

- Open your web browser and visit **http://127.0.0.1:8000/.**




- **Type or copy this http://127.0.0.1:8000/index/**

**Final Output By Clicking Register Project and Register Student**

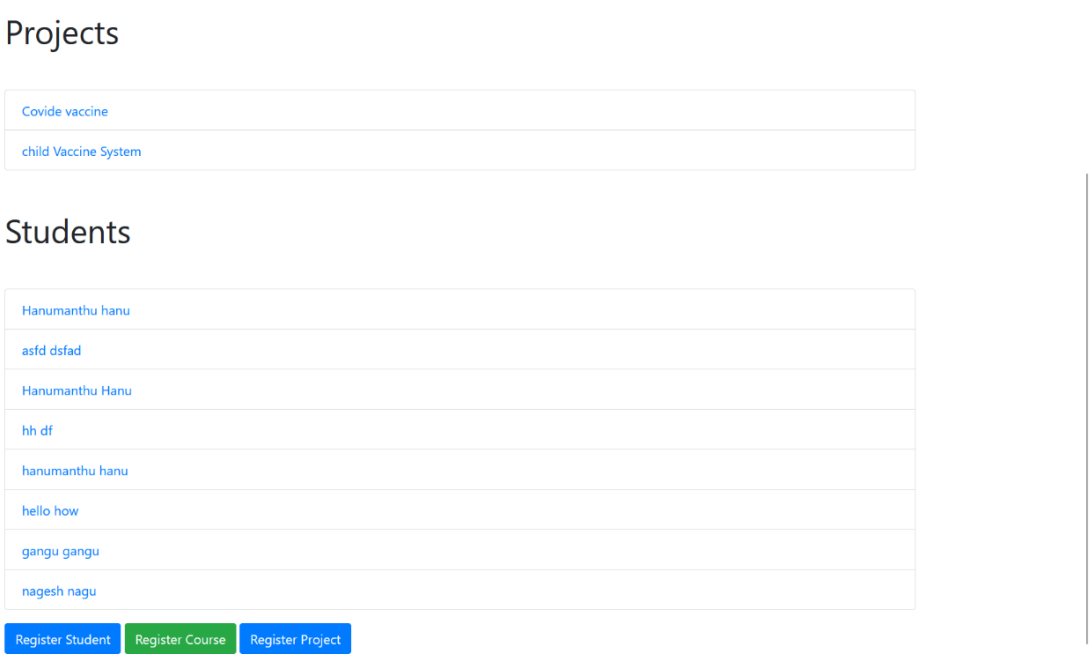



**Fig: Home Page Screen**

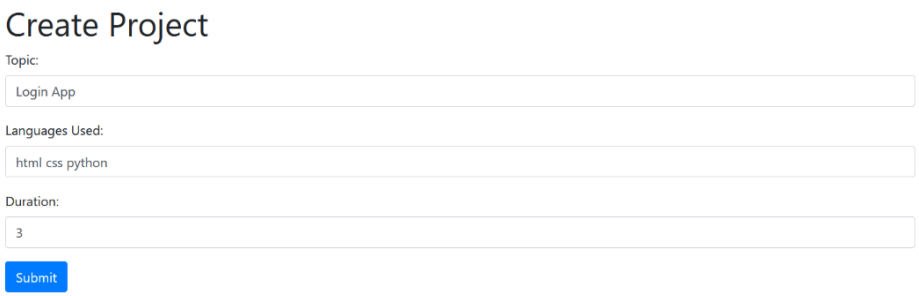



**Fig: Project Creation Screen**


**Fig: Project Successfully Creation Screen**


**Fig: Student Register the Project**


**Fig: Student Registered Particular Projects**

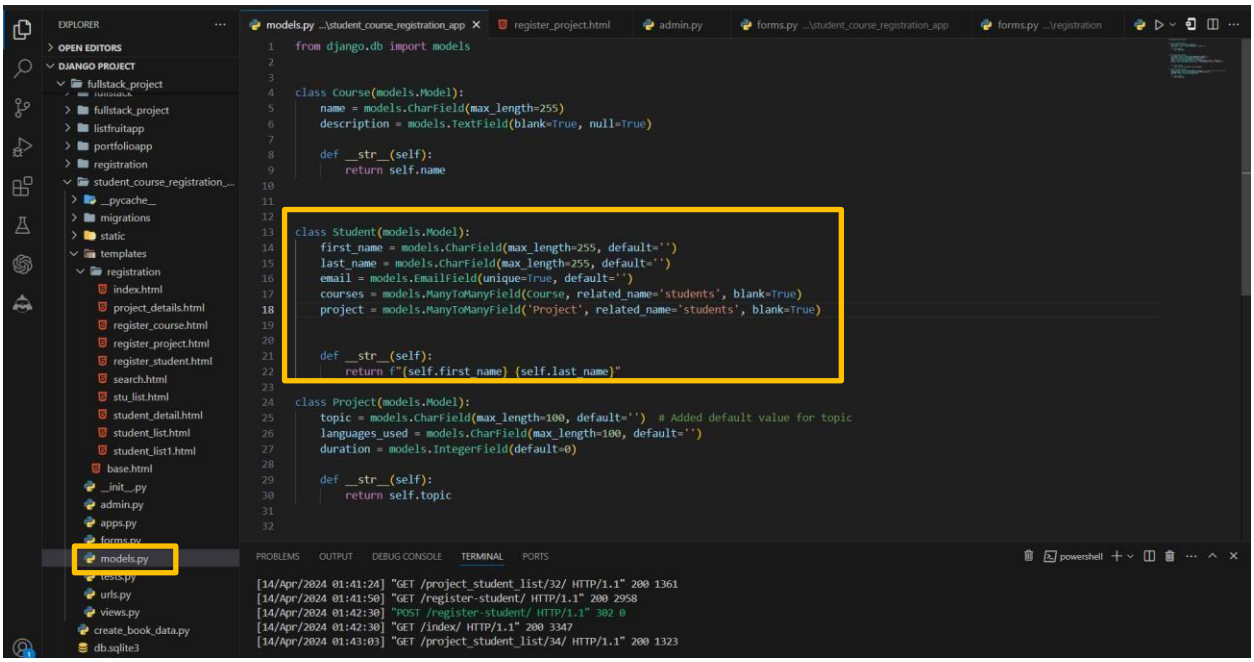
## Experiment-10

For students enrolment developed in Module 2, create a generic class view which displays list of students and detail view that displays student details for any selected student in the list.

**Step-1: First, let's create models for Student and Course if you haven't already:**

## Models.py

```
class Student(models.Model):
    first_name = models.CharField(max_length=255, default='')
    last_name = models.CharField(max_length=255, default='')
    email = models.EmailField(unique=True, default='')
    courses = models.ManyToManyField(Course, related_name='students', blank=True)
    project = models.ManyToManyField('Project', related_name='students', blank=True)
```



**Step-2: Next, create views for the list and detail views and import necessary packages:**

**from django.views.generic import ListView, DetailView**

**class StudentListView(ListView):**

**model = Student**

**template_name = 'registration/stu_list.html'**

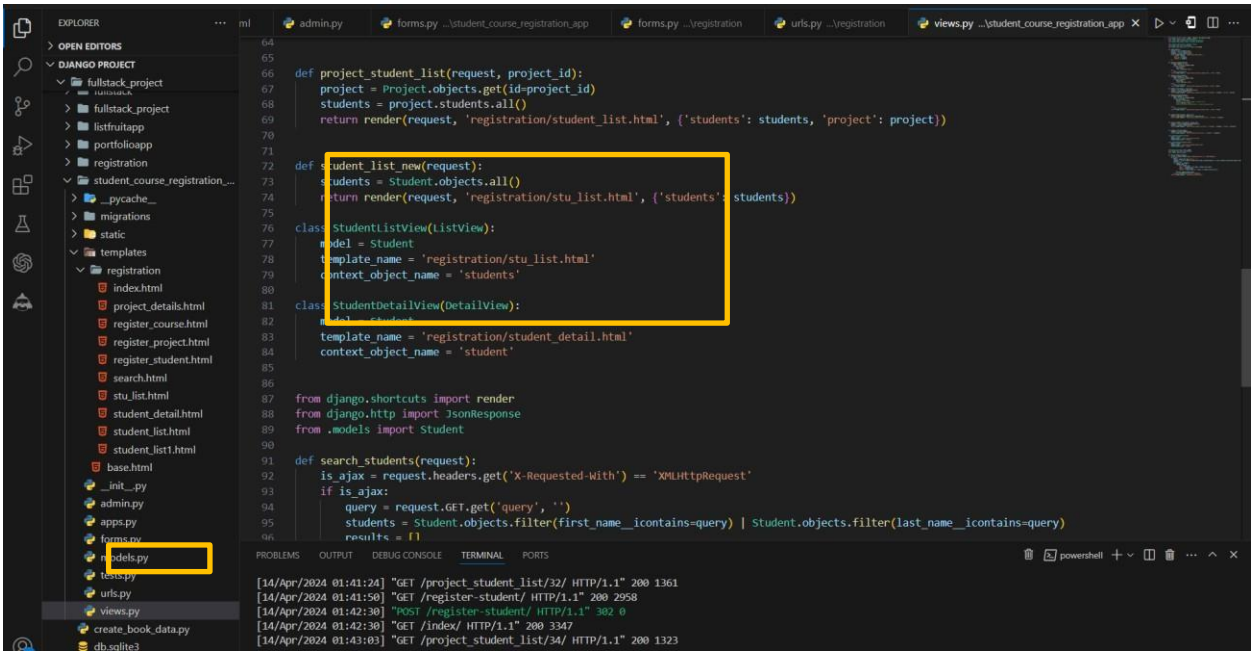
**context_object_name = 'students'**

**class StudentDetailView(DetailView):**

**model = Student**

**template_name = 'registration/student_detail.html'**

**context_object_name = 'student'**



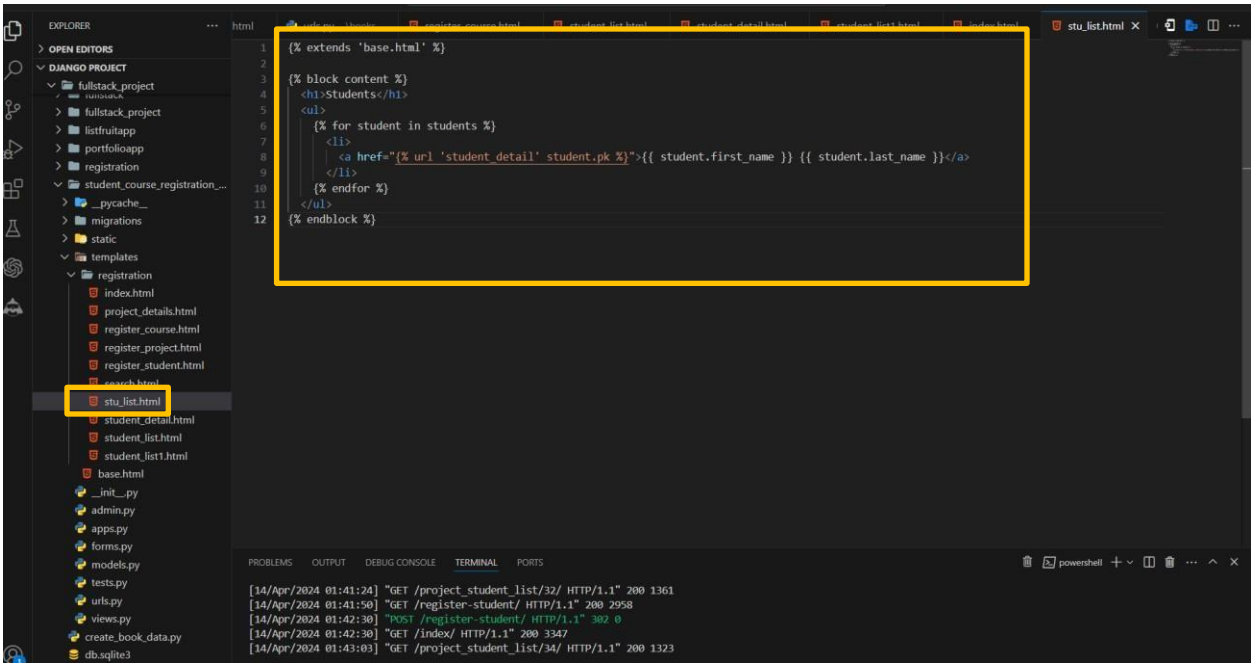

- In the StudentListView, we've overridden the get_queryset method to filter the students based on the course_id parameter in the URL. If the course_id is

provided, it will return the students associated with that course; otherwise, it will return all students.

**Step-3: Now, let's create the templates:**

**stu_list.html**

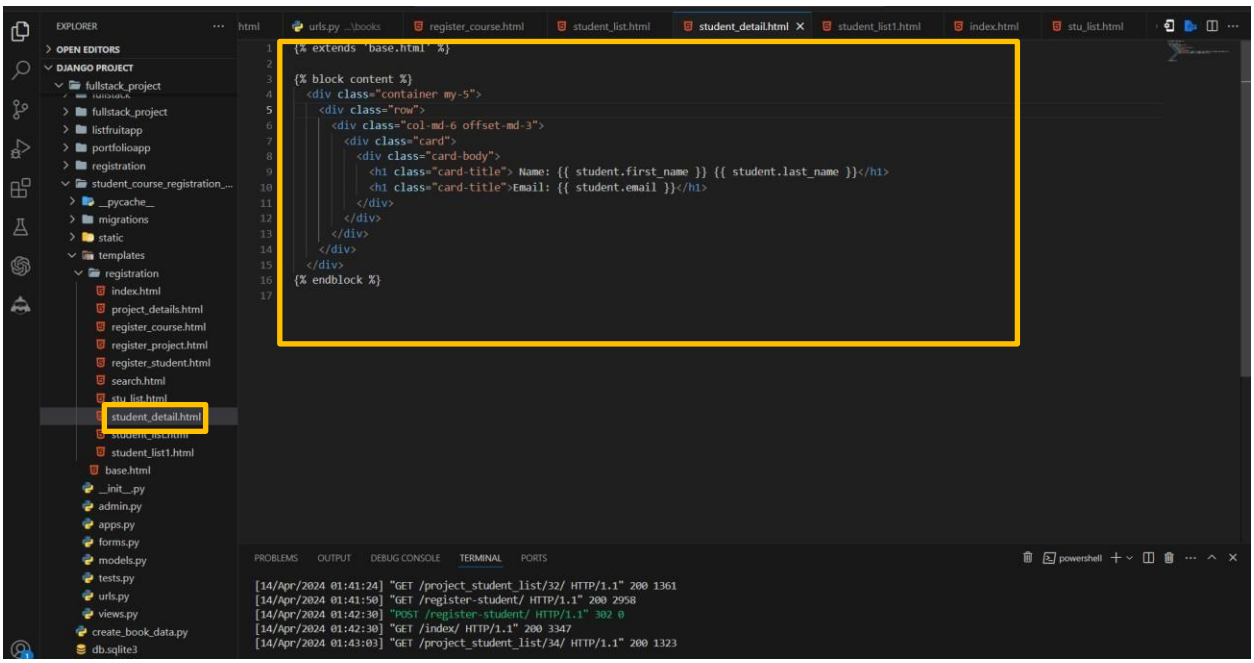
```
{% extends 'base.html' %}
{% block content %}
  <h1>Students</h1>
  <ul>
    {% for student in students %}
      <li>
        <a href="{% url 'student_detail' student.pk %}">{{ student.first_name }} {{ student.last_name }}</a>
      </li>
    {% endfor %}
  </ul>
{% endblock %}
```




**student_detail.html**

```
{% extends 'base.html' %}
{% block content %}
  <div class="container my-5">
    <div class="row">
      <div class="col-md-6 offset-md-3">
        <div class="card">
          <div class="card-body">
            <h1 class="card-title"> Name: {{ student.first_name }} {{ student.last_name }}</h1>
            <h1 class="card-title">Email: {{ student.email }}</h1>
          </div>
        </div>
```

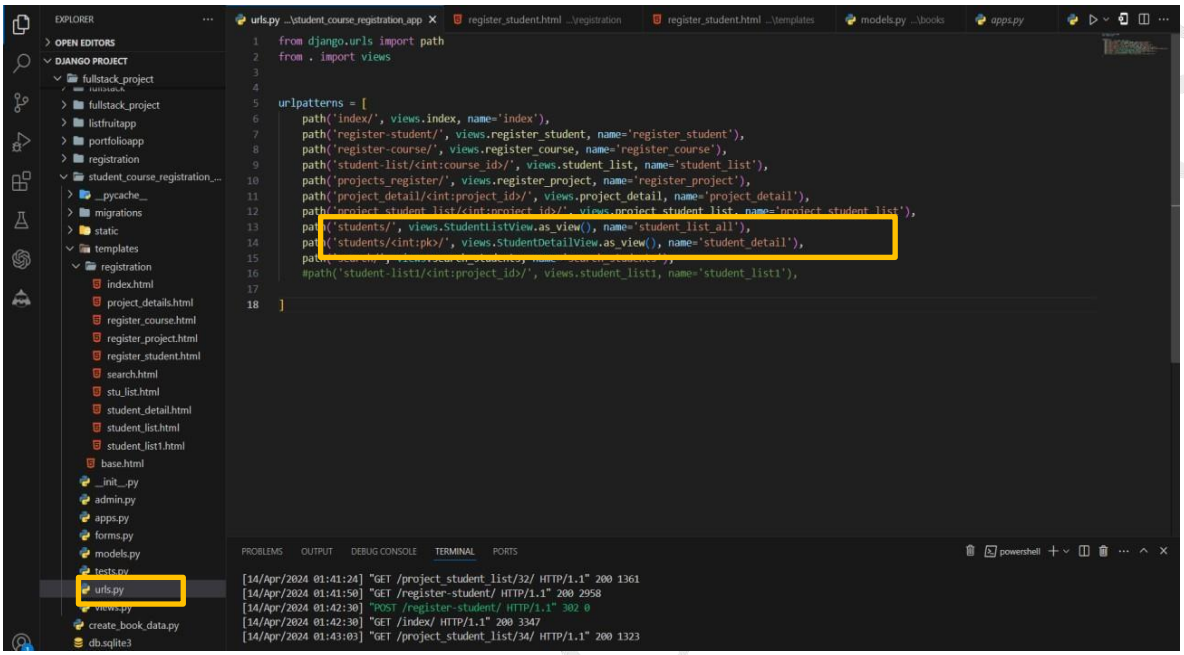
```
    </div>
  </div>
</div>
{% endblock %}
```



**Step-4: add the URL patterns:**

path('students/', views.StudentListView.as_view(), name='student_list_all'),

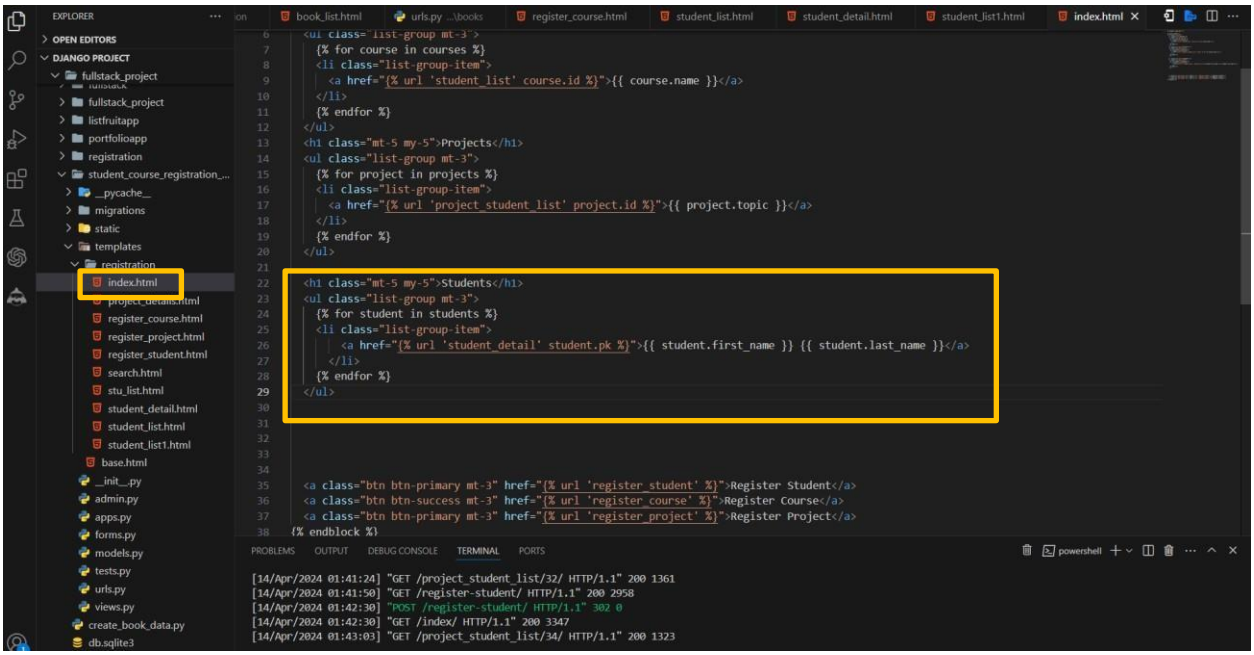
path('students/<int:pk>/', views.StudentDetailView.as_view(), name='student_detail'),




**Step-5: add Index.html For showing the Students In Main Page: Index.html**

```
<h1 class="mt-5 my-5">Students</h1>
<ul class="list-group mt-3">
  {% for student in students %}
  <li class="list-group-item">
    <a href="{% url 'student_detail' student.pk %}">{{ student.first_name }} {{
student.last_name }}</a>
    </li>
{% endfor %}
```

</ul>

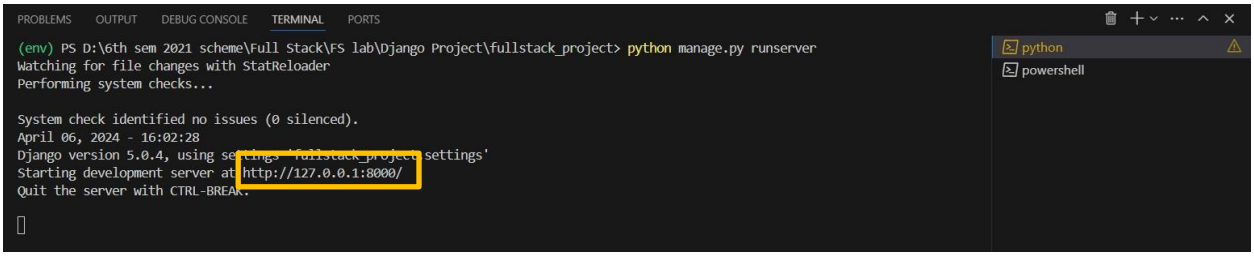



### Step-6: Run the development server

- In the VS Code terminal, navigate to your Django project's directory (the one containing manage.py).
- Run the development server

**python manage.py runserver**

- Open your web browser and visit **http://127.0.0.1:8000/.**




- **Type or copy this http://127.0.0.1:8000/index/**

## Final Output By Clicking Register Project and Register Student

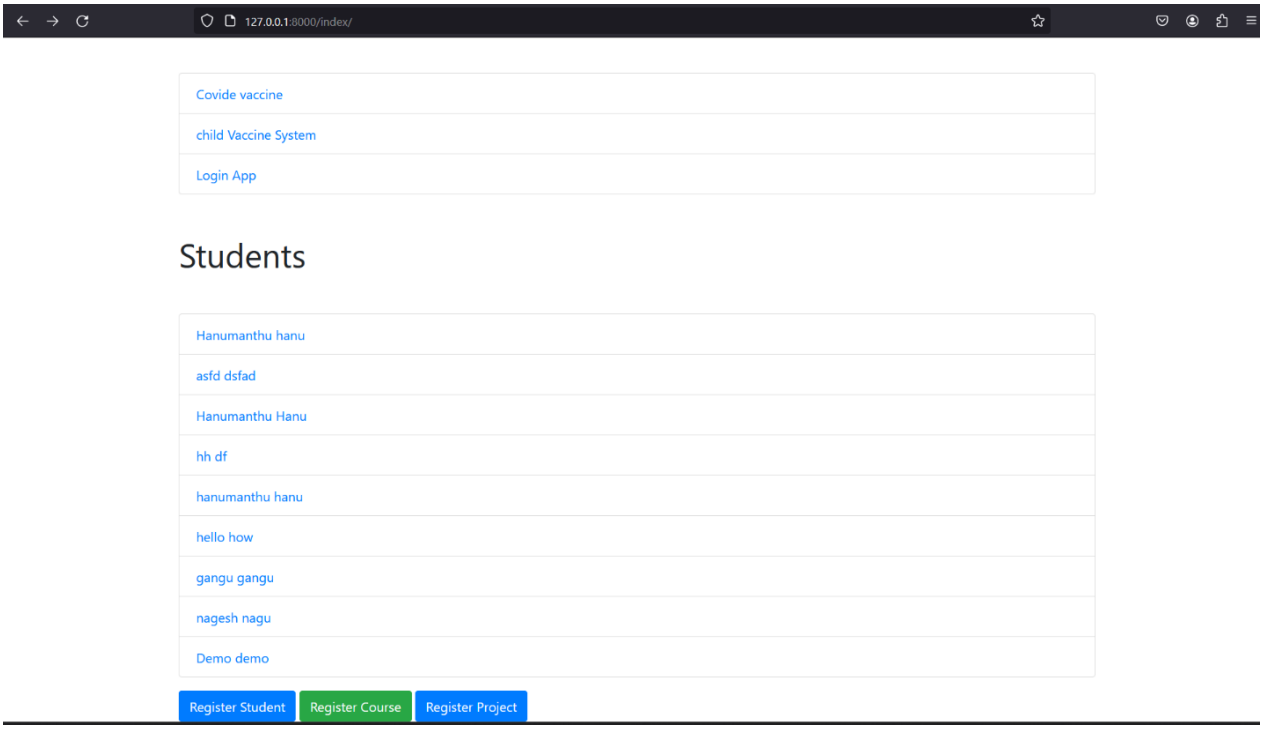



**Fig: Main Screen**

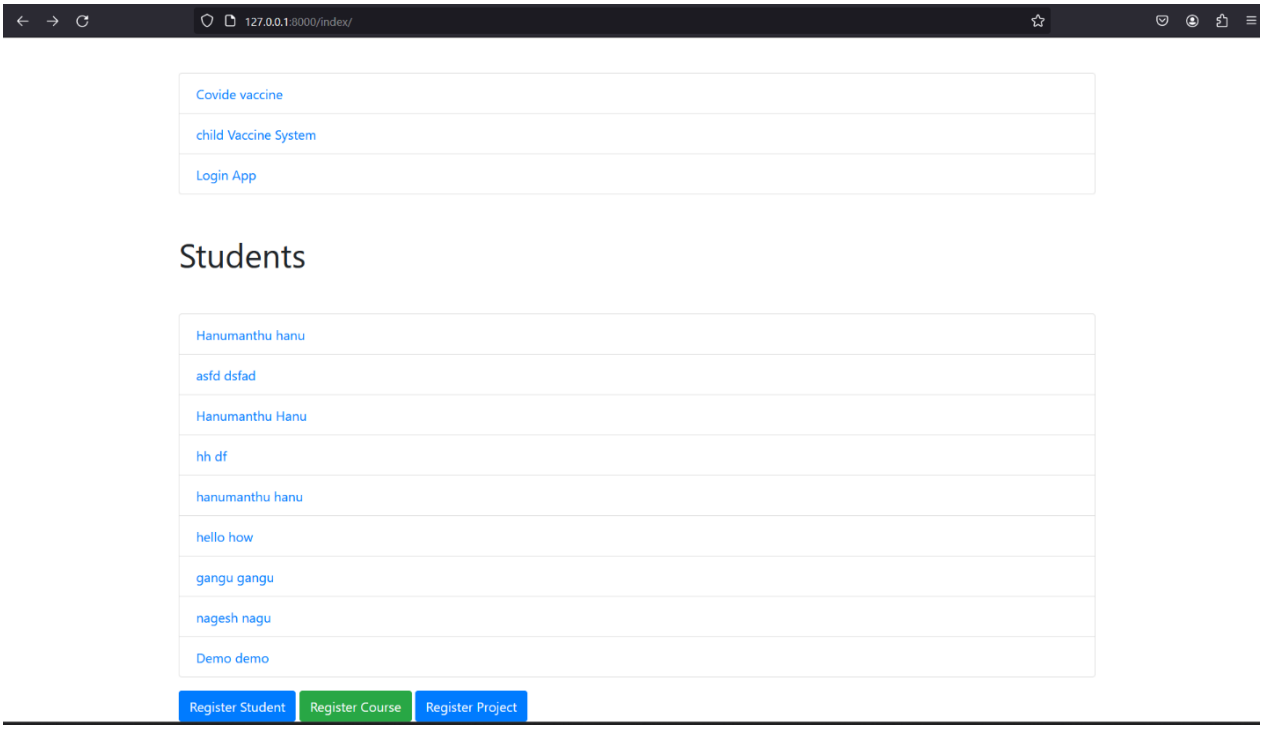



**Fig: List View of Students Screen**

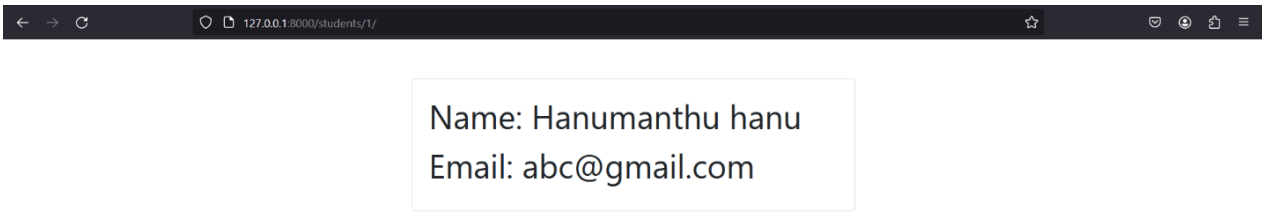



**Fig: Detailed View of Student Screen**

# Experiment-11

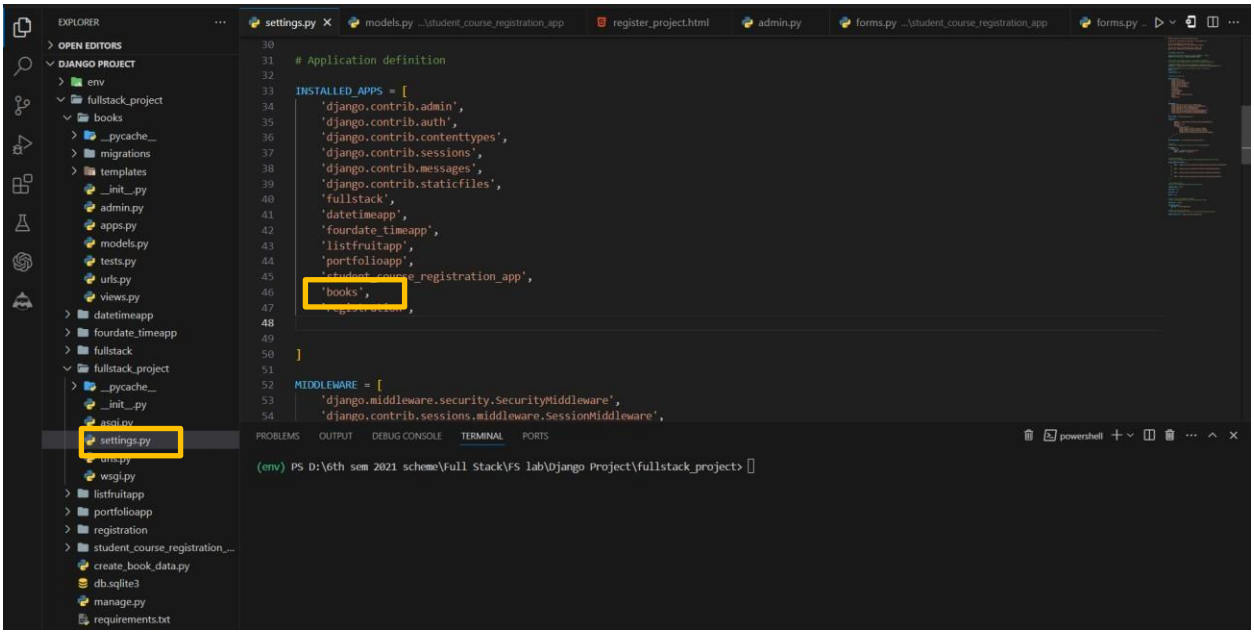
Develop example Django app that performs CSV and PDF generation for any models created in previous laboratory component.

**Step-01**: **This app will be created in the Django project we made earlier.**


**Step-02: Add the app to INSTALLED_APPS**

Open the settings.py file in your project's directory (e.g., **fullstack_project/settings.py**).

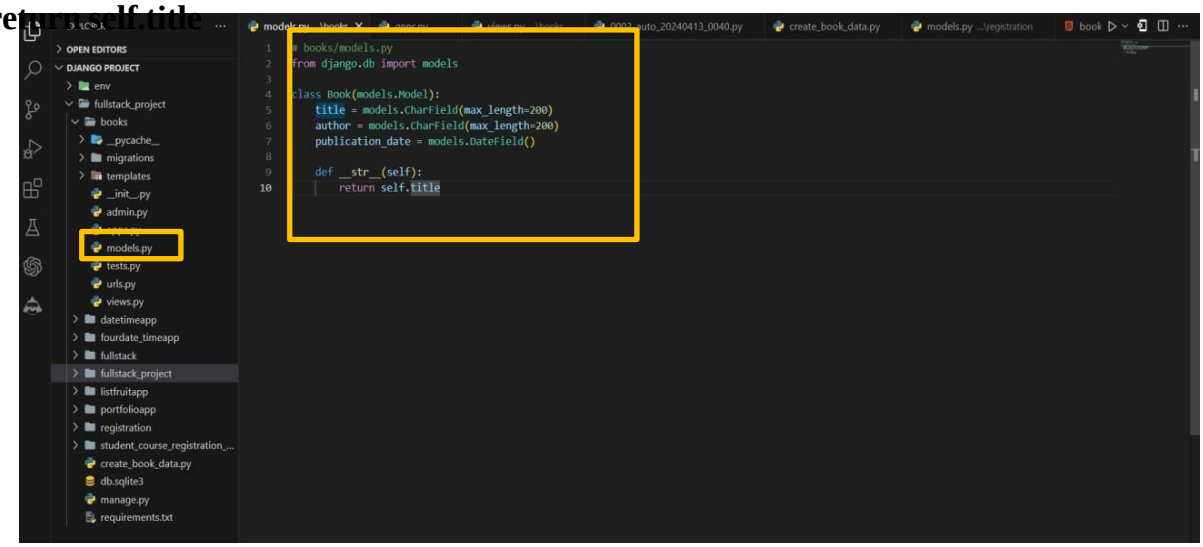
Locate the **INSTALLED_APPS** list and add the name of your new app to the list:



**Step-03: Create models**

- Open the **models.py** file inside the student_course_registration_app and define your models:

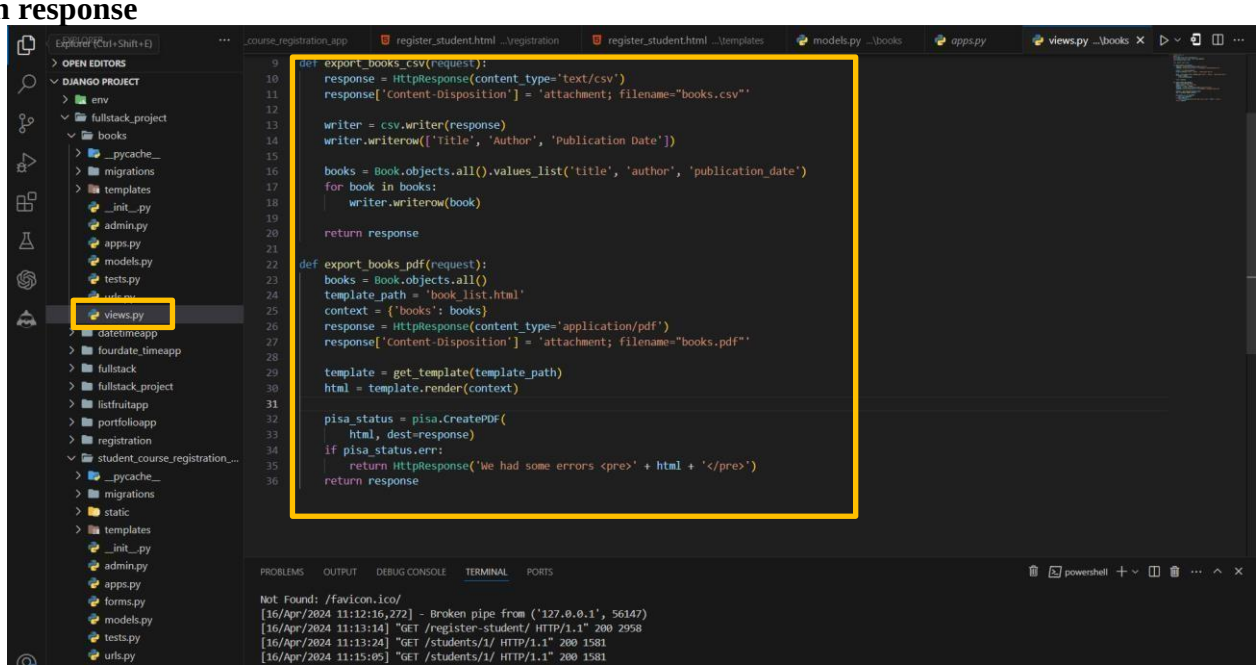
```
from django.db import models

class Book(models.Model):
    title = models.CharField(max_length=200)
    author = models.CharField(max_length=200)
    publication_date = models.DateField()
    def __str (self):
        return self.title
```



**Step-04: Create a views.py file in your books app and define the views for generating CSV and PDF files**

```
import csv

from django.http import HttpResponse

from django.template.loader import get_template

from xhtml2pdf import pisa

from .models import Book


def export_books_csv(request):

    response = HttpResponse(content_type='text/csv')

    response['Content-Disposition'] = 'attachment; filename="books.csv"'

    writer = csv.writer(response)

    writer.writerow(['Title', 'Author', 'Publication Date'])

    books = Book.objects.all().values_list('title', 'author', 'publication_date')

    for book in books:

        writer.writerow(book)

    return response

def export_books_pdf(request):

    books = Book.objects.all()
    template_path = 'book_list.html'

    context = {'books': books}
```

```
        response = HttpResponse(content_type='application/pdf')
        response['Content-Disposition'] = 'attachment; filename="books.pdf"'
        template = get_template(template_path)
html = template.render(context)
pisa_status = pisa.CreatePDF(
    html, dest=response)
if pisa_status.err:
    return HttpResponse('We had some errors <pre>' + html + '</pre>')
return response
```



**Step-04: In your books app, create a templates directory and an html file for rendering the PDF:**

**books/**

**templates/**

**books/**

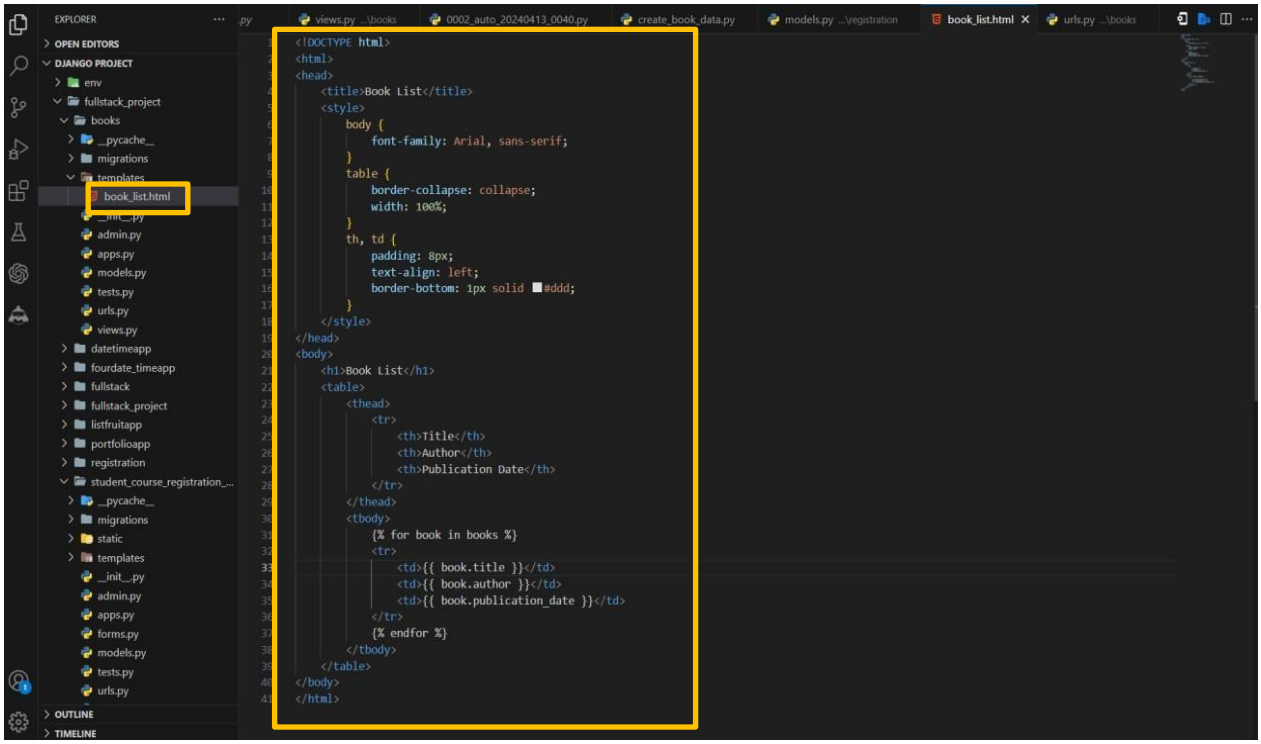
**book_list.html**

**book_list.html**

```
<!DOCTYPE html>
<html>
<head>
    <title>Book List</title>
    <style>
        body {
            font-family: Arial, sans-serif;
        }
        table {
            border-collapse: collapse;
            width: 100%;
                }
                th, td {
                    padding: 8px;
```

```
            text-align: left;
            border-bottom: 1px solid #ddd;
        }
    </style>
</head>
<body>
    <h1>Book List</h1>
    <table>
        <thead>
            <tr>
                <th>Title</th>
                <th>Author</th>
                <th>Publication Date</th>
            </tr>
        </thead>
        <tbody>
            {% for book in books %}
            <tr>
                <td>{{ book.title }}</td>
                <td>{{ book.author }}</td>
                <td>{{ book.publication_date }}</td>
```

```
        </tr>
        {% endfor %}
    </table>
</body>
</html>
```




```
        </tbody>
```

**Step-05: add the URL patterns and import Necessary Packages:**

from django.urls import path

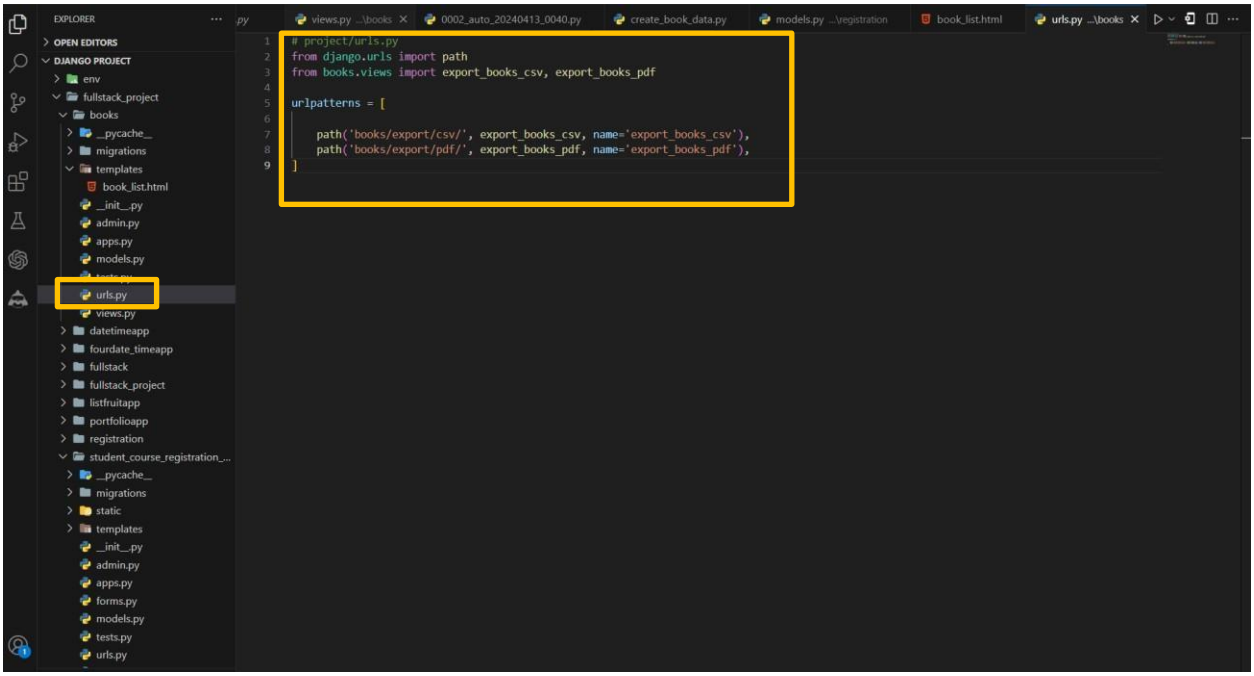
from books.views import export_books_csv, export_books_pdf

urlpatterns = [

path('books/export/csv/', export_books_csv, name='export_books_csv'),

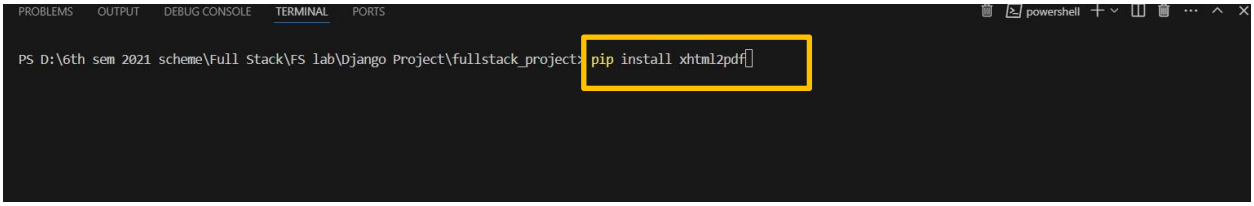
path('books/export/pdf/', export_books_pdf, name='export_books_pdf'),

]




**Step-06: Install the xhtml2pdf library for generating PDF files:**

pip install xhtml2pdf



easy way to create objects interactively.

**Data Migration**



- If you want to include the book data as part of your project's initial data setup, you can create a data migration. Here's how you can do it

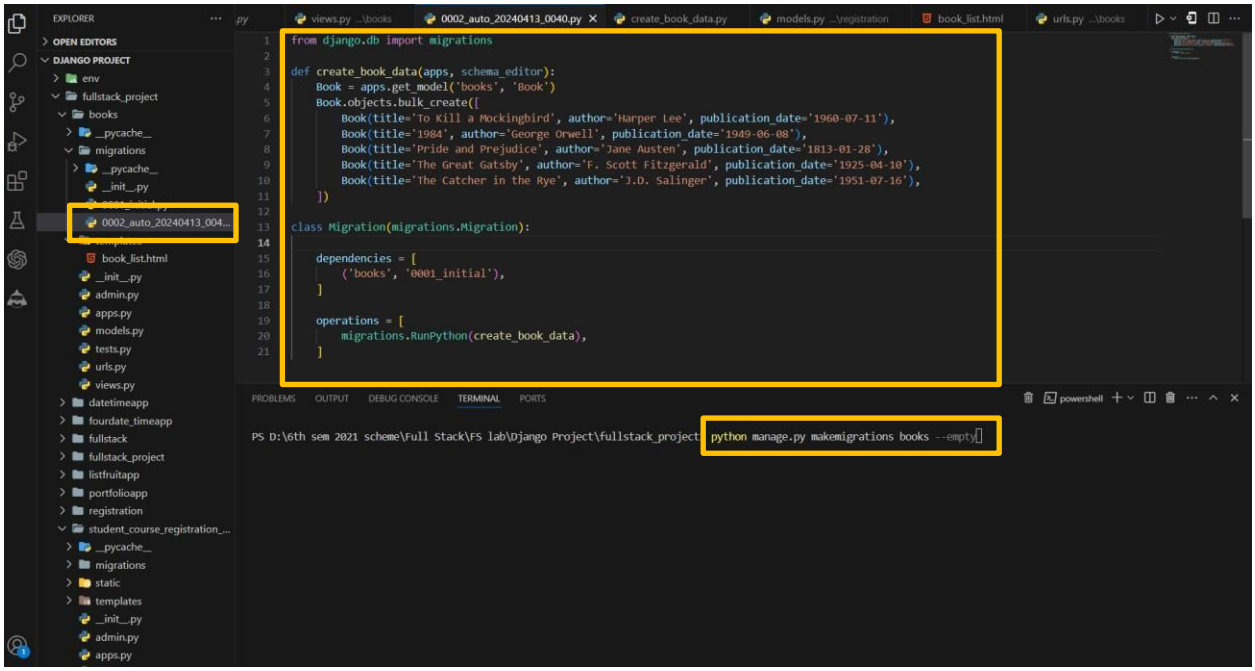
**Step-07: Create the book titles and author and publication date**

**Django Shell**

**Run the following command to create a new data migration**

- If you just want to create some sample data for testing or development purposes, you can run the code in the Django shell as shown in the previous example. This is a quick and

python manage.py makemigrations books –empty




- This will create a new empty migration file in the book's/migrations/

- Open the newly created migration file (e.g., books/migrations/0002_auto_<...>.py) and add the code to create the book objects in the operations list of the Migration class.

```
from django.db import migrations

def create_book_data(apps, schema_editor):
    Book = apps.get_model('books', 'Book')
```

python manage.py migrate

- This will create the book objects in your database.

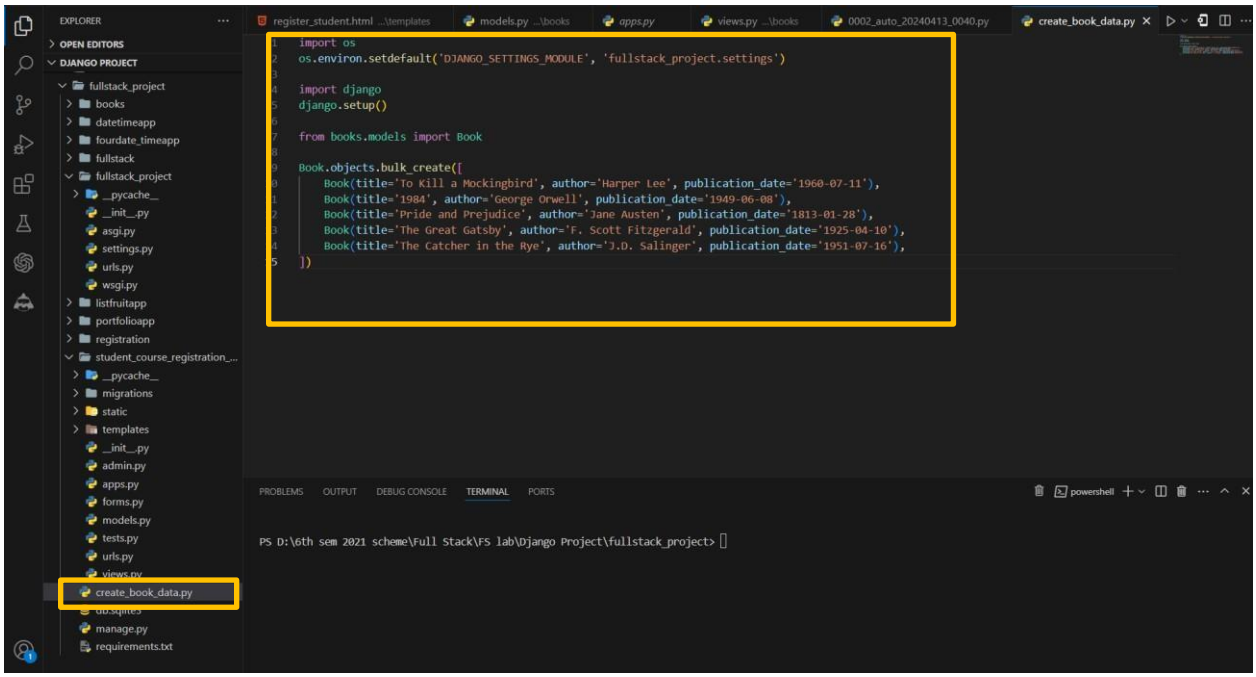
```
Book.objects.bulk_create([
    Book(title='To Kill a Mockingbird', author='Harper Lee', publication_date='1960-07-11'),
    Book(title='1984', author='George Orwell', publication_date='1949-06-08'),
    Book(title='Pride and Prejudice', author='Jane Austen', publication_date='1813-01-28'),
    Book(title='The Great Gatsby', author='F. Scott Fitzgerald', publication_date='1925-04-10'),
    Book(title='The Catcher in the Rye', author='J.D. Salinger', publication_date='1951-07-16'),
])

class Migration(migrations.Migration):
    dependencies = [
        ('books', '0001_initial'),
    ]
    operations = [
        migrations.RunPython(create_book_data),
    ]
```

**Run the following command to apply the data migration**

**Step-08: Custom Script**

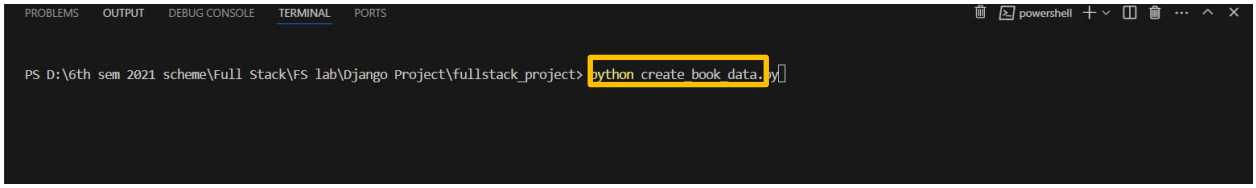
- If you need to create the book data programmatically (e.g., during deployment or as part of a data import process), you can create a custom Python script and run it as needed.
- Create a new Python file (e.g., create_book_data.py) in your project's root directory, and add the code to create the book objects:

```
import os

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'fullstack_project.settings')

import django

django.setup()

from books.models import Book

Book.objects.bulk_create([

    Book(title='To Kill a Mockingbird', author='Harper Lee', publication_date='1960-07-11'),

    Book(title='1984', author='George Orwell', publication_date='1949-06-08'),

    Book(title='Pride and Prejudice', author='Jane Austen', publication_date='1813-01-28'),

    Book(title='The Great Gatsby', author='F. Scott Fitzgerald', publication_date='1925-04-10'),

    Book(title='The Catcher in the Rye', author='J.D. Salinger', publication_date='1951-07-16'),

])
```




Run the script using the following command:

python create_book_data.py




### Step-09: Update project URLs

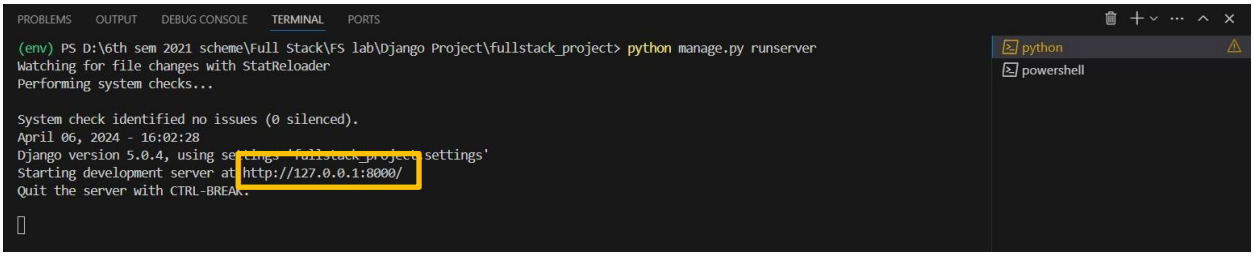
- Open the **urls.py** file inside your project and include the URLs from the book app:

**Step-09: Run the development server**

- In the VS Code terminal, navigate to your Django project's directory (the one containing manage.py).
- Run the development server

**python manage.py runserver**

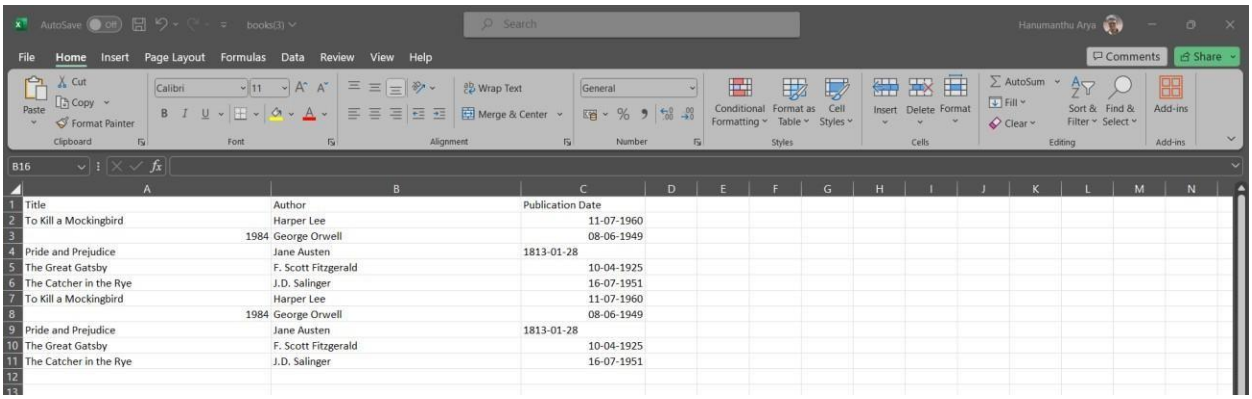
- Open your web browser and visit **http://127.0.0.1:8000/.**




- **Type or copy this http://127.0.0.1:8000/books/export/csv/**

**Final Output By Clicking Register Project and Register Student**


**Fig: CSV File Download Screen**



| | A | B | C |
|---|---|---|---|
| 1 | Title | Author | Publication Date |
| 2 | To Kill a Mockingbird | Harper Lee | 11-07-1960 |
| 3 | 1984 | George Orwell | 08-06-1949 |
| 4 | Pride and Prejudice | Jane Austen | 1813-01-28 |
| 5 | The Great Gatsby | F. Scott Fitzgerald | 10-04-1925 |
| 6 | The Catcher in the Rye | J.D. Salinger | 16-07-1951 |
| 7 | To Kill a Mockingbird | Harper Lee | 11-07-1960 |
| 8 | 1984 | George Orwell | 08-06-1949 |
| 9 | Pride and Prejudice | Jane Austen | 1813-01-28 |
| 10 | The Great Gatsby | F. Scott Fitzgerald | 10-04-1925 |
| 11 | The Catcher in the Rye | J.D. Salinger | 16-07-1951 |

- **Type or copy this [http://127.0.0.1:8000/books/export/pdf/](http://127.0.0.1:8000/books/export/pdf/)**

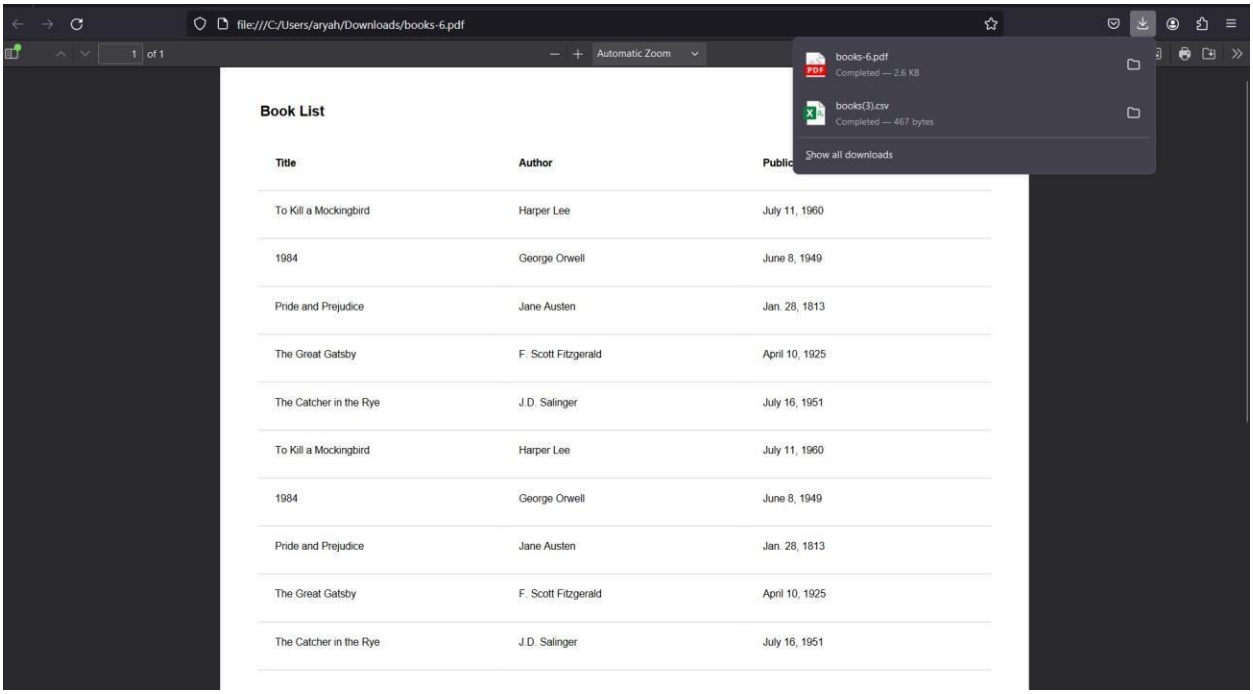
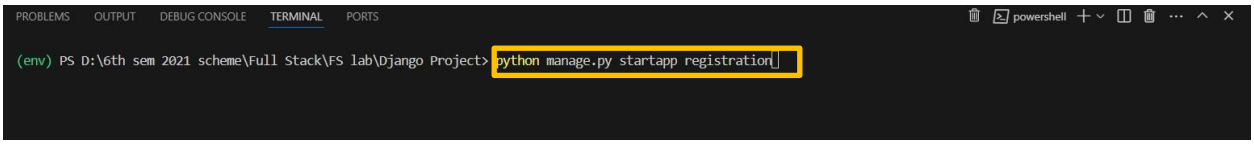



**Fig: PDF File Download Screen**

# Experiment-12

Develop a registration page for student enrolment as done in Module 2 but without page refresh using AJAX.

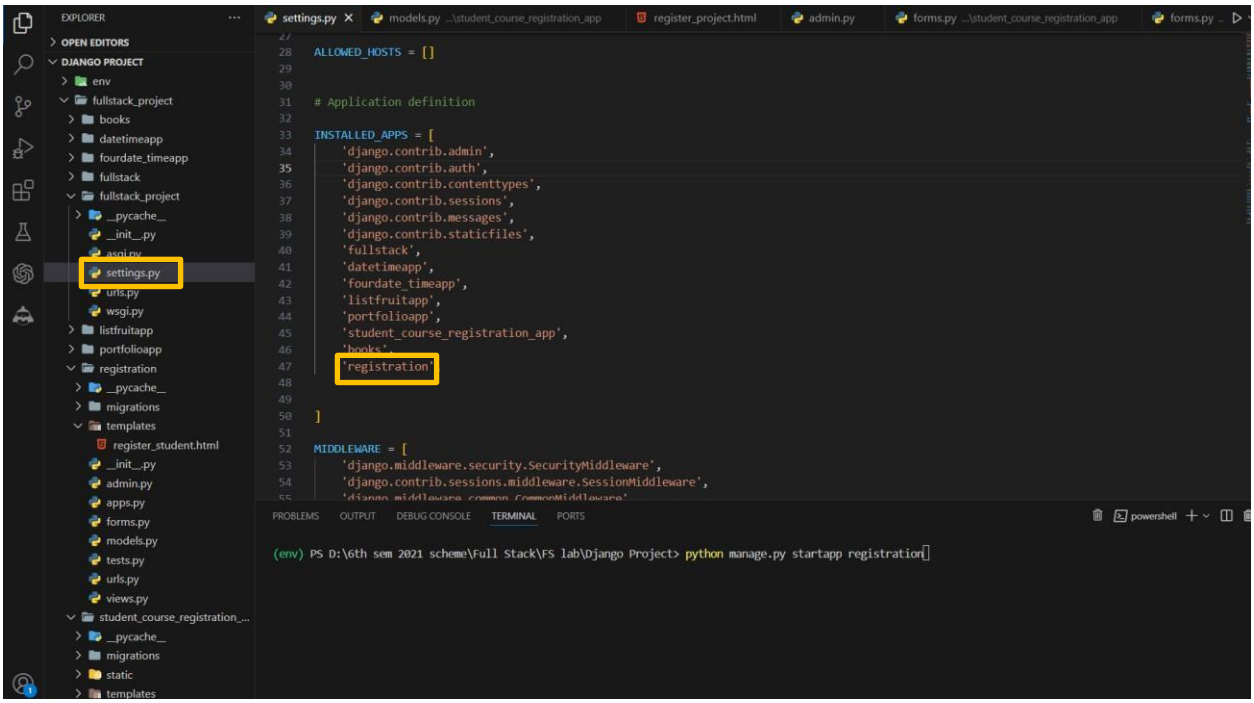
**Step-01**: **This app will be created in the Django project we made earlier.**




**Step-02: Add the app to INSTALLED_APPS**

Open the settings.py file in your project's directory (e.g., **fullstack_project/settings.py**).

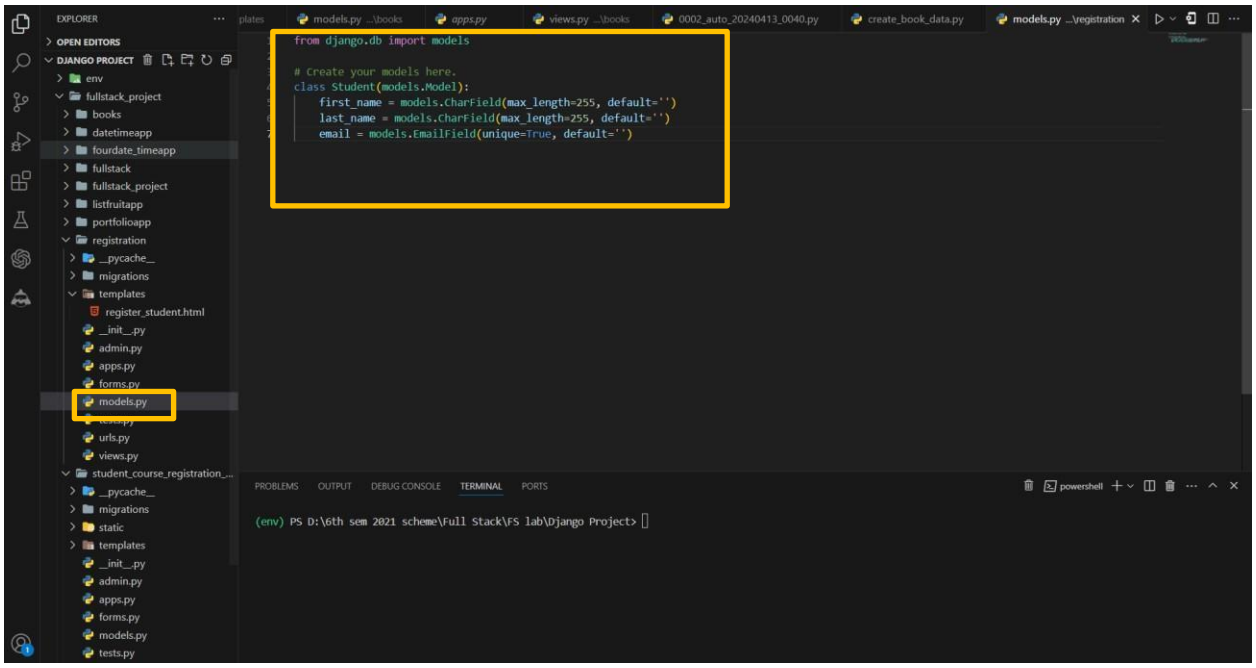
Locate the **INSTALLED_APPS** list and add the name of your new app to the list:



**Step-03: Create models**

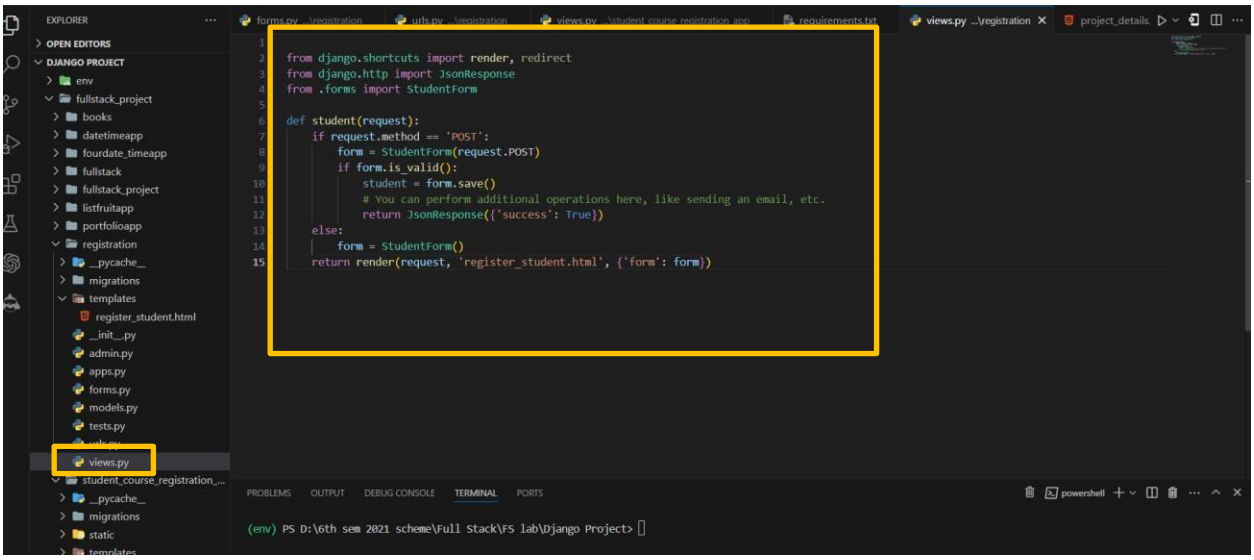
- Open the **models.py** file inside the registration app and define your models:

```
from django.db import models


# Create your models here.
class Student(models.Model):
    first_name = models.CharField(max_length=255, default='')
    last_name = models.CharField(max_length=255, default='')
    email = models.EmailField(unique=True, default='')
```



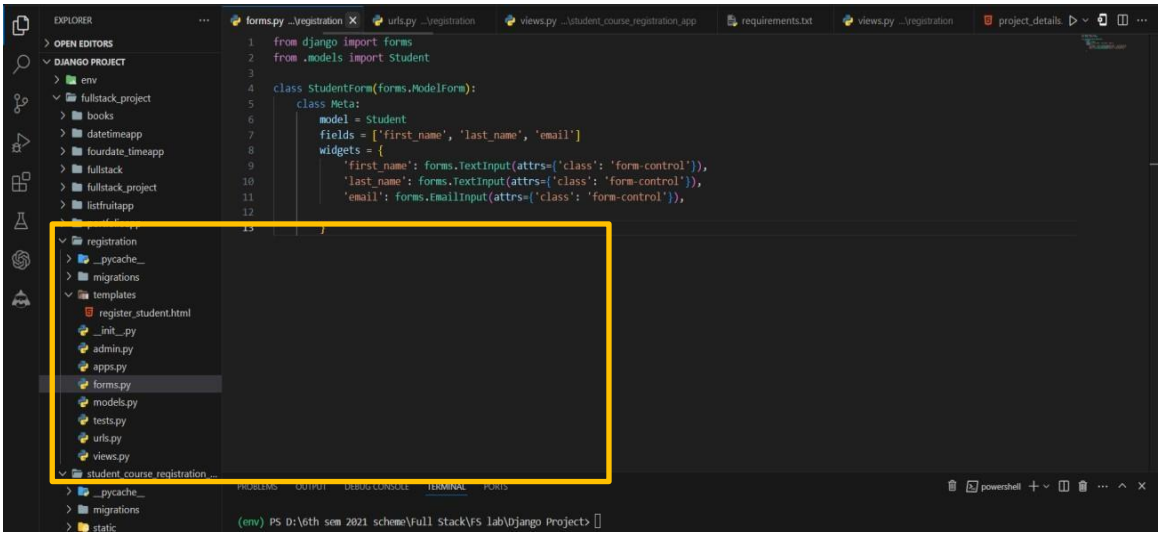
**Step-04: Create a views.py file in your app and define the views**

```
from django.shortcuts import render, redirect
from django.http import JsonResponse
from .forms import StudentForm
def student(request):
    if request.method == 'POST':
        form = StudentForm(request.POST)
        if form.is_valid():
            student = form.save()
            # You can perform additional operations here, like sending an email, etc.
            return JsonResponse({'success': True})
    else:
        form = StudentForm()
    return render(request, 'register_student.html', {'form': form})
```



**Step-05: Create a forms.py file in your app and define the forms**

```
from django import forms
from .models import Student
class StudentForm(forms.ModelForm): class Meta:
    model = Student
    fields = ['first_name', 'last_name', 'email'] widgets = {
        'first_name': forms.TextInput(attrs={'class': 'form-control'}), 'last_name': forms.TextInput(attrs={'class': 'form-control'}), 'email': forms.EmailInput(attrs={'class': 'form-control'}),
    }
```



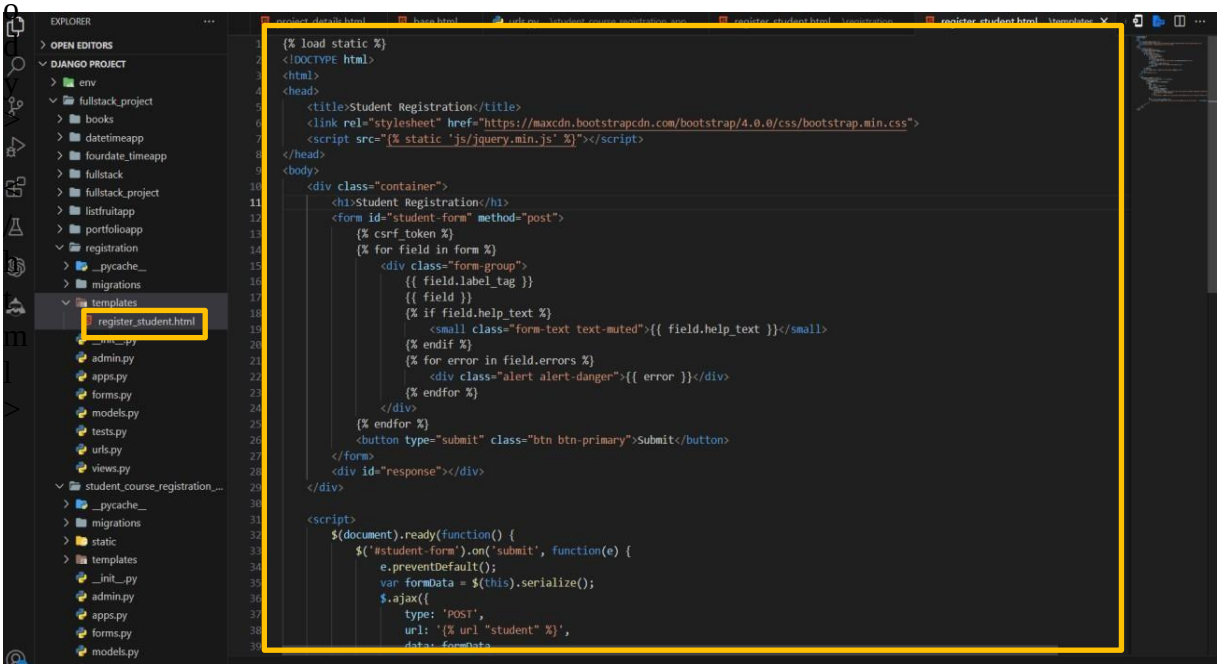
**Step-06: In your app, create a templates directory and an html file**

**register_student.html**

```
{% load static %}
<!DOCTYPE html>
<html>
<head>
    <title>Student Registration</title>
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
    <script src="{% static 'js/jquery.min.js' %}"></script>
</head>
<body>
    <div class="container">
        <h1>Student Registration</h1>
        <form id="student-form" method="post">
            {% csrf_token %}
            {% for field in form %}
                <div class="form-group">
                    {{ field.label_tag }}
                    {{ field }}
                    {% if field.help_text %}
                <small class="form-text text-muted">{{ field.help_text }}</small>
                    {% endif %}
                    {% for error in field.errors %}
                        <div class="alert alert-danger">{{ error }}</div>
```

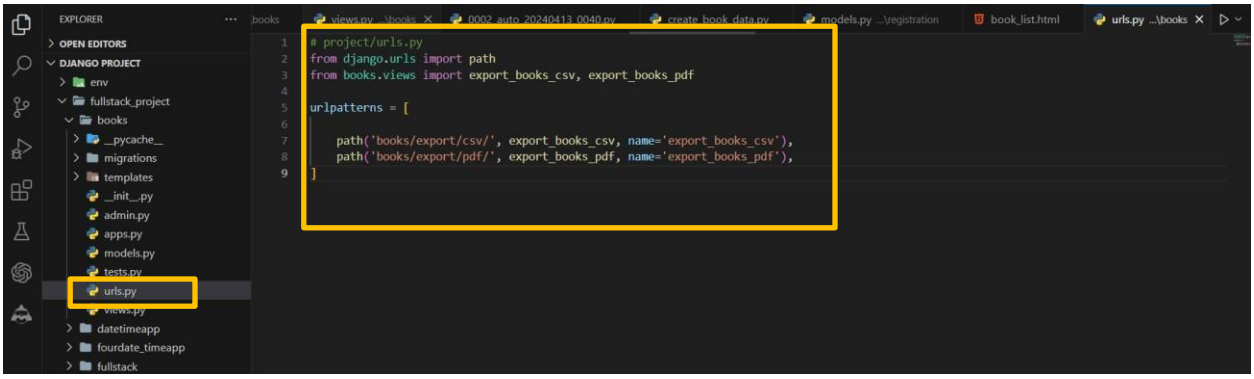
```
            {% endfor %}
          </div>
        {% endfor %}
        <button type="submit" class="btn btn-primary">Submit</button>
      </form>
      <div id="response"></div>
    </div>
    <script>
      $(document).ready(function() {
        $('#student-form').on('submit',
          function(e) { e.preventDefault();
          var formData = $(this).serialize();
          $.ajax({
            type: 'POST',
            url: '{% url "student"
            %}', data: formData,
            success:
              function(response)
              { if
              (response.success)
              {
                $('#response').html('<div class="alert alert-success">Student
                $('#response').html('<div class="alert alert-danger">An error occurred. Please
try again.</div>');
              }
            },
```

```
            error: function(xhr, errmsg, err) {

                $('#response').html('<div class="alert alert-danger">An error occurred: ' +
xhr.status + ' ' + xhr.responseText + '</div>');

            }

        });

    });

}
)
;

    </script>

<
/
b
```



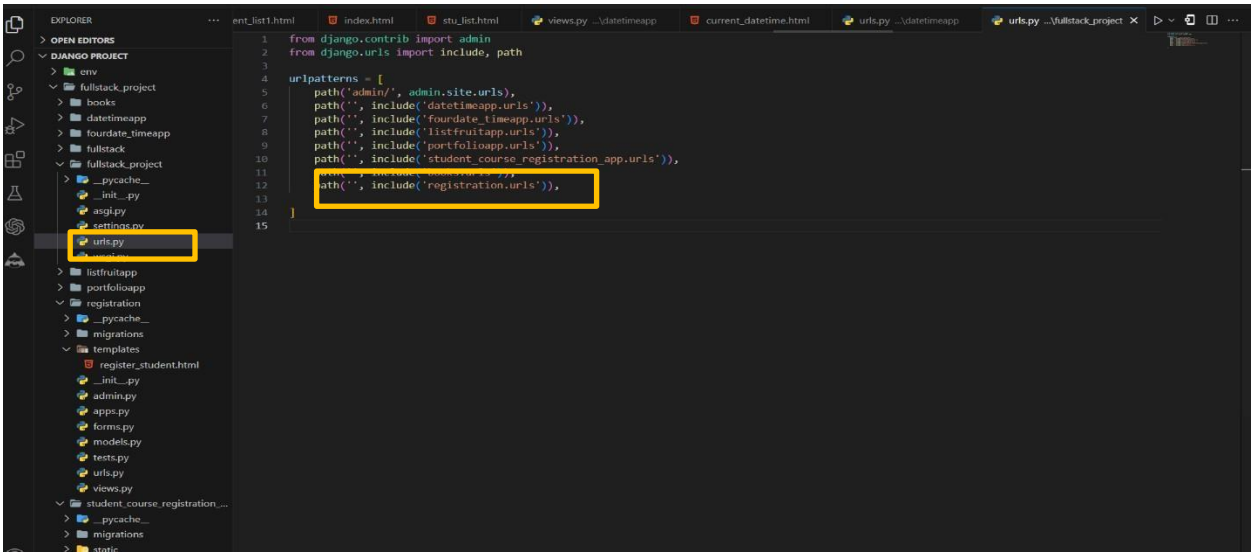
**Step-07: Update app URLs**

```
# project/urls.py
from django.urls import path
from books.views import export_books_csv, export_books_pdf
urlpatterns = [
    path('books/export/csv/', export_books_csv, name='export_books_csv'),
    path('books/export/pdf/', export_books_pdf, name='export_books_pdf'),
]
```




**Step-08: Update project URLs**

- Open the **urls.py** file inside your project and include the URLs from the registration app:

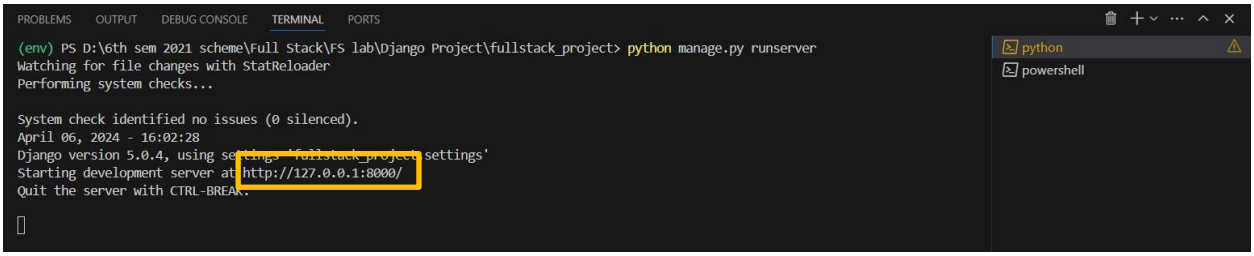


**Step-09: Run the development server**

- In the VS Code terminal, navigate to your Django project's directory (the one containing manage.py).
- Run the development server

**python manage.py runserver**

- Open your web browser and visit **http://127.0.0.1:8000/.**




- **Type or copy this http://127.0.0.1:8000/student/**


**Fig: Student Registration Screen**

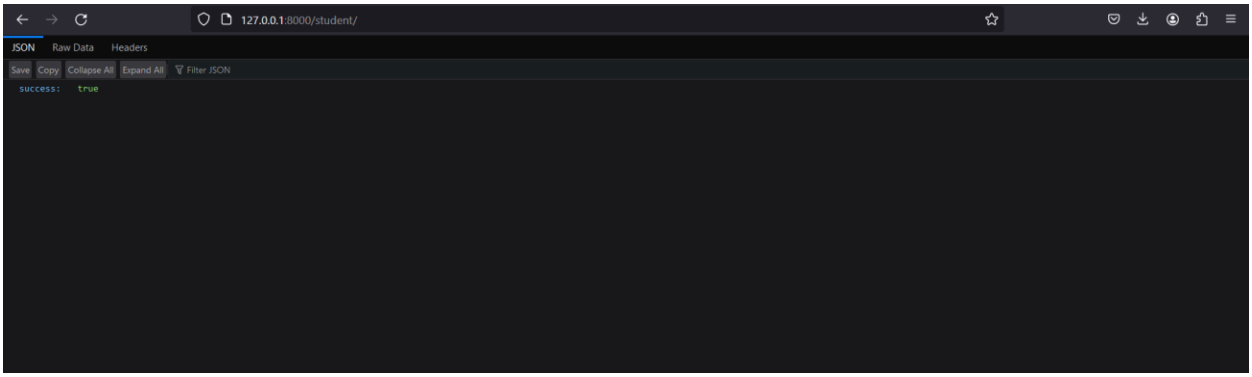



**Fig: Student Registration Success Message Screen**

# Experiment-13

Develop a search application in Django using AJAX that displays courses enrolled by a student being searched.

**Step-01: Use an existing app i.e. student_course_registration_app**

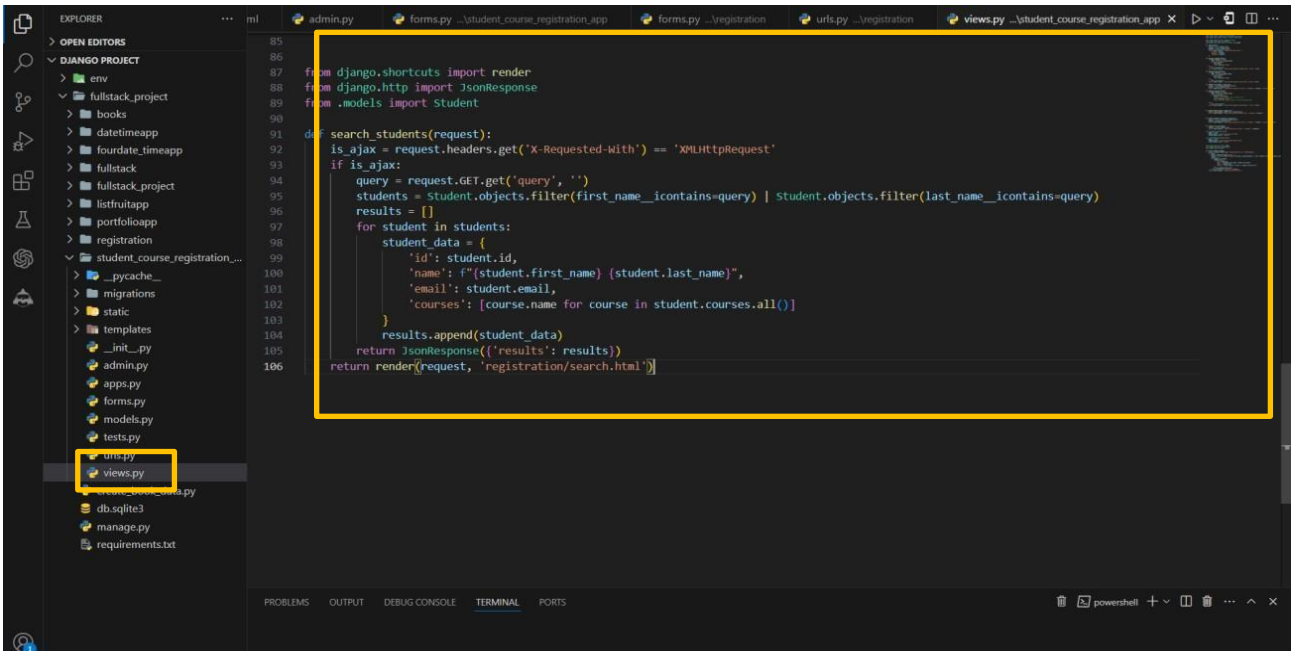
```
[08/Apr/2024 00:21:00] "GET /student-list/1/ HTTP/1.1" 200 1333
[08/Apr/2024 00:21:02] "GET /student-list/2/ HTTP/1.1" 200 1326
PS D:\6th sem 2021 scheme\Full Stack\FS lab\Django Project\fullstack_project> python manage.py makemigrations student_course_registration_app
```

**Step-02: Create View Function in a student_course_registration_app views.py**

```
def search_students(request):
    is_ajax = request.headers.get('X-Requested-With') == 'XMLHttpRequest'
    if is_ajax:
        query = request.GET.get('query', '')
        students = Student.objects.filter(first_name_icontains=query) |
Student.objects.filter(last_name_icontains=query)
        results = []
        for student in students:
            student_data = {
                'id': student.id,
                'name': f"{student.first_name} {student.last_name}",
                'email': student.email,
                'courses': [course.name for course in student.courses.all()]
            }
            results.append(student_data)
```

**return JsonResponse({'results': results})**

**return render(request, 'registration/search.html')**



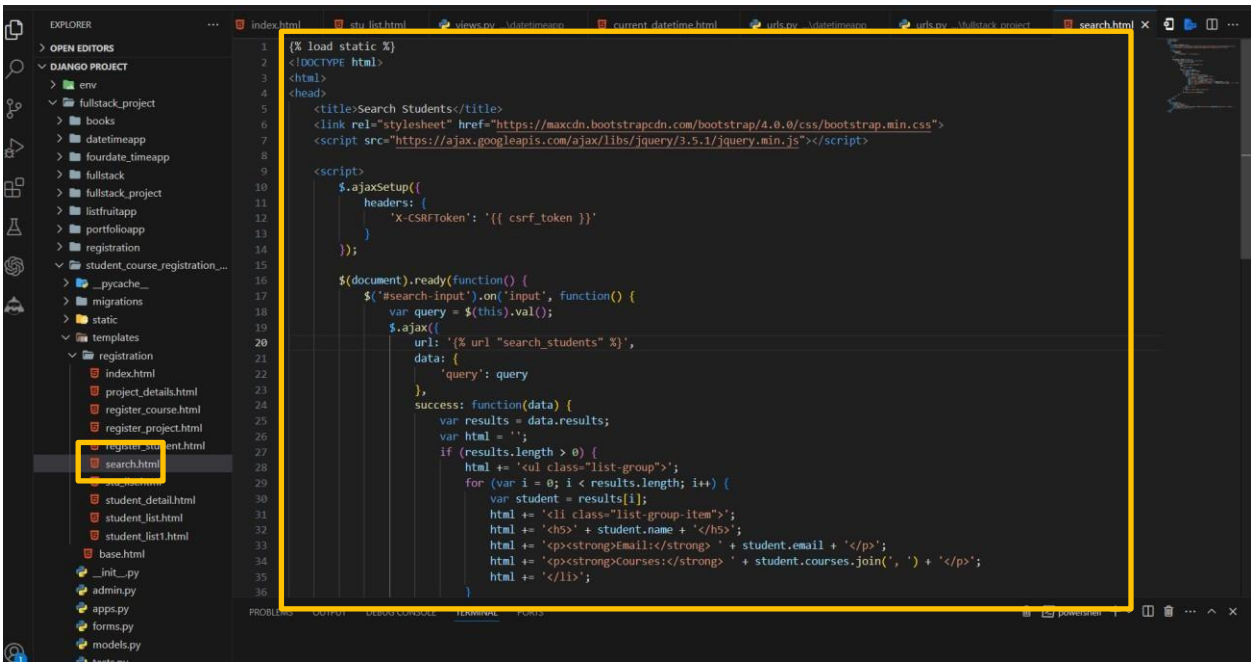

**Step-02: Create Templates in a student_course_registration_app**

**Registration/search.html**

```
{% load static %}
<!DOCTYPE html>
<html>
<head>
    <title>Search Students</title>
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

    <script>
        $.ajaxSetup({
```

```
    headers: {
        'X-CSRFToken': '{{ csrf_token }}'
    }
});

$(document).ready(function() {
    $('#search-input').on('input',
        function() { var query =
        $(this).val();
        $.ajax({
            url: '{% url "search_students"
            %}', data: {
                'query': query
            },
            success: function(data)
                { var results =
                data.results; var
                html = '';
                if (results.length > 0) {
                    html += '<ul class="list-group">';
                    for (var i = 0; i < results.length;
                        i++) { var student = results[i];
                        html += '<p><strong>Email:</strong> ' + student.email + '</p>';
                        html += '<p><strong>Courses:</strong> ' + student.courses.join(', ') + '</p>';
                        html += '</li>';
```

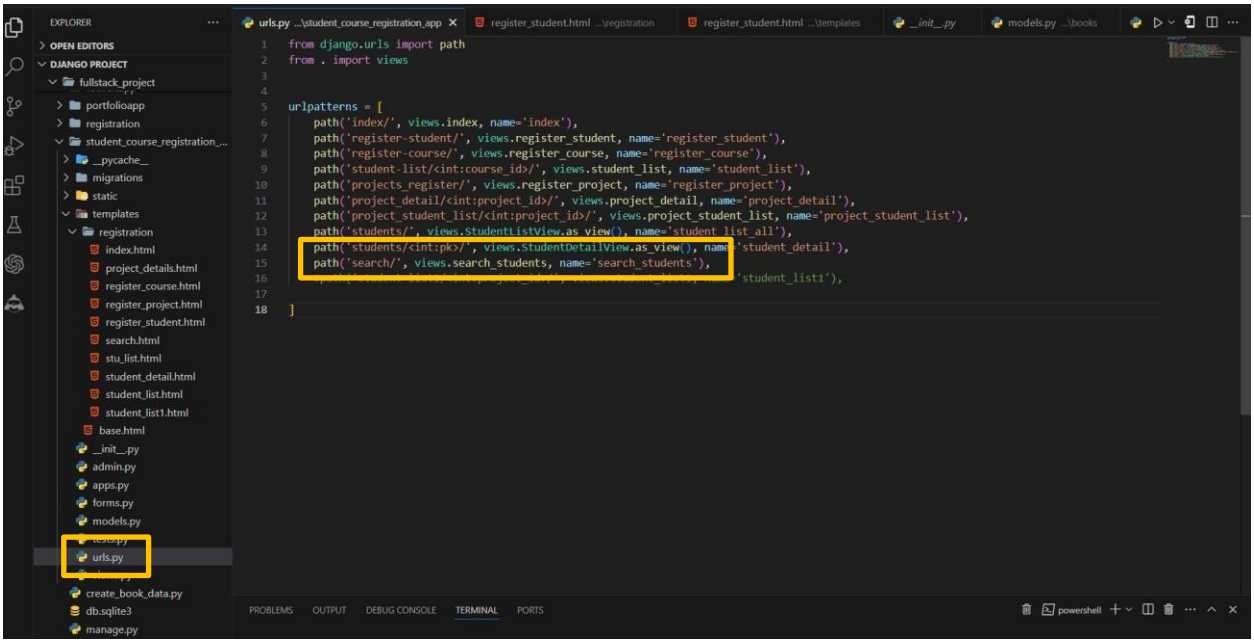
```
                    }
                    html += '</ul>';
                } else {
                    html += '<p>No students found.</p>';
                }
                $('#search-results').html(html);
            }
        });
    });
  </script>
</head>
<body>

  {% csrf_token %}
  <div class="container">
    <h1>Search Students</h1>
    <input type="text" id="search-input" class="form-control" placeholder="Search for a student...">
    <div id="search-results"></div>
  </div>
</body>
</html>
```



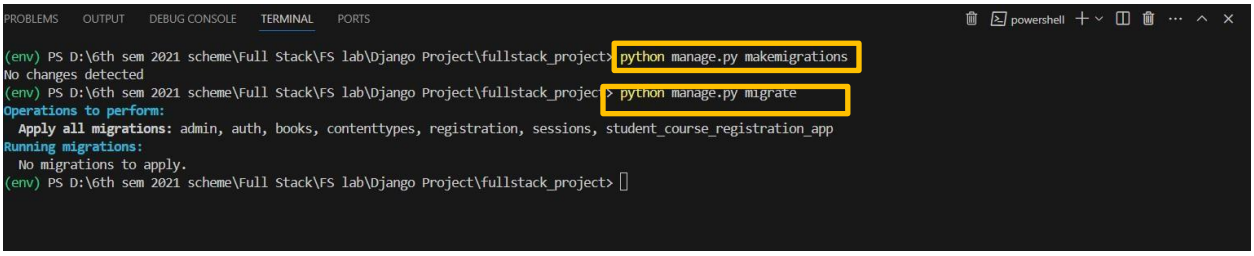

- Very important to add the all The necessary AJAX scripts we provided

### Step-03: Update app URLs

- Open the **urls.py** file inside your app and include the URLs



**Step-04: Makemigrations and migrate if any changes are made in your application eg. Models etc..**




**Step-05: Run the development server**

- In the VS Code terminal, navigate to your Django project's directory (the one containing manage.py).
- Run the development server

**python manage.py runserver**

- Open your web browser and visit **http://127.0.0.1:8000/.**


- **Type or copy this http://127.0.0.1:8000/search/**

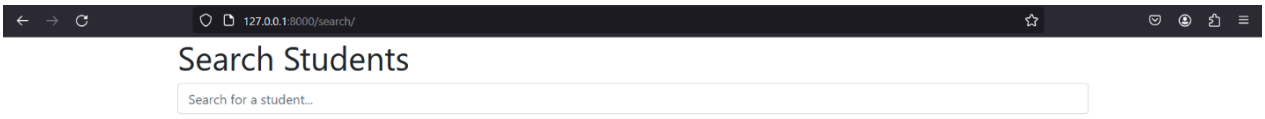



**Fig: Before Search Result contains only Search Bar Screen**

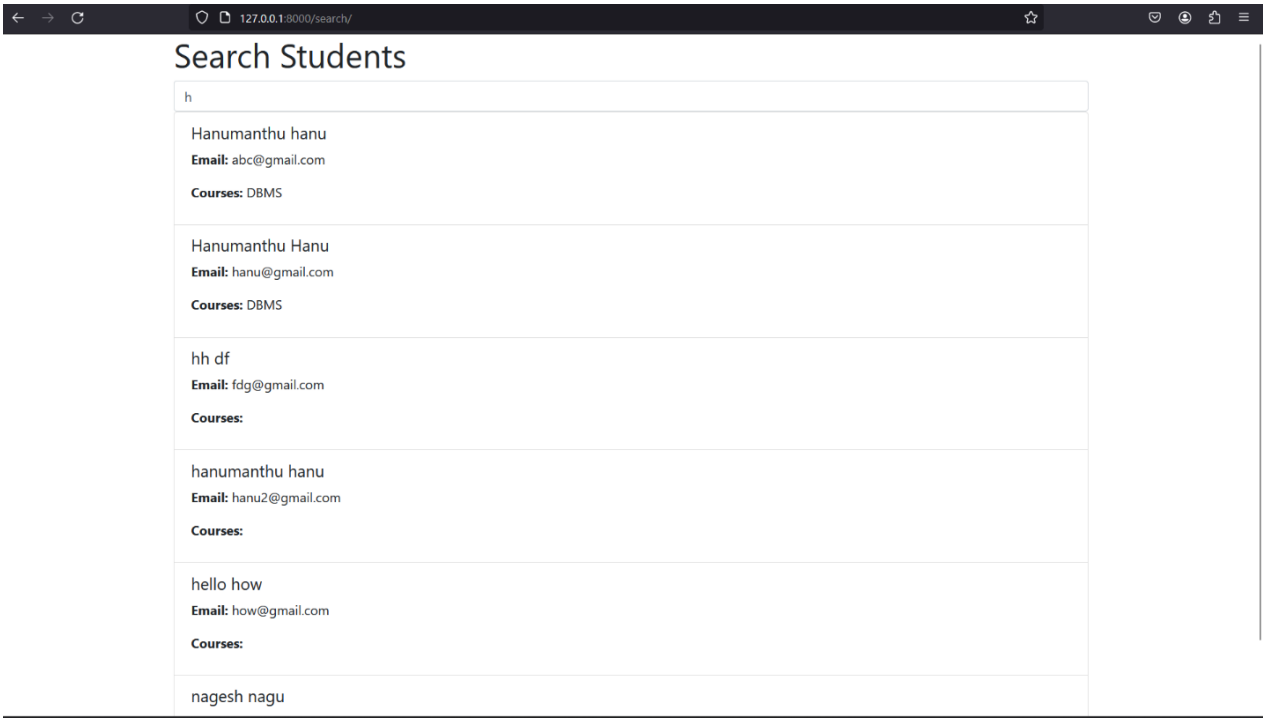



**Fig: Search Screen**