



Search Creators

21CS62

FULL STACK DEVELOPMENT DJANGO LAB MANUAL

Prepared for 6th semester VTU 2021
Scheme Students

Prepared By

HANUMANTHU

Designed By

NAGESH

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.

The screenshot shows the official Python website at https://www.python.org. The page features a dark blue header with the Python logo and the word "python" in white. Below the header is a navigation bar with links for About, Downloads, Documentation, Community, Success Stories, News, and Events. A prominent feature is a code editor window displaying Python code related to list comprehensions and the enumerate function. To the right of the code, there's a section titled "Compound Data Types" with a brief description of lists. At the bottom of the page, there's a footer with links for Get Started, Download, Docs, and Jobs, along with a "Learn More" button.

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

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

The screenshot shows the Python.org homepage. At the top, there are navigation links for Python, PSF, Docs, PyPI, Jobs, and Community. Below the header is the Python logo and a search bar with a magnifying glass icon and a 'GO' button. A blue navigation bar contains links for About, Downloads, Documentation, Community, Success Stories, News, and Events. The main content area features a large illustration of two boxes descending from the sky on yellow and white striped parachutes. A prominent yellow button says "Download the latest version for Windows". Below it, a smaller button says "Download Python 3.12.2". Text below the buttons indicates support for other OSes like Windows, Linux/UNIX, macOS, and others, and mentions Prereleases and Docker images. A section titled "Active Python Releases" provides a table of current versions:

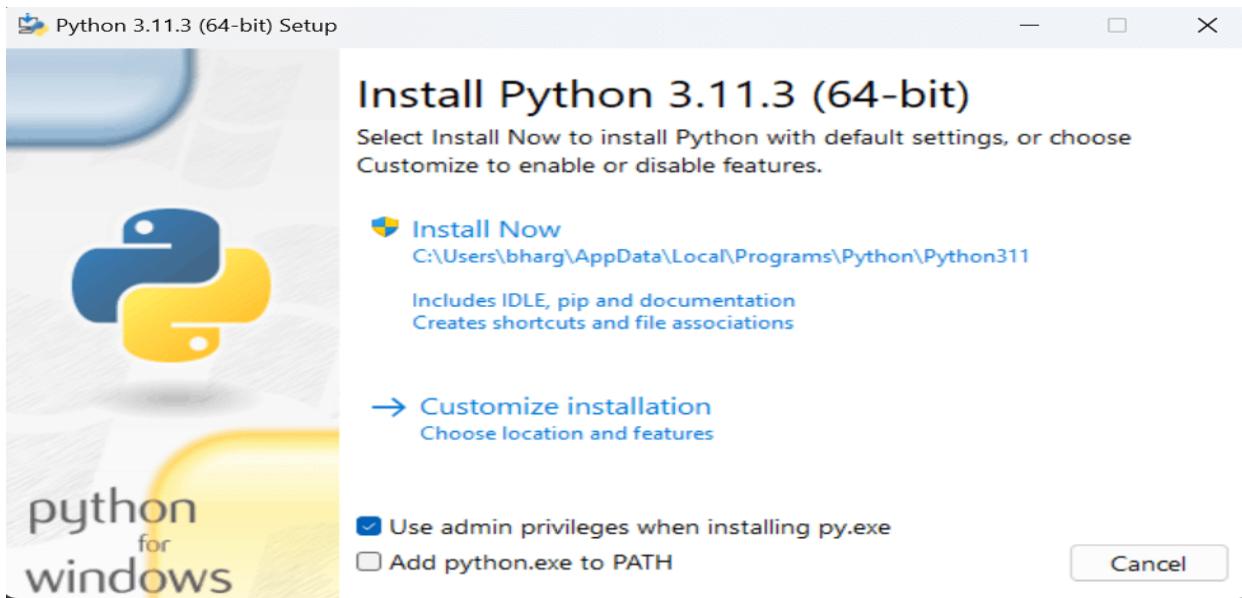
Python version	Maintenance status	First released	End of support	Release schedule
3.13	prerelease	2024-10-01 (planned)	2029-10	PEP 719
3.12	bugfix	2023-10-02	2028-10	PEP 693
3.11	bugfix	2022-10-24	2027-10	PEP 664
3.10	security	2021-10-04	2026-10	PEP 610

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.

The screenshot shows a list of specific Python releases. At the top, a heading says "Looking for a specific release? Python releases by version number:". Below is a table with columns for Release version, Release date, and Click for more (with Download and Release Notes links). The table includes rows for Python 3.11.9 (April 2, 2024), Python 3.10.14 (March 19, 2024), Python 3.9.19 (March 19, 2024), Python 3.8.19 (March 19, 2024), Python 3.11.8 (Feb. 6, 2024), Python 3.12.2 (Feb. 6, 2024), and Python 3.12.1 (Dec. 8, 2023). A link "View older releases" is at the bottom of the table. Below the table is a "Sponsors" section featuring logos for Google, Meta, fastly, and Bloomberg Engineering. At the very bottom, there are links for Licenses, Sources, Alternative, and History.

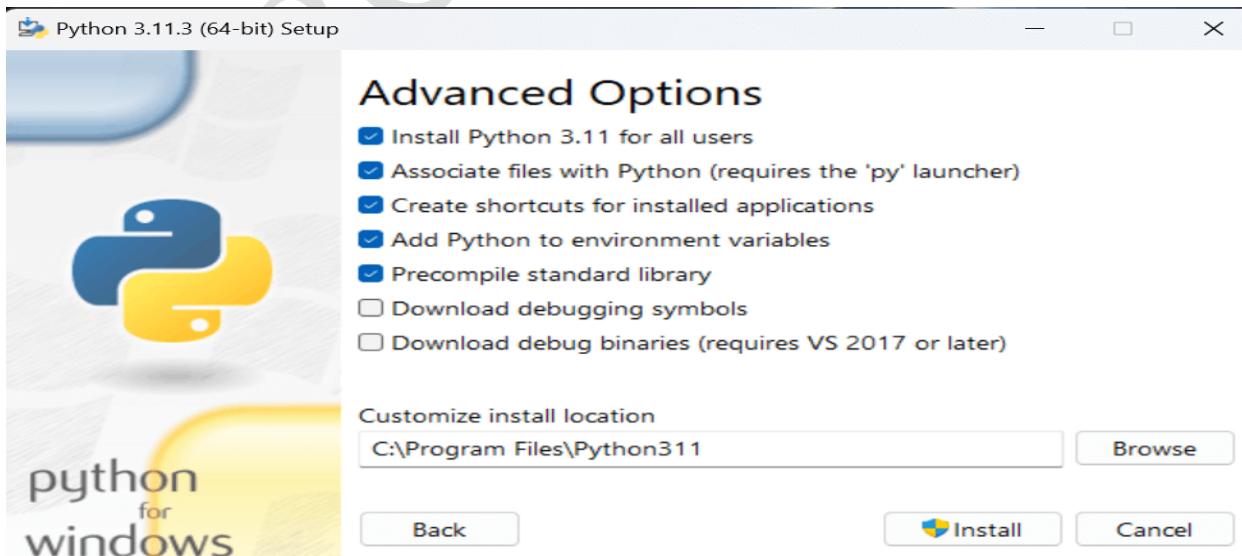
Step - 2: Click on the Install Now

Double-click the executable file, which is downloaded.

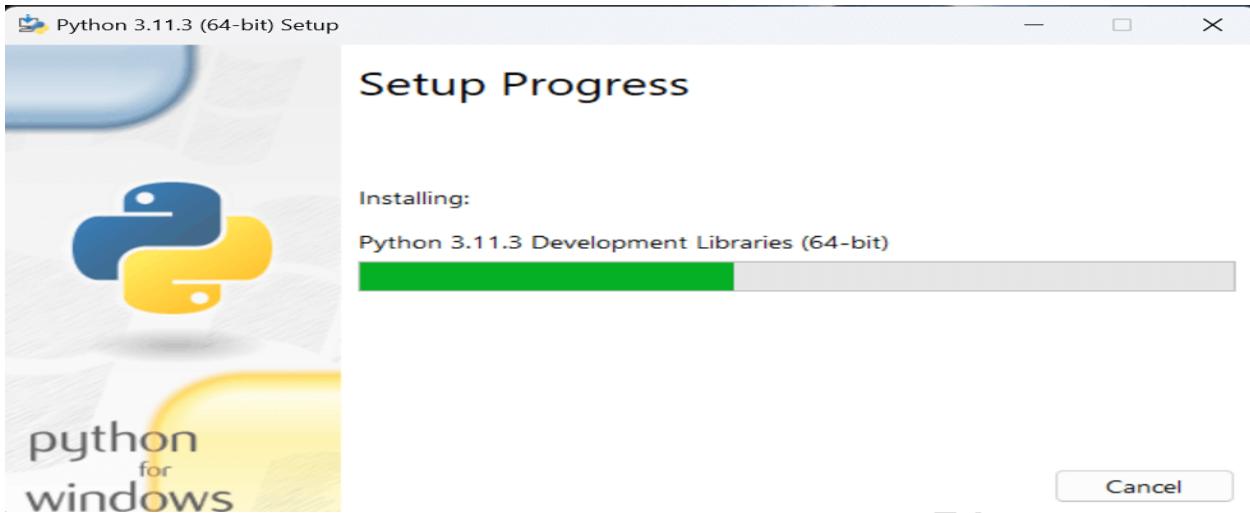


The following window will open. Click on the Add Path check box, it will set the Python path automatically.

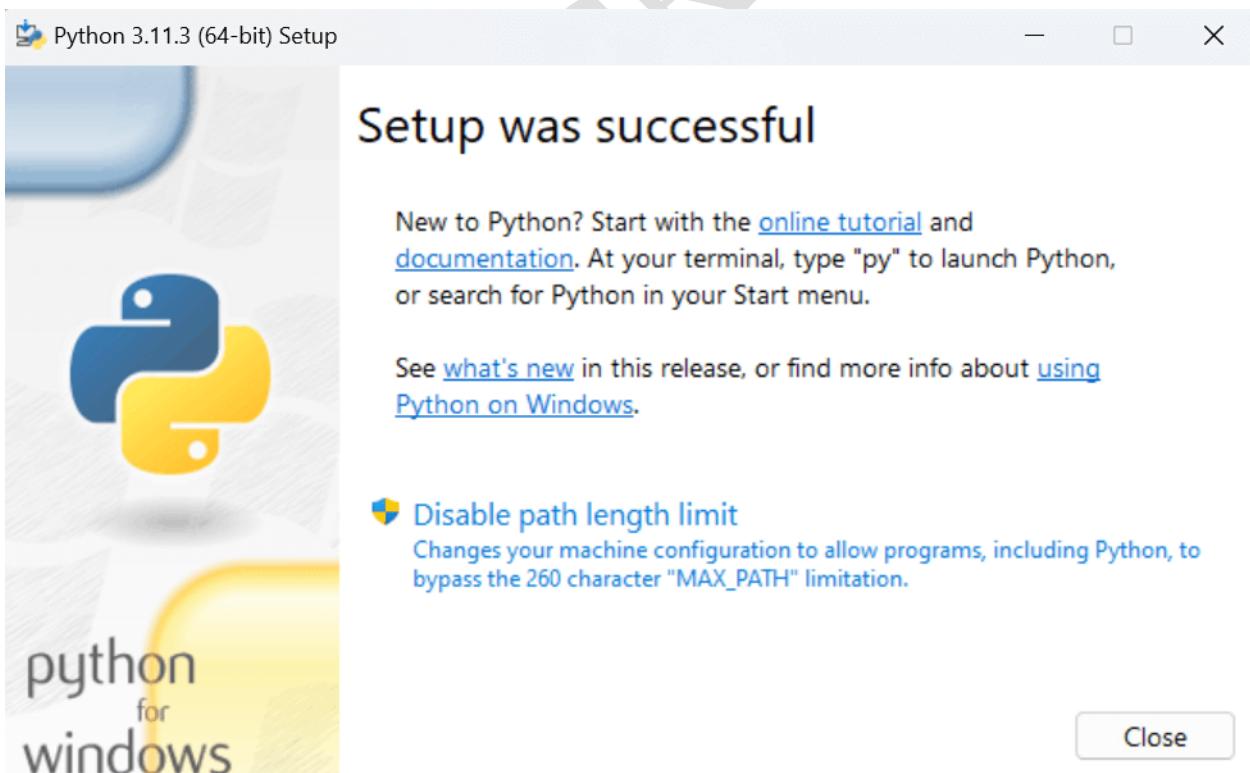
Now, Select Customize installation and proceed. We can also click on the customize installation to choose desired location and features. Other important thing is installing launcher for the all user must be checked.



Step - 3 Installation in Process



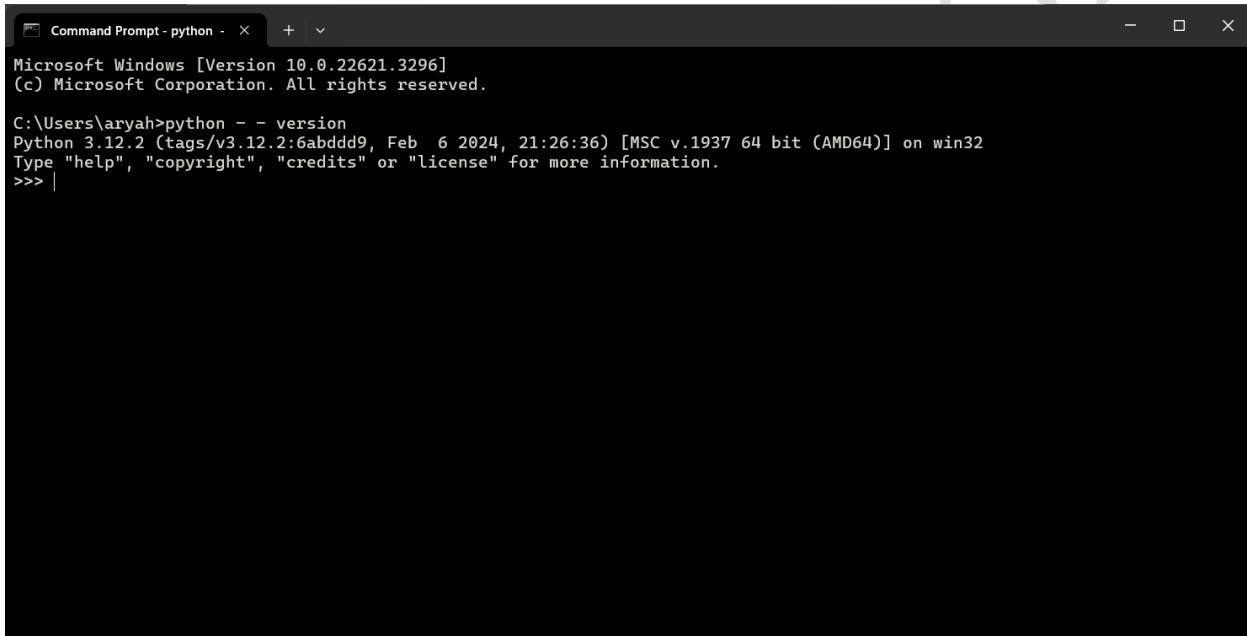
The set up is in progress. All the python libraries, packages, and other python default files will be installed in our system. Once the installation is successful, the following page will appear saying " Setup was successful ".



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. ".



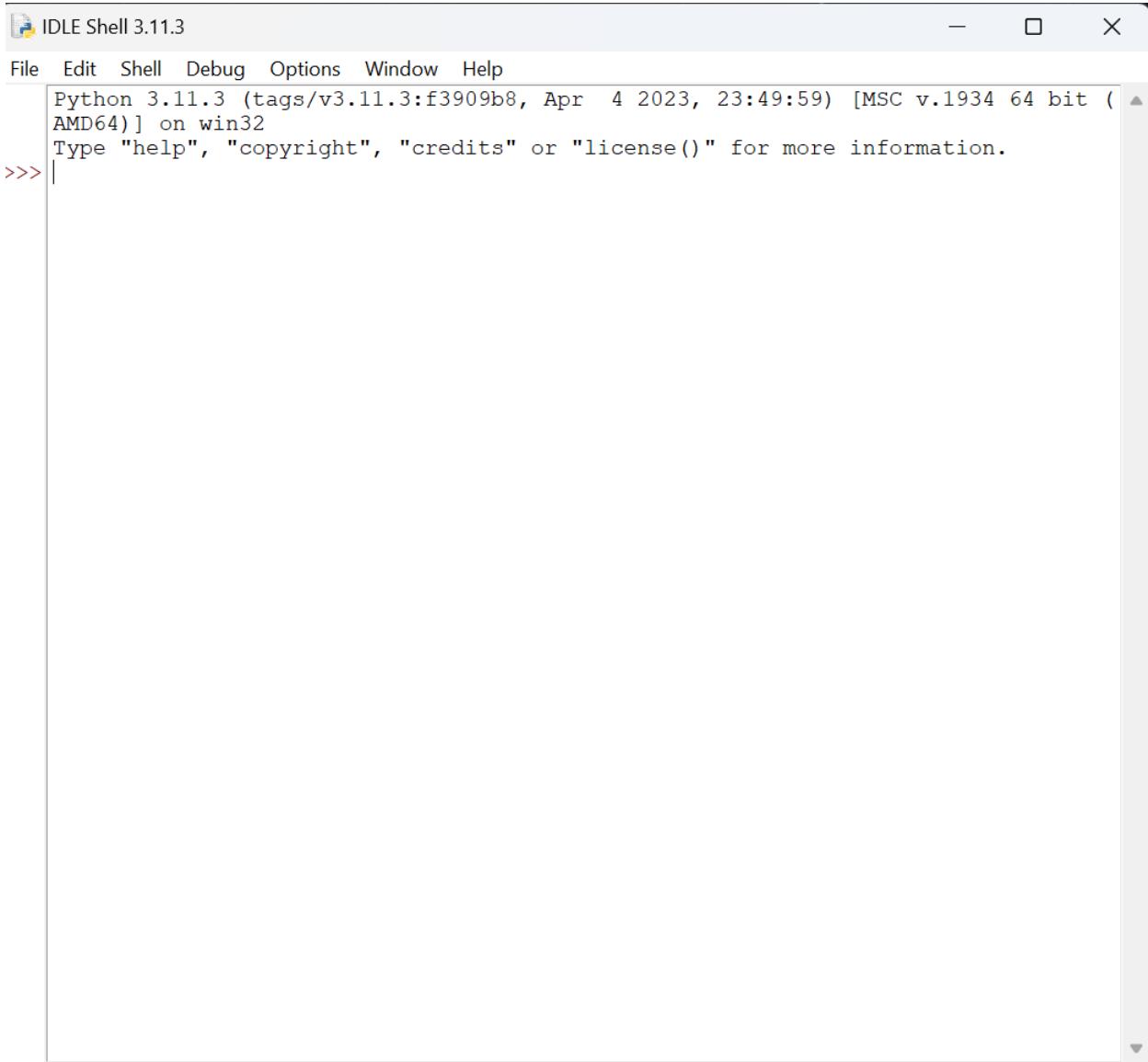
The screenshot shows a Windows Command Prompt window titled "Command Prompt - python -". The window displays the following text:

```
Microsoft Windows [Version 10.0.22621.3296]
(c) Microsoft Corporation. All rights reserved.

C:\Users\aryah>python --version
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> |
```

We are ready to work with the Python.

Step - 5: Opening idle

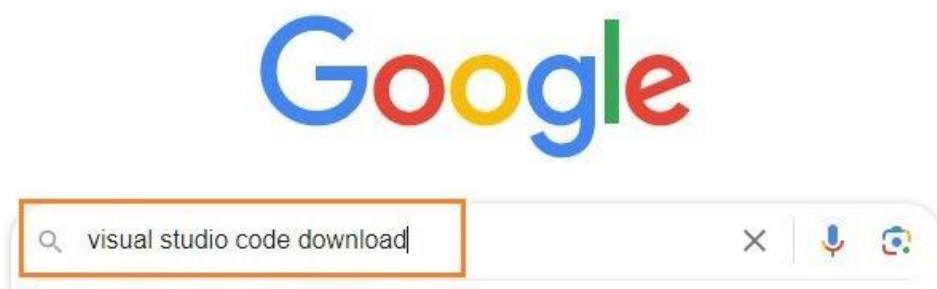


The screenshot shows the IDLE Shell 3.11.3 window. The title bar reads "IDLE Shell 3.11.3". The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The main window displays the Python 3.11.3 interactive interpreter. It shows the version information: "Python 3.11.3 (tags/v3.11.3:f3909b8, Apr 4 2023, 23:49:59) [MSC v.1934 64 bit (AMD64)] on win32". Below this, it says "Type 'help', 'copyright', 'credits' or 'license()' for more information." A red cursor is visible at the start of a new line, indicating where input can be typed.

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

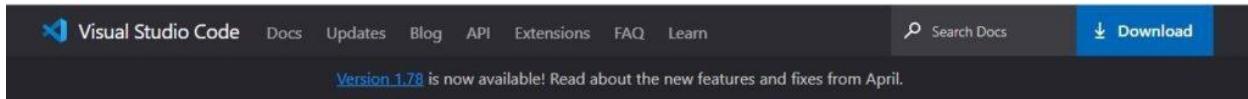
Step – 1: Open Google and type Visual Studio Code download in the search bar.



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

A screenshot of a Google search results page. The search bar at the top contains the query "visual studio code download". The results page shows several links, with the first one from Microsoft highlighted by a red box. The highlighted link is titled "Download Visual Studio Code - Mac, Linux, Windows" and includes a snippet of text: "Visual Studio Code is free and available on your favorite platform - Linux, macOS, and Windows. Download Visual Studio Code to experience a redefined code ...".

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



Download Visual Studio Code

Free and built on open source. Integrated Git, debugging and extensions.

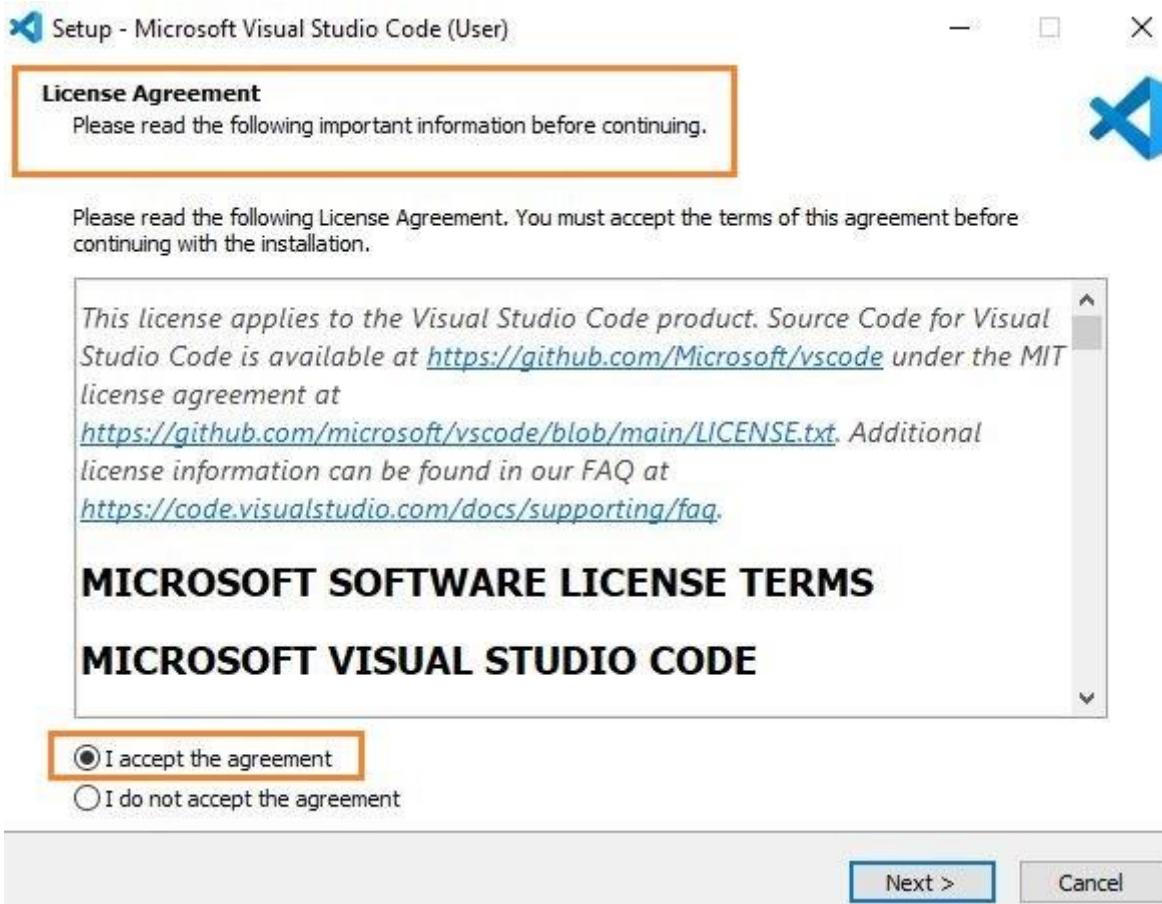


Step – 4: The file will be downloaded onto your system. Open the file and then click on Install.

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

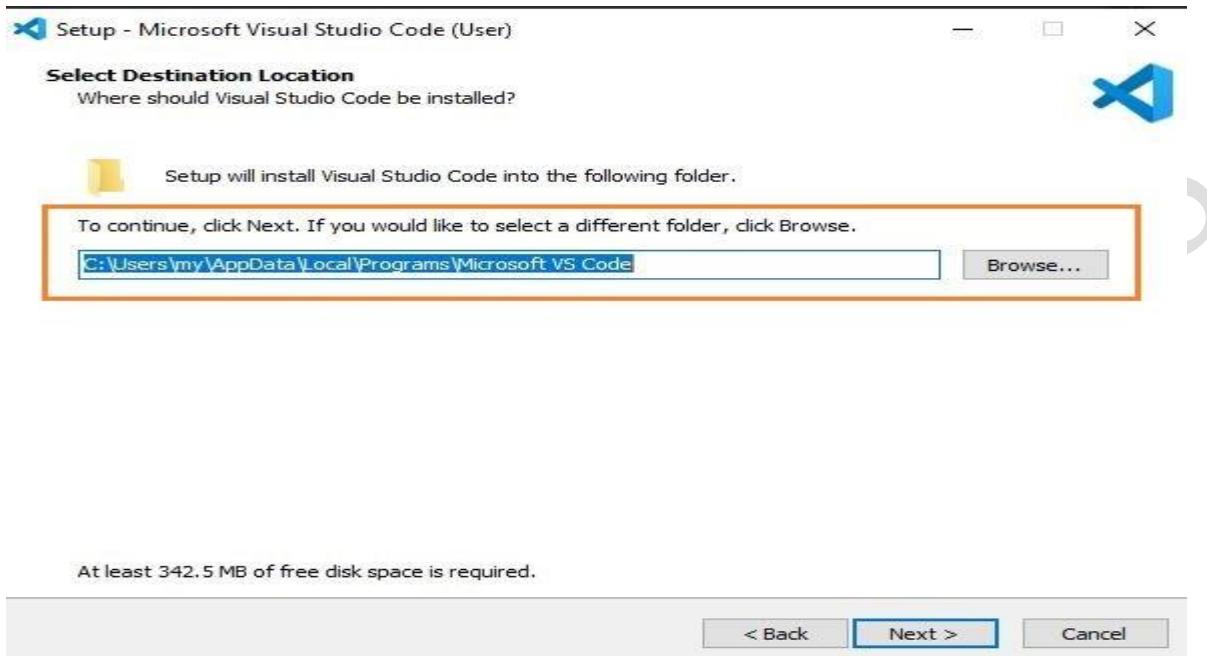
A screenshot of the Visual Studio Code website after download. The navigation bar and banner are the same as the previous screenshot. A central box contains the text "Thanks for downloading VS Code for Windows!". Below it, there's a link to a direct download and a survey invitation. To the left is a sidebar with links like Overview, SETUP, GET STARTED, USER GUIDE, SOURCE CONTROL, TERMINAL, LANGUAGES, NODEJS / JAVASCRIPT, TYPESCRIPT, PYTHON, and JAVA. To the right is a "GETTING STARTED" sidebar with links for VS Code in Action, Top Extensions, First Steps, Keyboard Shortcuts, Downloads, Privacy, and social media sharing options (Twitter, RSS, Ask questions, Follow @code, Request features, Report issues, Watch videos).

Step – 5: Now accept the license agreement.

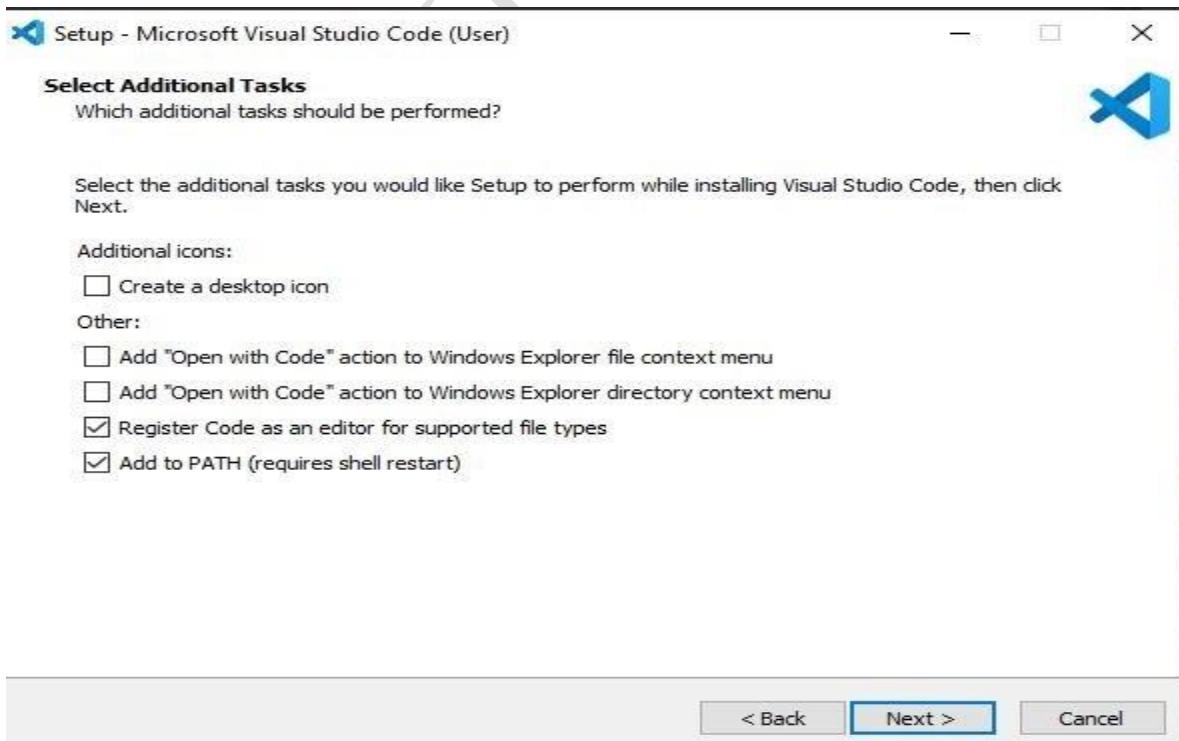


Step – 6: Then it prompts for the file location, where you want to save the VS Code file.

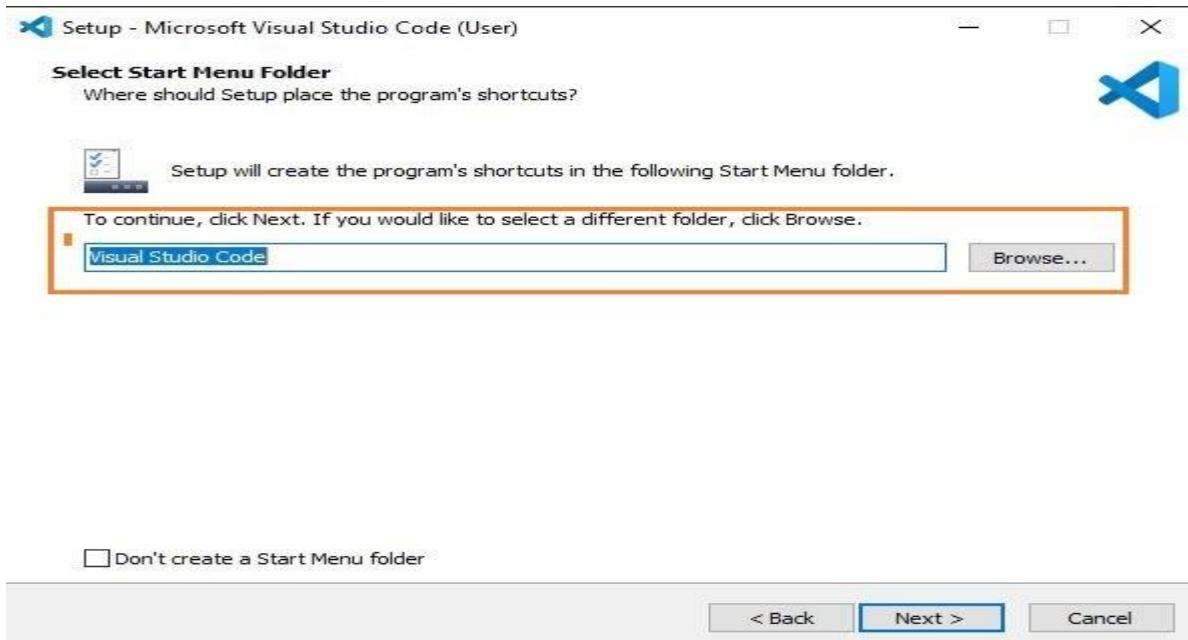
Browse the location and then click on Next.



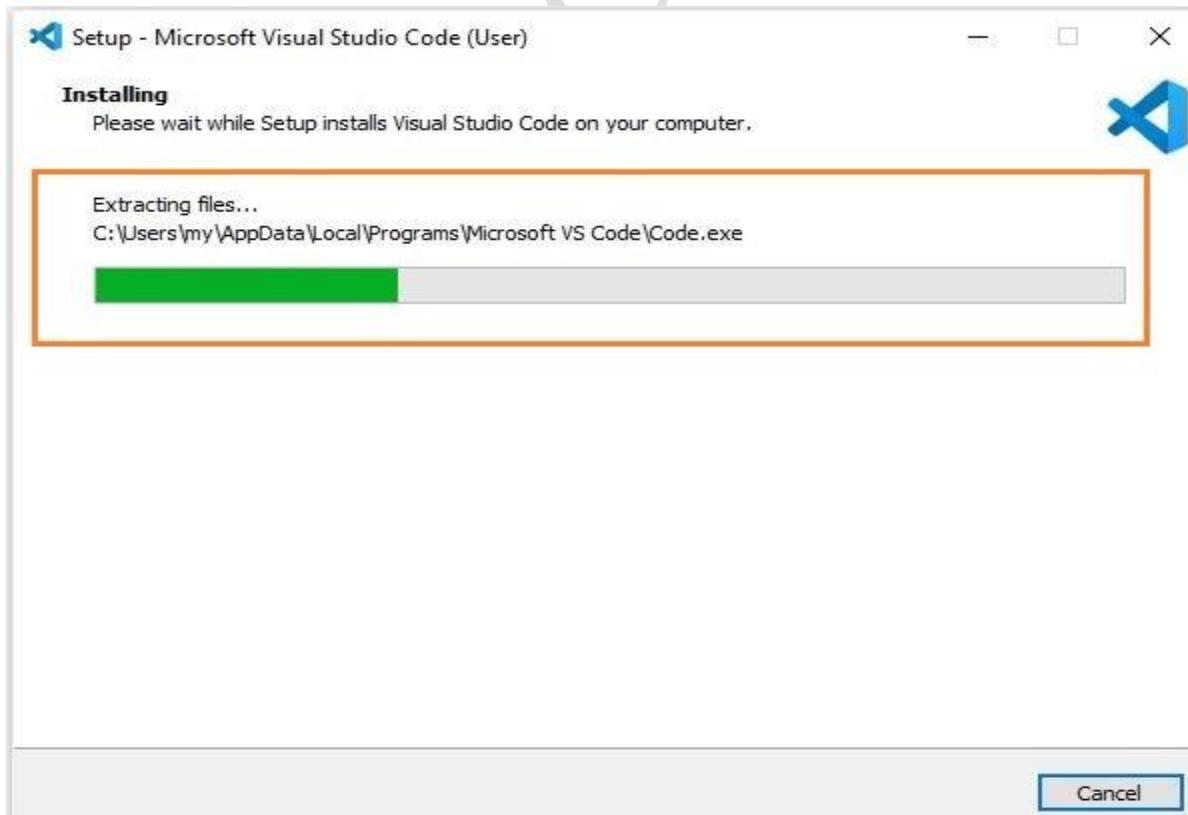
Step – 7: Next, you see the prompt for the additional task which we want the VS Code to perform. At this step, choose the default settings and then click on next.



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



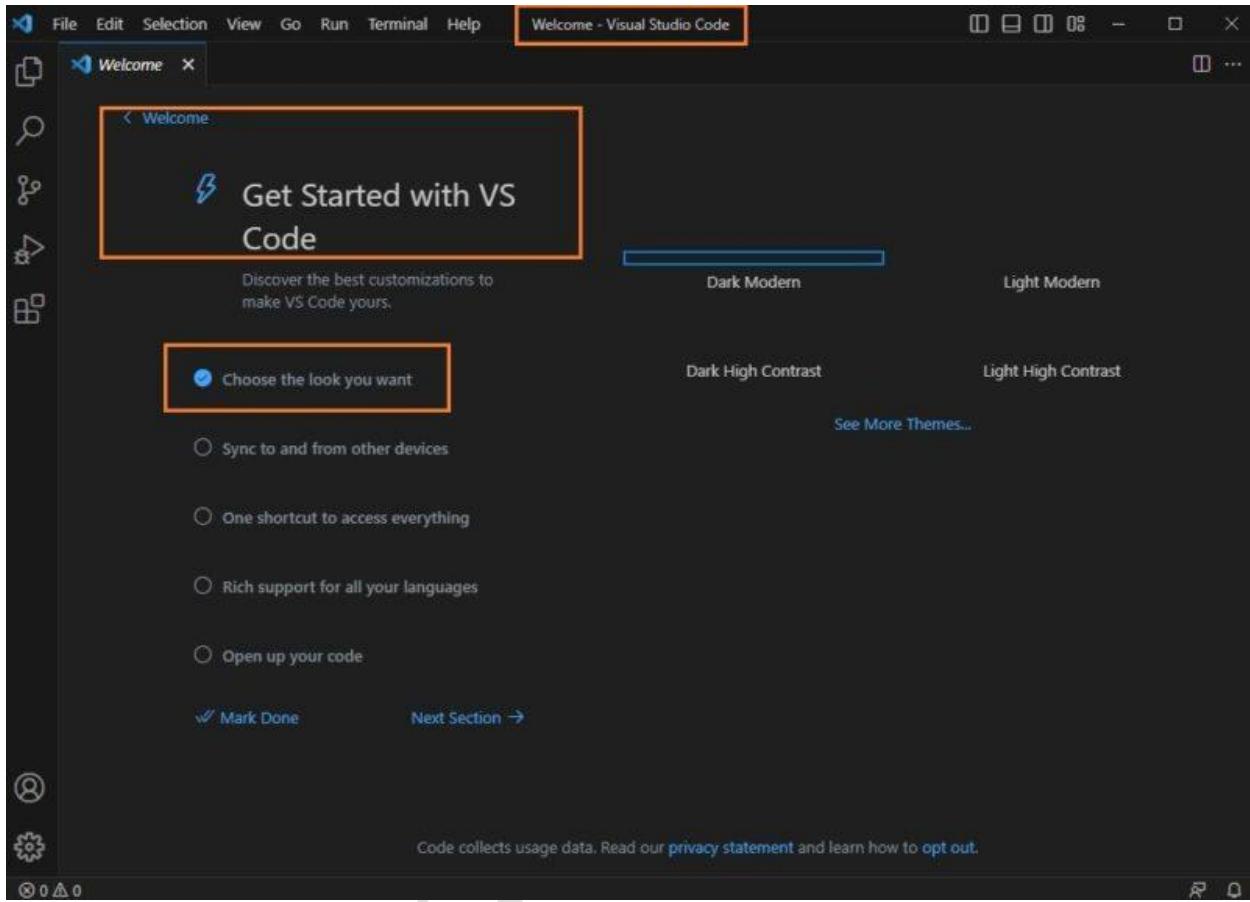
Step – 9: The installation of VS Code will now begin.



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.

The screenshot shows the official Django website at <https://www.djangoproject.com>. The header features the Django logo and navigation links for Overview, Download, Documentation, News, Community, Code, Issues, About, and Donate. A prominent green button labeled "Get started with Django" is centered on the page. Below it, a section titled "Meet Django" describes the framework as a high-level Python web framework for rapid development. To the right, there's a "Support Django!" section with a "Download latest release: 5.0.4" button and a "Happy Herbivore Inc" donation message. The footer includes social media links and a "All Bookmarks" link.

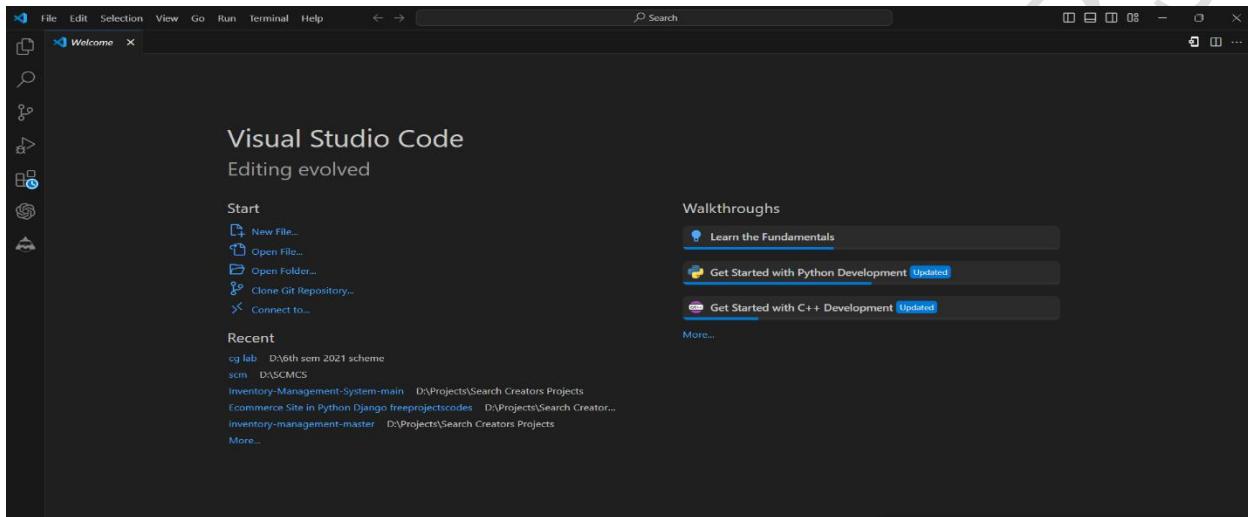
The screenshot shows the "Download" section of the Django website. It starts with a "How to get Django" section explaining the open-source nature and supported Python versions. Below this, two options for getting the latest version are listed: "Option 1: Get the latest official version" and "Option 2: Get the latest development version". Under Option 1, instructions for Linux/macOS and Windows are provided with command-line examples. The "Diamond and Platinum Members" section lists DEFNA and JetBrains as sponsors. A "Support Django!" section features a "StyleSeat" donation message. The footer includes a "For the impatient:" link.

Experiment-02

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

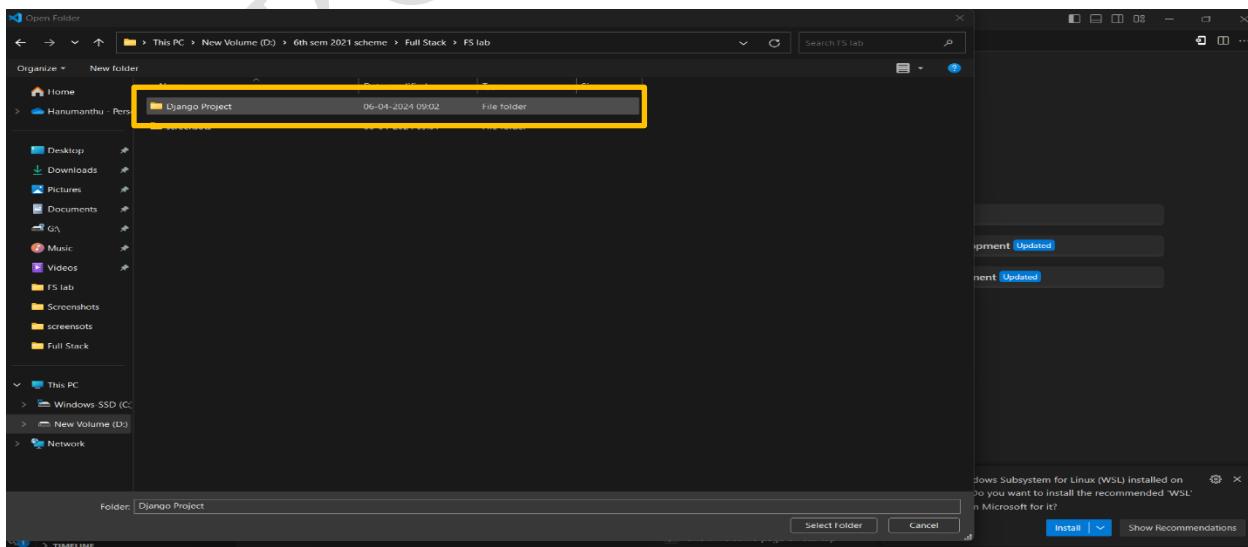
Here are the steps you can follow to create virtual environment: Steps to Create Virtual Environment, Django Project and App

Step-01: Open Visual Studio Code



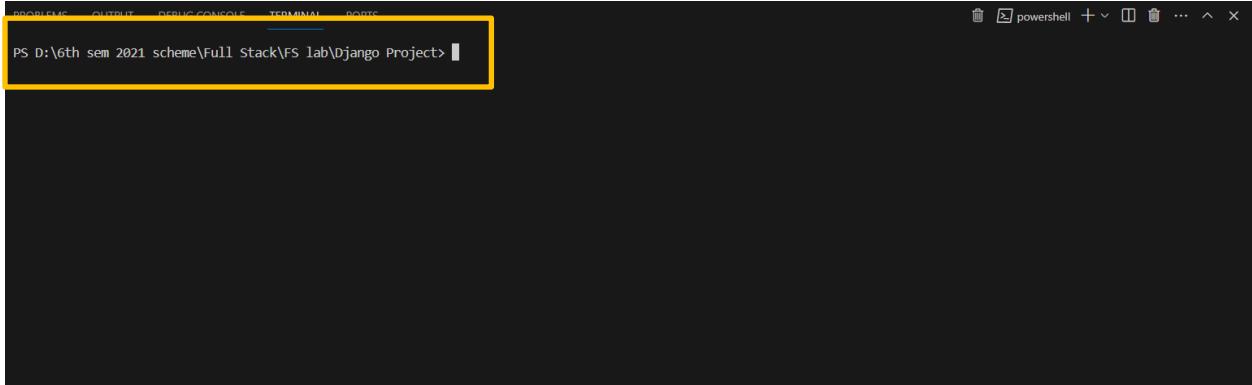
Step-02: Create a new folder for your project

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

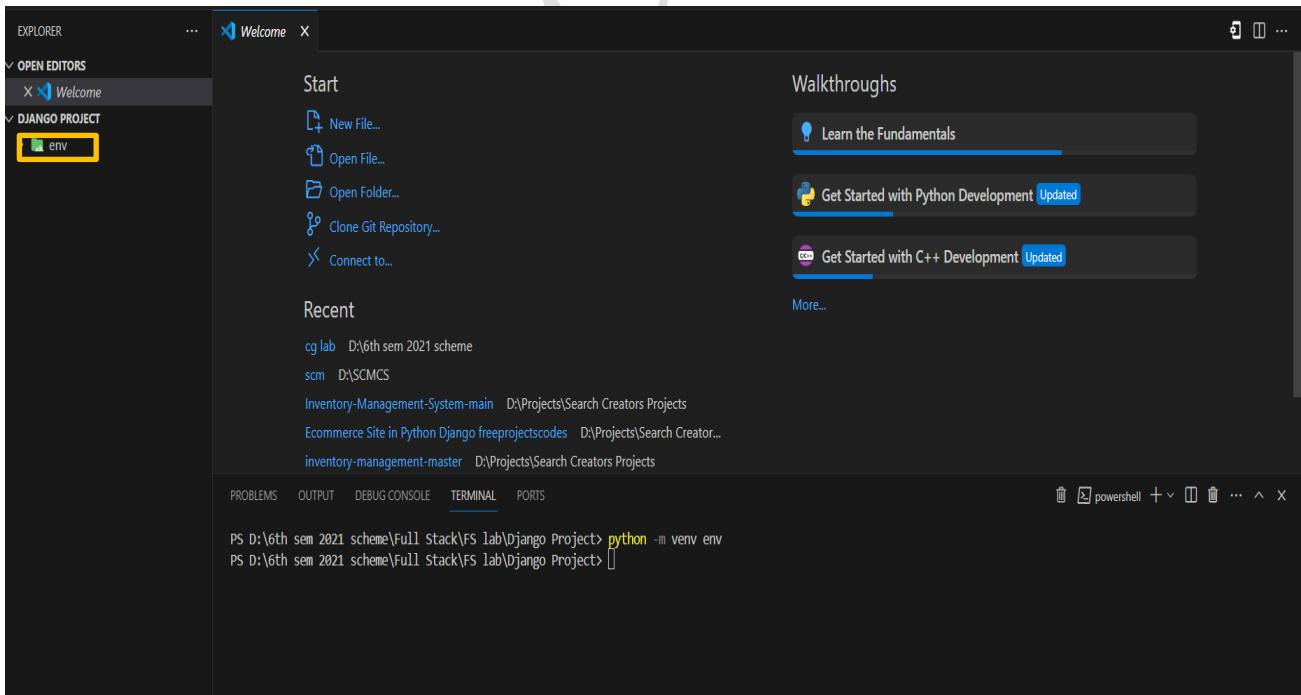
In VS Code, go to **View > Terminal** or use the keyboard shortcut **Ctrl+``** (Windows/Linux) or **Cmd+``** (macOS) to open the integrated terminal.



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

A screenshot of a terminal window in a dark-themed code editor. The terminal tab is selected at the top. The command `python -m venv env` is entered and executed. The next line shows the command to activate the environment: `env\Scripts\activate`. This command is highlighted with a yellow rectangular box. The final line shows the prompt `(env) PS D:\6th sem 2021 scheme\Full Stack\FS lab\ Django Project>`.

Step-05: Install Django

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

A screenshot of a terminal window in the Visual Studio Code interface. The terminal tab is selected at the bottom. The command `pip install django` is entered and executed. The output shows the download and installation of Django and its dependencies. The command `pip install django` is highlighted with a yellow rectangular box.

```
PS D:\6th sem 2021 scheme\Full Stack\FS lab\ Django Project> python -m venv env
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> pip install django
Collecting django
  Downloading Django-5.0.4-py3-none-any.whl.metadata (4.1 kB)
  Collecting asgiref<4,>=3.7.0 (from django)
    Downloading asgiref-3.8.1-py3-none-any.whl.metadata (9.3 kB)
  Collecting sqlparse>0.3.1 (from django)
    Using cached sqlparse-0.4.4-py3-none-any.whl.metadata (4.0 kB)
  Collecting tzdata (from django)
    Using cached tzdata-2024.1-py2.py3-none-any.whl.metadata (1.4 kB)
  Downloading Django-5.0.4-py3-none-any.whl (8.2 MB)
    8.2/8.2 MB 4.4 MB/s eta 0:00:00
  Downloading asgiref-3.8.1-py3-none-any.whl (23 kB)
  Using cached sqlparse-0.4.4-py3-none-any.whl (4 kB)
  Using cached tzdata-2024.1-py2.py3-none-any.whl (345 kB)
  Installing collected packages: tzdata, sqlparse, asgiref, django
  Successfully installed asgiref-3.8.1 django-5.0.4 sqlparse-0.4.4 tzdata-2024.1
(env) PS D:\6th sem 2021 scheme\Full Stack\FS lab\ Django Project>
```

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]

The screenshot shows the VS Code interface. In the Explorer sidebar, a folder named 'fullstack_project' is highlighted with a yellow box. The terminal tab is active, displaying the command 'django-admin startproject fullstack_project'. The output of the command shows the process of installing Django and other dependencies, with a yellow box highlighting the final message: 'Successfully installed asgiref-3.8.1 django-5.0.4 sqlparse-0.4.1 tzdata-2024.1'.

Step-07: Create a new Django app

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

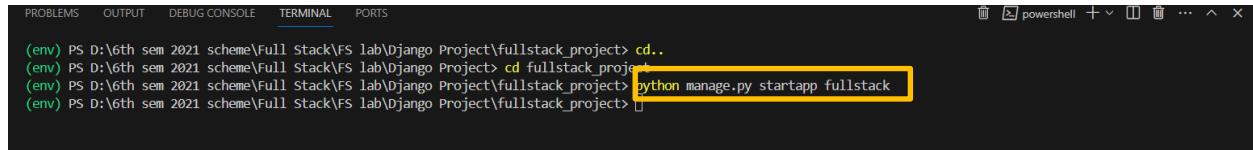
cd myproject [change Project name as You Desired]

The screenshot shows the VS Code terminal. The user has navigated to the project directory by running the command 'cd myproject'. The terminal shows the current path as 'D:\6th sem 2021 scheme\Full Stack\FS lab\Django Project\myproject'. A yellow box highlights the command 'cd fullstack_project'.

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.]

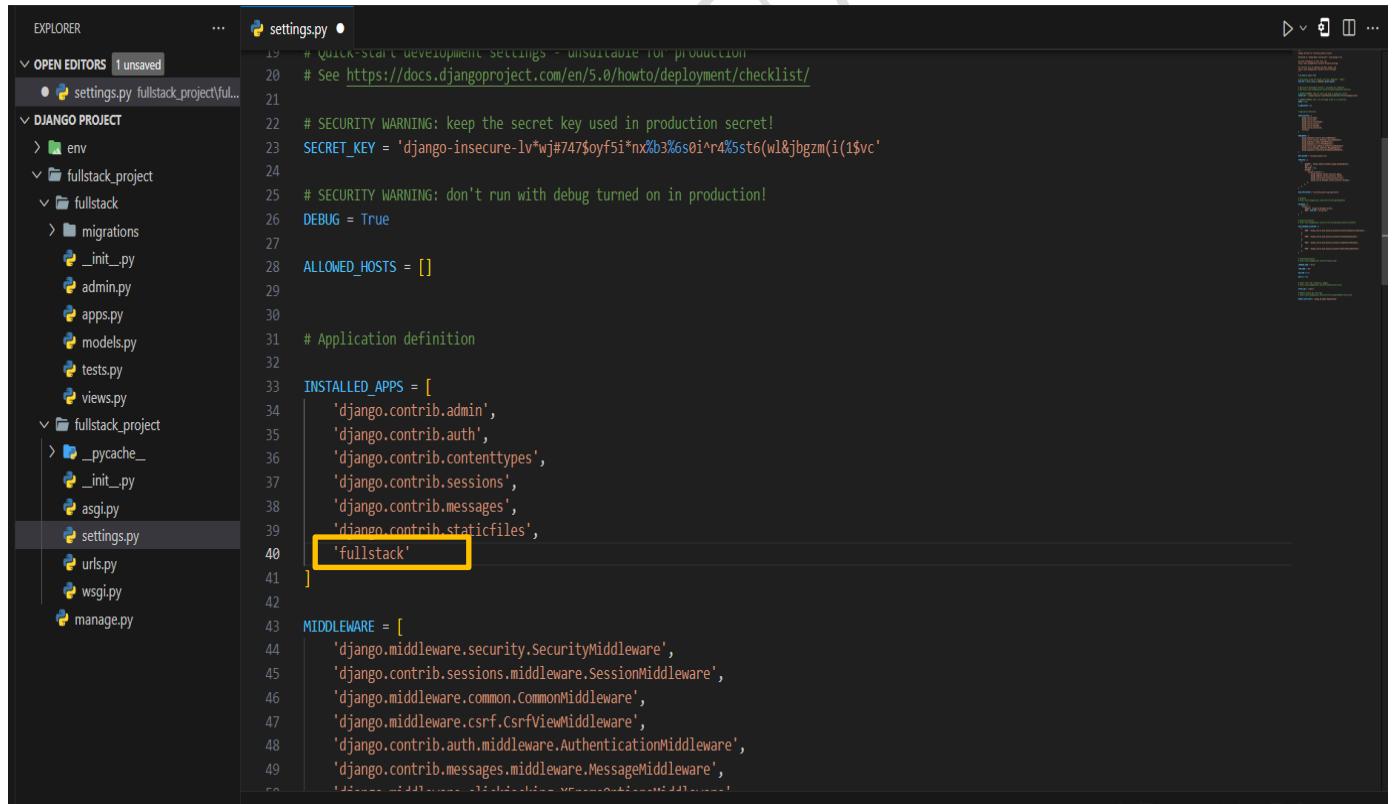


```
(env) PS D:\6th sem 2021 scheme\Full Stack\Fs lab\ Django Project\fullstack project> cd..
(env) PS D:\6th sem 2021 scheme\Full Stack\Fs lab\ Django Project> cd fullstack project
(env) PS D:\6th sem 2021 scheme\Full Stack\Fs lab\ Django Project\fullstack project> python manage.py startapp fullstack
(env) PS D:\6th sem 2021 scheme\Full Stack\Fs lab\ Django Project\fullstack project>
```

Step-08: Add the app to the INSTALLED_APPS list

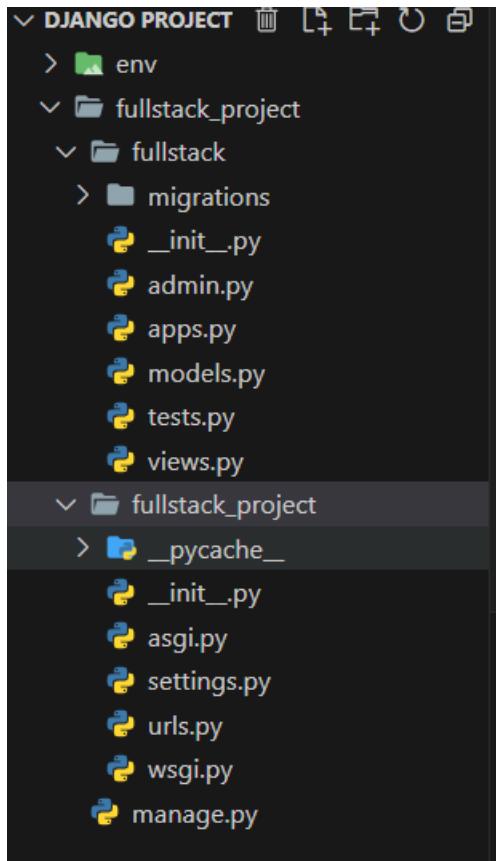
In the VS Code file explorer, locate the settings.py file (**usually located in the myproject directory**) and open it.

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



```
EXPLORER OPEN EDITORS 1 unsaved ... settings.py ●
19 # Quick-start development settings - unsuitable for production
20 # See https://docs.djangoproject.com/en/5.0/howto/deployment/checklist/
21
22 # SECURITY WARNING: keep the secret key used in production secret!
23 SECRET_KEY = 'django-insecure-lv*wj#7oyf5i^nx%b3%6s0i^r4x5st6(wl&jbgzm(i1$vc'
24
25 # SECURITY WARNING: don't run with debug turned on in production!
26 DEBUG = True
27
28 ALLOWED_HOSTS = []
29
30
31 # Application definition
32
33 INSTALLED_APPS = [
34     'django.contrib.admin',
35     'django.contrib.auth',
36     'django.contrib.contenttypes',
37     'django.contrib.sessions',
38     'django.contrib.messages',
39     'django.contrib.staticfiles',
40     'fullstack'
41 ]
42
43 MIDDLEWARE = [
44     'django.middleware.security.SecurityMiddleware',
45     'django.contrib.sessions.middleware.SessionMiddleware',
46     'django.middleware.common.CommonMiddleware',
47     'django.middleware.csrf.CsrfViewMiddleware',
48     'django.contrib.auth.middleware.AuthenticationMiddleware',
49     'django.contrib.messages.middleware.MessageMiddleware',
50 ]
```

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)

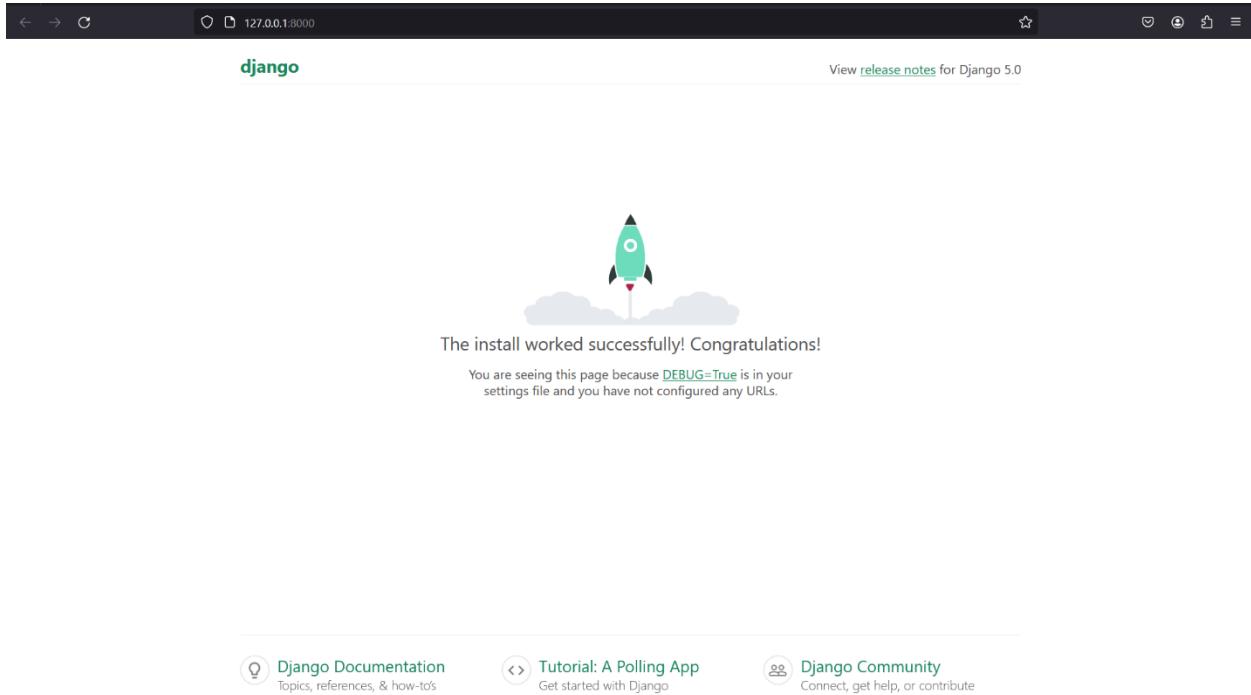
```
(env) PS D:\6th sem 2021 scheme\Full Stack\FS lab\ Django Project\fullstack_project> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

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



SEARCH CR

Experiment-03

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

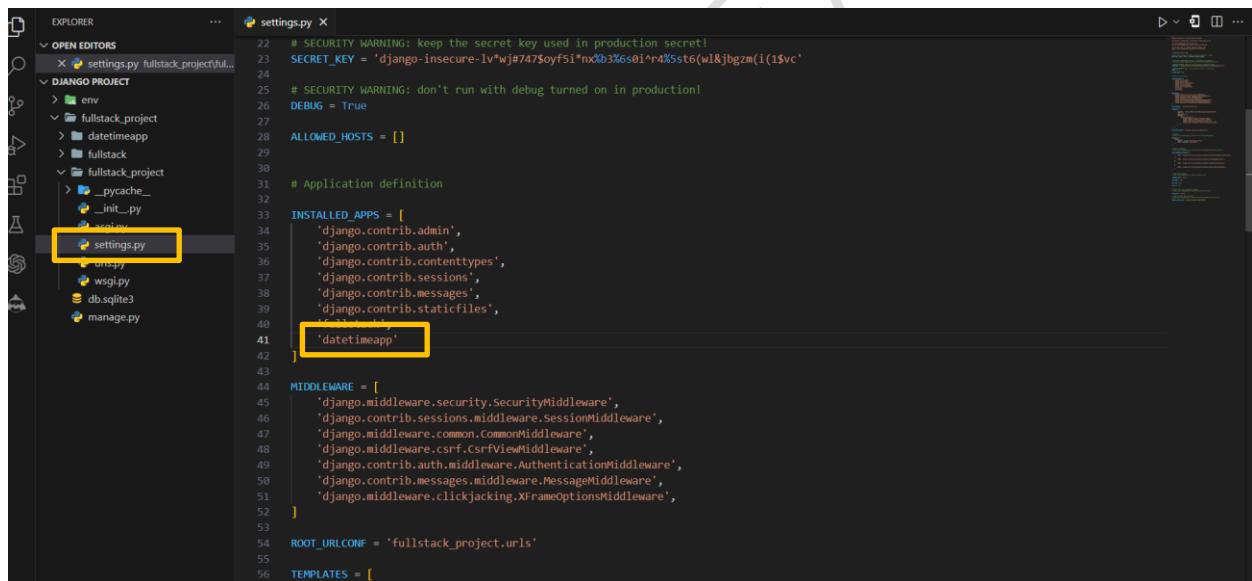
Step-01: This app will be created in the Django project we made earlier.

```
(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).

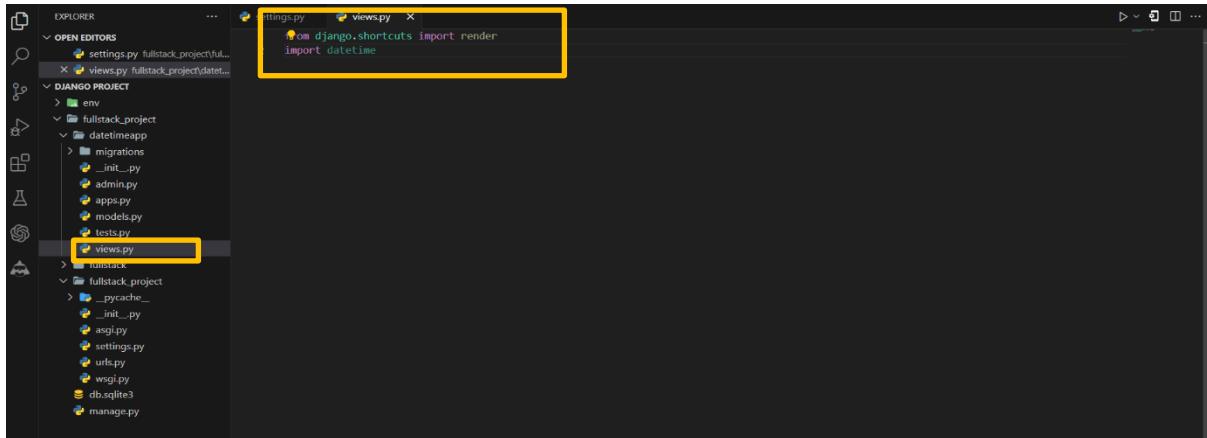
Locate the **INSTALLED_APPS** list and add the name of your new app to the list:



```
settings.py
22 # SECURITY WARNING: keep the secret key used in production secret!
23 SECRET_KEY = 'django-insecure-lv*wj#747$oyf5l^nx0B3%6s01^r4X$st6(wl&jbgzm(i1$vc'
24
25 # SECURITY WARNING: don't run with debug turned on in production!
26 DEBUG = True
27
28 ALLOWED_HOSTS = []
29
30
31 # Application definition
32
33 INSTALLED_APPS = [
34     'django.contrib.admin',
35     'django.contrib.auth',
36     'django.contrib.contenttypes',
37     'django.contrib.sessions',
38     'django.contrib.messages',
39     'django.contrib.staticfiles',
40
41     'datetimeapp'
42 ]
43
44 MIDDLEWARE = [
45     'django.middleware.security.SecurityMiddleware',
46     'django.contrib.sessions.middleware.SessionMiddleware',
47     'django.middleware.common.CommonMiddleware',
48     'django.middleware.csrf.CsrfViewMiddleware',
49     'django.contrib.auth.middleware.AuthenticationMiddleware',
50     'django.contrib.messages.middleware.MessageMiddleware',
51     'django.middleware.clickjacking.XFrameOptionsMiddleware',
52 ]
53
54 ROOT_URLCONF = 'fullstack_project.urls'
55
56 TEMPLATES = [
```

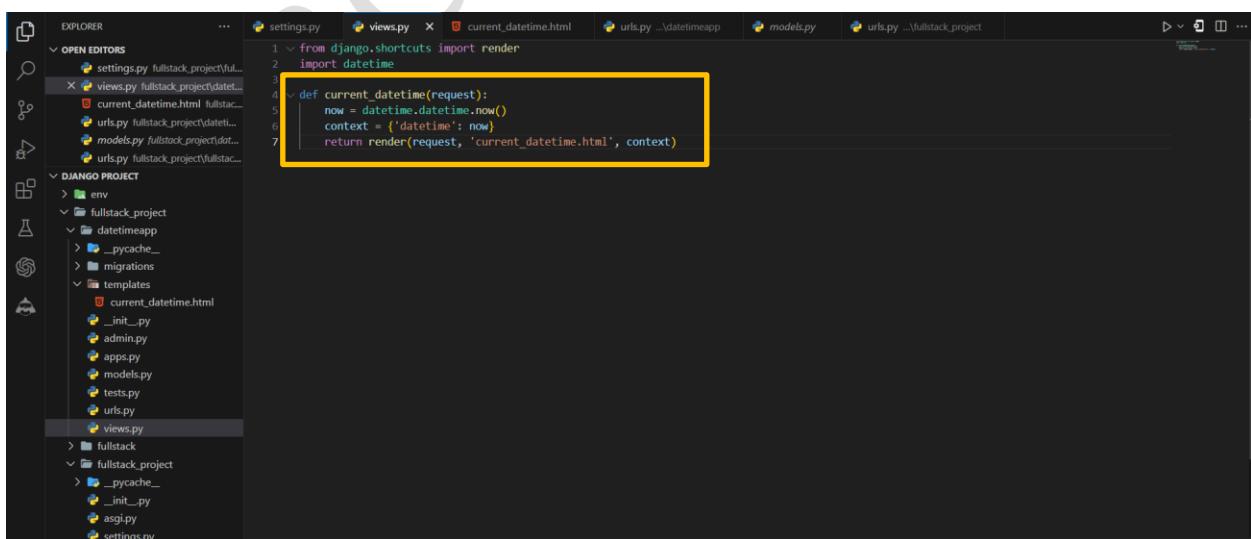
Step-03: Create a view function

- 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>

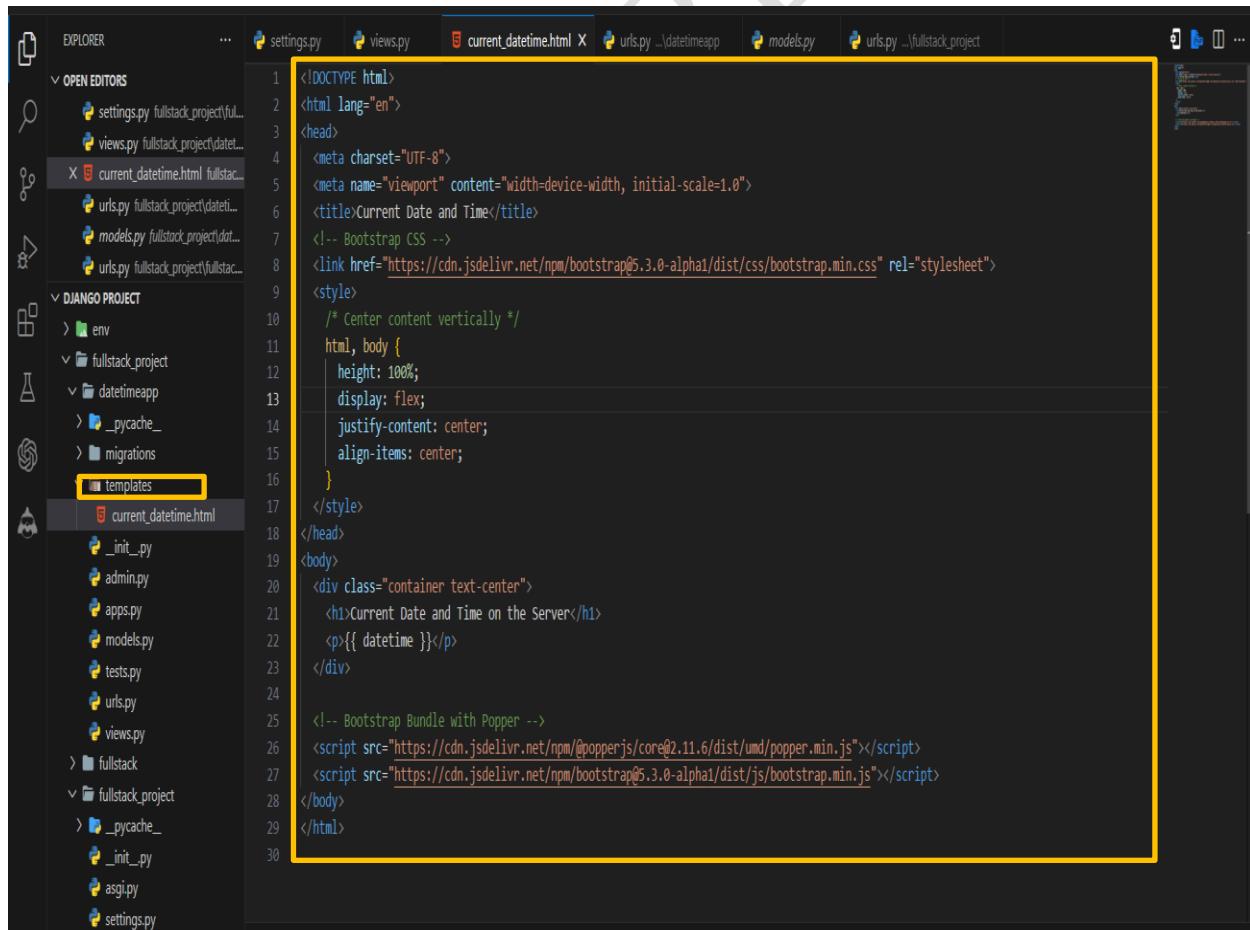
<!-- Bootstrap Bundle with Popper -->

<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>
```



```
<!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

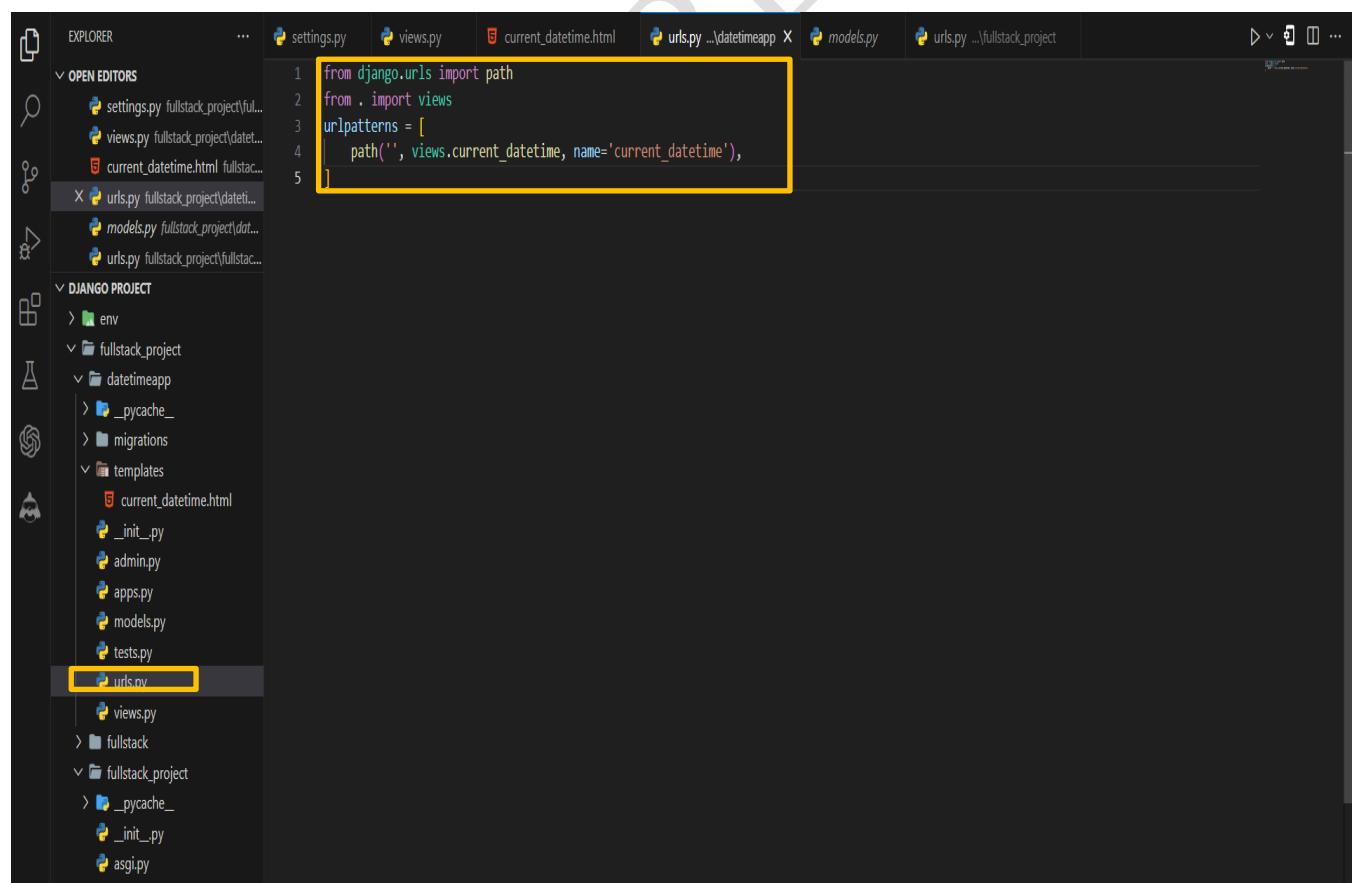
- Open the **urls.py** file in your Django app's directory (e.g., `datetimeapp/urls.py`).
- Import the view function at the top of the file
- Add a new URL pattern to the `urlpatterns` list

```
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 `/`.



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

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

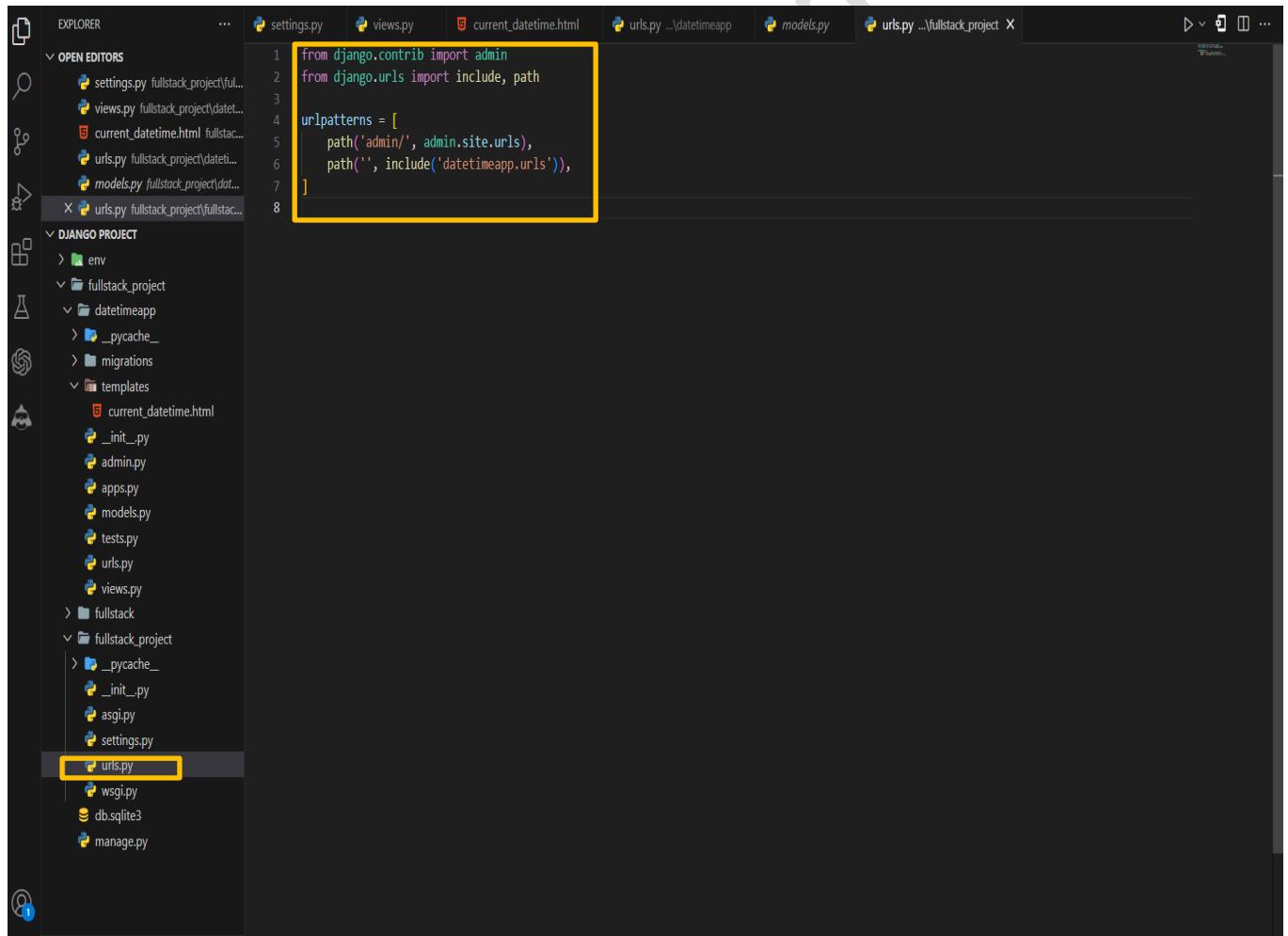
- Open the **urls.py** file in your project's directory (e.g., **fullstack_project/urls.py**).
- Import the include function from **django.urls** and the path function from **django.urls**:

```
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.



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

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

The screenshot shows a code editor interface with a dark theme. On the left is an Explorer sidebar showing the project structure:

- OPEN EDITORS**: settings.py, views.py, current_datetime.html, urls.py (highlighted), models.py
- DJANGO PROJECT**: env, fullstack_project (highlighted), datetimapp, _pycache_, migrations, templates (highlighted), fullstack, fullstack_project (highlighted), _pycache_, __init__.py, admin.py, apps.py, models.py, tests.py, urls.py (highlighted), views.py
- urls.py (highlighted again)
- wsgi.py, db.sqlite3, manage.py

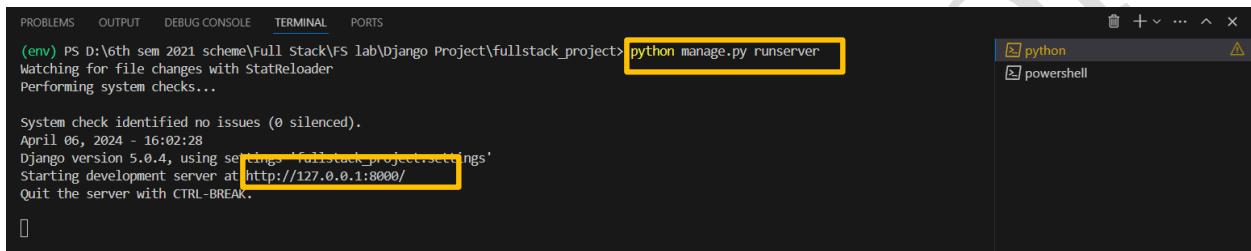
The main editor area displays the `urls.py` file content, which includes imports for `django.contrib.admin` and `django.urls`, and defines a `urlpatterns` list with two entries: a path to the admin site and a path that includes the `datetimeapp.urls`.

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.
```

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.

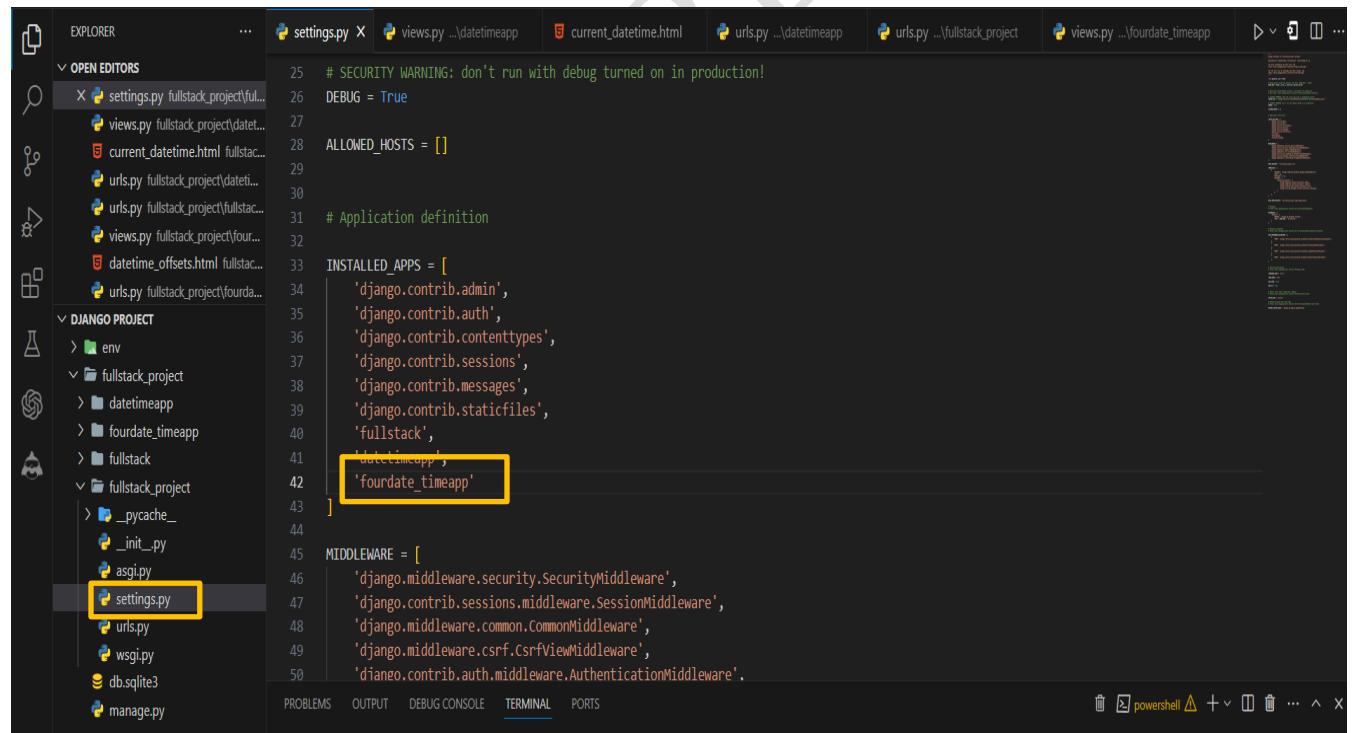
Step-01: This app will be created in the Django project we made earlier.

```
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:



```
25 # SECURITY WARNING: don't run with debug turned on in production!
26 DEBUG = True
27
28 ALLOWED_HOSTS = []
29
30 # Application definition
31
32 INSTALLED_APPS = [
33     'django.contrib.admin',
34     'django.contrib.auth',
35     'django.contrib.contenttypes',
36     'django.contrib.sessions',
37     'django.contrib.messages',
38     'django.contrib.staticfiles',
39     'fullstack',
40     'datetimeapp',
41     'fourdate_timeapp',
42     'fourdate_timeapp'
43 ]
44
45 MIDDLEWARE = [
46     'django.middleware.security.SecurityMiddleware',
47     'django.contrib.sessions.middleware.SessionMiddleware',
48     'django.middleware.common.CommonMiddleware',
49     'django.middleware.csrf.CsrfViewMiddleware',
50     'django.contrib.auth.middleware.AuthenticationMiddleware',
```

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,

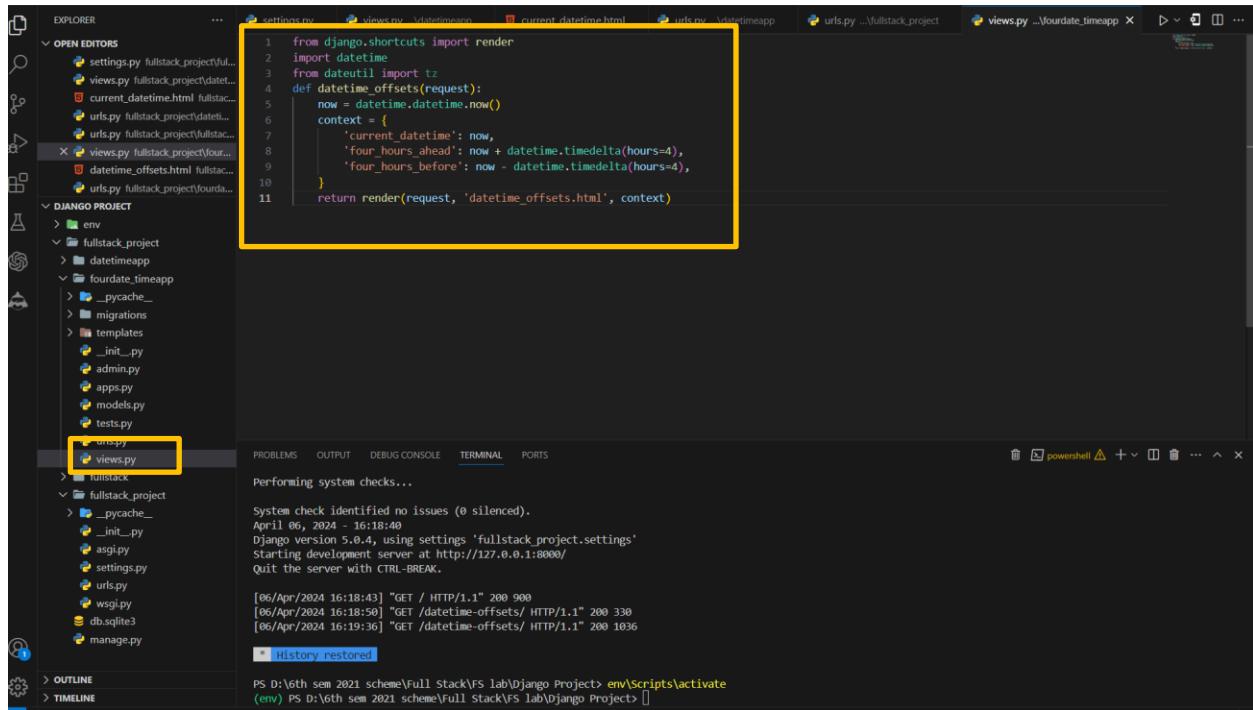
        'four_hours_ahead': now + datetime.timedelta(hours=4),

        '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.



```

1  from django.shortcuts import render
2  import datetime
3  from dateutil import tz
4  def datetime_offsets(request):
5      now = datetime.datetime.now()
6      context = {
7          'current_datetime': now,
8          'four_hours_ahead': now + datetime.timedelta(hours=4),
9          'four_hours_before': now - datetime.timedelta(hours=4),
10     }
11    return render(request, 'datetime_offsets.html', context)

```

The screenshot shows the VS Code interface with the Django project structure in the Explorer sidebar. The views.py file in the foudate_timeapp directory is open in the editor, containing the provided code. The terminal below shows the server starting and log entries for three requests. The status bar at the bottom indicates the current environment is '(env)'.

Step-04: Create a new template

- In your app's directory (**e.g., foudate_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., foudate_timeapp).
- Inside the foudate_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 */

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>{{ 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 --&gt;

&lt;script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd/popper.min.js"&gt;&lt;/script&gt;

&lt;script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.min.js"&gt;&lt;/script&gt;

&lt;/body&gt;

&lt;/html&gt;</pre>
```

The screenshot shows the Visual Studio Code interface with the following details:

- EXPLORER:** On the left, the project structure is visible under the "DJANGO PROJECT" section. A yellow box highlights the "templates" folder, which contains a file named "datetime_offsets.html".
- CODE EDITOR:** The main area displays the content of "datetime_offsets.html". The code includes HTML, CSS, and JavaScript. A large yellow box highlights the entire content of the file.
- STATUS BAR:** At the bottom, the status bar shows the file path "datetime_offsets.html" and other project files like "urls.py", "views.py", and "models.py".

```
<!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 */
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>{{ 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 -->
<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>
```

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

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

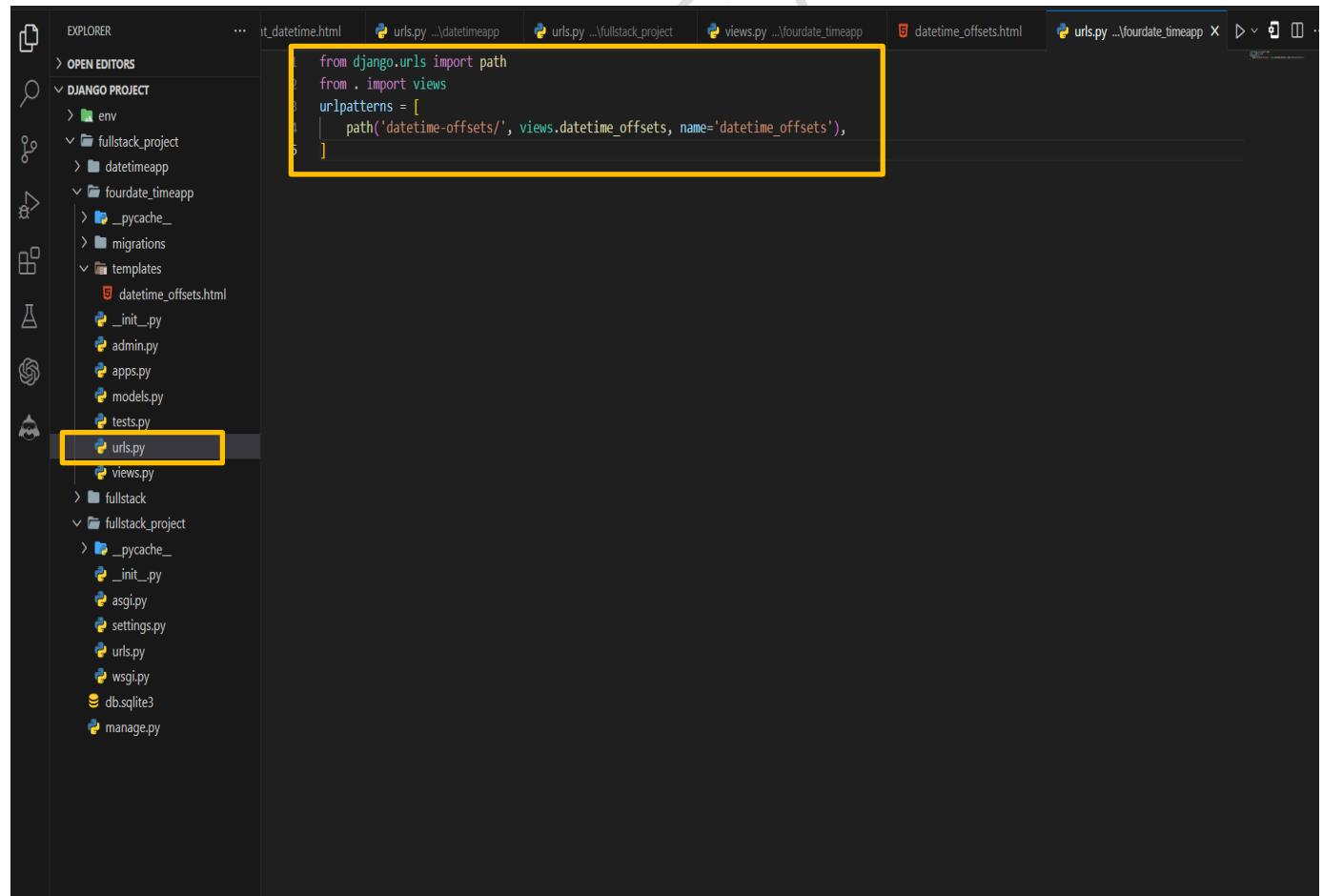
```
from django.urls import path
```

```
from . import views
```

```
urlpatterns = [
```

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

```
]
```



```
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.

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

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

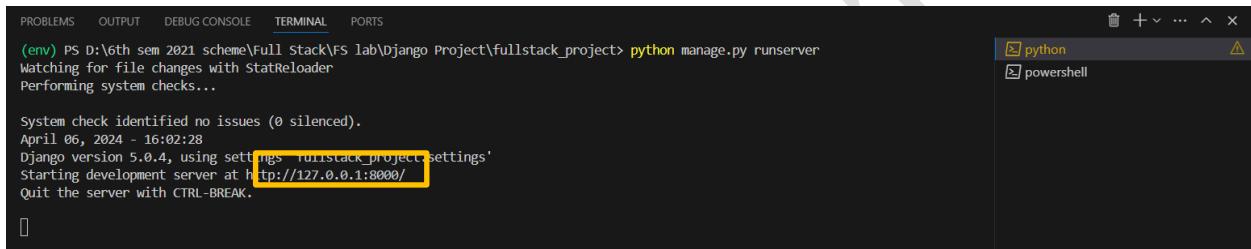
The screenshot shows a code editor interface with several tabs open at the top: settings.py, views.py, current\_datetime.html, urls.py, urls.py, urls.py, and views.py. The main editor area displays Python code for a Django project's urls.py file. The code imports admin and urls from django.contrib and django.urls respectively. It defines a urlpatterns list containing three paths: one for the admin site, one for the datetimeapp, and one for the fourdate\_timeapp, each using the include function to import the urls from those respective apps. The file is located in the fullstack\_project directory, specifically in the fourdate\_timeapp folder. The code editor has a dark theme, and the file is highlighted with syntax coloring. A yellow box highlights the 'urls.py' file in the Explorer sidebar on the left, and another yellow box highlights the 'path('', include('fourdate\_timeapp.urls'))' line in the code editor.

## 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

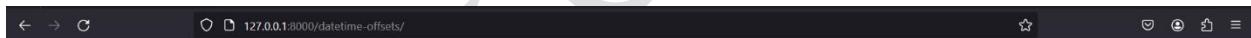
**python manage.py runserver**

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



```
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.
```

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

## **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:

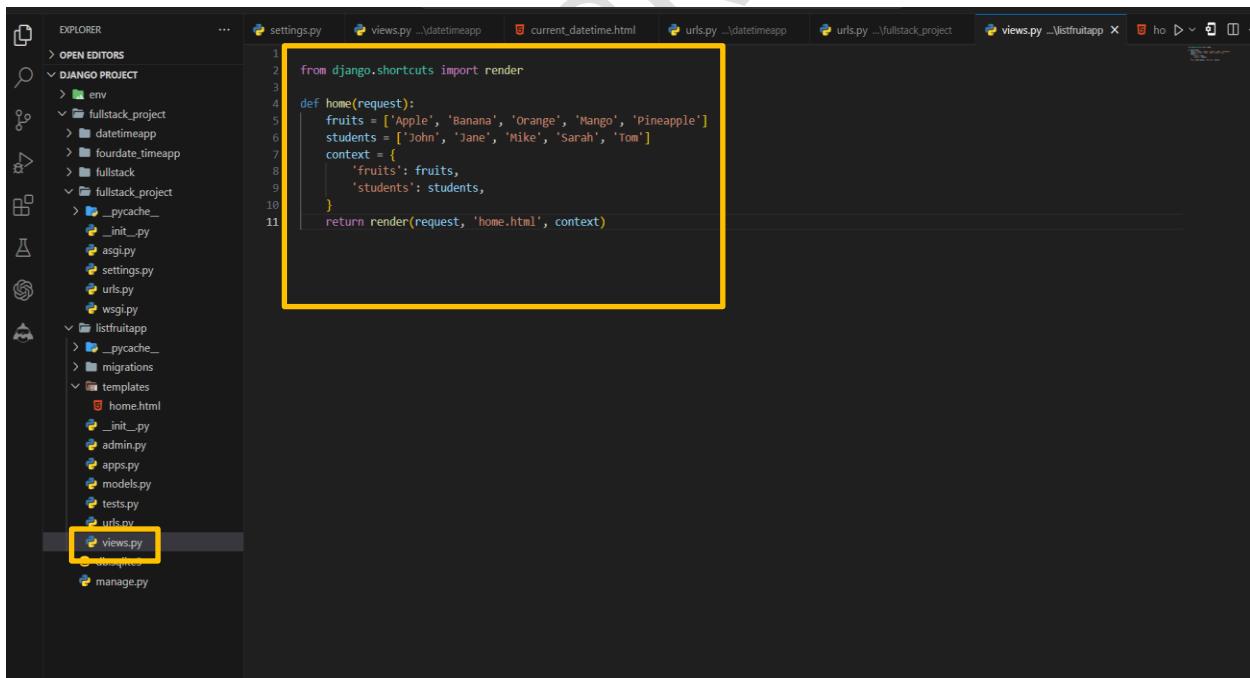
```
settings.py x views.py .../datetimeapp current_datetime.html urls.py .../datetimeapp urls.py .../fullstack_project views.py .../listfruitapp ho > ...
```

```
30
31 # Application definition
32
33 INSTALLED_APPS = [
34 'django.contrib.admin',
35 'django.contrib.auth',
36 'django.contrib.contenttypes',
37 'django.contrib.sessions',
38 'django.contrib.messages',
39 'django.contrib.staticfiles',
40 'fullstack',
41 'datetimeapp',
42 'foundate_timeapp',
43 'listfruitapp',
44]
45
46 MIDDLEWARE = [
47 'django.middleware.security.SecurityMiddleware',
48 'django.contrib.sessions.middleware.SessionMiddleware',
49 'django.middleware.common.CommonMiddleware',
50 'django.middleware.csrf.CsrfViewMiddleware',
51 'django.contrib.auth.middleware.AuthenticationMiddleware',
52 'django.contrib.messages.middleware.MessageMiddleware',
53 'django.middleware.clickjacking.XFrameOptionsMiddleware',
54]
55
56 ROOT_URLCONF = 'fullstack_project.urls'
57
58 TEMPLATES = [
59 {
60 'BACKEND': 'django.template.backends.django.DjangoTemplates',
61 'DIRS': [],
62 'APP_DIRS': True,
63 'OPTIONS': {
64 'context_processors': [
65 'django.template.context_processors.debug',
66 'django.template.context_processors.request',
67 'django.contrib.auth.context_processors.auth',
68],
69 },
70 },
71]
```

### 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)
```



The screenshot shows a code editor interface with the following details:

- EXPLORER:** Shows the project structure with the `listfruitapp` folder selected.
- OPEN EDITORS:** Shows multiple tabs: `settings.py`, `views.py` (highlighted), `current_datetime.html`, `urls.py`, and `views.py`.
- views.py Content:**

```
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., **listfruitapp**).
- Inside the **listfruitapp** 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 {

display: flex;

justify-content: center;

align-items: center;

}
```

```
}

.container {
 text-align: center;
}

/* Style for lists */

.list-container {
 display: 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">
 {% for fruit in fruits %}
 <li class="list-group-item">{{ fruit }}
 {% endfor %}

 </div>
 </div>
</div>
```

```
</div>

<div class="col">

 <h1>Selected Students</h1>

 <ol class="list-group list-container">

 { % for student in students % }

 <li class="list-group-item">{ { student } }

 { % endfor % }

</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>
```

The screenshot shows the Visual Studio Code interface with the Django project structure in the Explorer sidebar. The current file is `home.html`, which is highlighted with a yellow box. The code in `home.html` uses Django template tags (`{% %}`) to display fruit and student lists.

```

<!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 {
 display: flex;
 justify-content: center;
 align-items: center;
 }
 .container {
 text-align: center;
 }
 /* Style for lists */
 .list-container {
 display: 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">
 {% for fruit in fruits %}
 <li class="list-group-item">{{ fruit }}
 {% endfor %}

 </div>
 <div class="col">
 <h1>Selected Students</h1>
 </div>
 </div>
 </div>
</body>

```

- 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

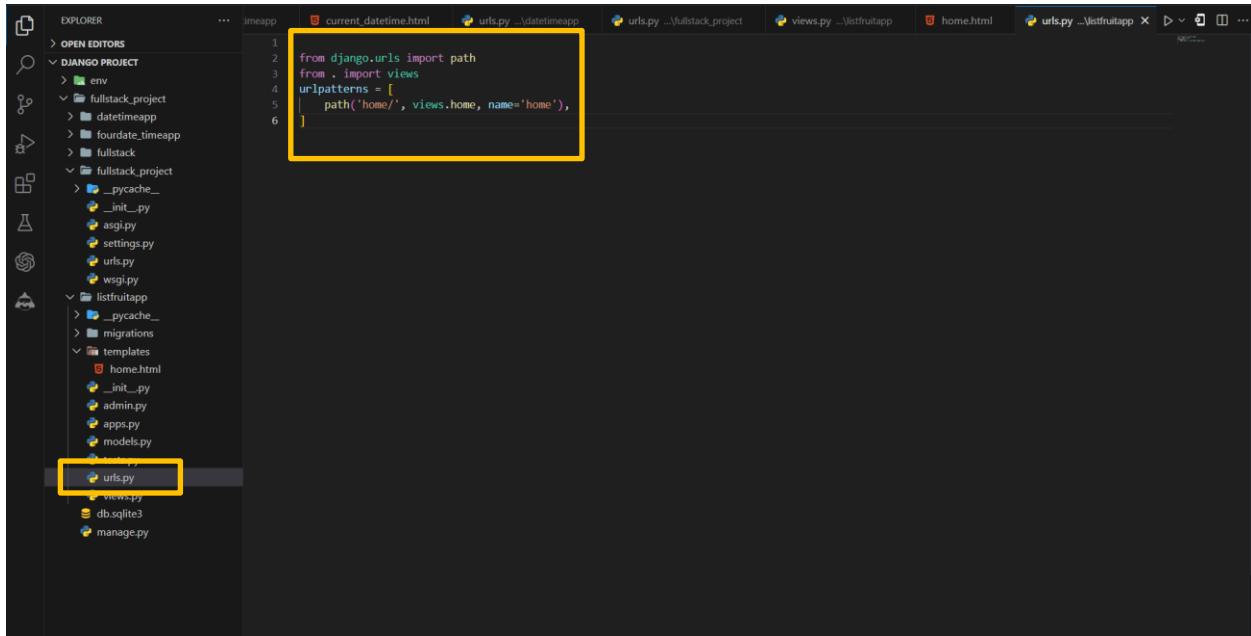
```
from django.urls import path
```

```
from . import views
```

```
urlpatterns = [
```

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

```
]
```



The screenshot shows the VS Code interface with the following details:

- EXPLORER:** Shows the project structure under "DJANGO PROJECT". The "fullstack\_project" folder contains "env", "datetimeapp", "fourdate\_timeapp", "fullstack", and "listfruitapp" subfolders. Each of these subfolders has its own ".pycache", "migrations", and "templates" subfolders.
- CODE EDITOR:** The "urls.py" file for the "datetimeapp" app is open. It contains the following code:

```

1 from django.urls import path
2 from . import views
3 urlpatterns = [
4 path('home/', views.home, name='home'),
5]

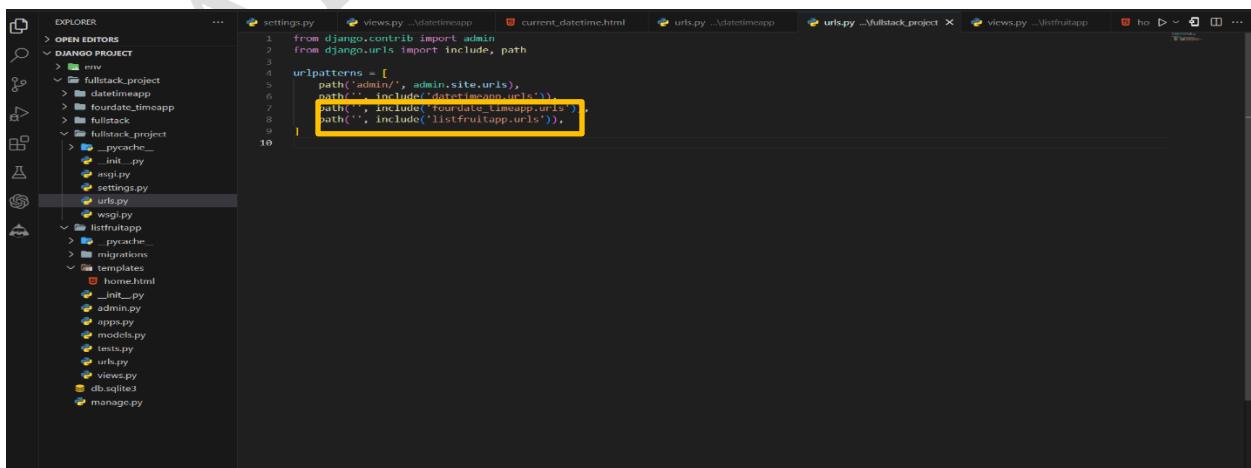
```
- STATUS BAR:** Shows file tabs for "current\_datetime.html", "urls.py ...datetimeapp", "urls.py ...fullstack\_project", "views.py ...listfruitapp", "home.html", "urls.py ...listfruitapp", and "urls.py ...listfruitapp".

## 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.



The screenshot shows the VS Code interface with the following details:

- EXPLORER:** Shows the project structure under "DJANGO PROJECT". The "fullstack\_project" folder contains "env", "datetimeapp", "fourdate\_timeapp", "fullstack", and "listfruitapp" subfolders. Each of these subfolders has its own ".pycache", "migrations", and "templates" subfolders.
- CODE EDITOR:** The "urls.py" file for the "fullstack\_project" is open. It contains the following code:

```

1 from django.contrib import admin
2 from django.urls import include, path
3
4 urlpatterns = [
5 path('admin/', admin.site.urls),
6 path('', include('fourdate_timeapp.urls')),
7 path('', include('listfruitapp.urls')),
8]

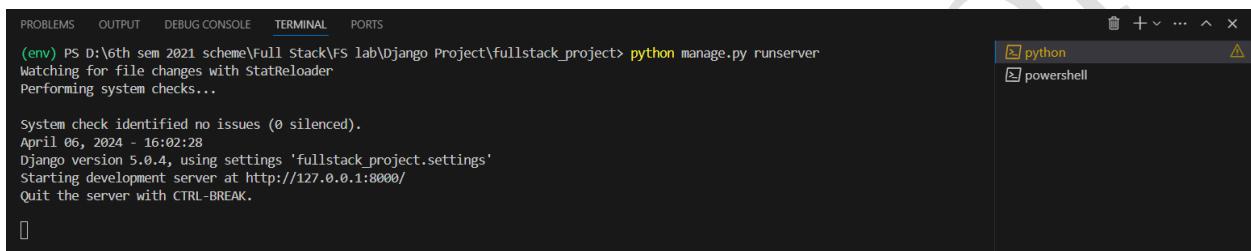
```
- STATUS BAR:** Shows file tabs for "settings.py", "views.py ...datetimeapp", "current\_datetime.html", "urls.py ...datetimeapp", "urls.py ...fullstack\_project", "views.py ...listfruitapp", and "urls.py ...listfruitapp".

## 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**

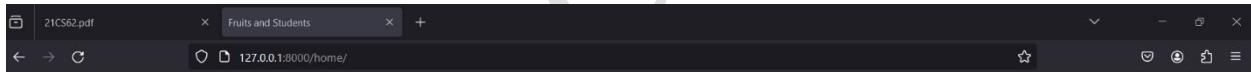
- Open your web browser and visit <http://127.0.0.1:8000/>.



```
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/home/>

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

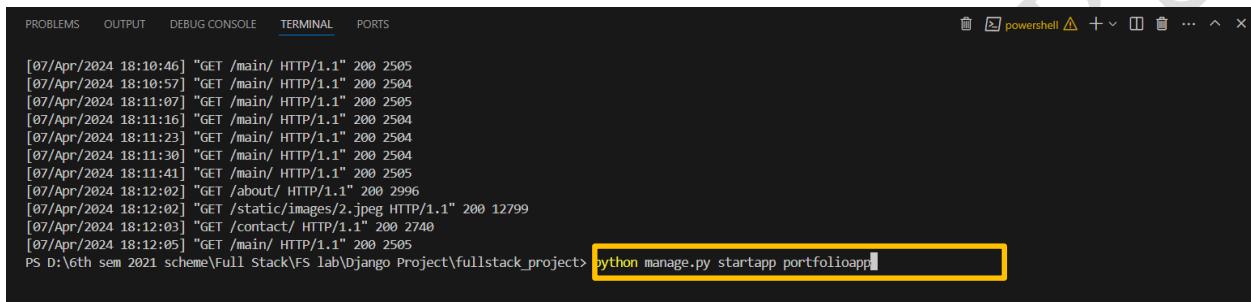


Fruits	Selected Students
Apple	John
Banana	Jane
Orange	Mike
Mango	Sarah
Pineapple	Tom

## 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.

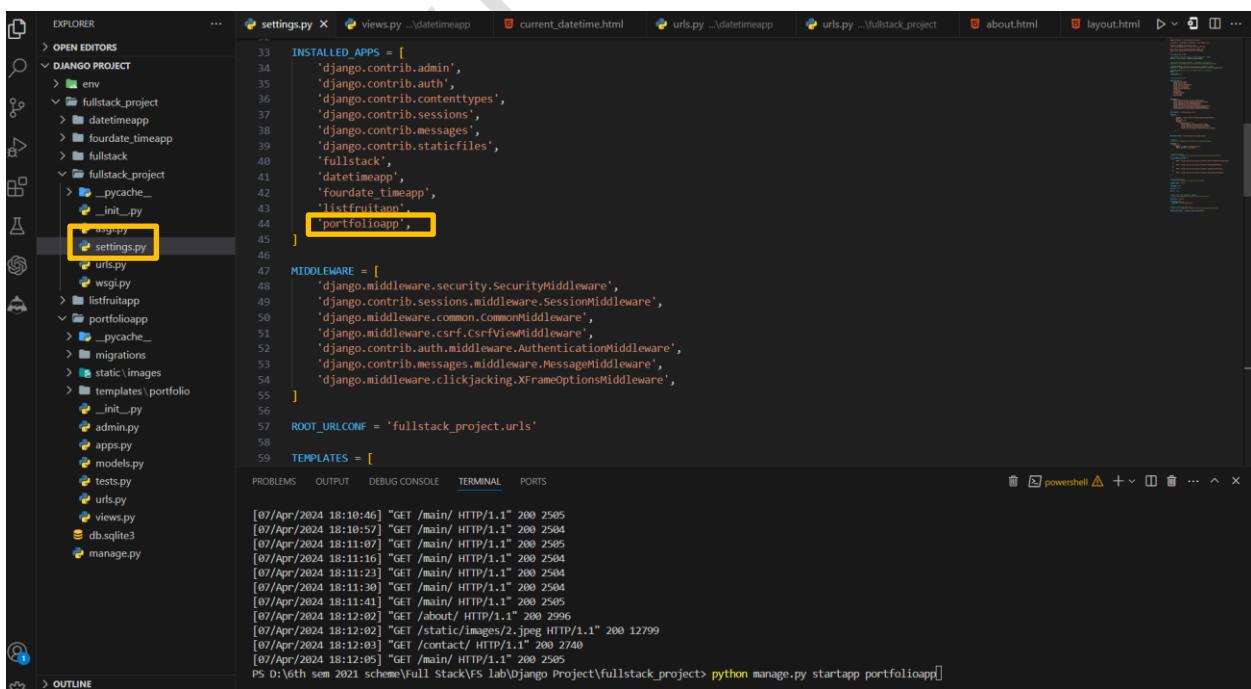


```
[07/Apr/2024 18:10:46] "GET /main/ HTTP/1.1" 200 2505
[07/Apr/2024 18:10:57] "GET /main/ HTTP/1.1" 200 2504
[07/Apr/2024 18:11:07] "GET /main/ HTTP/1.1" 200 2505
[07/Apr/2024 18:11:16] "GET /main/ HTTP/1.1" 200 2504
[07/Apr/2024 18:11:23] "GET /main/ HTTP/1.1" 200 2504
[07/Apr/2024 18:11:30] "GET /main/ HTTP/1.1" 200 2504
[07/Apr/2024 18:11:41] "GET /main/ HTTP/1.1" 200 2505
[07/Apr/2024 18:12:02] "GET /about/ HTTP/1.1" 200 2996
[07/Apr/2024 18:12:02] "GET /static/images/2.jpeg HTTP/1.1" 200 12799
[07/Apr/2024 18:12:03] "GET /contact/ HTTP/1.1" 200 2740
[07/Apr/2024 18:12:05] "GET /main/ HTTP/1.1" 200 2505
PS D:\6th sem 2021 scheme\Full Stack\F5 lab\ Django Project\fullstack_project> python manage.py startapp portfolioapp
```

**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:



```
INSTALLED_APPS = [
 'django.contrib.admin',
 'django.contrib.auth',
 'django.contrib.contenttypes',
 'django.contrib.sessions',
 'django.contrib.messages',
 'django.contrib.staticfiles',
 'fullstack',
 'datetimeapp',
 'foudate_timeapp',
 'listfruitapp',
 'portfolioapp',
]
```

### 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 */

.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>

Home

About Us

Contact Us


```

```
</nav>

</header>

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

<footer>
 <p>© 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>
```

```

<!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 */
 .center-content {
 display: flex;
 justify-content: center;
 }
</style>

```

## 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>

</div>

</div>

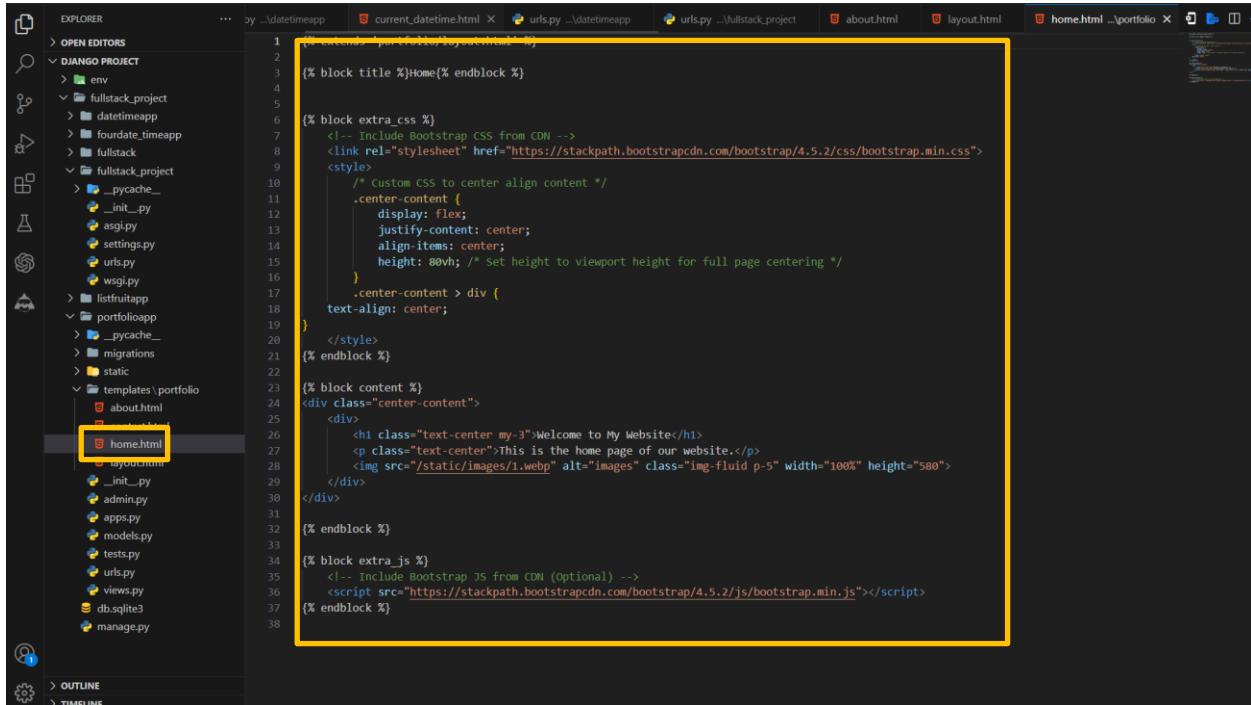
{% endblock %}

{% 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 %}



```
1 {% extends 'portfolio/layout.html' %} 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38
```

```
{% 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>

</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 %}
```

## 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="container">

<div class="row justify-content-center">

```
<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">

 <p>Web Development</p>

 </div>

 <div class="col-md-3 text-center">

 <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 %}
```

```
% 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="container">
 <div class="row justify-content-center">
 <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">

 <p>Web Development</p>
 </div>
 <div class="col-md-3 text-center">

 <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 class="text-center">Phone: 123-456-7890
 <li class="text-center">Address: 123 Main Street, City, State

 </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 %}
```

The screenshot shows a code editor interface with a dark theme. On the left is the Explorer sidebar, which lists the project structure. It includes a 'DJANGO PROJECT' section with 'fullstack\_project' containing 'env', 'datetimeapp', 'foudrate\_timeapp', and 'fullstack'. Under 'fullstack' are files like '\_pycache\_/\_init\_.py', 'asgi.py', 'settings.py', 'urls.py', and 'wsgi.py'. There are also 'listfruitapp' and 'portfolioapp' sections. Inside 'portfolioapp', there are 'migrations', 'static', and 'templates/portfolio' directories. The 'templates/portfolio' directory contains files: 'about.html', 'home.html', 'layout.html', 'portfolio.html', and 'contact.html'. The 'contact.html' file is selected and shown in the main editor area. This file contains HTML and Django template code. A yellow box highlights the code block.

```

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

{% block content %}

Contact Us

You can reach us at:

- Email: contact@mywebsite.com
- Phone: 123-456-7890
- Address: 123 Main Street, City, State

{% endblock %}

<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')
```

The screenshot shows the VS Code interface with a Django project structure. The Explorer sidebar on the left lists files and folders: .env, fullstack\_project, fullstack, listfruitapp, portfolioapp, static, templates\portfolio, and manage.py. The portfolioapp folder contains \_\_init\_\_.py, admin.py, apps.py, models.py, tests.py, and views.py. The views.py file is open in the editor, showing Python code for rendering HTML templates:

```
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')
```

The views.py file is highlighted with a yellow box. The status bar at the bottom indicates "Search Creators....".

## 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'),
]
```

The screenshot shows the VS Code interface with the following details:

- EXPLORER:** Shows the project structure under "Django Project". It includes "fullstack\_project" (containing "env", "fullstack", "fullstack\_project", "listfruityapp", "portfolioapp" (containing ".pycache-", "migrations", "static", "templates\portfolio"), and "tests.py").
- OPEN EDITORS:** Shows multiple tabs: "urls.py" (highlighted with a yellow box), "about.html", "layout.html", "home.html", "contact.html", "views.py", and "urls.py" (another tab).
- Code Editor:** Displays the content of "urls.py" in the portfolioapp directory:

```

1 from django.urls import path
2 from . import views
3
4 urlpatterns = [
5 path('main/', views.main, name='main'),
6 path('about/', views.about, name='about'),
7 path('contact/', views.contact, name='contact'),
8]

```

## 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 exists]**

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

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

The screenshot shows the VS Code interface with the following details:

- EXPLORER:** Shows the project structure under "Django Project". It includes "fullstack-project" (containing "env", "fullstack", "fullstack\_project", "listfruityapp", "portfolioapp" (containing ".pycache-", "migrations", "static", "templates\portfolio"), and "tests.py").
- OPEN EDITORS:** Shows multiple tabs: "urls.py" (highlighted with a yellow box), "about.html", "layout.html", "home.html", "contact.html", "views.py", and "urls.py" (another tab).
- Code Editor:** Displays the content of "urls.py" in the fullstack\_project directory:

```

1 from django.contrib import admin
2 from django.urls import include, path
3
4 urlpatterns = [
5 path('admin/', admin.site.urls),
6 path('', include('listfruityapp.urls')),
7 path('', include('fullstack_timeapp.urls')),
8 path('', include('portfolioapp.urls')),
9]

```

## 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 StackFS 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/main/>

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

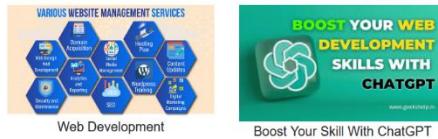


**Fig: Home Page Screen**

Home About Us Contact Us

## About Us

We are a company that provides awesome products and services.



© 2024 Search Creators. Developed by Hanumanthu.

**Fig: About Us Screen**

Home About Us Contact Us

## Contact Us

You can reach us at:

Email: contact@mywebsite.com  
Phone: 123-456-7890  
Address: 123 Main Street, City, State

© 2024 Search Creators. Developed by Hanumanthu.

**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:

```
EXPLORER
OPEN EDITORS
DJANGO PROJECT
env
fullstack_project
 datetimemapp
 foudrdate_timeapp
 fullstack
 __pycache__
 __init__.py
 asgi.py
 settings.py
 wsgi.py
listfruitapp
portfolioapp
student_course_registration...
 __pycache__
 migrations
 templates
 registration
 base.html
 __init__.py
 admin.py
 apps.py
 forms.py
 models.py
 tests.py
settings.py X models.py X admin.py X forms.py X views.py ...student_course_registration_app X defaults.py X course_students.html
35 'django.contrib.auth',
36 'django.contrib.contenttypes',
37 'django.contrib.sessions',
38 'django.contrib.messages',
39 'django.contrib.staticfiles',
40 'fullstack',
41 'datetimemapp',
42 'foudrdate_timeapp',
43 'listfruitapp',
44 'portfolioapp',
45 'student_course_registration_app',
46]
47 MIDDLEWARE = [
48 'django.middleware.security.SecurityMiddleware',
49 'django.contrib.sessions.middleware.SessionMiddleware',
50 'django.middleware.common.CommonMiddleware',
51 'django.middleware.csrf.CsrfViewMiddleware',
52 'django.contrib.auth.middleware.AuthenticationMiddleware',
53 'django.contrib.messages.middleware.MessageMiddleware',
54 'django.middleware.clickjacking.XFrameOptionsMiddleware',
55]
56 ROOT_URLCONF = 'fullstack_project.urls'
57
58 TEMPLATES = [
59 {
60 'BACKEND': 'django.template.backends.django.DjangoTemplates',
61 'DIRS': [],
62 'APP_DIRS': True,
63 'OPTIONS': {
64 'context_processors': [
65 'django.template.context_processors.debug',
```

### 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}'
```

```

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)

```

```

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

admin.site.register(Course)
admin.site.register(Student)

```

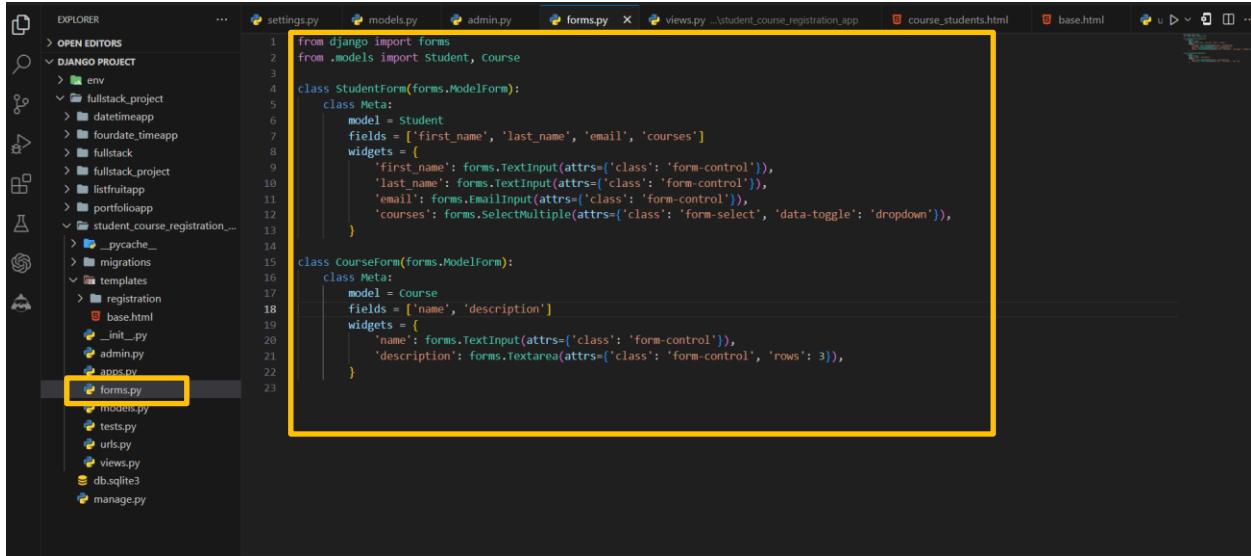
### Step-05: Create forms

- 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}),
 }
```



```
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:
```

21CS62 | Full Stack Django Development|

```
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()

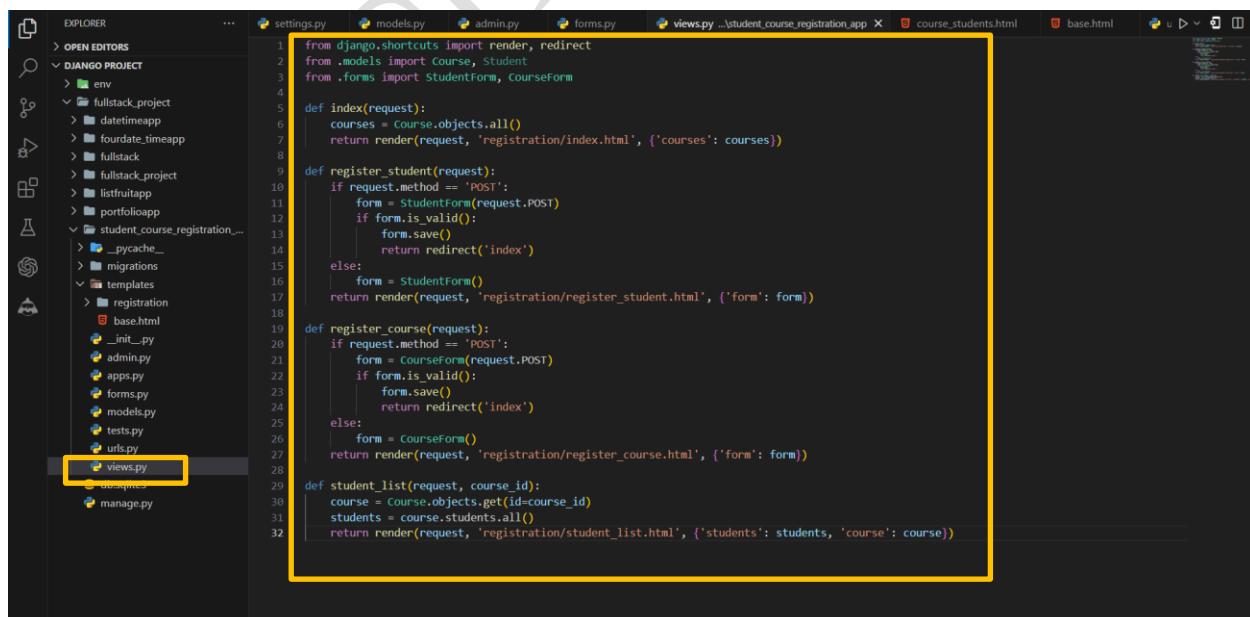
 return render(request, 'registration/register_course.html', {'form': form})

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">{{ course.name }}

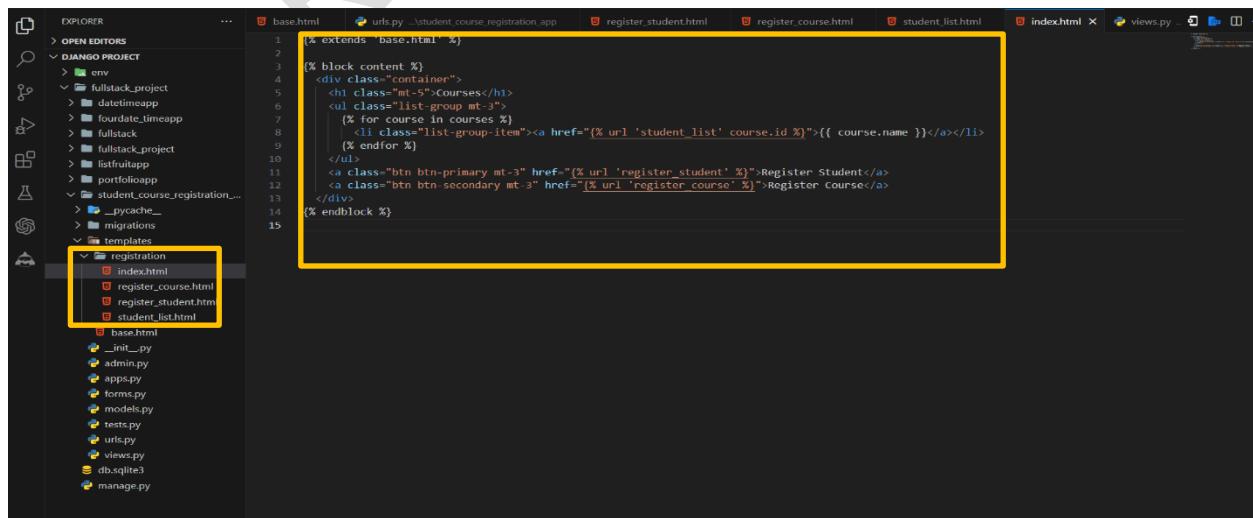
 {% endfor %}

Register Student

Register Course

</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

 </script>
```

```
var dropdownElementList = [].slice.call(document.querySelectorAll('.dropdown-toggle'))

var dropdownList = dropdownElementList.map(function (dropdownToggleEl) {

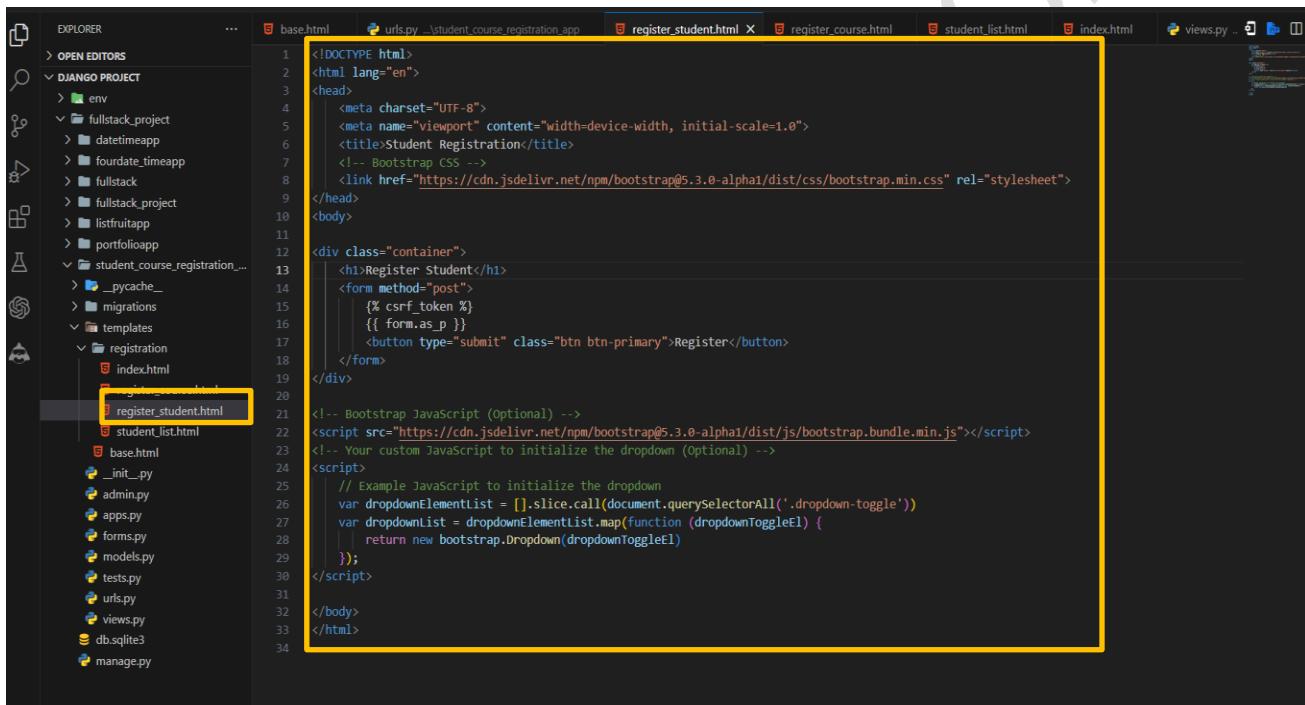
 return new bootstrap.Dropdown(dropdownToggleEl)

});

</script>

</body>

</html>
```



```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 <meta charset="UTF-8">
5 <meta name="viewport" content="width=device-width, initial-scale=1.0">
6 <title>Student Registration</title>
7 <!-- Bootstrap CSS -->
8 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css" rel="stylesheet">
9 </head>
10 <body>
11
12 <div class="container">
13 <h1>Register Student</h1>
14 <form method="post">
15 {% csrf_token %}
16 {{ form.as_p }}
17 <button type="submit" class="btn btn-primary">Register</button>
18 </form>
19 </div>
20
21 <!-- Bootstrap JavaScript (Optional) -->
22 <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.bundle.min.js"></script>
23 <!-- Your custom JavaScript to initialize the dropdown (Optional) -->
24 <script>
25 // Example JavaScript to initialize the dropdown
26 var dropdownElementList = [].slice.call(document.querySelectorAll('.dropdown-toggle'))
27 var dropdownList = dropdownElementList.map(function (dropdownToggleEl) {
28 return new bootstrap.Dropdown(dropdownToggleEl)
29 });
30 </script>
31
32 </body>
33 </html>
```

## register\_course.html

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

{% load static %}

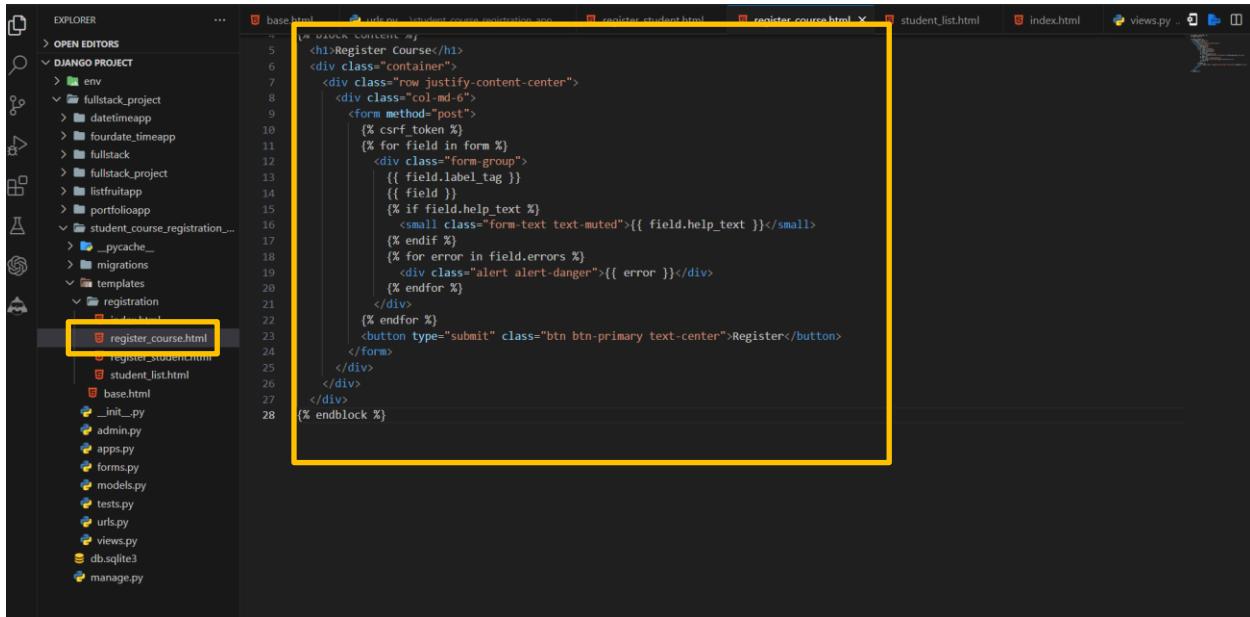
{% block content %}

<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>
{% endblock %}
```



The screenshot shows a code editor interface with a sidebar labeled 'EXPLORER' containing a 'DJANGO PROJECT' tree. The tree includes 'env', 'fullstack\_project', 'datetimeapp', 'foundate\_timeapp', 'fullstack', 'listfruitapp', 'portfolioapp', and 'student\_course\_registration...'. Under 'student\_course\_registration...', there are 'migrations', 'templates', and 'registration' folders. Inside 'registration', the 'register\_course.html' file is selected and highlighted with a yellow box. The main editor area displays the content of 'register\_course.html', which is a Django template for registering a course.

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

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

## student\_list.html

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

{% block content %}

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

 {% for student in students %}

 {{ student.first_name }} {{ student.last_name }}

 {% endfor %}

{% endblock %}
```

The screenshot shows a code editor interface with a sidebar on the left displaying a project structure. The project structure includes a 'templates' directory containing 'registration' and 'student\_course\_registration' sub-directories. Inside 'student\_course\_registration', there are files like 'index.html', 'register\_course.html', and 'student.list.html'. The 'student.list.html' file is highlighted with a yellow box and is open in the main editor area. The code in 'student.list.html' is a Django template:

```
1 {% extends 'base.html' %}
2
3 {% block content %}
4 <h1>Students Registered for {{ course.name }}</h1>
5
6 {% for student in students %}
7 {{ student.first_name }} {{ student.last_name }}
8 {% endfor %}
9
10 {% endblock %}
```

## Step-08: Create a base template

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

### base.html

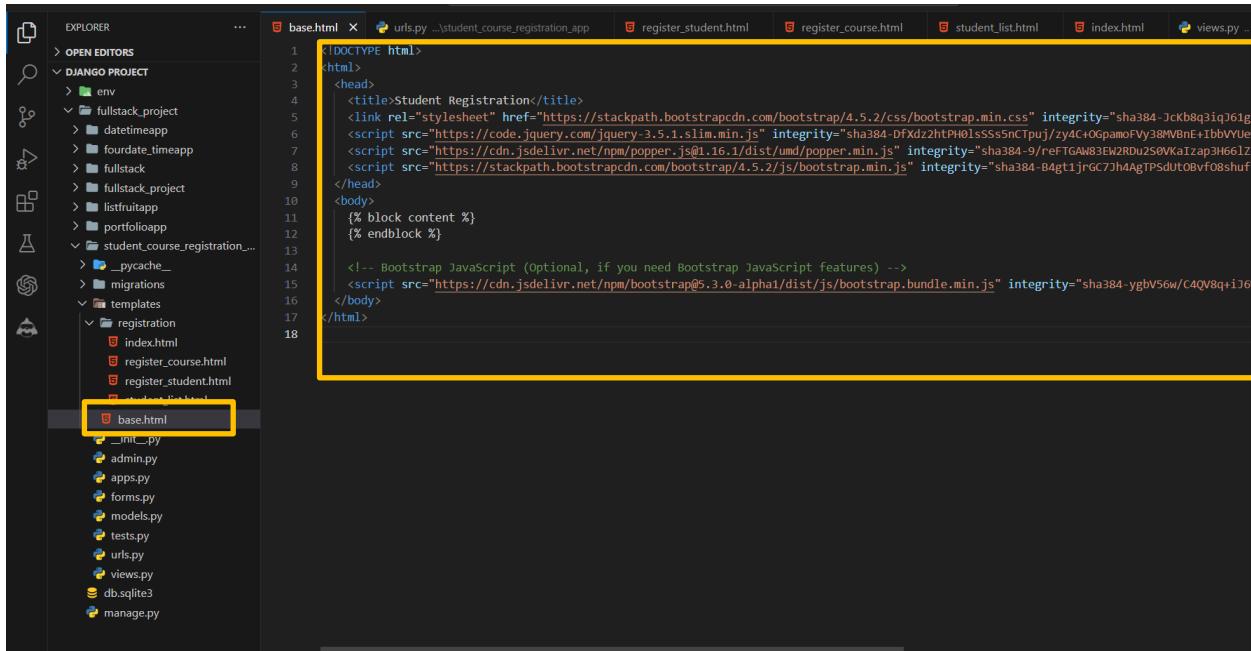
```
<!DOCTYPE html>

<html>

 <head>
 <title>Student Registration</title>
 </head>

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

</html>
```



```

1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>Student Registration</title>
5 <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" integrity="sha384-JcKb8q3iqJ61g61g7WVUeFvQfHdI5t58qZ9FEJH+q5O+Q>">
6 <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-Dfxz2htPH0lSSs5nCTpuj/zY4C+0GpamOFvy38MVRnE+ThbVVUe">
7 <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js" integrity="sha384-9/rFTGAw83EW2RDv250VK3Tzap3H6612">
8 <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js" integrity="sha384-B4gt1jrGC7Jh4AgTPSdUtoVfo8shuf">
9 </head>
10 <body>
11 {% block content %}
12 {% endblock %}
13
14 <!-- Bootstrap JavaScript (Optional, if you need Bootstrap JavaScript features) -->
15 <script src="https://cdn.jsdelivr.net/npm/bootstrap.bundle.min.js" integrity="sha384-ygbV56w/C4QV8q+iJ6">
16 </body>
17 </html>
18

```

## 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 = [
```

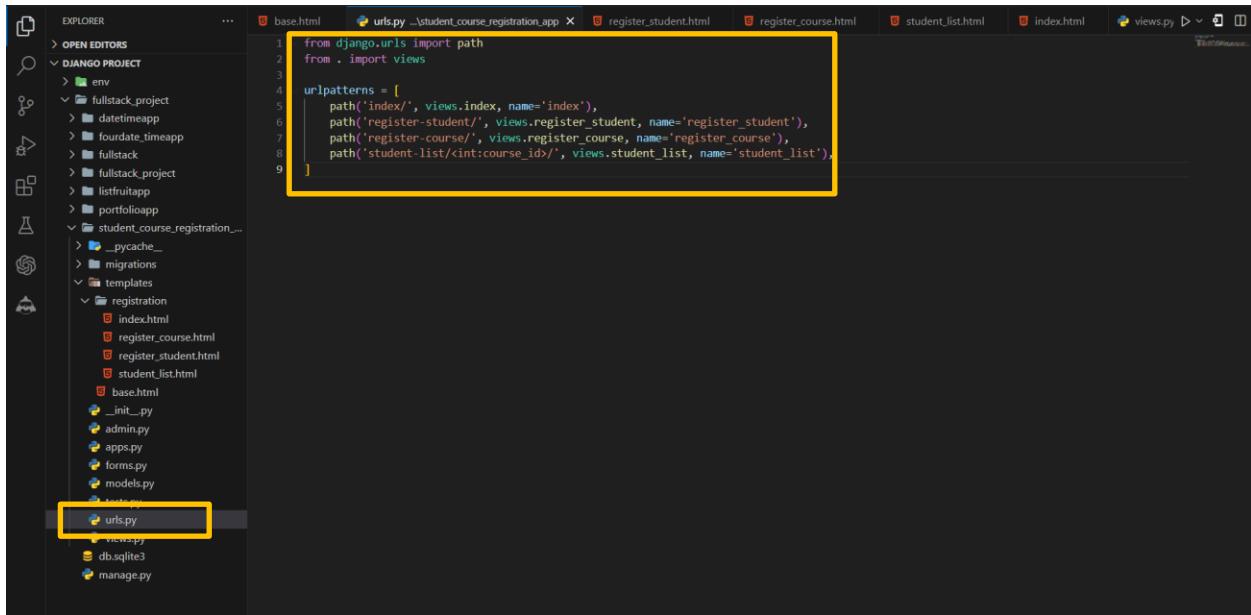
```
 path('index/', views.index, name='index'),
```

```
 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'),
```

```
]
```



```

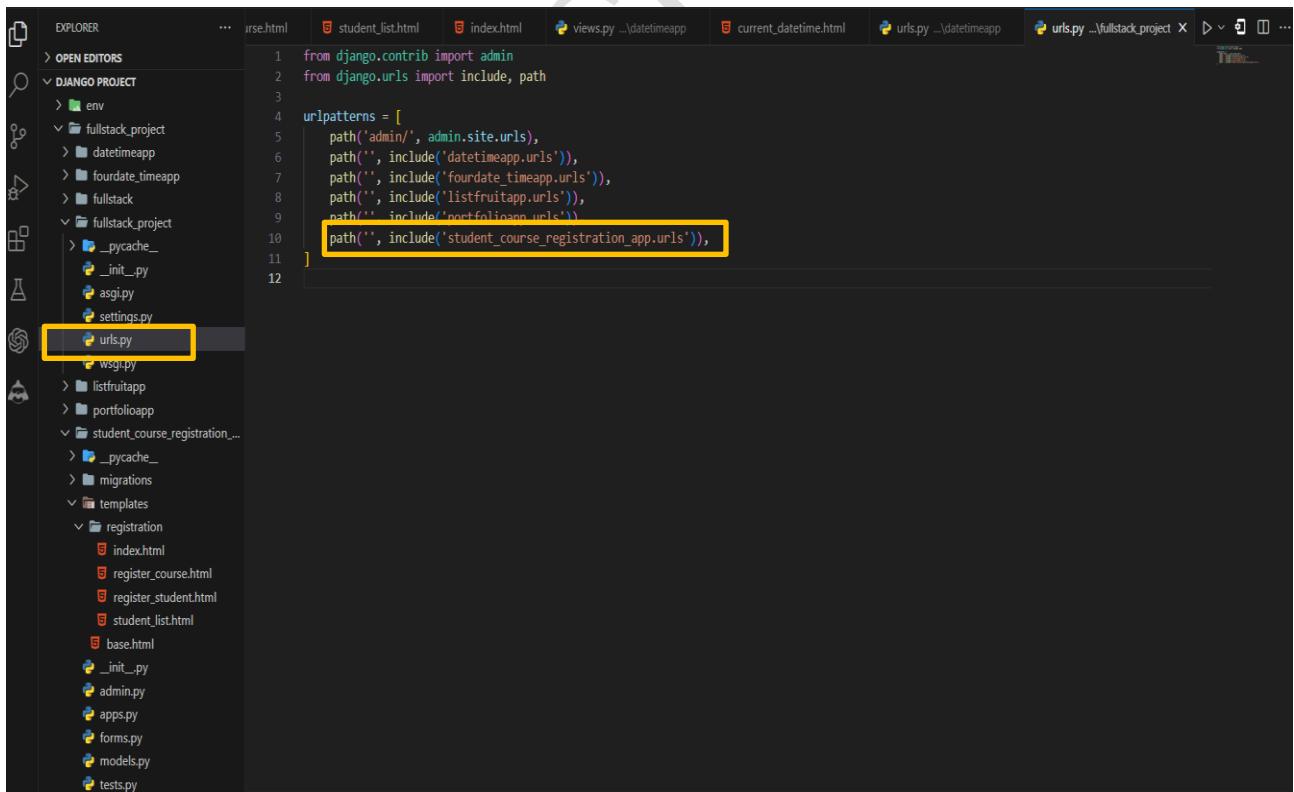
from django.urls import path
from . import views

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:



```

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

urlpatterns = [
 path('admin/', admin.site.urls),
 path('', include('datetimeapp.urls')),
 path('', include('fourdate_timeapp.urls')),
 path('', include('listfruitapp.urls')),
 path('', include('portfolioapp.urls')),
 path('', include('student_course_registration_app.urls'))
]

```

## 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

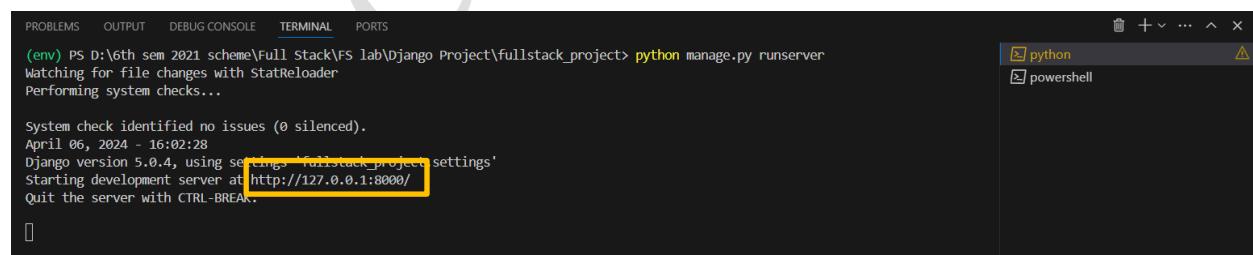
```
PS D:\6th sem 2021 scheme\Full Stack\FS lab\ Django Project\fullstack_project> python manage.py migrate
Operations to perform:
 Apply all migrations: admin, auth, contenttypes, sessions, student_course_registration_app
Running migrations:
 Applying student_course_registration_app.0002_remove_student_name_student_courses_and_more... 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 3.0.4, 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**



Courses

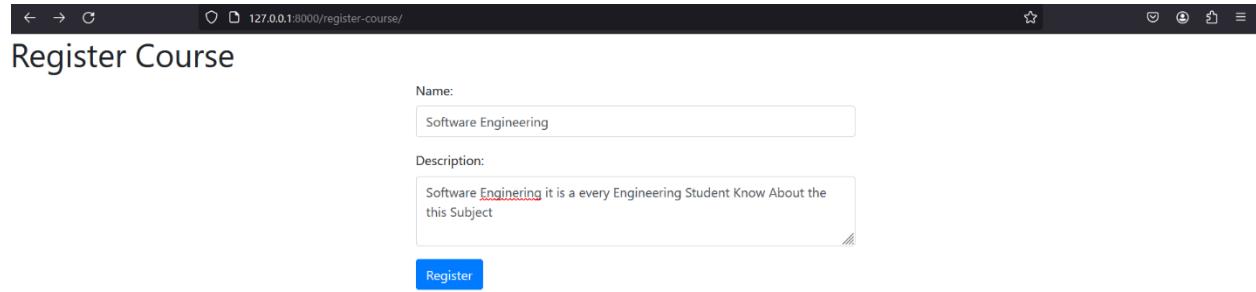
DBMS
CN

Register Student Register Course

**Fig: Index Screen**



**Fig: Student Register Screen**



A screenshot of a web browser showing a course registration form. The URL in the address bar is 127.0.0.1:8000/register-course/. The page title is "Register Course". There are two input fields: "Name:" containing "Software Engineering" and "Description:" containing "Software Engineering it is a every Engineering Student Know About the this Subject". A blue "Register" button is at the bottom.

**Fig: Course Register Screen**



A screenshot of a web browser showing a list of registered students for a course. The URL in the address bar is 127.0.0.1:8000/student-list/1/. The page title is "Students Registered for DBMS". It lists two students: "Hanumanthu hanu" and "Hanumanthu Hanu".

**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.

### 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):

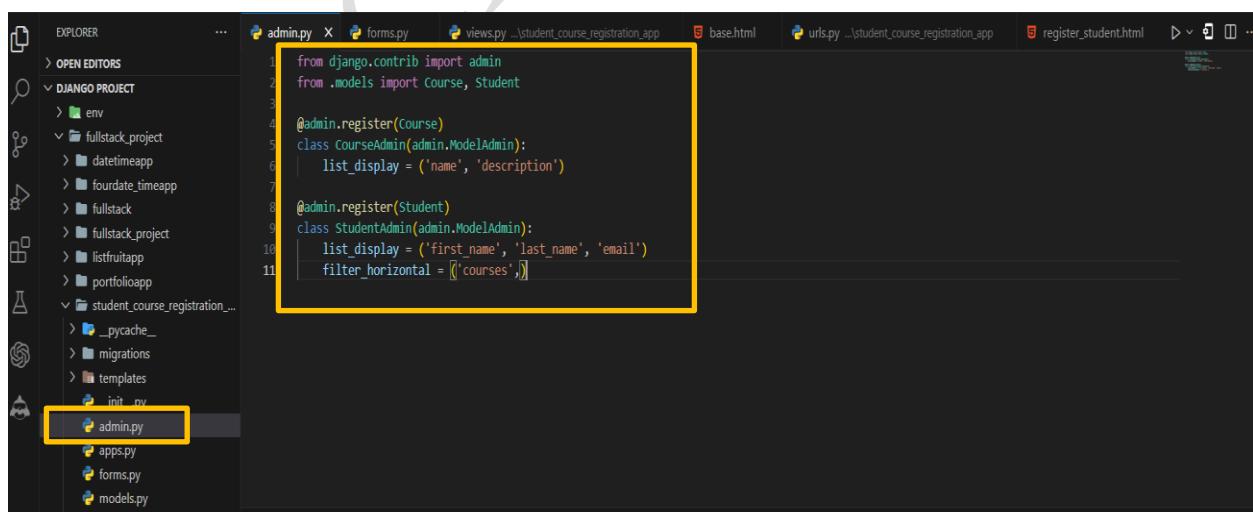
 list_display = ('name', 'description')

@admin.register(Student)

class StudentAdmin(admin.ModelAdmin):

 list_display = ('first_name', 'last_name', 'email')

 filter_horizontal = ('courses',)
```



```
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',)
```

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

- 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`

```
(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**

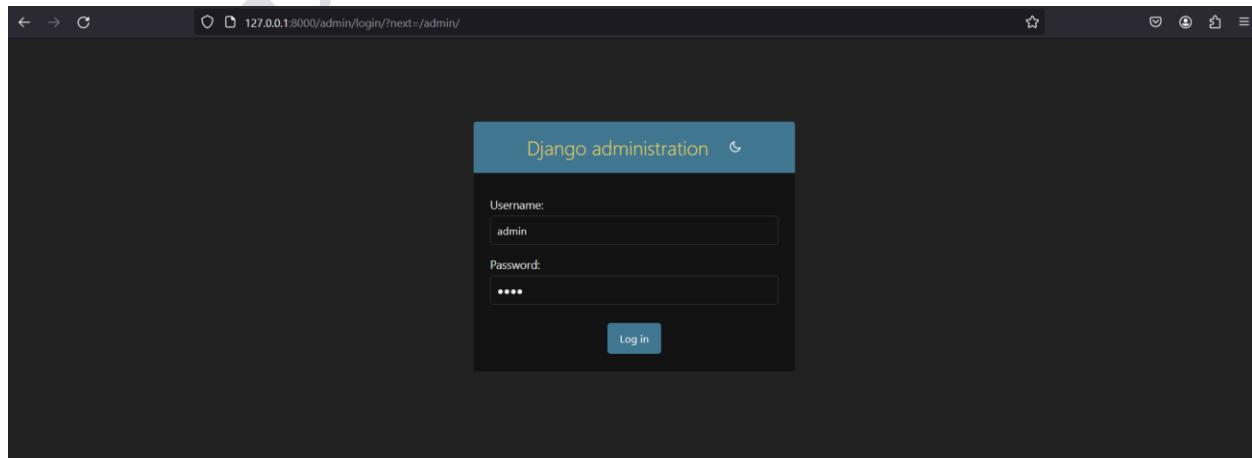
- Open your web browser and visit <http://127.0.0.1:8000/>.

A screenshot of the VS Code terminal window. The tab bar at the top shows 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', and 'PORTS'. The 'TERMINAL' tab is active, displaying the command 'python manage.py runserver' and its output. The output shows the server is watching for file changes with StatReloader, performing system checks, and starting the development server at <http://127.0.0.1:8000/>. A yellow box highlights the URL in the output.

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

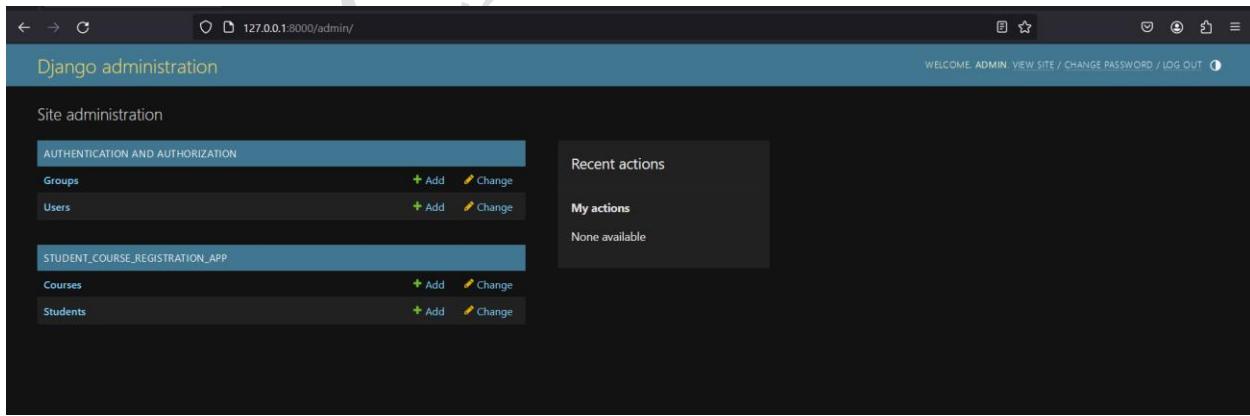
Next, click on "Students" in the left sidebar to add a new student.

- 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 Main Screen**

The screenshot shows the Django Admin interface for the 'Courses' model. The left sidebar lists 'Groups' and 'Users' under 'AUTHENTICATION AND AUTHORIZATION', and 'Courses' and 'Students' under 'STUDENT COURSE REGISTRATION APP'. The main area is titled 'Select course to change' and shows a table with four entries:

	NAME	DESCRIPTION
<input type="checkbox"/>	Software Engineering	Software Engineering it is a every Engineering Student Know About the this Subject
<input type="checkbox"/>	Software Engineering	Software Engineering it is a every Engineering Student Know About the this Subject
<input type="checkbox"/>	CN	nice
<input type="checkbox"/>	DBMS	It is Nice Code Version

Below the table, it says '4 courses'. There are 'ADD COURSE' and 'ADD STUDENT' buttons at the top right.

**Fig: Admin Course Screen**

The screenshot shows the Django Admin interface for the 'Students' model. The left sidebar lists 'Groups' and 'Users' under 'AUTHENTICATION AND AUTHORIZATION', and 'Courses' and 'Students' under 'STUDENT COURSE REGISTRATION APP'. The main area is titled 'Select student to change' and shows a table with three entries:

	FIRST NAME	LAST NAME	EMAIL
<input type="checkbox"/>	Hanumanthu	Hanu	hanu@gmail.com
<input type="checkbox"/>	asdf	difad	shivu@gmail.com
<input type="checkbox"/>	Hanumanthu	hanu	abc@gmail.com

Below the table, it says '3 students'. There are 'ADD COURSE' and 'ADD STUDENT' buttons at the top right.

**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):

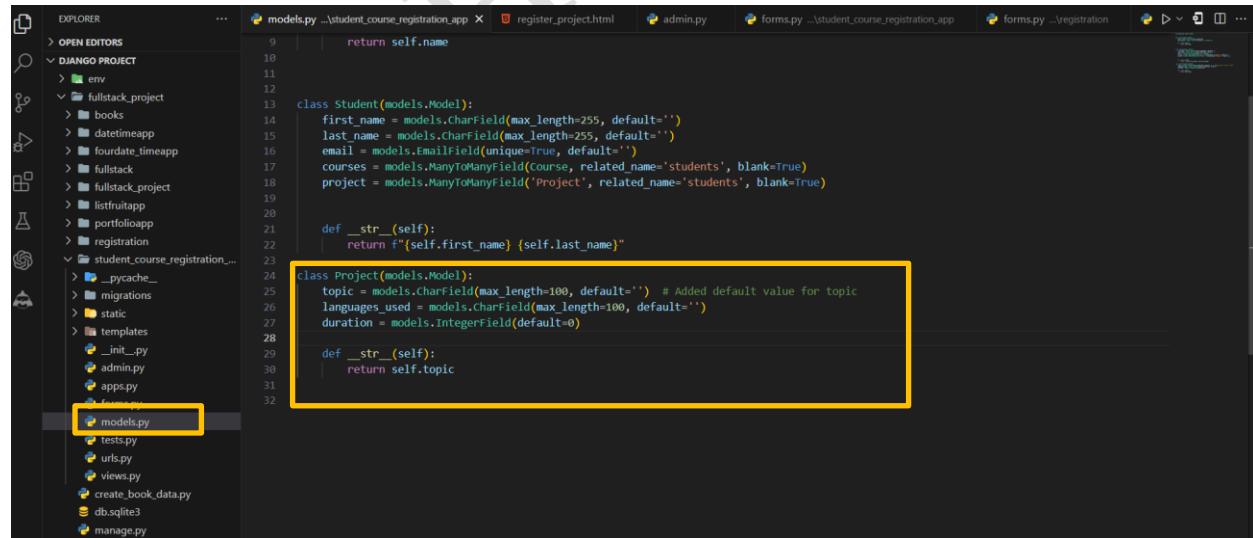
```
topic = models.CharField(max_length=100, default="") # Added default value for topic
```

```
languages_used = models.CharField(max_length=100, default="")
```

```
duration = models.IntegerField(default=0)
```

```
def __str__(self):
```

```
 return self.topic
```



```
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
```

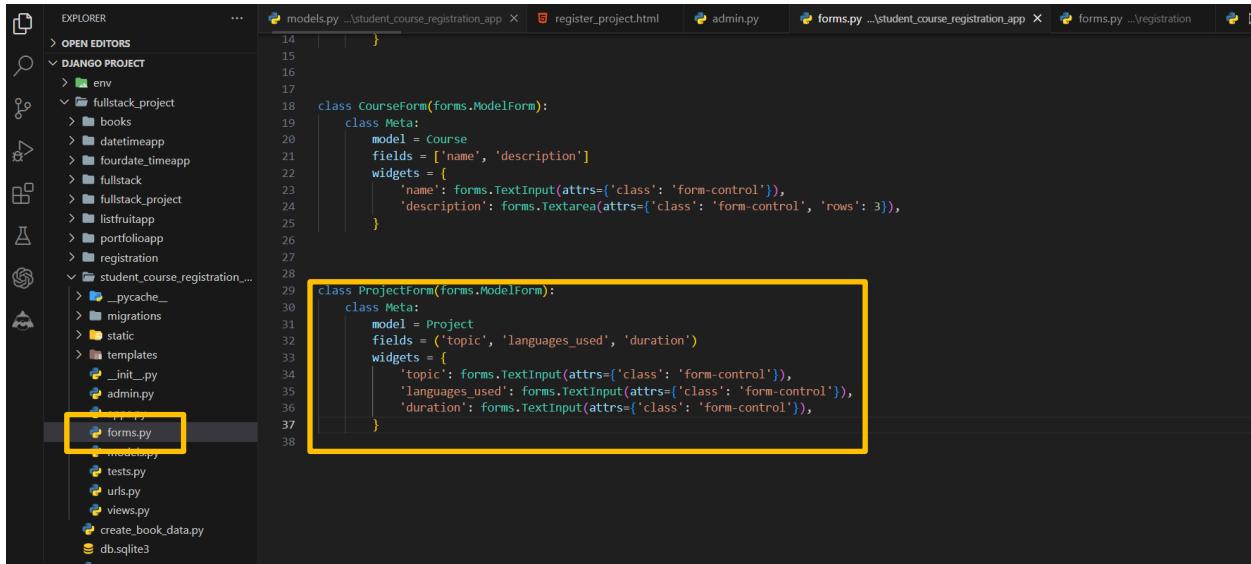
**python manage.py migrate**

```
(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'}),
 }
```



```

models.py ...\\student_course_registration_app X register_project.html admin.py forms.py ...\\student_course_registration_app X forms.py ...\\registration ...
14
15
16
17
18 class CourseForm(forms.ModelForm):
19 class Meta:
20 model = Course
21 fields = ['name', 'description']
22 widgets = {
23 'name': forms.TextInput(attrs={'class': 'form-control'}),
24 'description': forms.Textarea(attrs={'class': 'form-control', 'rows': 3}),
25 }
26
27
28
29 class ProjectForm(forms.ModelForm):
30 class Meta:
31 model = Project
32 fields = ('topic', 'languages_used', 'duration')
33 widgets = {
34 'topic': forms.TextInput(attrs={'class': 'form-control'}),
35 'languages_used': forms.TextInput(attrs={'class': 'form-control'}),
36 'duration': forms.TextInput(attrs={'class': 'form-control'})
37 }
38

```

**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})
```

```

OPEN EDITORS
DJANGO PROJECT
 > env
 > fullstack_project
 > books
 > datetimeapp
 > foudrate_timeapp
 > fullstack
 > listfruitapp
 > portfolioapp
 > registration
 > student_course_registration...
 > __pycache__
 > migrations
 > static
 > templates
 > __init__.py
 > admin.py
 > forms.py
 > models.py
 > tests.py
 > urls.py
 > views.py
 > wsgi.py
 db.sqlite3

ml admin.py forms.py ...student_course_registration_app forms.py ...registration forms.py ...registration urls.py ...registration views.py ...student_course_registration_app

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})

def project_detail(request, project_id):
 project = get_object_or_404(Project, pk=project_id)
 return render(request, 'registration/project_details.html', {'project': project})

def project_student_list(request, project_id):
 project = Project.objects.get(id=project_id)
 students = project.students.all()
 return render(request, 'registration/student_list.html', {'students': students, 'project': project})

```

**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,
})

```

```

OPEN EDITORS
DJANGO PROJECT
 > env
 > fullstack_project
 > books
 > datetimeapp
 > foudrate_timeapp
 > fullstack
 > listfruitapp
 > portfolioapp
 > registration
 > student_course_registration...
 > __pycache__
 > migrations
 > static
 > templates
 > __init__.py
 > admin.py
 > forms.py
 > models.py
 > tests.py
 > urls.py
 > views.py
 db.sqlite3

ml admin.py forms.py ...student_course_registration_app forms.py ...registration forms.py ...registration urls.py ...registration views.py ...student_course_registration_app

from django.shortcuts import render, get_object_or_404
from .models import Course, Student, Project
from .forms import StudentForm, CourseForm, ProjectForm
from django.views.generic import ListView, DetailView
from django.http import JsonResponse
from django.views.decorators.csrf import csrf_exempt

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

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})

```

**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' % }

{ % 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>

 {{ form.duration }}

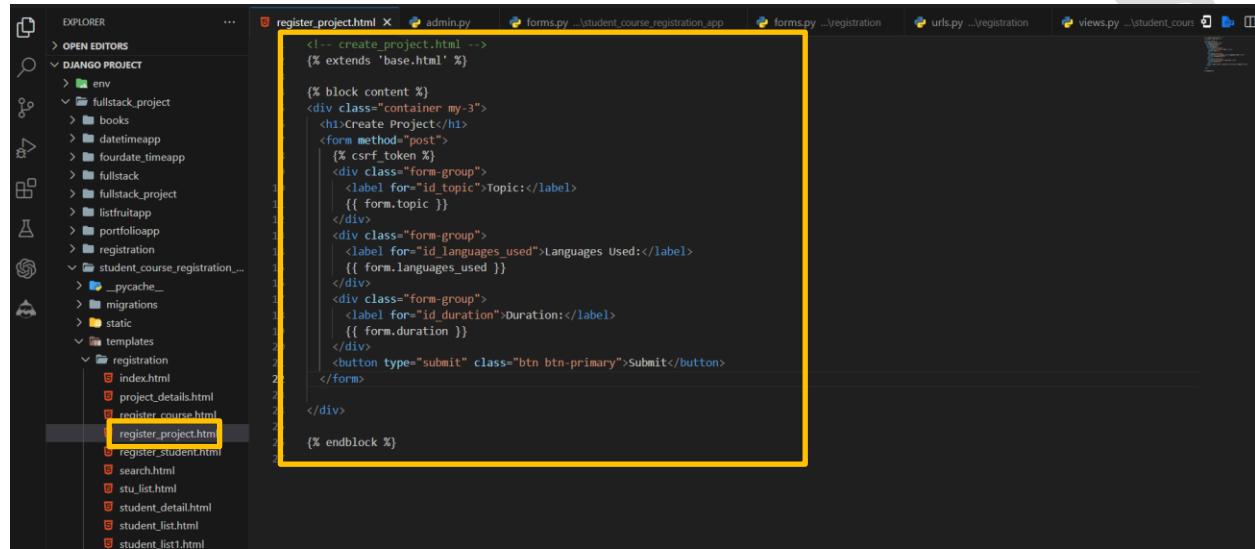
 </div>
```

```
<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 %}

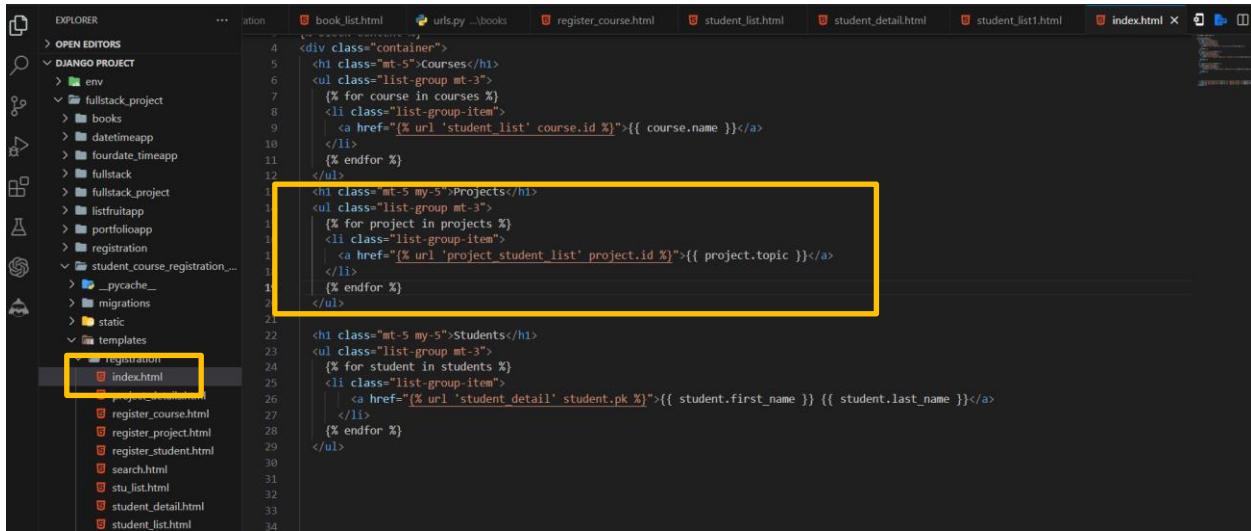
 <li class="list-group-item">

 {{ project.topic }}

 {% endfor %}


```

# 21CS62 | Full Stack Django Development|



```
<div class="container">
<h1 class="mt-5">Courses</h1>
<ul class="list-group mt-3">
{% for course in courses %}
<li class="list-group-item">
| {{ course.name }}

{% endfor %}

<h1 class="mt-5 my-5">Projects</h1>
<ul class="list-group mt-3">
{% for project in projects %}
<li class="list-group-item">
| {{ project.topic }}

{% endfor %}

<h1 class="mt-5 my-5">Students</h1>
<ul class="list-group mt-3">
{% for student in students %}
<li class="list-group-item">
| {{ student.first_name }} {{ student.last_name }}

{% endfor %}

```

- Add the **register\_student.html** to this is Student can Register the Created Project

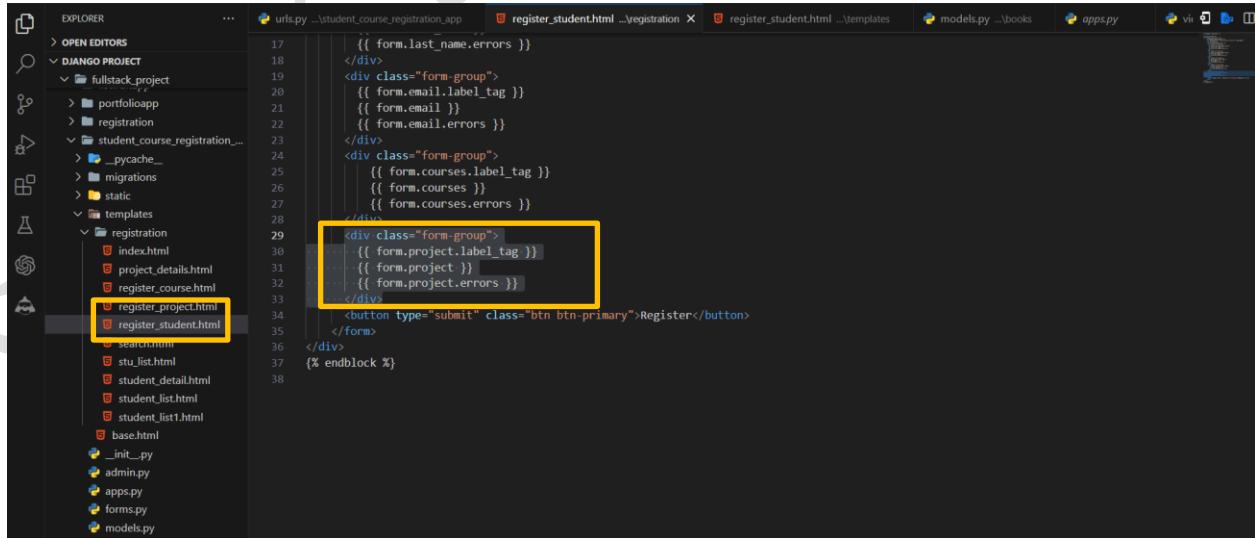
```
<div class="form-group">

{{ form.project.label_tag }}

{{ form.project }}

{{ form.project.errors }}

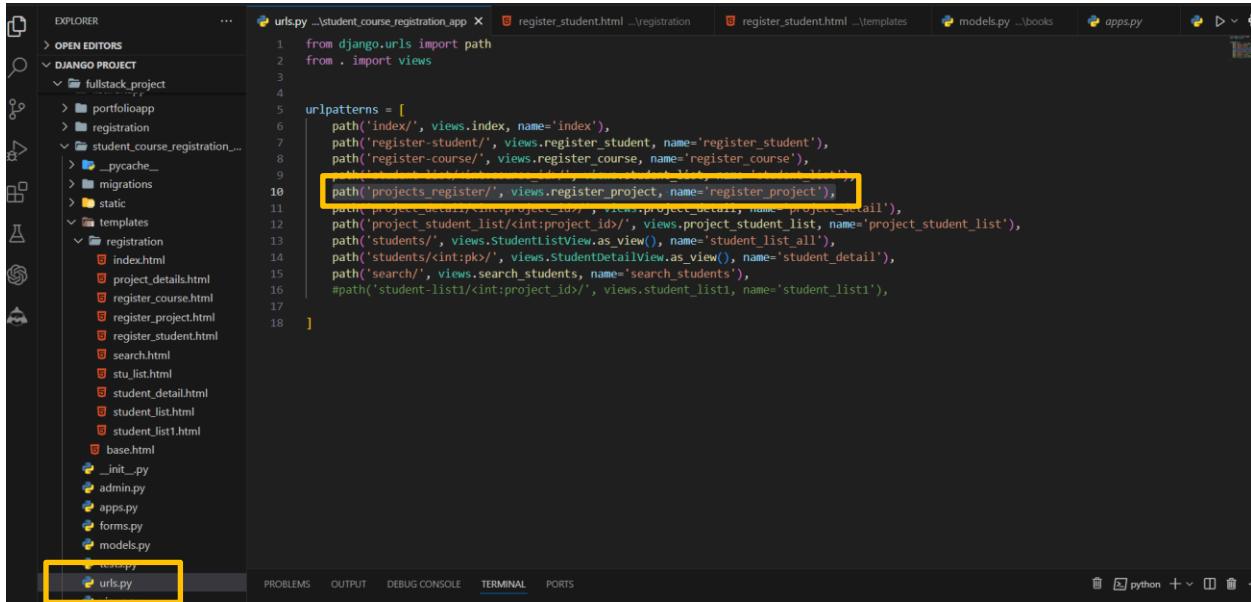
</div>
```



```
<div class="form-group">
{{ form.last_name.errors }}
</div>
<div class="form-group">
{{ form.email.label_tag }}
{{ form.email }}
{{ form.email.errors }}
</div>
<div class="form-group">
{{ form.courses.label_tag }}
{{ form.courses }}
{{ form.courses.errors }}
</div>
<div class="form-group">
{{ form.project.label_tag }}
{{ form.project }}
{{ form.project.errors }}
</div>
<button type="submit" class="btn btn-primary">Register</button>
</form>
</div>
{% endblock %}
```

## Step 9: Add a URL pattern for the register\_project in your urls.py file

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



```

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('projects_register/', views.register_project, name='register_project'),
 path('project_student_list/<int:project_id>', views.project_student_list, name='project_student_list'),
 path('students/', views.StudentListView.as_view(), name='student_list_all'),
 path('students/<int:pk>', views.StudentDetailView.as_view(), name='student_detail'),
 path('search/', views.search_students, name='search_students'),
 #path('student-list1/<int:project_id>', views.student_list1, name='student_list1'),
]

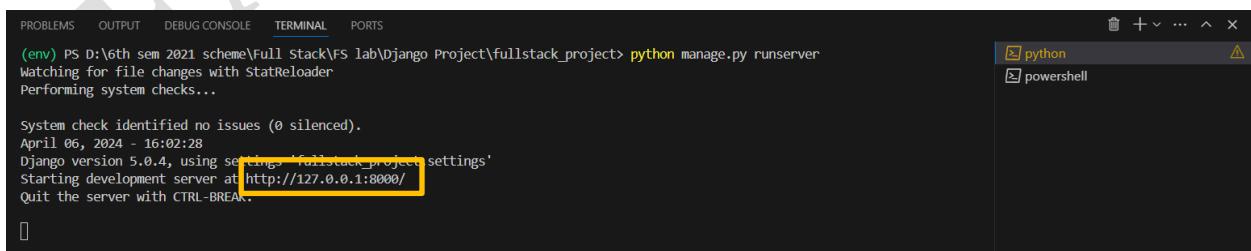
```

## 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/>.



```

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...

```

- Type or copy this <http://127.0.0.1:8000/index/>

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

---

#### Projects

Covide vaccine
child Vaccine System

#### Students

Hanumanthu hanu
asfd dsfad
Hanumanthu Hanu
hh df
hanumanthu hanu
hello how
gangu gangu
nagesh nagu

[Register Student](#) [Register Course](#) [Register Project](#)

**Fig: Home Page Screen**

### Create Project

Topic:

Login App

Languages Used:

html css python

Duration:

3

**Submit**

**Fig: Project Creation Screen**

Software Engineering

Software Engineering

### Projects

Covide vaccine

child Vaccine System

Login App

**Fig: Project Successfully Creation Screen**

### Student Registration

First name:

Last name:

Email:

Courses:

DBMS

CN

Software Engineering

Software Engineering

Project:

Covide vaccine

child Vaccine System

Login App

**Fig: Student Register the Project**

---

### Students Registered for

- Demo demo

**Fig: Student Registered Particular Projects**

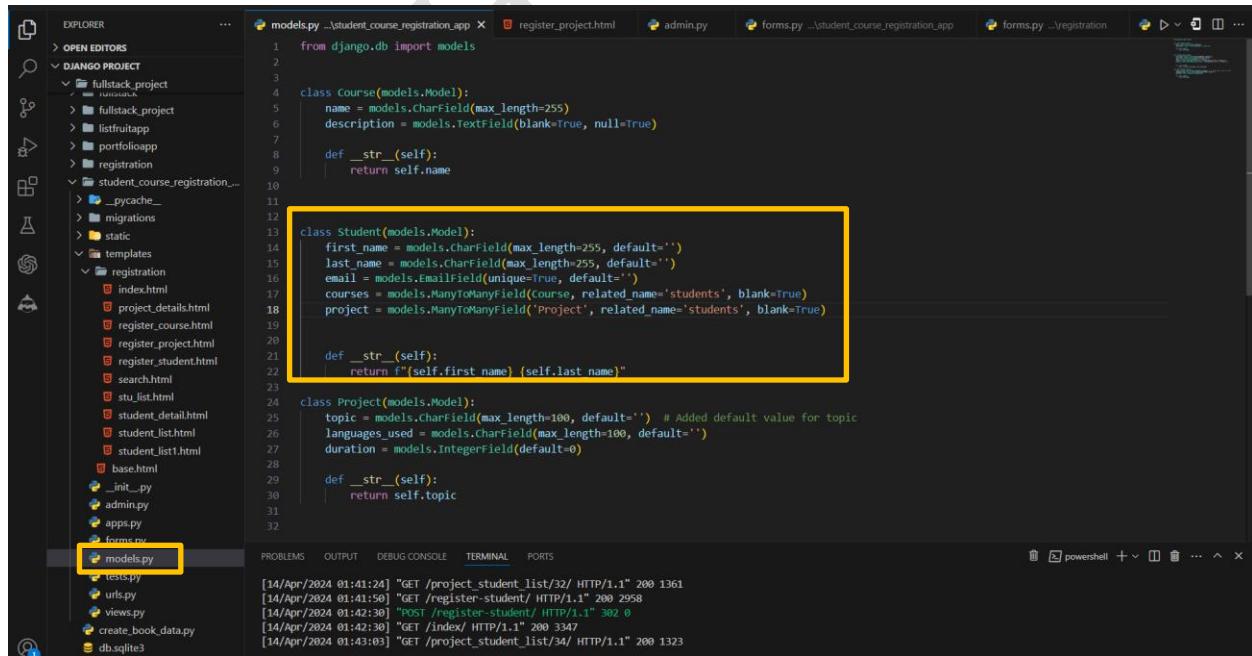
## Experiment-10

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)
```



```
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)
 project = models.ManyToManyField('Project', related_name='students', blank=True)

 def __str__(self):
 return f'{self.first_name} {self.last_name}'

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-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'
```

The screenshot shows a code editor interface with the following details:

- EXPLORER** sidebar: Shows the project structure. The `student_course_registration` app contains `__pycache__, migrations, static, templates, registration, forms.py, models.py, tests.py, urls.py, views.py`. The `registration` folder contains `index.html, project_details.html, register_course.html, register_project.html, register_student.html, search.html, stu_list.html, student_detail.html, student_list.html, student_list1.html, base.html, __init__.py, admin.py, apps.py, forms.py, models.py, tests.py, urls.py, views.py`.
- CODE** tab: Displays `views.py` content. A yellow box highlights the `ListView` and `DetailView` classes.
- OUTPUT** tab: Shows terminal logs with several "INFO" messages related to the application's startup and database queries.

```
64
65
66 def project_student_list(request, project_id):
67 project = Project.objects.get(id=project_id)
68 students = project.students.all()
69 return render(request, 'registration/student_list.html', {'students': students, 'project': project})
70
71
72 def student_list_new(request):
73 students = Student.objects.all()
74 return render(request, 'registration/stu_list.html', {'students': students})
75
76 class StudentListView(ListView):
77 model = Student
78 template_name = 'registration/stu_list.html'
79 context_object_name = 'students'
80
81 class StudentDetailView(DetailView):
82 model = Student
83 template_name = 'registration/student_detail.html'
84 context_object_name = 'student'
85
86
87 from django.shortcuts import render
88 from django.http import JsonResponse
89 from .models import Student
90
91 def search_students(request):
92 if 'X-Requested-With' in request.headers and request.headers['X-Requested-With'] == 'XMLHttpRequest':
93 if request.is_ajax():
94 query = request.GET.get('query', '')
95 students = Student.objects.filter(first_name__icontains=query) | Student.objects.filter(last_name__icontains=query)
96 results = []
97
98
99
100 PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
101
102 [14/Apr/2024 01:41:24] "GET /project_student_list/32/ HTTP/1.1" 200 1361
103 [14/Apr/2024 01:41:50] "GET /register-student/ HTTP/1.1" 200 2958
104 [14/Apr/2024 01:42:30] "POST /register-student/ HTTP/1.1" 302 6
105 [14/Apr/2024 01:42:30] "GET /index/ HTTP/1.1" 200 3347
106 [14/Apr/2024 01:43:03] "GET /project_student_list/34/ HTTP/1.1" 200 1323
```

- 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 %}

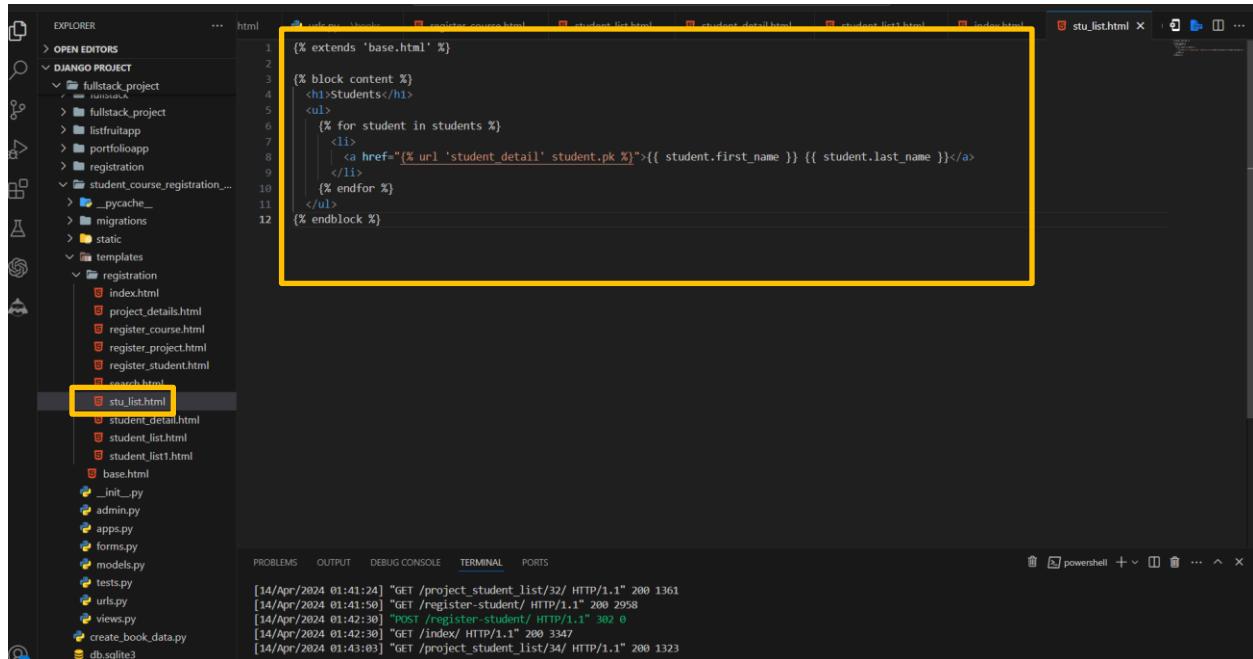
<h1>Students</h1>

 {% for student in students %}

 {{ student.first_name }} {{ student.last_name }}

 {% endfor %}

{% endblock %}
```



The screenshot shows the VS Code interface with the Django project structure in the Explorer sidebar. The current file being edited is `stu_list.html`, which is highlighted with a yellow box. The code in the editor is:

```

1 {% extends 'base.html' %}
2
3 {% block content %}
4 <h1>Students</h1>
5
6 {% for student in students %}
7
8 {{ student.first_name }} {{ student.last_name }}
9
10 {% endfor %}
11
12 {% endblock %}

```

The terminal at the bottom shows the command line history:

```

[14/Apr/2024 01:41:24] "GET /project_student_list/32/ HTTP/1.1" 200 1361
[14/Apr/2024 01:41:50] "GET /register-student/ HTTP/1.1" 200 2958
[14/Apr/2024 01:42:30] "POST /register-student/ HTTP/1.1" 302 0
[14/Apr/2024 01:42:30] "GET /index/ HTTP/1.1" 200 3347
[14/Apr/2024 01:43:03] "GET /project_student_list/34/ HTTP/1.1" 200 1323

```

## 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>

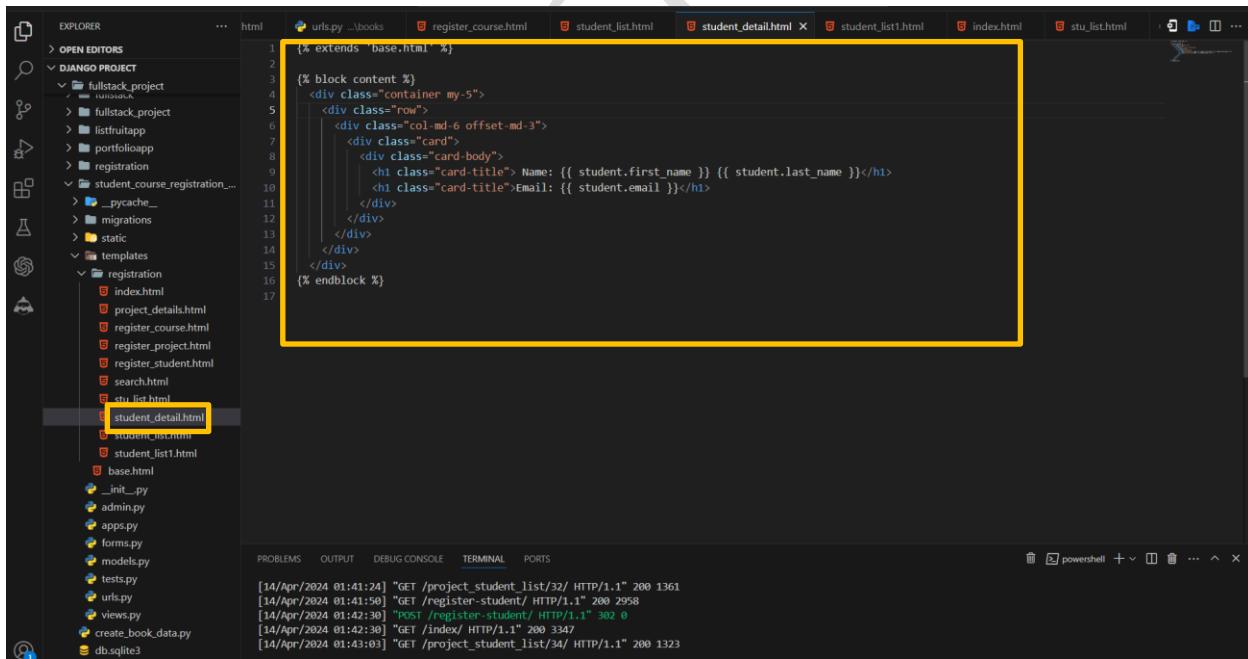
```

```
</div>

</div>

</div>

{% endblock %}
```



The screenshot shows a code editor interface with the following details:

- EXPLORER:** Shows the project structure under "DJANGO PROJECT".
- OPEN EDITORS:** Shows several files: urls.py, register\_course.html, student\_list.html, student\_detail.html (highlighted with a yellow box), student\_list1.html, index.html, stu\_list.html, and student\_detail.html.
- CODE EDITOR:** Displays the content of the student\_detail.html template file. The code is as follows:

```
{% 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>
{% endblock %}
```

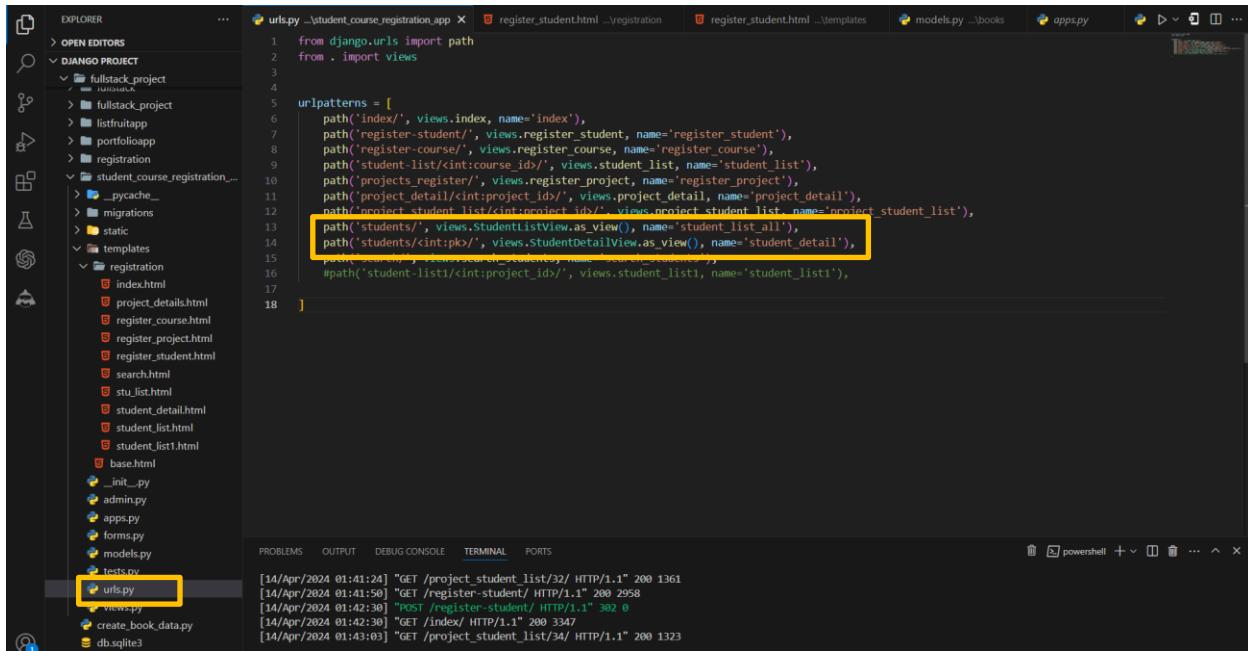
**PROBLEMS:** Shows log entries from the terminal:

```
[14/Apr/2024 01:41:24] "GET /project_student_list/32/ HTTP/1.1" 200 1361
[14/Apr/2024 01:41:50] "GET /register-student/ HTTP/1.1" 200 2958
[14/Apr/2024 01:42:30] "POST /register-student/ HTTP/1.1" 302 0
[14/Apr/2024 01:42:30] "GET /index/ HTTP/1.1" 200 3347
[14/Apr/2024 01:43:03] "GET /project_student_list/34/ HTTP/1.1" 200 1323
```

## 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'),
```



```

from django.urls import path
from . import views

urlpatterns = [
 path('/', 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'),
 path('projects/register/', views.register_project, name='register_project'),
 path('project/detail/<int:project_id>', views.project_detail, name='project_detail'),
 path('project/student/list/<int:project_id>', views.project_student_list, name='project_student_list'),
 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">

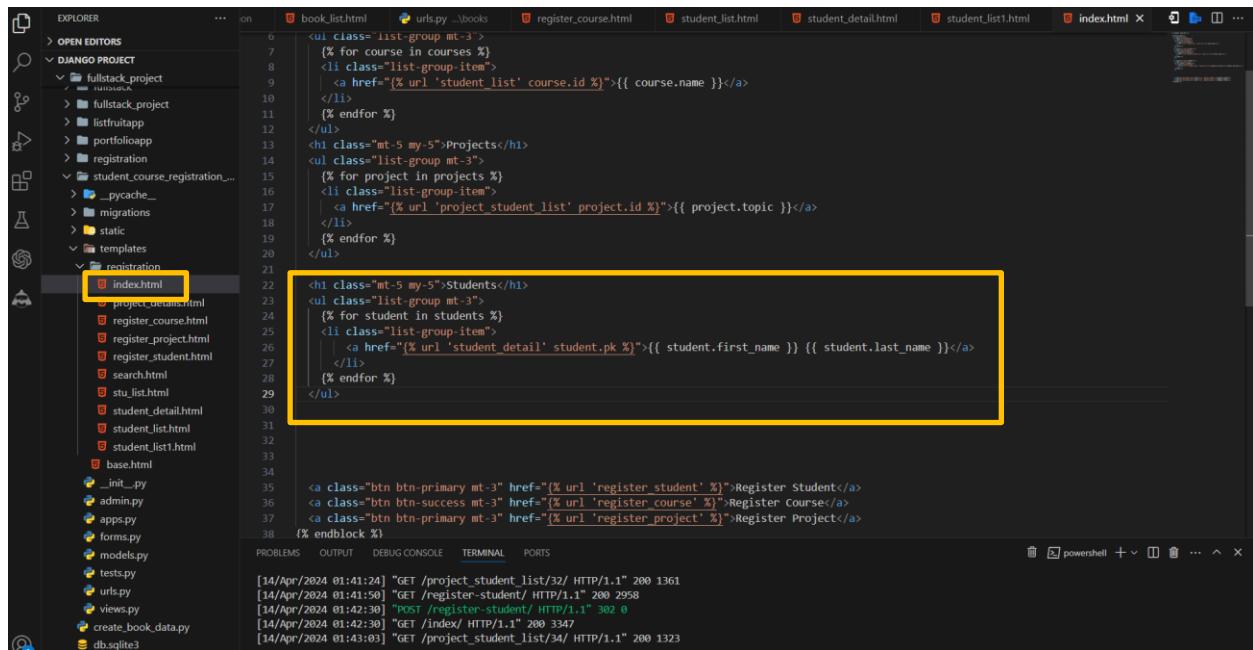
 {% for student in students %}

 <li class="list-group-item">
 {{ student.first_name }} {{ student.last_name }}

 {% endfor %}


```

</ul>



```

<ul class="list-group mt-3">
 {% for course in courses %}
 <li class="list-group-item">
 {{ course.name }}

 {% endfor %}

<h1 class="mt-5 my-5">Projects</h1>
<ul class="list-group mt-3">
 {% for project in projects %}
 <li class="list-group-item">
 {{ project.topic }}

 {% endfor %}

<h1 class="mt-5 my-5">Students</h1>
<ul class="list-group mt-3">
 {% for student in students %}
 <li class="list-group-item">
 {{ student.first_name }} {{ student.last_name }}

 {% endfor %}

Register Student
Register Course
Register Project
{% endblock %}

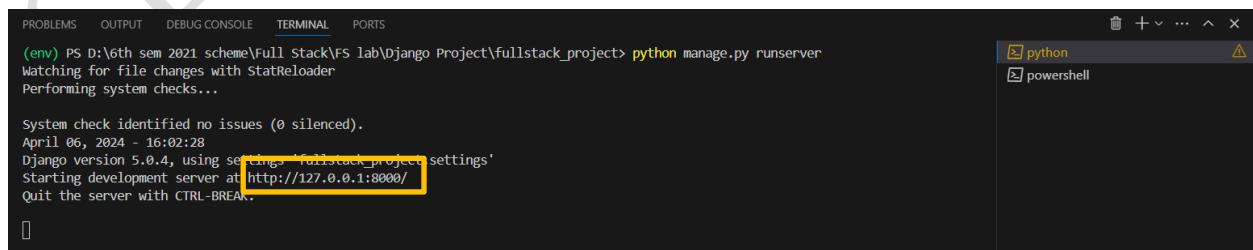
```

## 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/>.



```

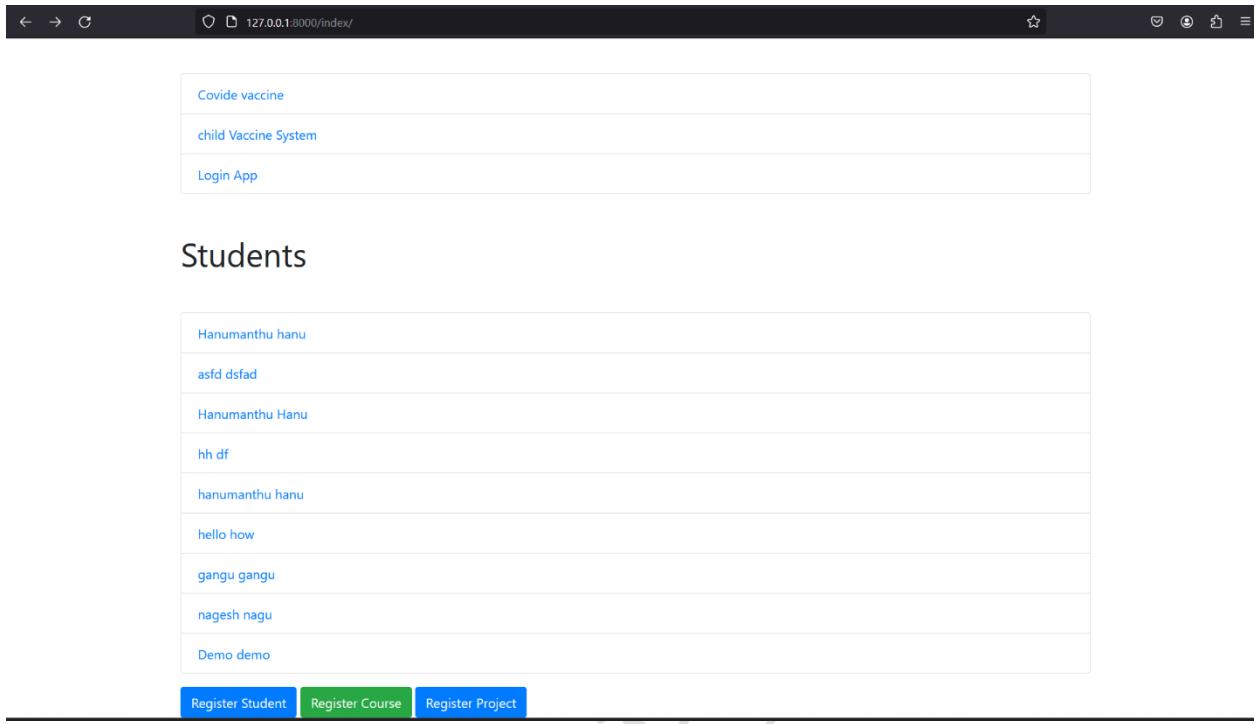
PS D:\6th sem 2021 scheme\Full StackFS lab\ Django Project\fullstack_project> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

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.

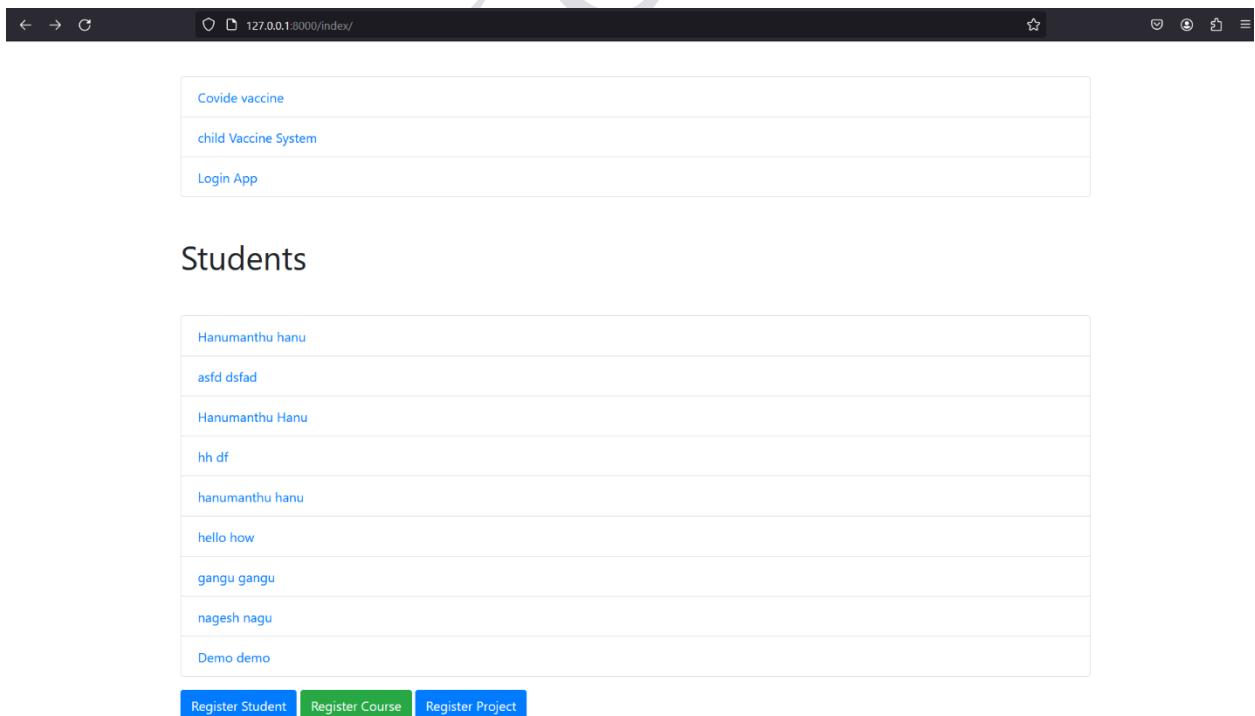
```

- Type or copy this <http://127.0.0.1:8000/index/>

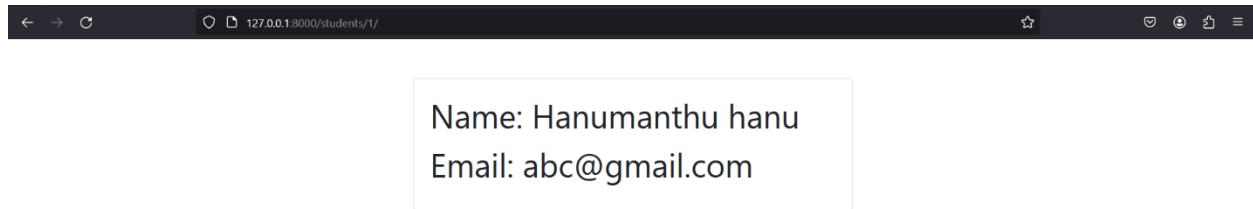
## Final Output By Clicking Register Project and Register Student



**Fig: Main Screen**



**Fig: List View of Students Screen**



**Fig: Detailed View of Student Screen**

SEARCH CREATORS

## 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.

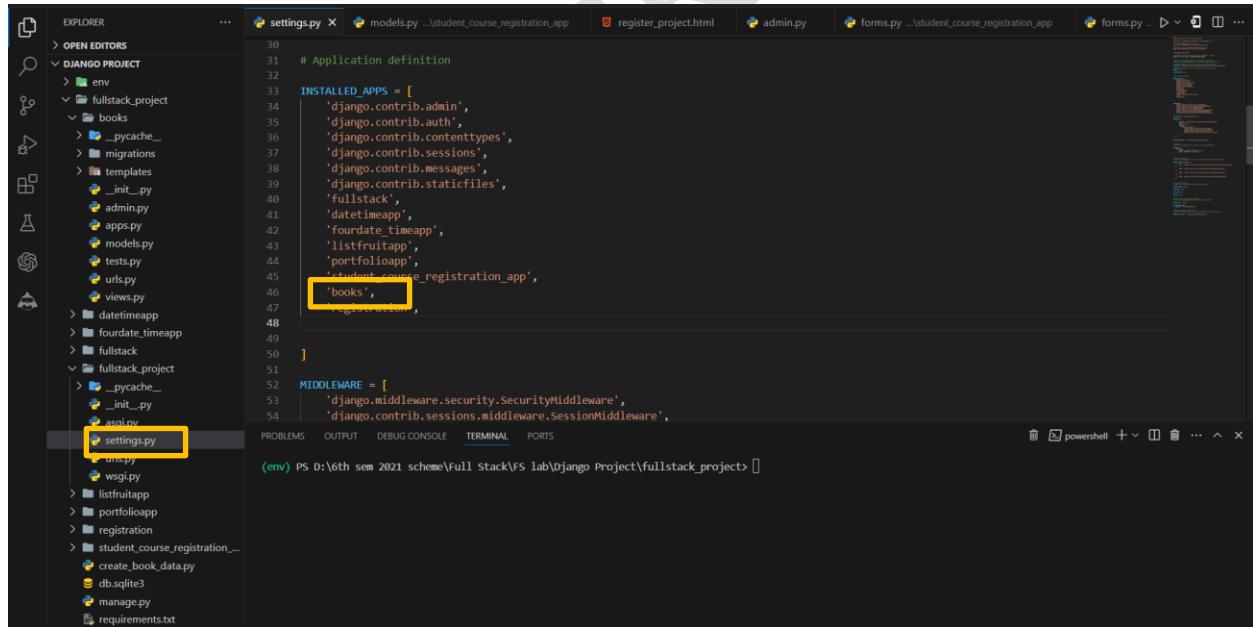


```
(env) PS D:\6th sem 2021 scheme\Full Stack\FS lab\ Django Project\fullstack_project> django-admin startapp books
```

**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:



```
30 # Application definition
31
32 INSTALLED_APPS = [
33 'django.contrib.admin',
34 'django.contrib.auth',
35 'django.contrib.contenttypes',
36 'django.contrib.sessions',
37 'django.contrib.messages',
38 'django.contrib.staticfiles',
39 'fullstack',
40 'datetimapp',
41 'foundate_timeapp',
42 'listfruitapp',
43 'portfolioapp',
44 'student_course_registration_app',
45 'books',
46]
47
48 MIDDLEWARE = [
49 'django.middleware.security.SecurityMiddleware',
50 'django.contrib.sessions.middleware.SessionMiddleware',
51 'django.middleware.common.CommonMiddleware',
52 'django.middleware.csrf.CsrfViewMiddleware',
53 'django.middleware.clickjacking.XFrameOptionsMiddleware',
54]
```

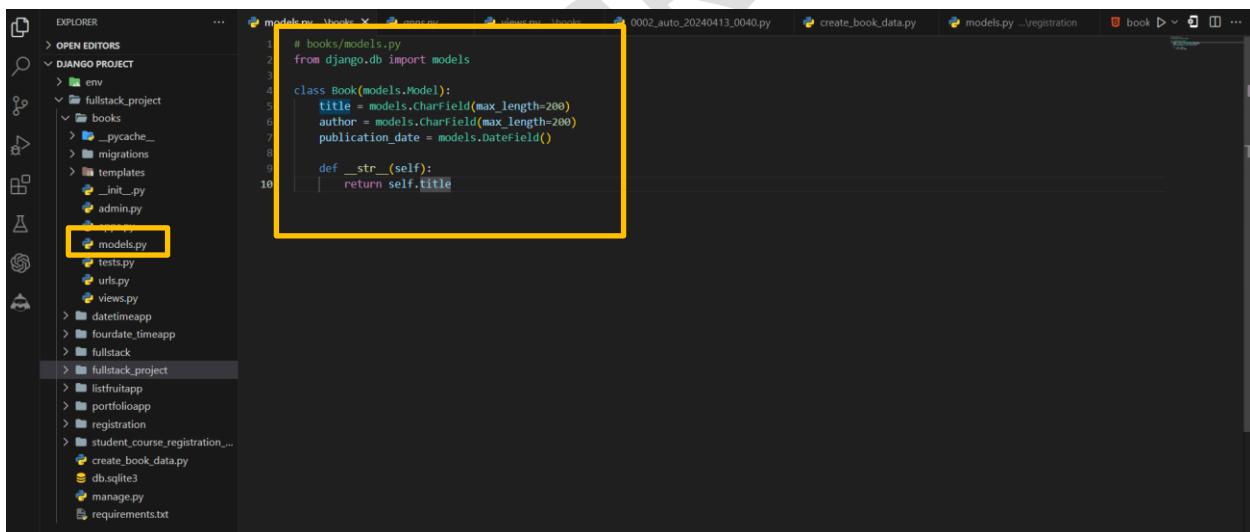
## Step-03: Create models

- 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)

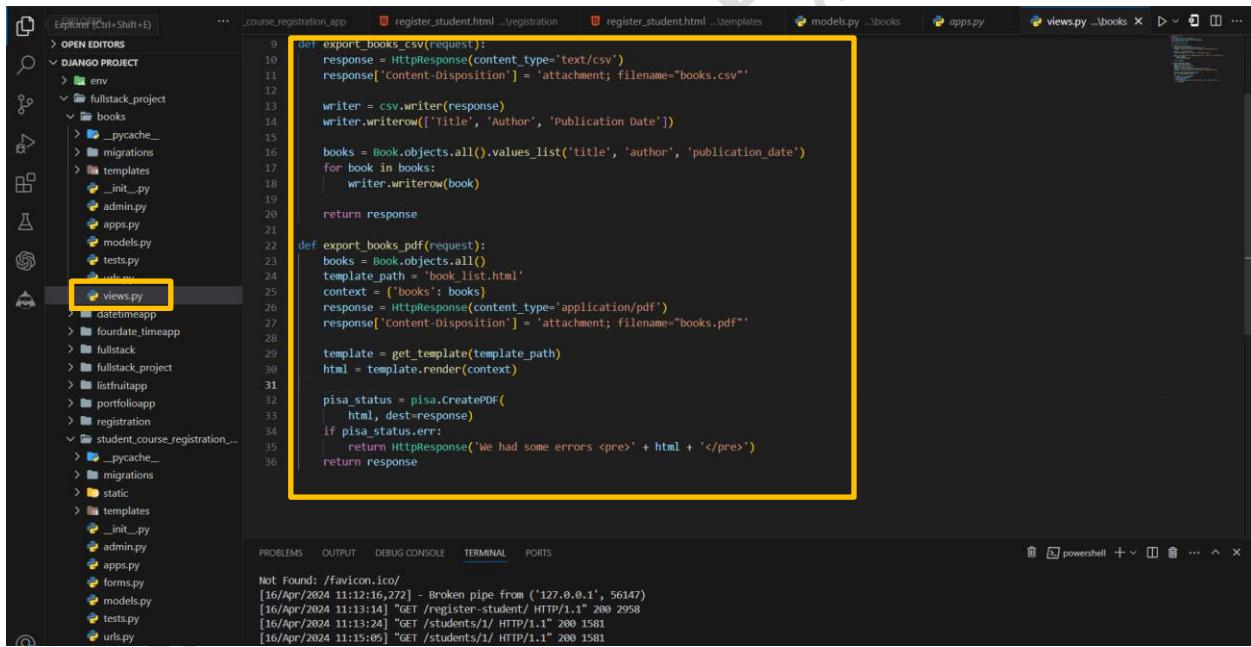
pisa_status = pisa.CreatePDF(

 html, dest=response)

if pisa_status.err:

 return HttpResponse('We had some errors <pre>' + html + '</pre>')

return response
```



The screenshot shows a code editor interface with a yellow box highlighting the section of code that generates both CSV and PDF files. The code uses the `HttpResponse` class with `content\_type` set to 'text/csv' or 'application/pdf'. It also sets the `Content-Disposition` header to 'attachment; filename="books.csv"' or 'books.pdf'. The code then creates a CSV writer for the CSV file and a PDF writer for the PDF file, both using the same template path ('book\_list.html'). The code then returns the response.

```
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 %}
```

```
</tbody>
```

```
</table>
```

```
</body>
```

```
</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 %}
 </tbody>
 </table>
</body>
</html>
```

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

```
from django.urls import path
```

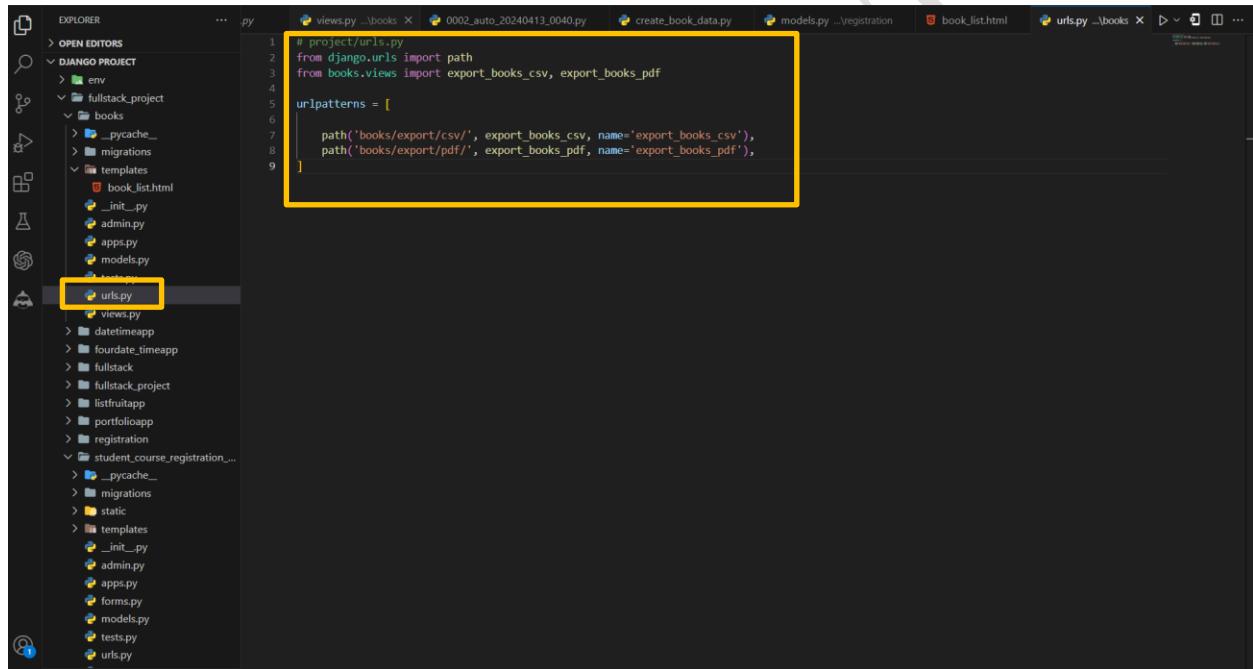
```
from books.views import export_books_csv, export_books_pdf
```

```
urlpatterns = [
```

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

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

```
]
```



The screenshot shows the VS Code interface with the 'urls.py' file open in the center editor tab. The code in the editor is:

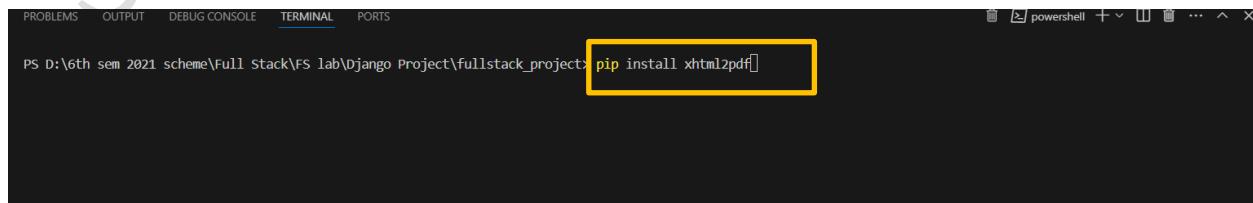
```
project/urls.py
from django.urls import path
from books.views import export_books_csv, export_books_pdf

urlpatterns = [
 path('books/export/csv/', export_books_csv, name='export_books_csv'),
 path('books/export/pdf/', export_books_pdf, name='export_books_pdf'),
]
```

The 'urls.py' file is highlighted in the Explorer sidebar on the left.

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

```
pip install xhtml2pdf
```



The screenshot shows a terminal window with the command 'pip install xhtml2pdf' entered and highlighted with a yellow box. The terminal is located in a dark-themed environment with tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', and 'PORTS'. The command is being run from a directory path: 'PS D:\6th sem 2021 scheme\Full Stack\FS lab\ Django Project\fullstack\_project'.

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

### Django Shell

- 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 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

Run the following command to create a new data migration

```
python manage.py makemigrations books --empty
```

```
from django.db import migrations

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

class Migration(migrations.Migration):

 dependencies = [
 ('books', '0001_initial'),
]

 operations = [
 migrations.RunPython(create_book_data),
]
```

PS D:\6th sem 2021 scheme\Full Stack\FS lab\ Django Project\fullstack\_project python manage.py makemigrations books --empty

- This will create a new empty migration file in the books/migrations/ directory.
- 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')
 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**

```
python manage.py migrate
```

- This will create the book objects in your database.

### 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

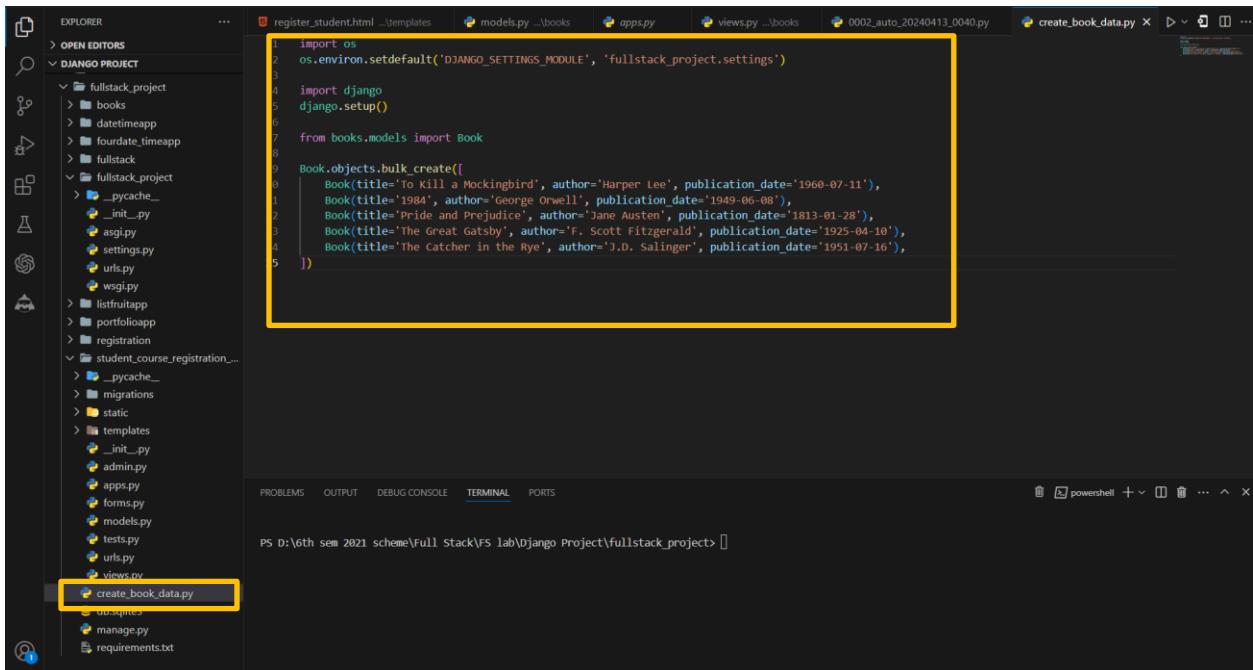
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'fullstack_project.settings')

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'),
])
```



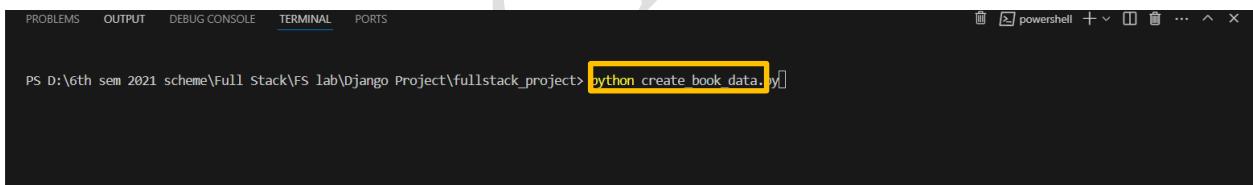
```

1 import os
2 os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'fullstack_project.settings')
3
4 import django
5 django.setup()
6
7 from books.models import Book
8
9 Book.objects.bulk_create([
10 Book(title='To Kill a Mockingbird', author='Harper Lee', publication_date='1960-07-11'),
11 Book(title='1984', author='George Orwell', publication_date='1949-06-08'),
12 Book(title='Pride and Prejudice', author='Jane Austen', publication_date='1813-01-28'),
13 Book(title='The Great Gatsby', author='F. Scott Fitzgerald', publication_date='1925-04-10'),
14 Book(title='The Catcher in the Rye', author='J.D. Salinger', publication_date='1951-07-16'),
15])

```

Run the script using the following command:

`python create_book_data.py`



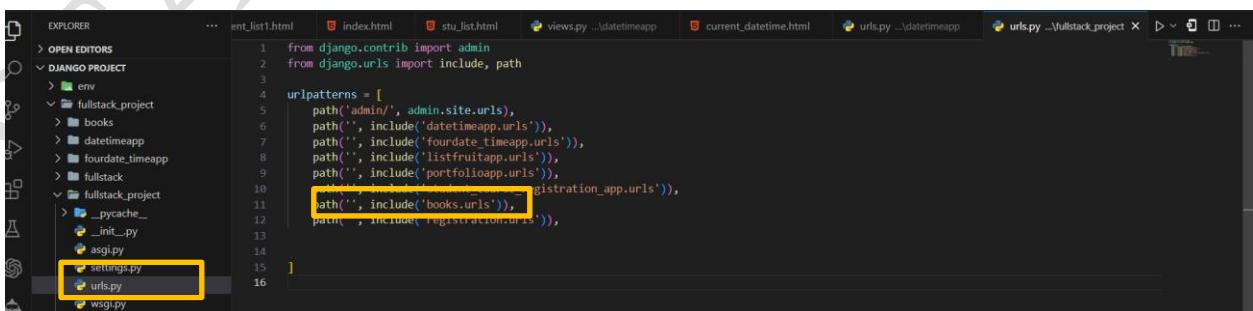
```

PS D:\6th sem 2021 scheme\Full Stack\Fs lab\ Django Project\fullstack_project> 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:



```

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

urlpatterns = [
 path('admin/', admin.site.urls),
 path('', include('datetimeapp.urls')),
 path('', include('foundate_timeapp.urls')),
 path('', include('listfruitapp.urls')),
 path('', include('portfolioapp.urls')),
 path('', include('registration_app.urls')),
 path('', include('books.urls')),
 path('', include('registration_app.urls')),
]

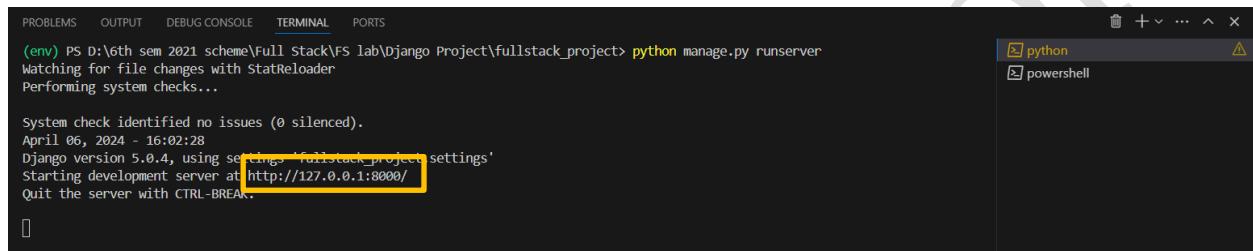
```

## 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/>.



```
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.
```

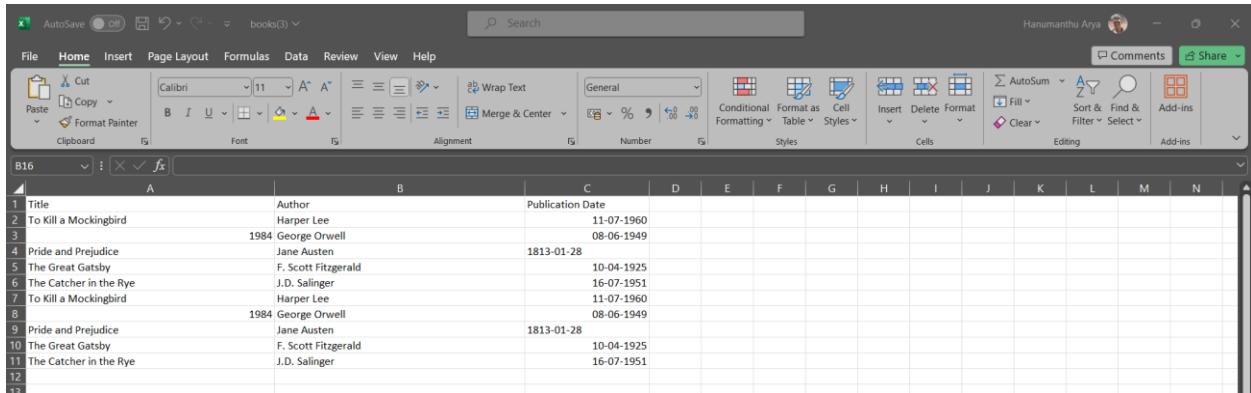
- Type or copy this <http://127.0.0.1:8000/books/export/csv/>

## Final Output By Clicking Register Project and Register Student



**Fig: CSV File Download Screen**

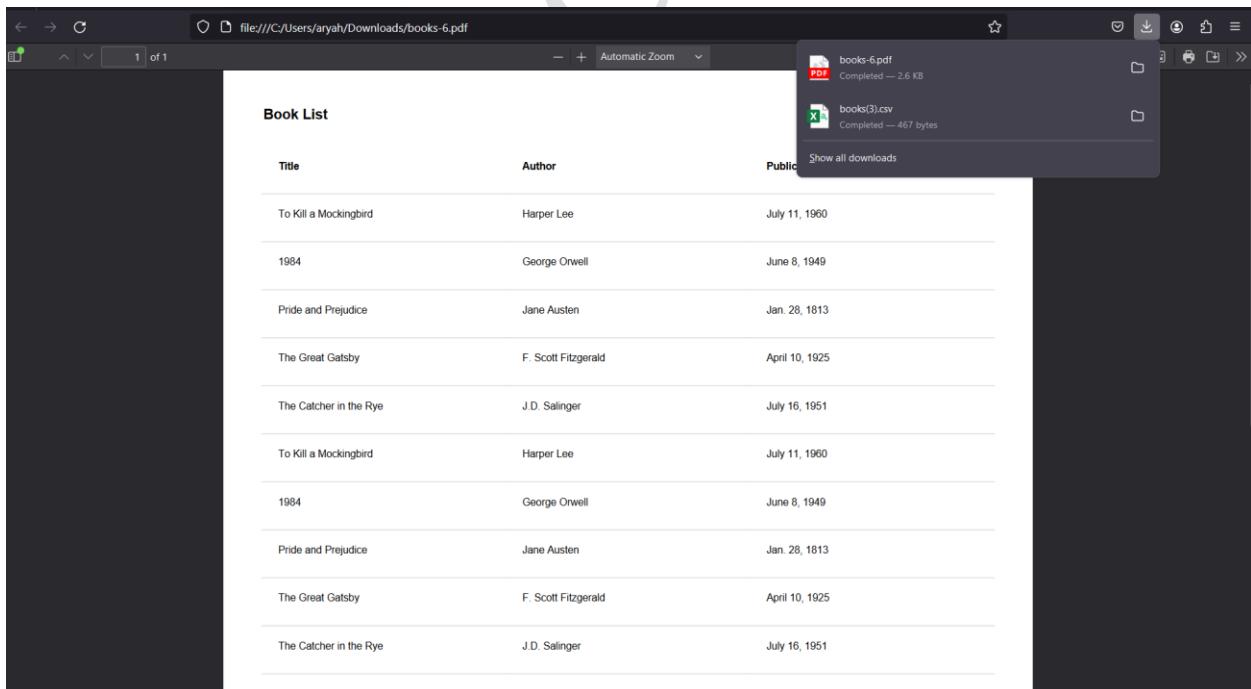
## 21CS62 | Full Stack Django Development|



A screenshot of Microsoft Excel showing a table of book data. The table has three columns: Title, Author, and Publication Date. The data includes multiple entries for books like "To Kill a Mockingbird" and "The Great Gatsby".

	Title	Author	Publication Date
1	To Kill a Mockingbird	Harper Lee	11-07-1960
2	1984	George Orwell	08-06-1949
3	Pride and Prejudice	Jane Austen	1813-01-28
4	The Great Gatsby	F. Scott Fitzgerald	10-04-1925
5	The Catcher in the Rye	J.D. Salinger	16-07-1951
6	To Kill a Mockingbird	Harper Lee	11-07-1960
7	1984	George Orwell	08-06-1949
8	Pride and Prejudice	Jane Austen	1813-01-28
9	The Great Gatsby	F. Scott Fitzgerald	10-04-1925
10	The Catcher in the Rye	J.D. Salinger	16-07-1951
11			
12			
13			

- Type or copy this <http://127.0.0.1:8000/books/export/pdf/>



A screenshot of a browser window showing a PDF file download screen. The page title is "Book List". The table data is identical to the one in the Excel screenshot. A download dialog box is open, showing two files: "books-6.pdf" (Completed — 2.6 KB) and "books3.csv" (Completed — 467 bytes). There is also a link to "Show all downloads".

Title	Author	Publication Date
To Kill a Mockingbird	Harper Lee	July 11, 1960
1984	George Orwell	June 8, 1949
Pride and Prejudice	Jane Austen	Jan. 28, 1813
The Great Gatsby	F. Scott Fitzgerald	April 10, 1925
The Catcher in the Rye	J.D. Salinger	July 16, 1951
To Kill a Mockingbird	Harper Lee	July 11, 1960
1984	George Orwell	June 8, 1949
Pride and Prejudice	Jane Austen	Jan. 28, 1813
The Great Gatsby	F. Scott Fitzgerald	April 10, 1925
The Catcher in the Rye	J.D. Salinger	July 16, 1951

**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.

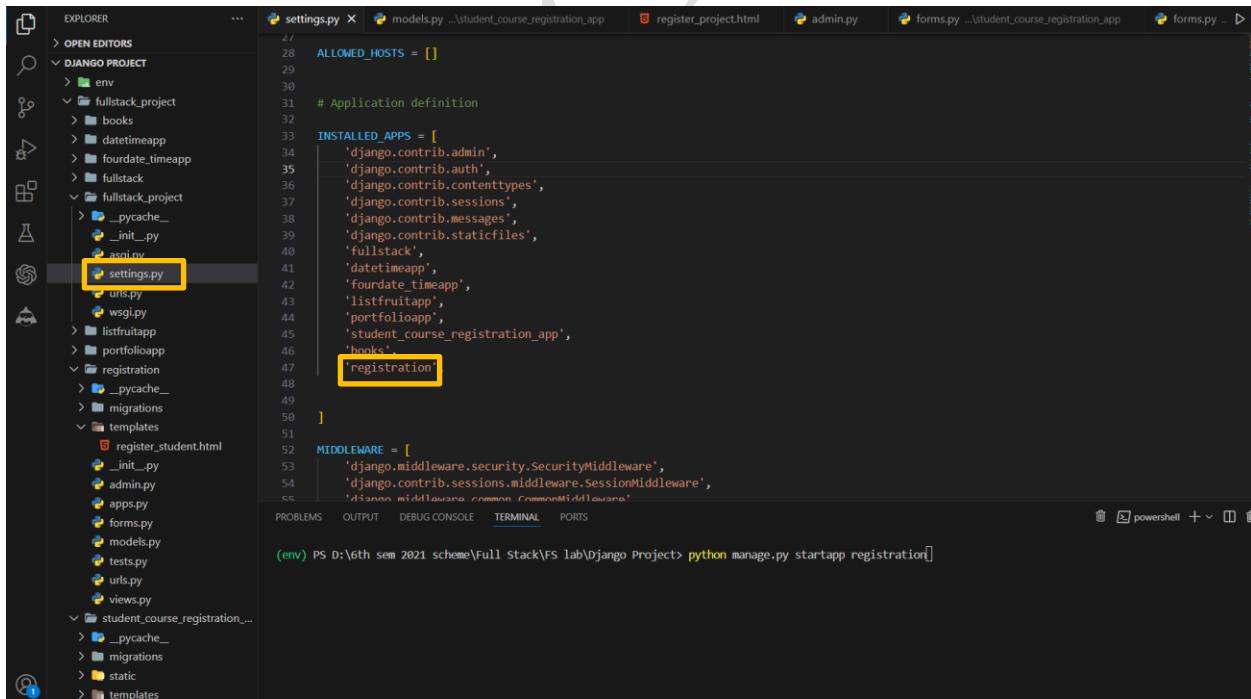


```
(env) PS D:\6th sem 2021 scheme\Full Stack\FS lab\ Django Project> python manage.py startapp registration
```

**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:



```
settings.py X models.py ...\\student_course_registration_app register_project.html admin.py forms.py ...\\student_course_registration_app forms.py ...
EXPLORER ...
OPEN EDITORS
DJANGO PROJECT
> env
fullstack_project
> books
> datetimeapp
> foudate_timeapp
> fullstack
fullstack_project
> __pycache__
 __init__.py
 asgi.py
 settings.py
 urls.py
 wsgi.py
listfruitapp
portfolioapp
registration
 __pycache__
 __init__.py
 admin.py
 apps.py
 forms.py
 models.py
 tests.py
 urls.py
 views.py
student_course_registration...
 __pycache__
 migrations
 static
 templates
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
(env) PS D:\6th sem 2021 scheme\Full Stack\FS lab\ Django Project> python manage.py startapp registration
```

## Step-03: Create models

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

```
from django.db import models
```

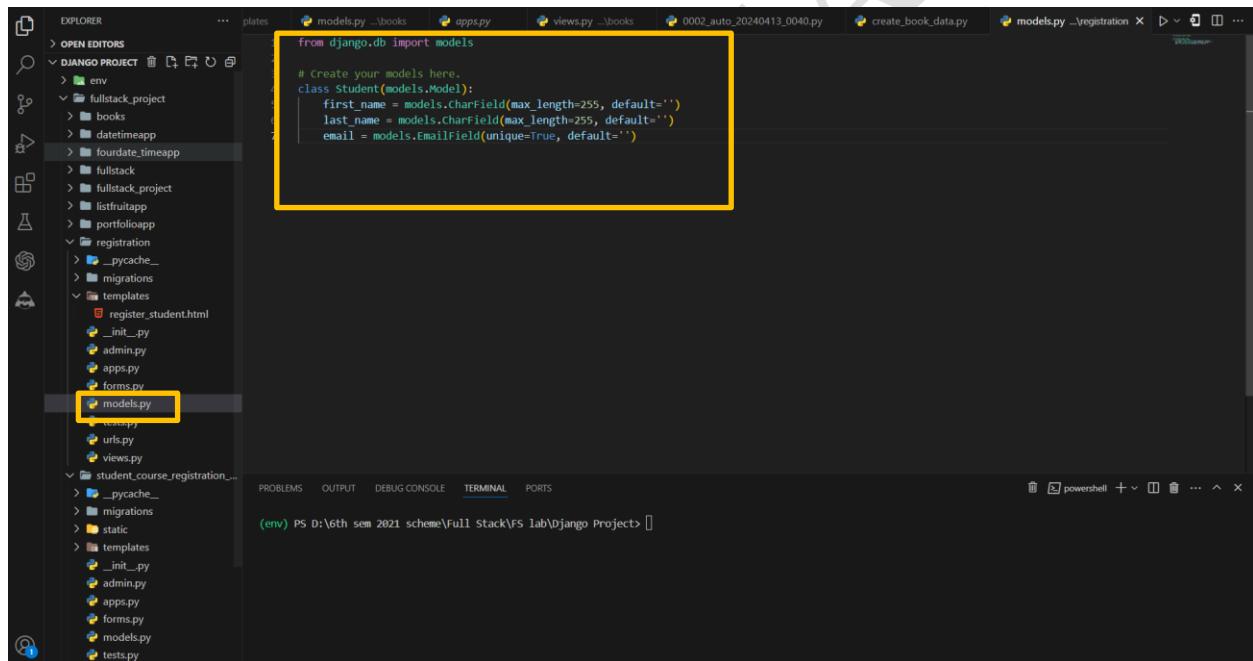
```
Create your models here.
```

```
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()

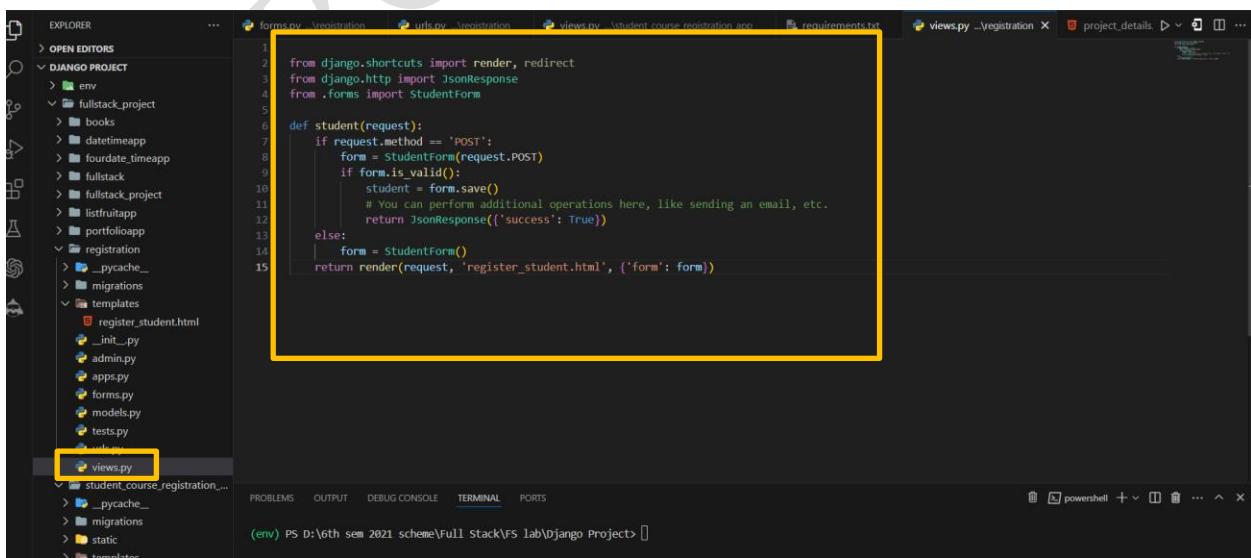
 # You can perform additional operations here, like sending an email, etc.

 return JsonResponse({'success': True})

 else:

 form = StudentForm()

 return render(request, 'register_student.html', {'form': form})
```



```
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 = {

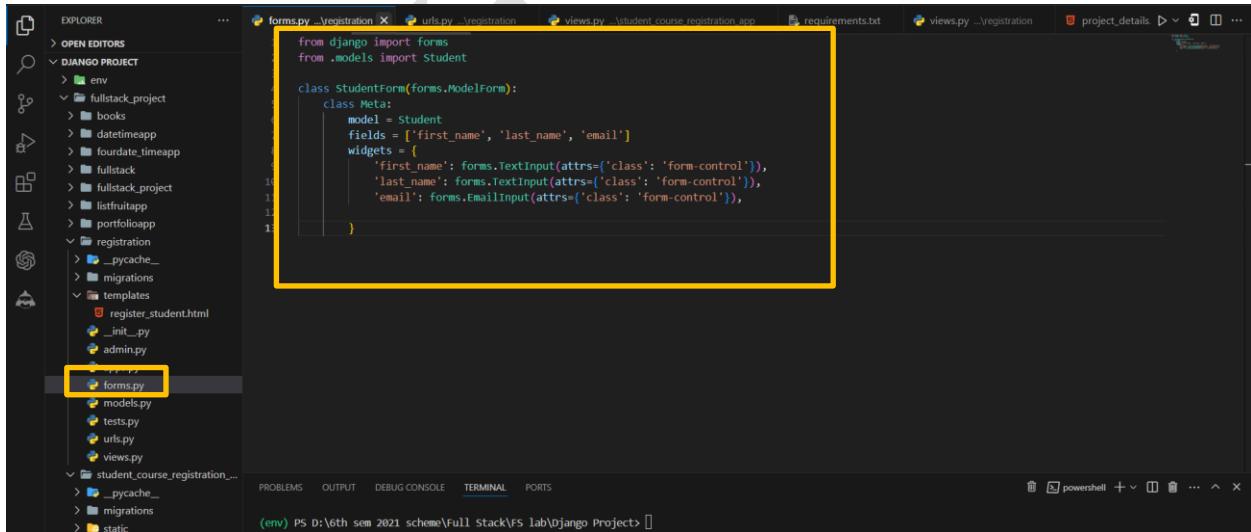
 'first_name': forms.TextInput(attrs={'class': 'form-control'}),

 'last_name': forms.TextInput(attrs={'class': 'form-control'}),

 'email': forms.EmailInput(attrs={'class': 'form-control'}),

 }

```



```
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 %}
 </form>
 </div>
 </body>

```

```
{% 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 registered
successfully.</div>');

 $('#student-form')[0].reset();

 } else {

 $('#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>');

}

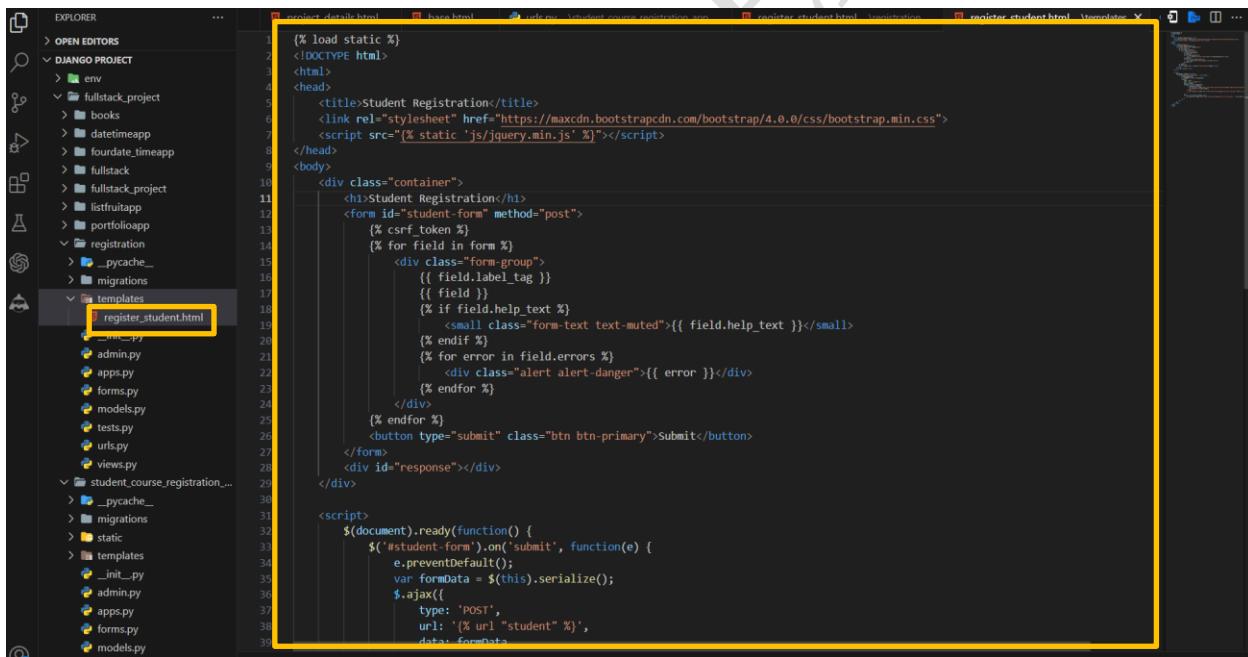
});

});

</script>

</body>

</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
 }).done(function(response) {
 $('#response').html(response);
 });
 });
});
```

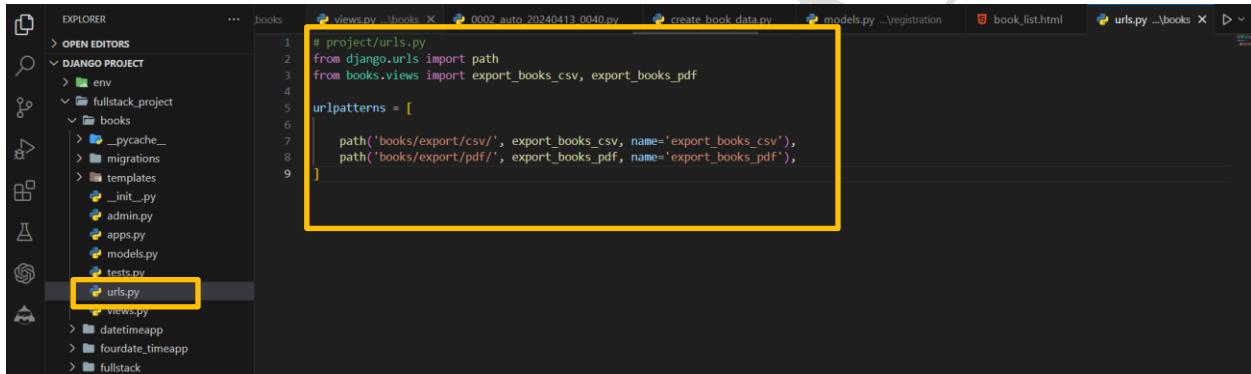
## Step-07: Update app URLs

```
project/urls.py

from django.urls import path

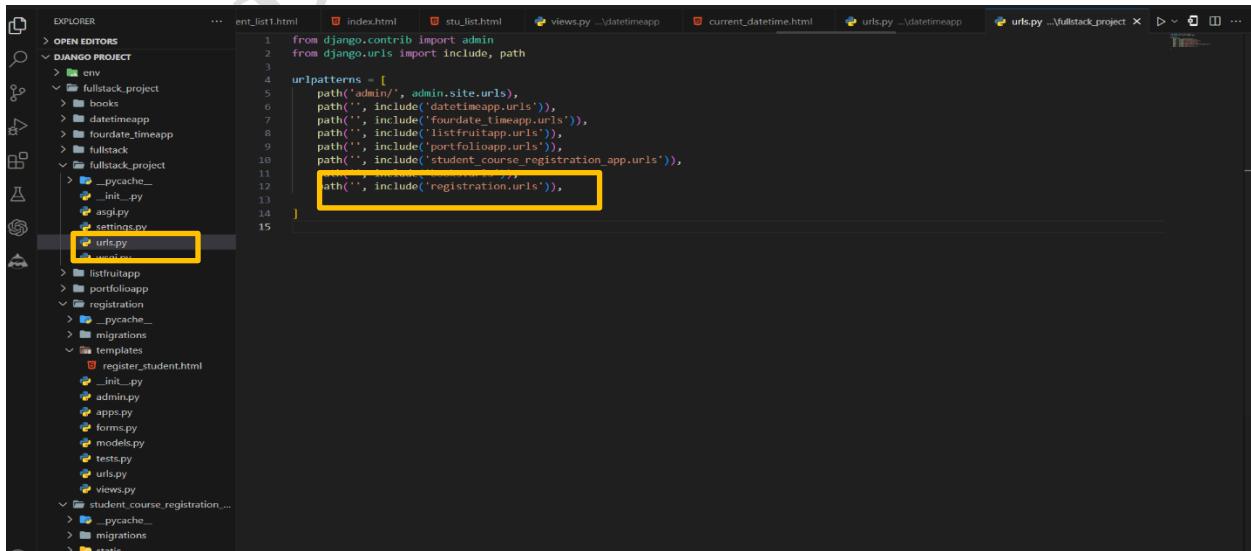
from books.views import export_books_csv, export_books_pdf

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:

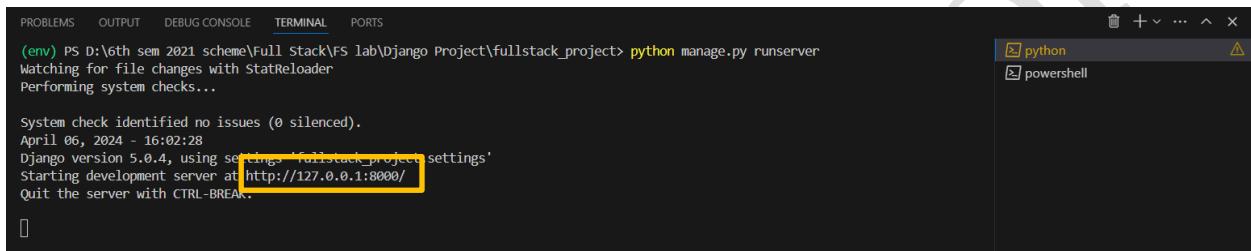


### Step-09: Run the development server

- 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/>.

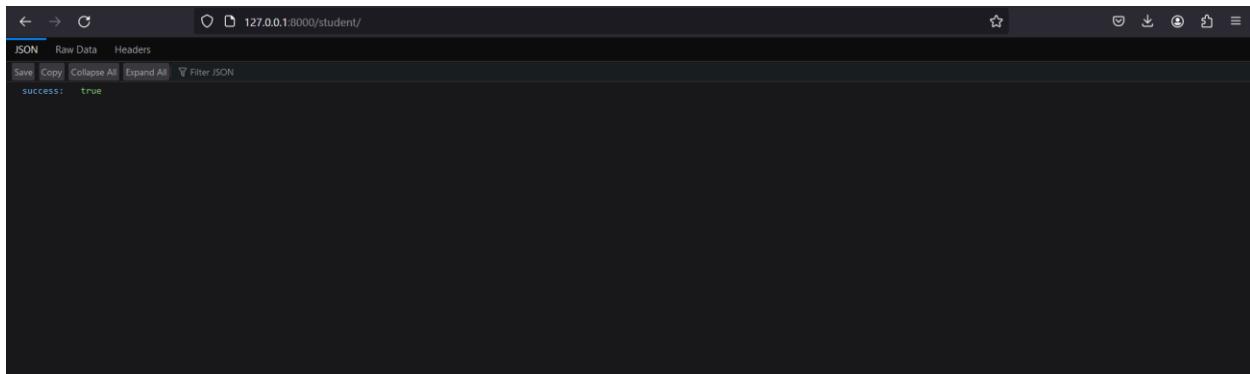


```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
(env) PS D:\6th sem 2021 scheme\Full Stack FS lab\django Project\fullstack_project> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...
System check identified no issues (0 silenced).
April 06, 2024 - 16:02:28
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/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
```

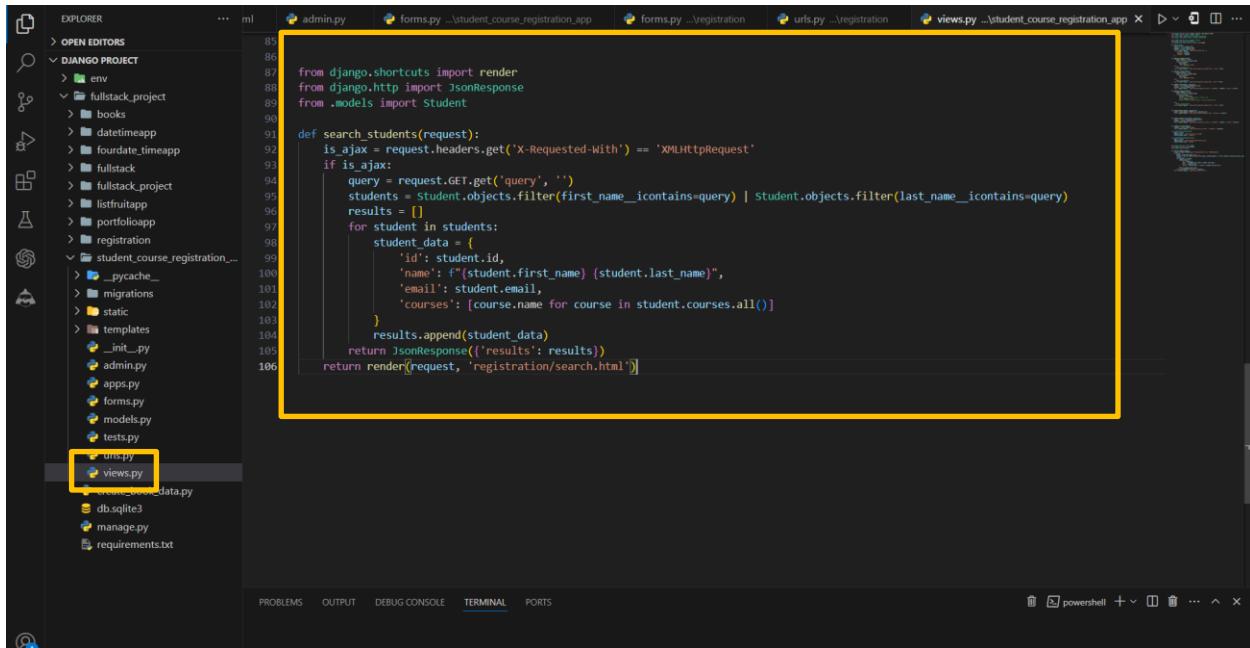
#### **Step-02: Create View Function in a student\_course\_registration\_app views.py**

```
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')
```



```
from django.shortcuts import render
from django.http import JsonResponse
from .models import Student

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 += '<li class="list-group-item">';
 html += '<h5>' + student.name + '</h5>';
 html += '<p>Email: ' + student.email + '</p>';
 html += '<p>Courses: ' + student.courses.join(', ') + '</p>';
 html += '';
 }
 html += '';
 }
 $('#search-results').html(html);
 }
 });
 });
});
```

```
 }

 html += '';

 } else {

 html += '<p>No students found.</p>';

 }

 $('#search-results').html(html);

}

});

});

});

</script>

</head>

<body>

{% csrf_token %}

<div class="container">

 <h1>Search Students</h1>

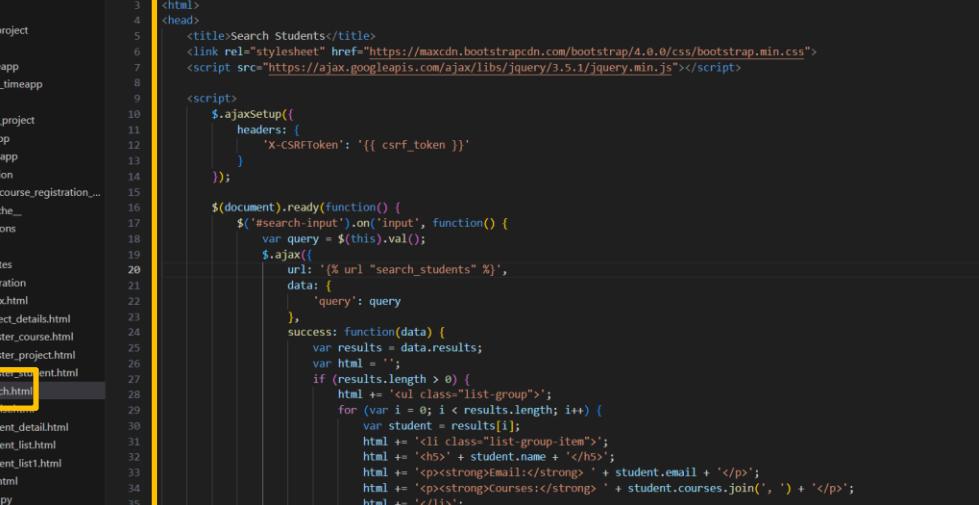
 <input type="text" id="search-input" class="form-control" placeholder="Search for a student...">

 <div id="search-results"></div>

</div>

</body>

</html>
```



The screenshot shows a code editor interface with a yellow box highlighting the search function in a file named `search.html`. The code uses jQuery to handle an input event on a search input field, sends an AJAX request to the URL `% url "search_students"`, and then processes the returned JSON data to generate a list of students with their names and courses.

```
(% 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 += '<li class="list-group-item">';
 html += '<h3>' + student.name + '</h3>';
 html += '<p>Email: ' + student.email + '</p>';
 html += '<p>Courses: ' + student.courses.join(', ') + '</p>';
 html += '';
 }
 html += '';
 }
 $('#search-results').html(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

The screenshot shows a code editor interface with two tabs open: `urls.py` and `register_student.html`. The `urls.py` tab contains URL patterns for a Django project, with the search functionality highlighted by a yellow box. The `register_student.html` tab shows the template code for a student registration form.

```
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'),
 path('projects_register/', views.register_project, name='register_project'),
 path('project_detail/int:project_id/', views.project_detail, name='project_detail'),
 path('project_student_list/int:project_id/', views.project_student_list, name='project_student_list'),
 path('students/', views.StudentListView.as_view(), name='student_list_all'),
 path('students/int:pk>/', views.StudentDetailView.as_view(), name='student_detail'),
 path('search/', views.search_students, name='search_students'),
]

```

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

A screenshot of a VS Code terminal window. The tab bar at the top shows 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL' (which is selected), and 'PORTS'. The terminal content shows a PowerShell session in an environment named '(env)'. The user runs 'python manage.py makemigrations' which outputs 'No changes detected'. Then, they run 'python manage.py migrate' which outputs 'Operations to perform:' followed by 'Apply all migrations: admin, auth, books, contenttypes, registration, sessions, student\_course\_registration\_app'. It also says 'Running migrations:' and 'No migrations to apply.' The command 'python manage.py' is shown again at the end.

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

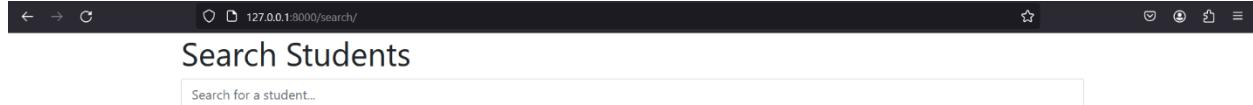
- In the VS Code terminal, navigate to your Django project's directory (the one containing manage.py).
- Run the development server

**python manage.py runserver**

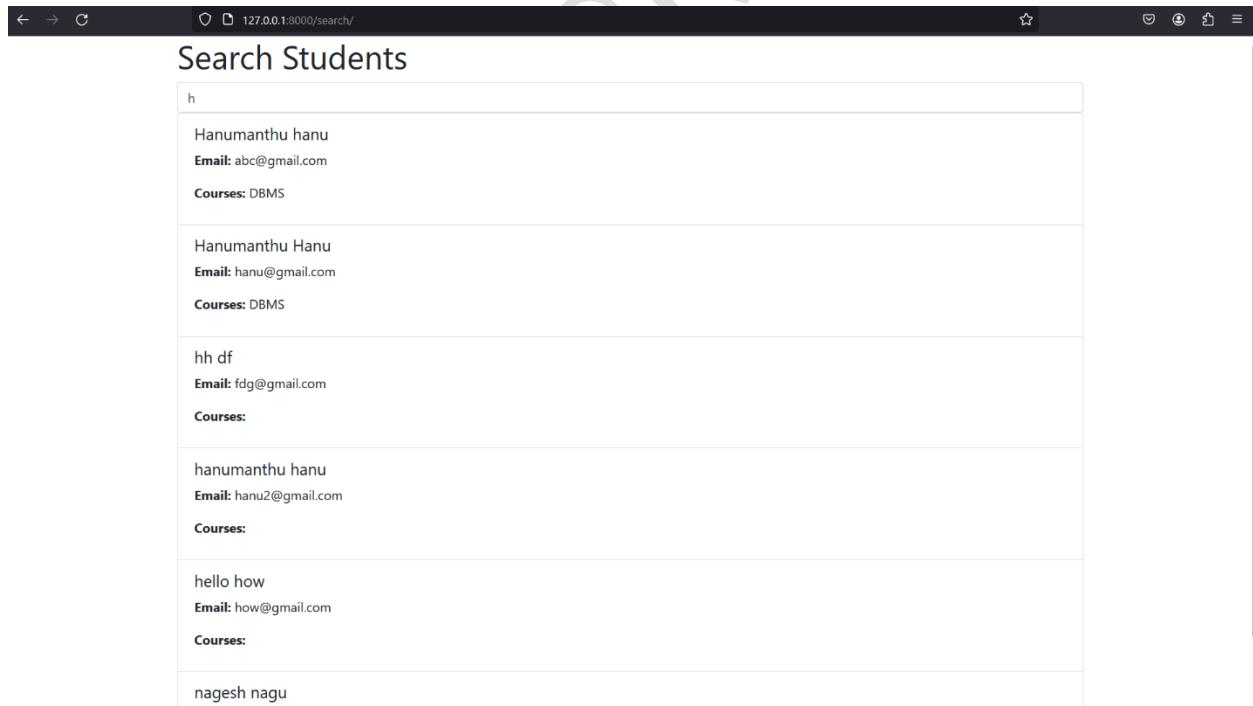
- Open your web browser and visit <http://127.0.0.1:8000/>.

A screenshot of a VS Code terminal window showing the output of the 'python manage.py runserver' command. The terminal starts with '(env) PS D:\6th sem 2021 scheme\Full Stack\FS lab\ Django Project\fullstack\_project> python manage.py runserver'. It then shows 'Watching for file changes with StatReloader' and 'Performing system checks...'. It continues with 'System check identified no issues (0 silenced). April 16, 2024 - 16:54:07'. It then lists 'Django version 5.0.4, using settings 'fullstack\_project.settings'', 'Starting development server at http://127.0.0.1:8000/' and 'Quit the server with CTRL-BREAK.'

- Type or copy this <http://127.0.0.1:8000/search/>



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



**Fig: Search Screen**