



**DEPARTMENT OF COMPUTER SCIENCE AND  
ENGINEERING**  
**(Accredited by NBA)**



**LABORATORY MANUAL  
FOR  
FULLSTACK DEVELOPMENT LABORATORY (21CS62)  
(VI SEMESTER)**

**Prepared By:**

**1. Prof. Bharath.M**

### **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 header features the Python logo and navigation links for Python, PSF, Docs, PyPI, Jobs, and Community. Below the header is a search bar with a 'GO' button. The main content area displays a code snippet demonstrating Python list comprehensions and enumerate functions:

```
# Python 3: List comprehensions
>>> fruits = ['Banana', 'Apple', 'Lime']
>>> loud_fruits = [fruit.upper() for fruit in
fruits]
>>> print(loud_fruits)
['BANANA', 'APPLE', 'LIME']

# List and the enumerate function
>>> list(enumerate(fruits))
[(0, 'Banana'), (1, 'Apple'), (2, 'Lime')]
```

To the right of the code, there is a section titled "Compound Data Types" with a brief description of lists and a "More about lists in Python 3" link. Below the code snippet is a "Learn More" button. At the bottom of the page, there are four footer sections: "Get Started", "Download", "Docs", and "Jobs".

## 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. A prominent banner in the center says "Download the latest version for Windows" with a "Download Python 3.12.2" button. To the right of the banner is an illustration of two parachutes carrying boxes. Below the banner, there are links for "Looking for Python with a different OS? Python for Windows, Linux/UNIX, macOS, Other" and "Want to help test development versions of Python 3.13? Prereleases, Docker images". Further down, a section titled "Active Python Releases" lists the following table:

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.

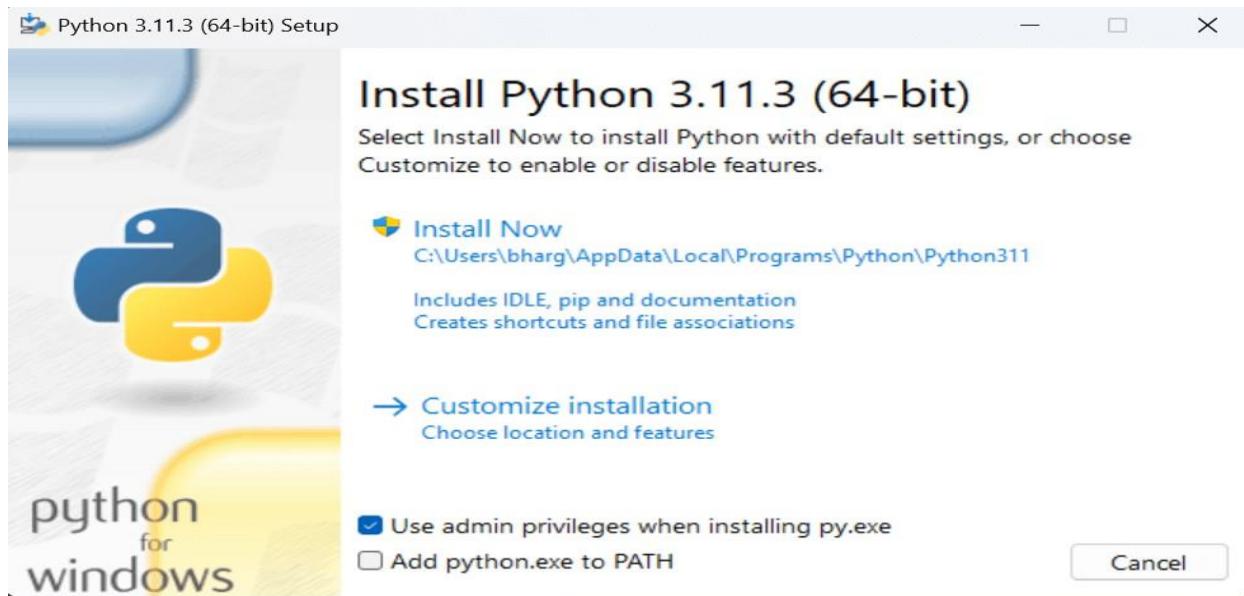
The screenshot shows a list of specific Python releases. At the top, it asks "Looking for a specific release? Python releases by version number:". Below is a table with columns: Release version, Release date, Download, and Click for more (Release Notes). The table includes the following rows:

Release version	Release date	Download	Click for more
Python 3.11.9	April 2, 2024	Download	Release Notes
Python 3.10.14	March 19, 2024	Download	Release Notes
Python 3.9.19	March 19, 2024	Download	Release Notes
Python 3.8.19	March 19, 2024	Download	Release Notes
Python 3.11.8	Feb. 6, 2024	Download	Release Notes
Python 3.12.2	Feb. 6, 2024	Download	Release Notes
Python 3.12.1	Dec. 8, 2023	Download	Release Notes

Below the table, there is a link "View older releases". Further down, there is a "Sponsors" section featuring logos for Google, Meta, fastly, Bloomberg, and Engineering. At the bottom, there are links for Licenses, Sources, Alternative, and History.

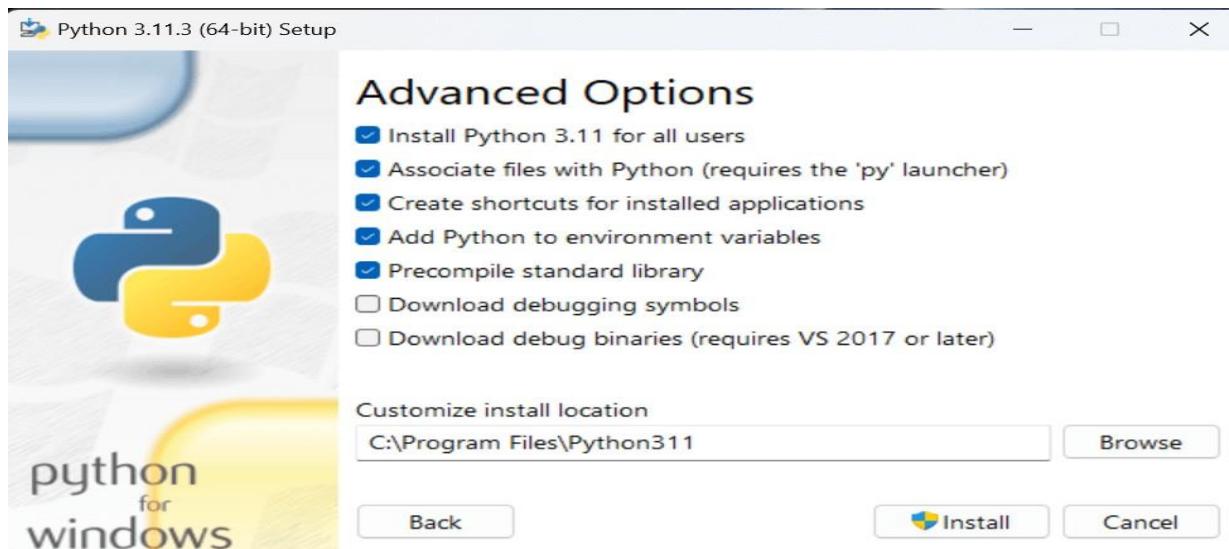
## Step - 2: Click on the Install Now

Double-click the executable file, which is downloaded.

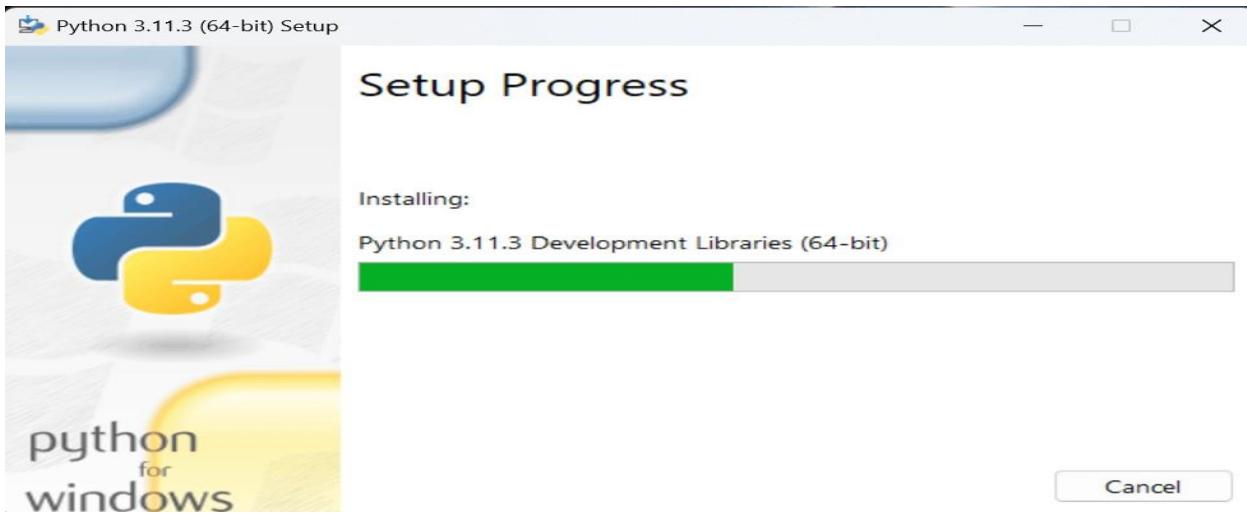


The following window will open. Click on the Add Path check box, it will set the Python path automatically.

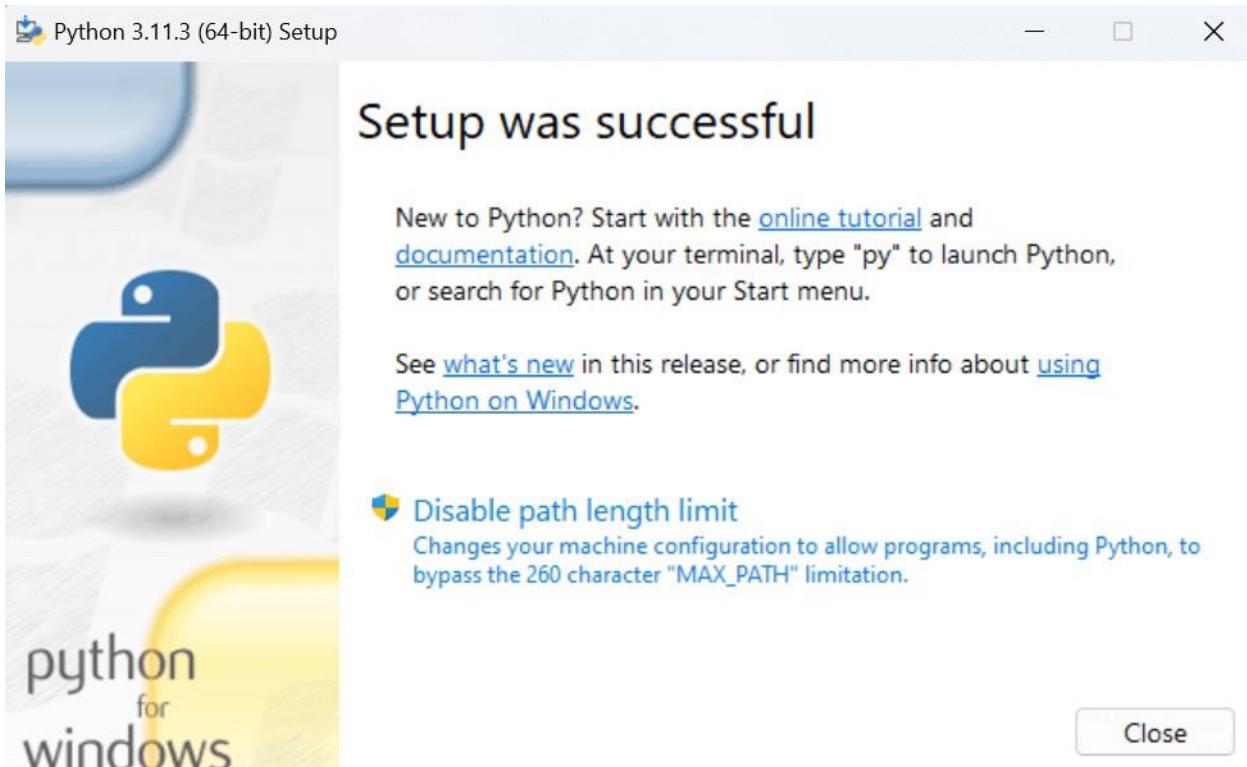
Now, Select Customize installation and proceed. We can also click on the customize installation to choose desired location and features. Other important thing is installing launcher for the all user must be checked.



### Step - 3 Installation in Process



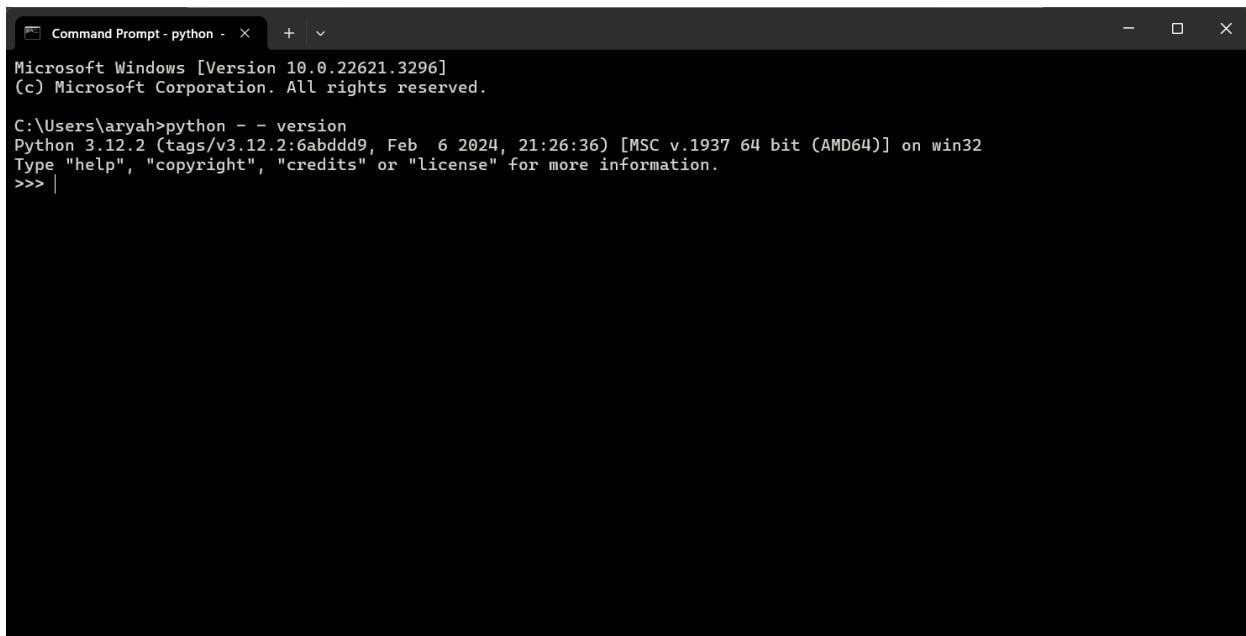
The set up is in progress. All the python libraries, packages, and other python default files will be installed in our system. Once the installation is successful, the following page will appear saying " Setup was successful ".



### 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. ".



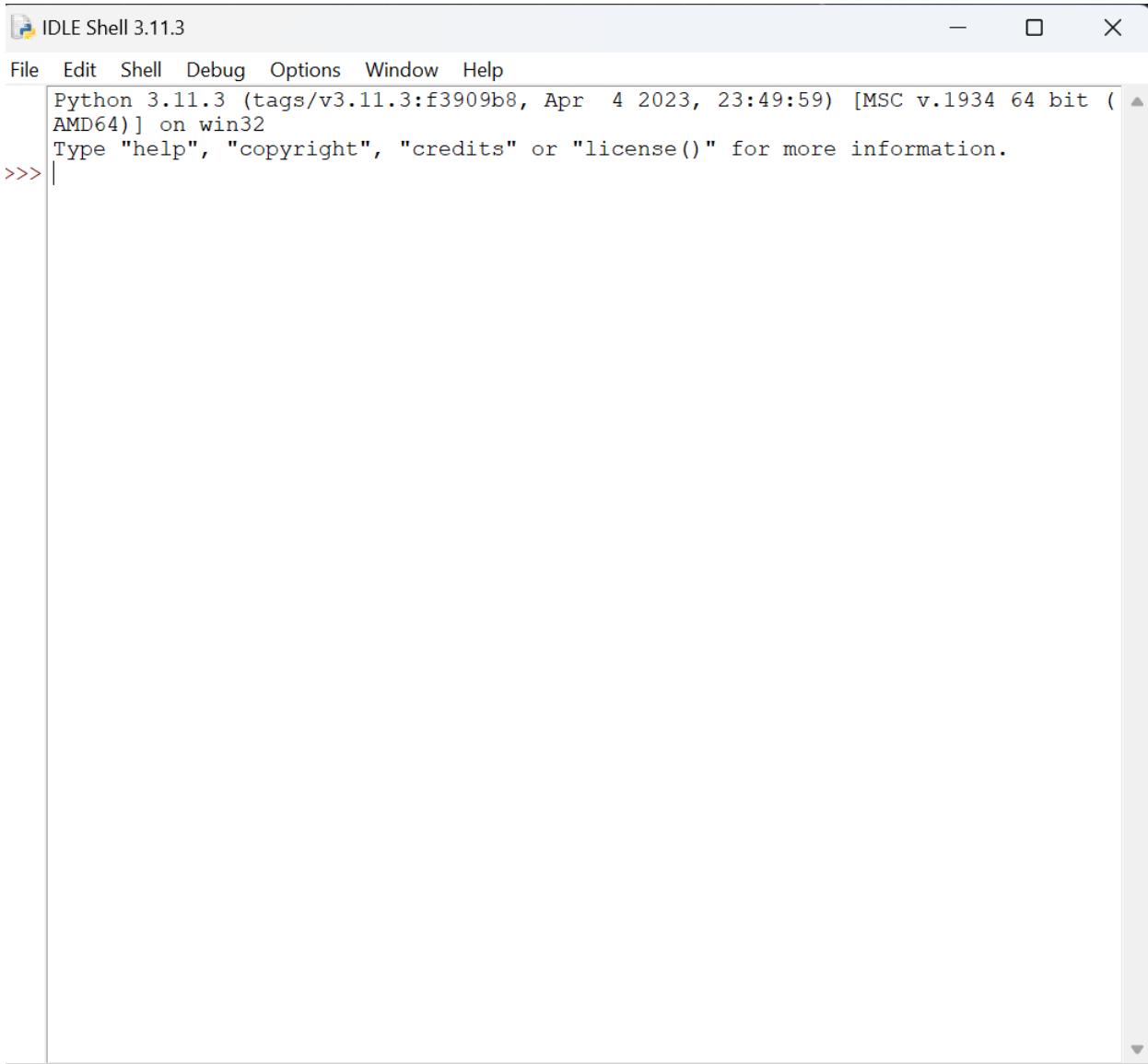
The screenshot shows a Windows Command Prompt window titled "Command Prompt - python". The window displays the following text:

```
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 following text:

```
Python 3.11.3 (tags/v3.11.3:f3909b8, Apr  4 2023, 23:49:59) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> |
```

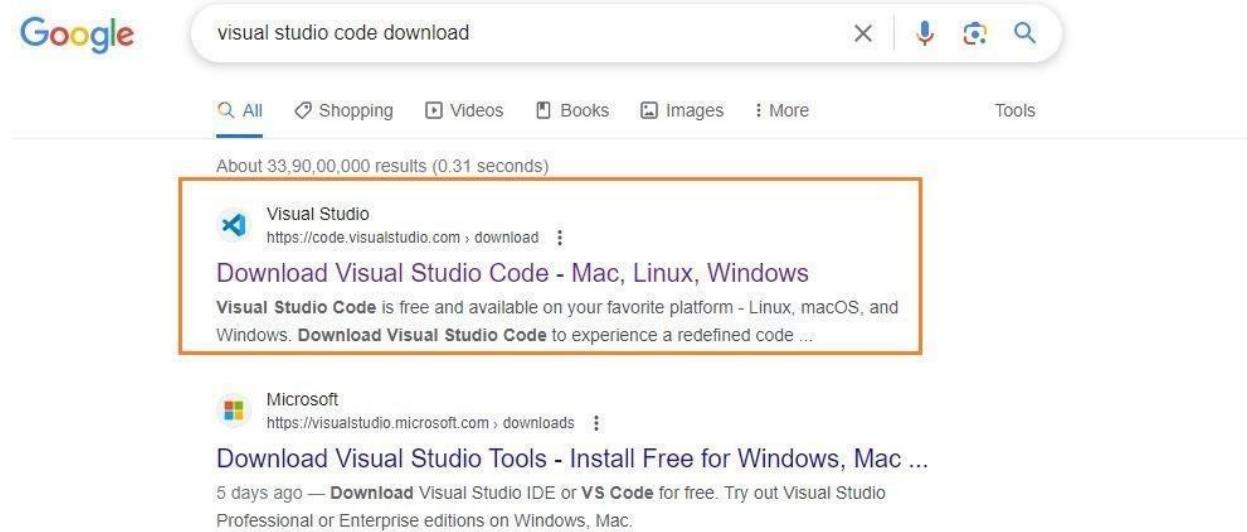
Now, to work on our first python program, we will go the interactive interpreter prompt(idle). To open this, go to "Start" and type idle. Then, click on open to start working on idle.

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

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



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



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

The screenshot shows the official Visual Studio Code website. At the top, there's a navigation bar with links for Visual Studio Code, Docs, Updates, Blog, API, Extensions, FAQ, Learn, a search bar labeled 'Search Docs', and a 'Download' button. Below the navigation bar, a message says 'Version 1.78 is now available! Read about the new features and fixes from April.' The main content area is titled 'Download Visual Studio Code' and includes the subtext 'Free and built on open source. Integrated Git, debugging and extensions.' Below this, there are download links for Windows (Windows 8, 10, 11), Linux (.deb for Debian, Ubuntu; .rpm for Red Hat, Fedora, SUSE), and Mac (macOS 10.11+). The Windows link is highlighted with a red box.

### Download Visual Studio Code

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

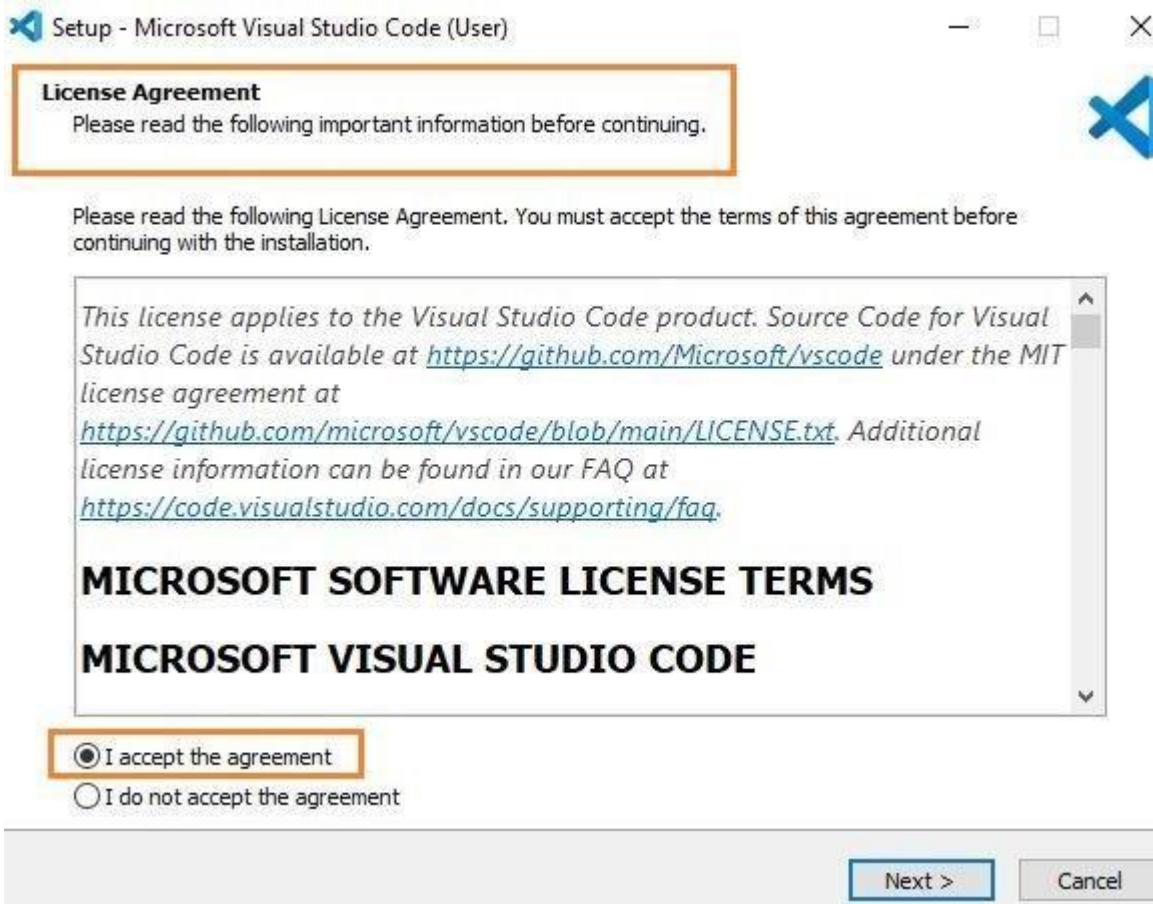


Step – 4: The file will be downloaded onto your system. Open the file and then click on Install.

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

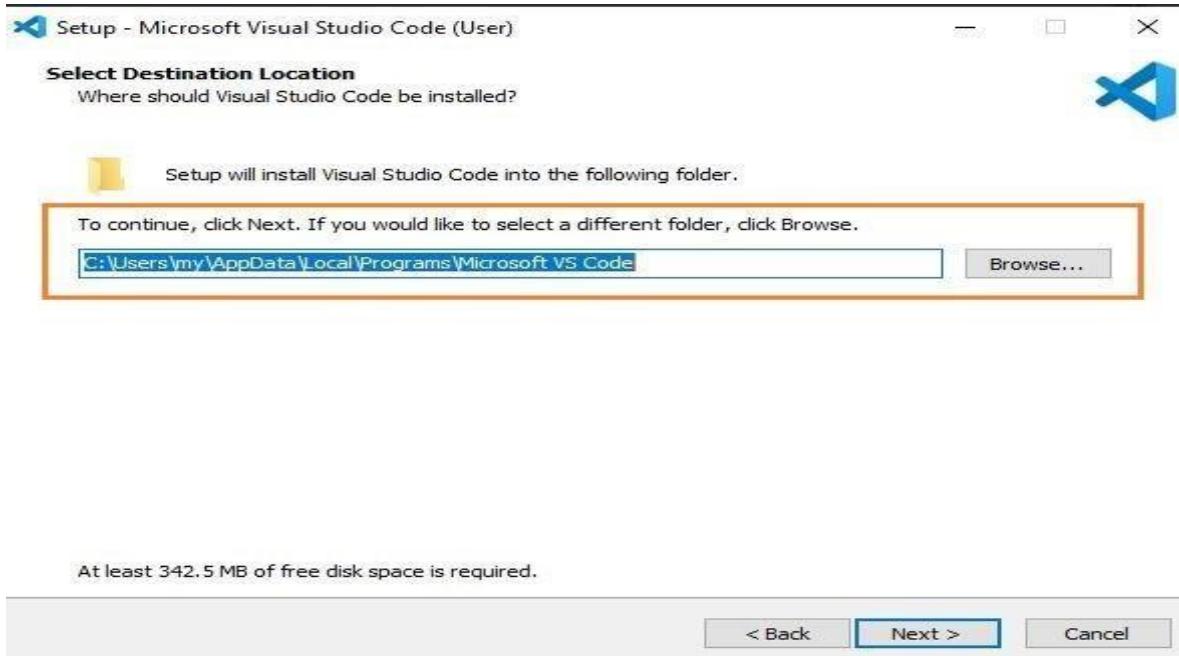
The screenshot shows the 'Getting Started' page of the Visual Studio Code website. On the left, there's a sidebar with links for Overview, SETUP, GET STARTED, USER GUIDE, SOURCE CONTROL, TERMINAL, LANGUAGES, NODEJS / JAVASCRIPT, TYPESCRIPT, PYTHON, and JAVA. The main content area has a box with the heading 'Thanks for downloading VS Code for Windows!' and a message about direct download links and user surveys. Below this, there's a section titled 'Getting Started' with a brief introduction to what VS Code is and its features. There's also a link to 'Visual Studio Code in Action'. On the right, there's a sidebar titled 'GETTING STARTED' with links for VS Code in Action, Top Extensions, First Steps, Keyboard Shortcuts, Downloads, and Privacy. At the bottom right, there are social media and community engagement links: Tweet this link, Subscribe, Ask questions, Follow @code, Request features, Report issues, and Watch videos.

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

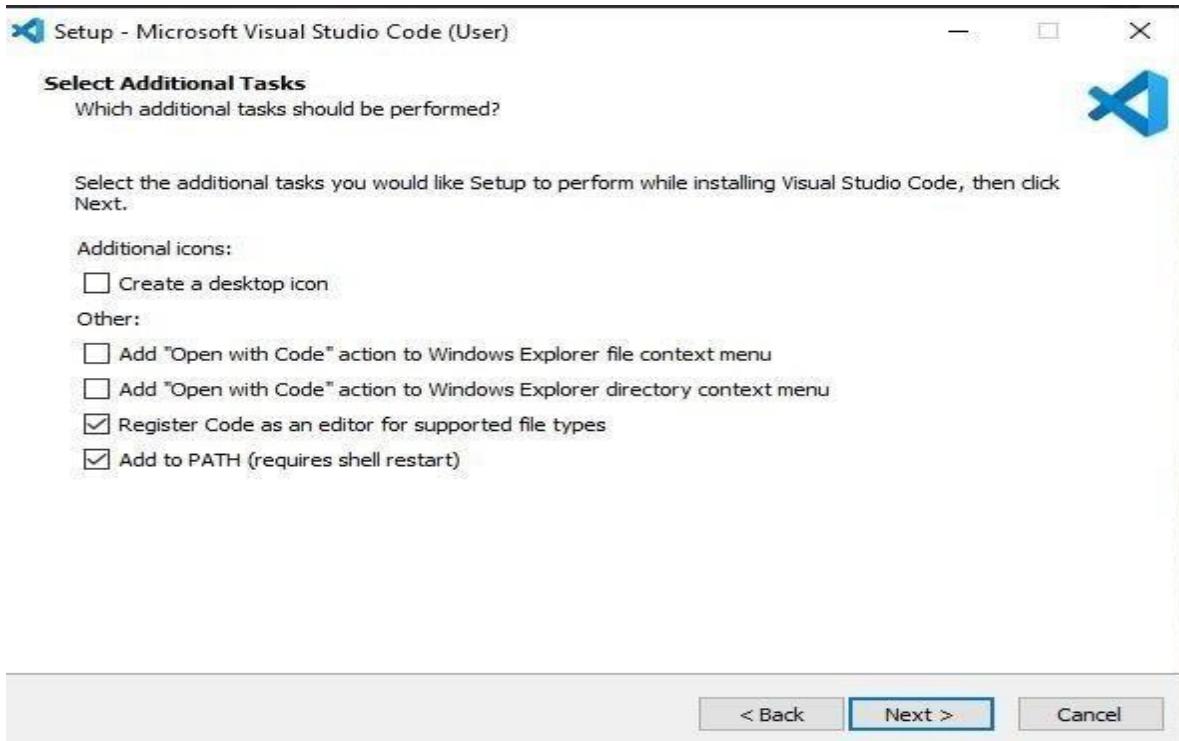


**Step – 6: Then it prompts for the file location, where you want to save the VS Code file.**

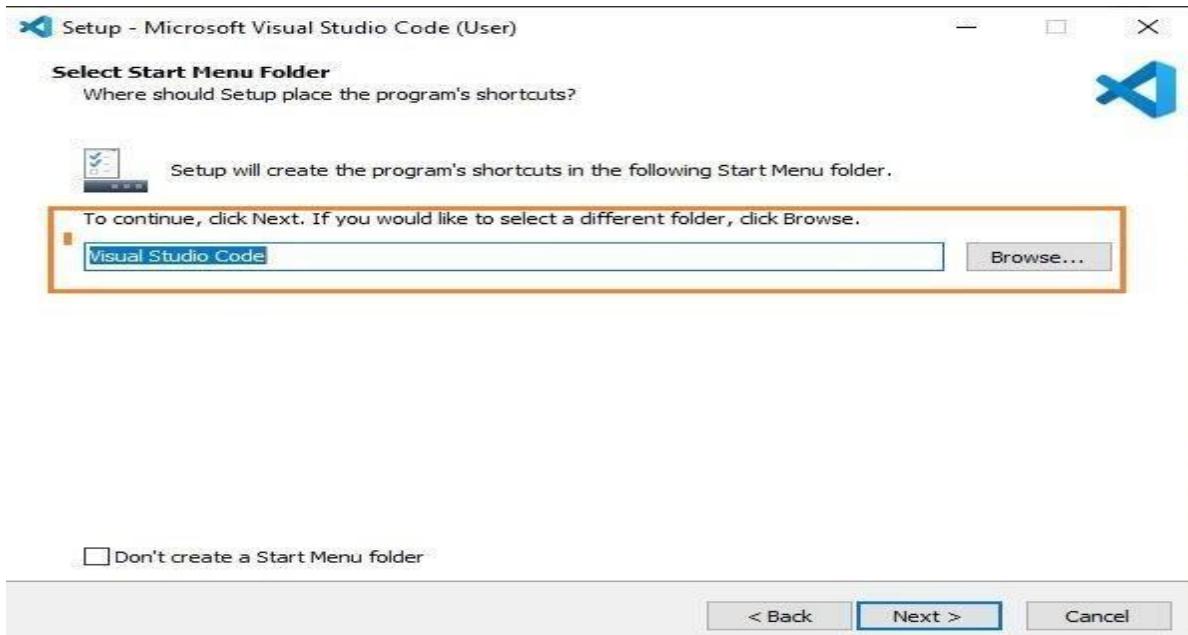
Browse the location and then click on Next.



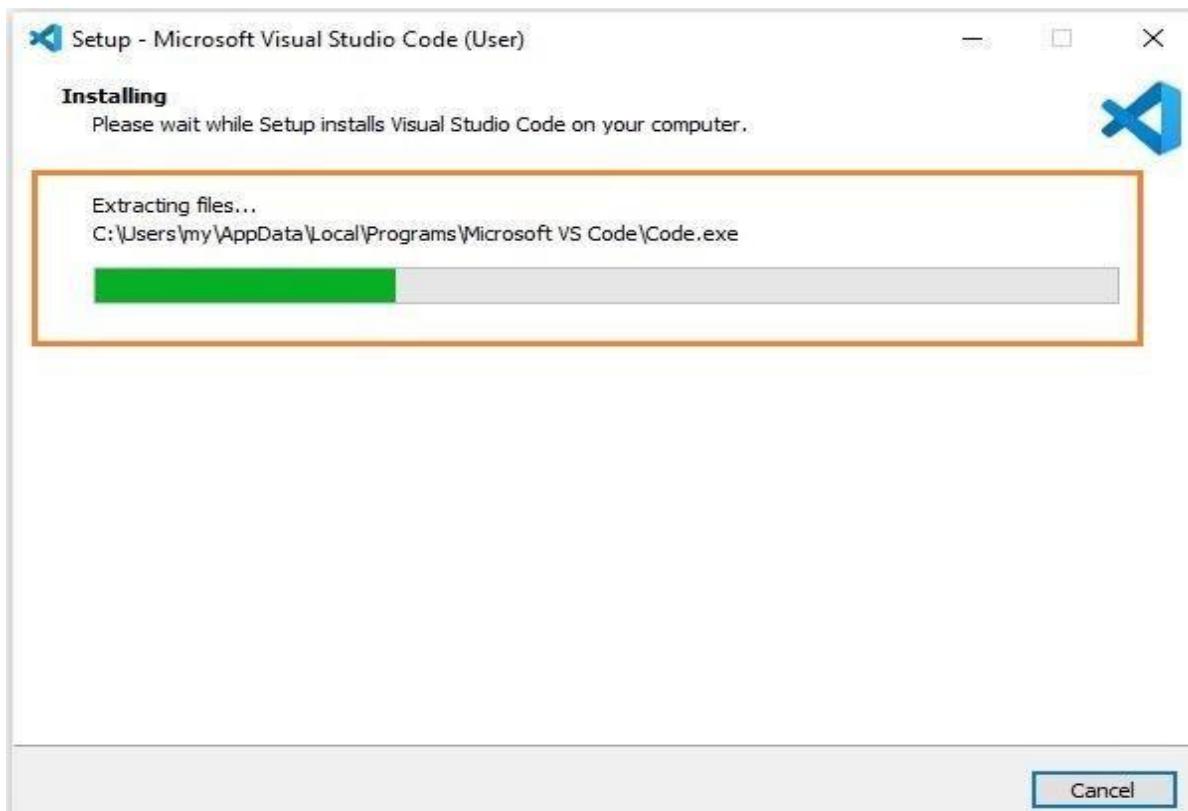
**Step – 7: Next, you see the prompt for the additional task which we want the VS Code to perform. At this step, choose the default settings and then click on next.**



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



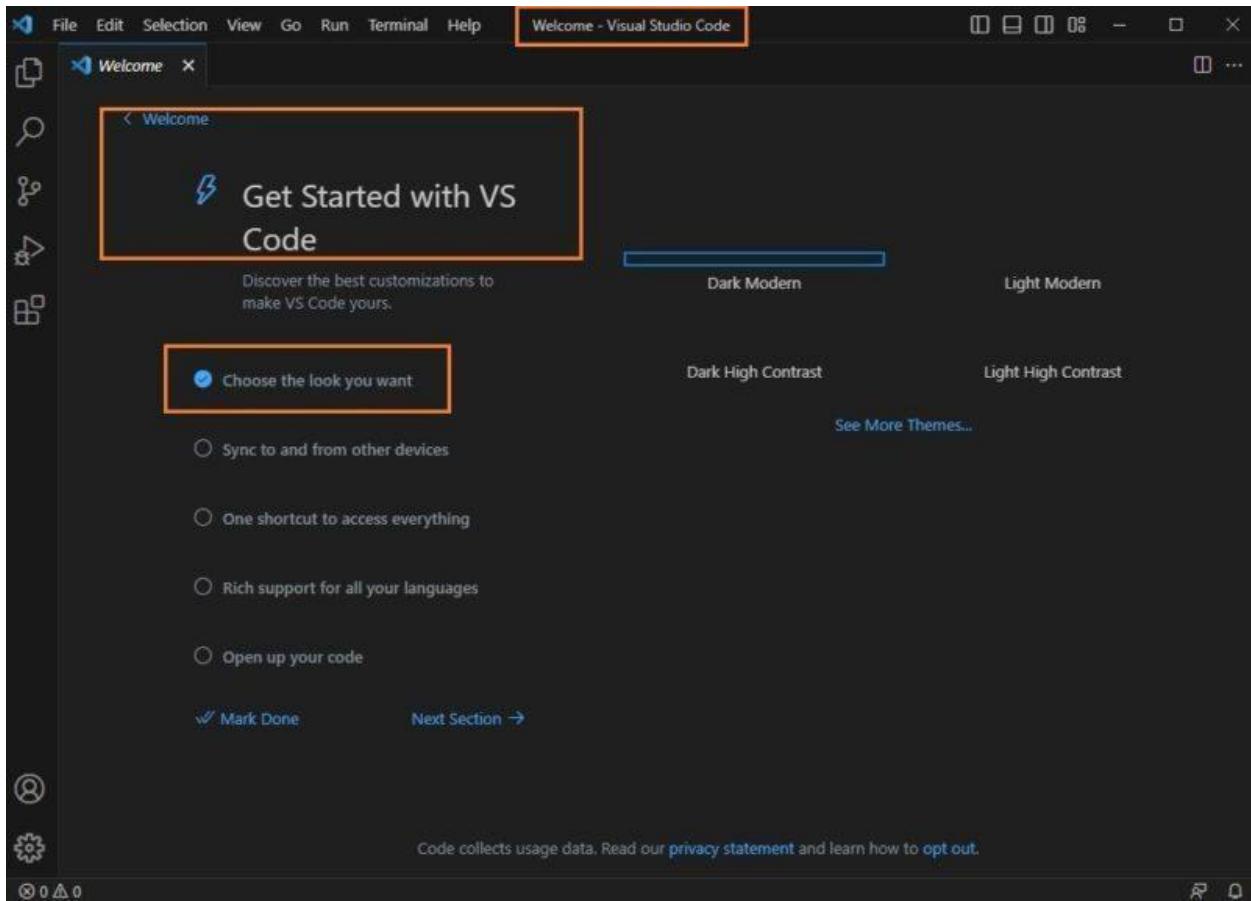
**Step – 9: The installation of VS Code will now begin.**



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 main Django website at <https://www.djangoproject.com>. The header features the Django logo and the tagline "The web framework for perfectionists with deadlines". Navigation links include OVERVIEW, DOWNLOAD, DOCUMENTATION, NEWS, COMMUNITY, CODE, ISSUES, ABOUT, and DONATE. A prominent green button says "Get started with Django". Below the header, a dark banner states "Django makes it easier to build better web apps more quickly and with less code." On the left, a section titled "Meet Django" describes it as a high-level Python web framework. It includes a "Ridiculously fast." badge with a lightning bolt icon. On the right, there's a "Download latest release: 5.0.4" button, a "Support Django!" section, and a note from Happy Herbivore Inc. about their donation to the Django Software Foundation.

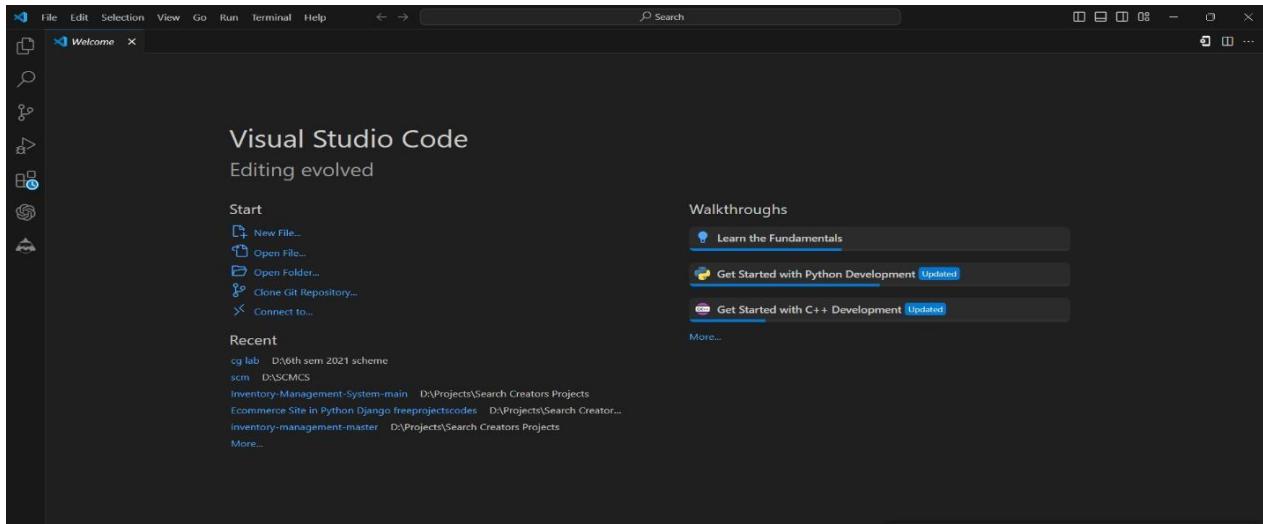
The screenshot shows the "Download" page of the Django website. The header and navigation bar are identical to the homepage. The main content area is titled "How to get Django". It explains that Django is available open-source under the [BSD license](#) and recommends using Python 3. It provides instructions for getting the latest official version (Option 1) or the latest development version (Option 2). For Option 1, it shows a terminal command: `python -m pip install Django==5.0.4`. For Option 2, it shows another terminal command: `py -m pip install Django==5.0.4`. To the right, there's a "Diamond and Platinum Members" section featuring DEFNA and JetBrains. DEFNA is described as a non-profit organization that hosts Django events in North America. JetBrains is described as the Python IDE for Professional Developers. Below this is a "Support Django!" section with a note from StyleSeat.

## Experiment-02

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

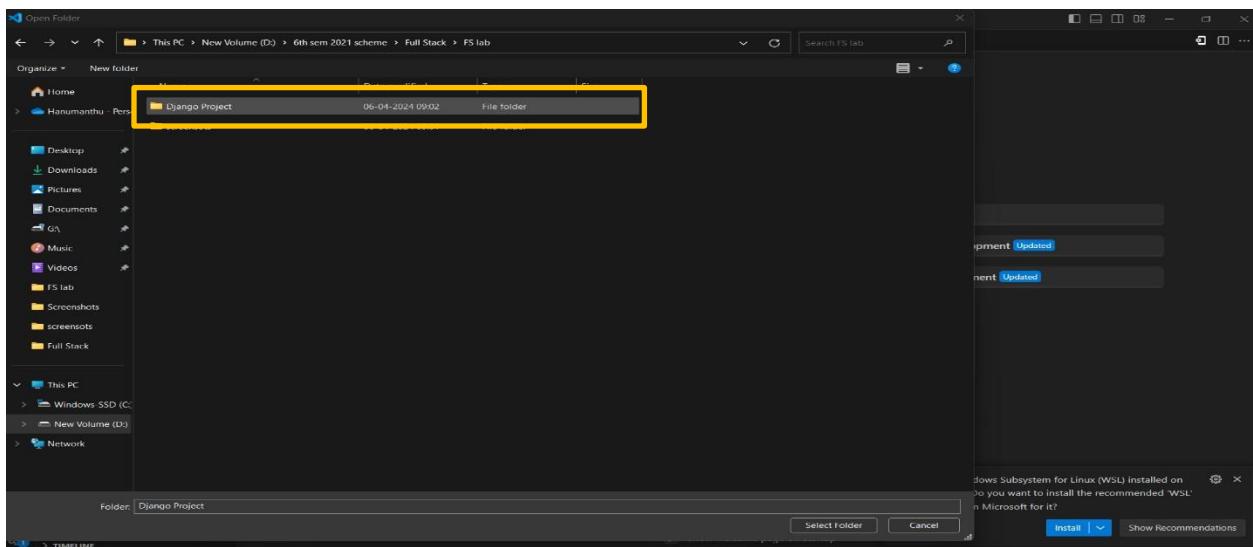
**Here are the steps you can follow to create virtual environment: Steps to Create Virtual Environment, Django Project and App**

### **Step-01: Open Visual Studio Code**



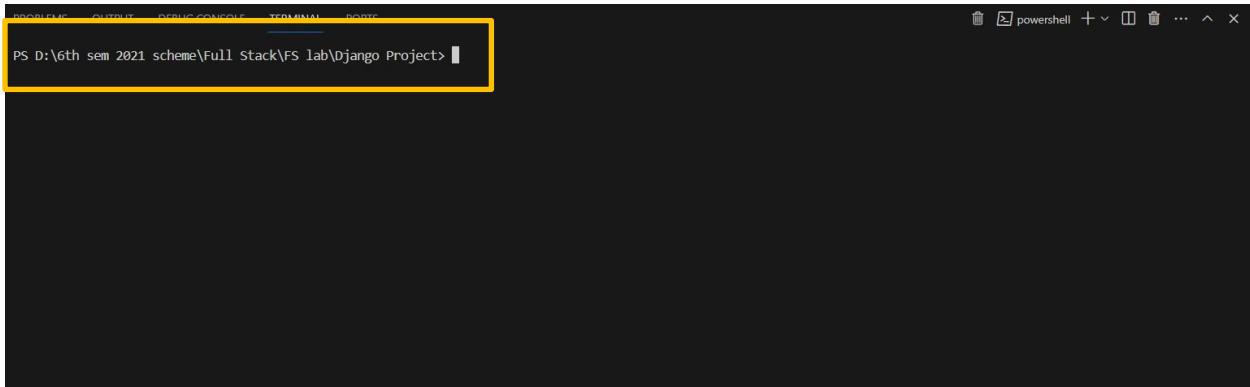
### **Step-02: Create a new folder for your project**

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

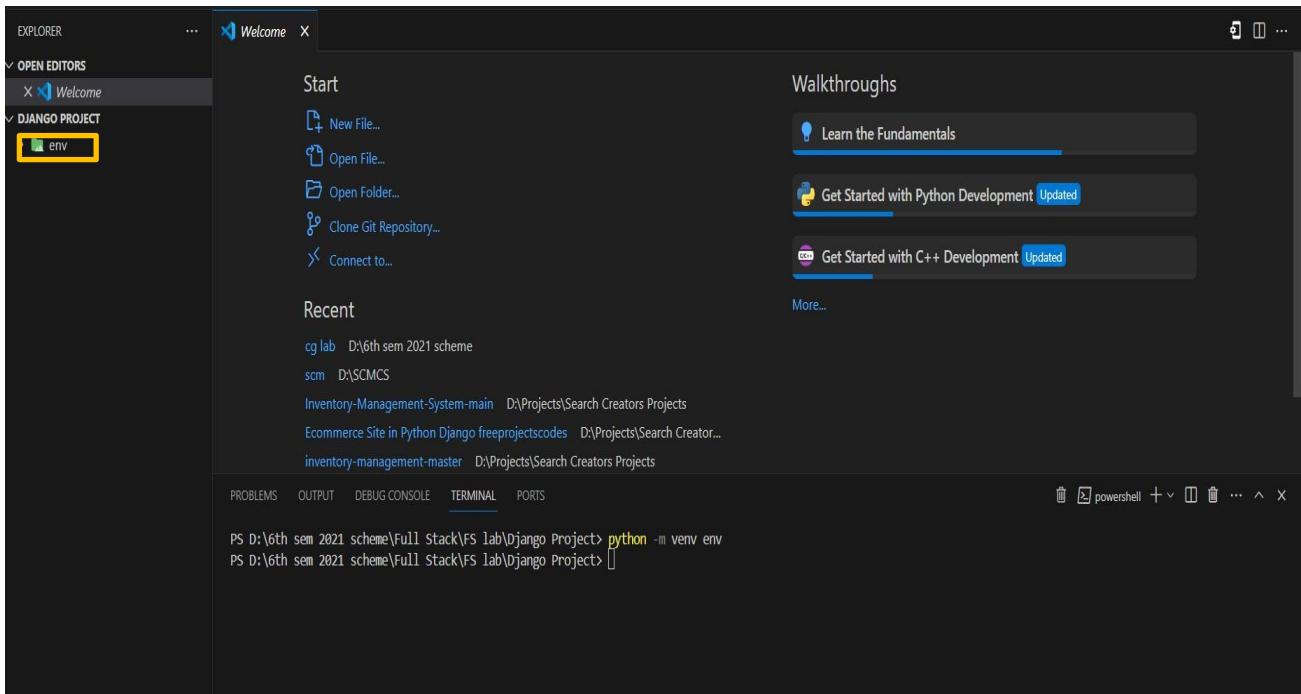
In VS Code, go to **View > Terminal** or use the keyboard shortcut **Ctrl+``** (Windows/Linux) or **Cmd+``** (macOS) to open the integrated terminal.



## 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

The screenshot shows a terminal window with the following command history:

```
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> 
```

The command `env\scripts\activate` is highlighted with a yellow box.

## Step-05: Install Django

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

The screenshot shows a terminal window in VS Code with the following command history:

```
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 (4.0 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> 
```

The command `pip install django` is highlighted with a yellow box.

## 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 with the terminal tab active. The terminal window displays the command `django-admin startproject fullstack_project` being run. The output shows the process of installing Django and other dependencies, including the creation of the project directory and the generation of initial files like `manage.py` and `settings.py`. The project 'fullstack\_project' is highlighted in the Explorer sidebar.

## 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 window with the command `cd fullstack_project` being run. The output shows the user navigating into the project directory. The directory path is shown as `D:\6th sem 2021 scheme\Full Stack\FS lab\ Django Project\fullstack_project`. The terminal tab is highlighted in the bottom navigation bar.

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.]

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 🗑️ 🖥️ powershell + ⬤ ⬧ ⬨ ⬩ ⬪ ⬫ ⬭ ⬮ ⬯ ⬱ ⬳
```

(env) PS D:\6th sem 2021\schema\Full StackFS lab\ Django Project\fullstack\_project> cd..  
(env) PS D:\6th sem 2021\schema\Full StackFS lab\ Django Project> cd fullstack project  
(env) PS D:\6th sem 2021\schema\Full StackFS lab\ Django Project\fullstack project> python manage.py startapp fullstack  
(env) PS D:\6th sem 2021\schema\Full StackFS 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.

The screenshot shows a code editor interface with a left sidebar for file navigation and a right sidebar for toolbars. The current file is `settings.py`, which contains configuration for a Django application. The `INSTALLED_APPS` section is highlighted with a yellow box around the entry `'fullstack'`. This indicates that the `fullstack` application is being developed or configured.

```
# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/5.0/howto/deployment/checklist/
#
# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-lv*wj#747$oyfsi*nx%b3%6s0i^r4%5st6(wl&jbgzm(i1$vc'
#
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

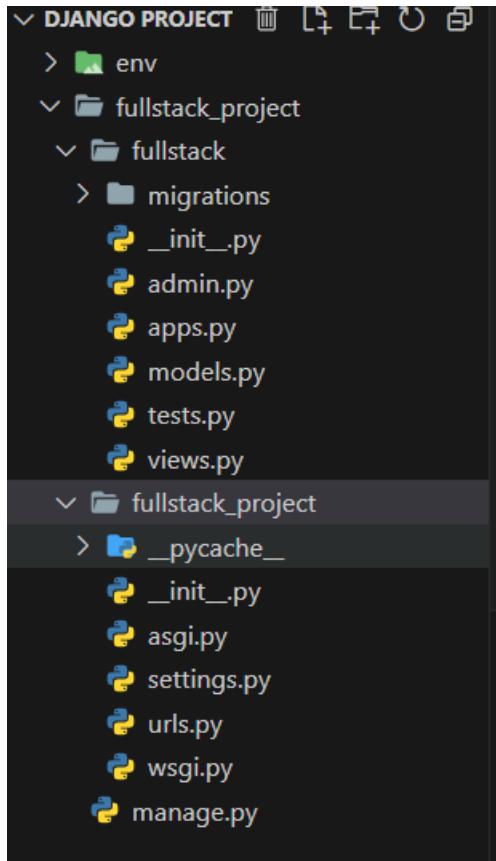
ALLOWED_HOSTS = []

# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'fullstack'
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    ...
]
```

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.
```

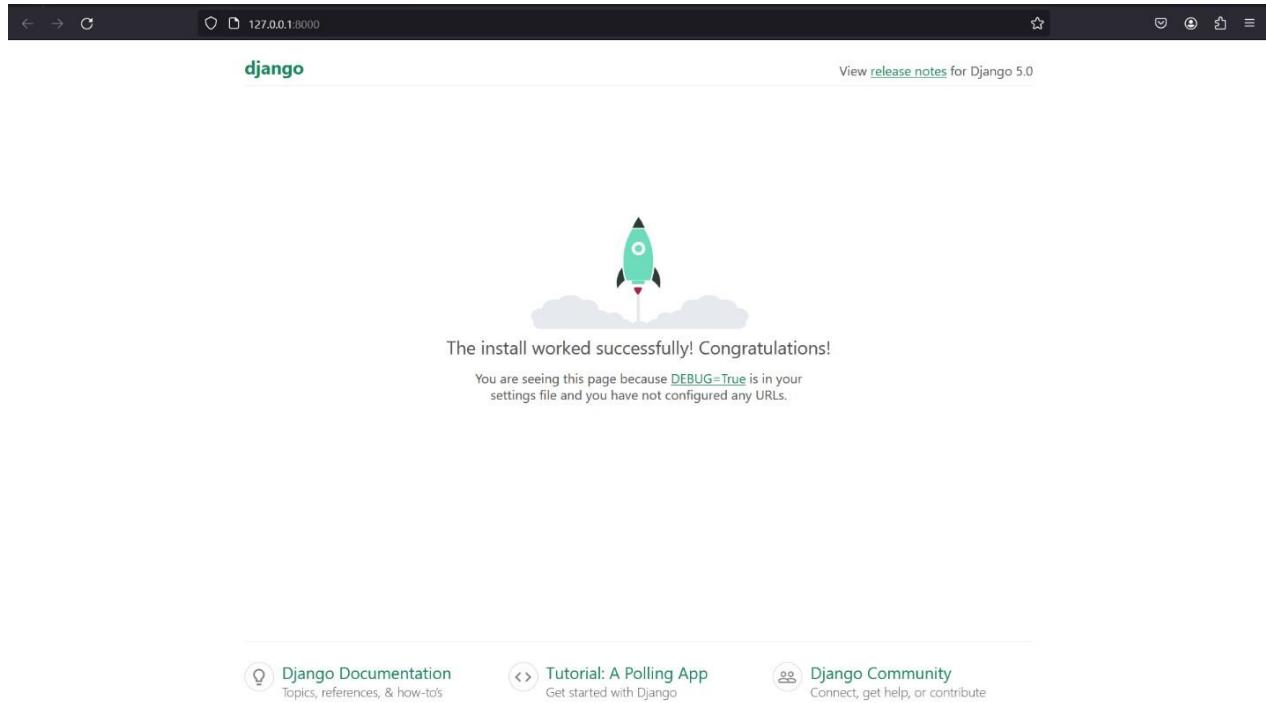
## Check the server output

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.

# 21CS62 | Full Stack Django Development|

---

Here Displays the Development Server After Running Server



## Experiment-03

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

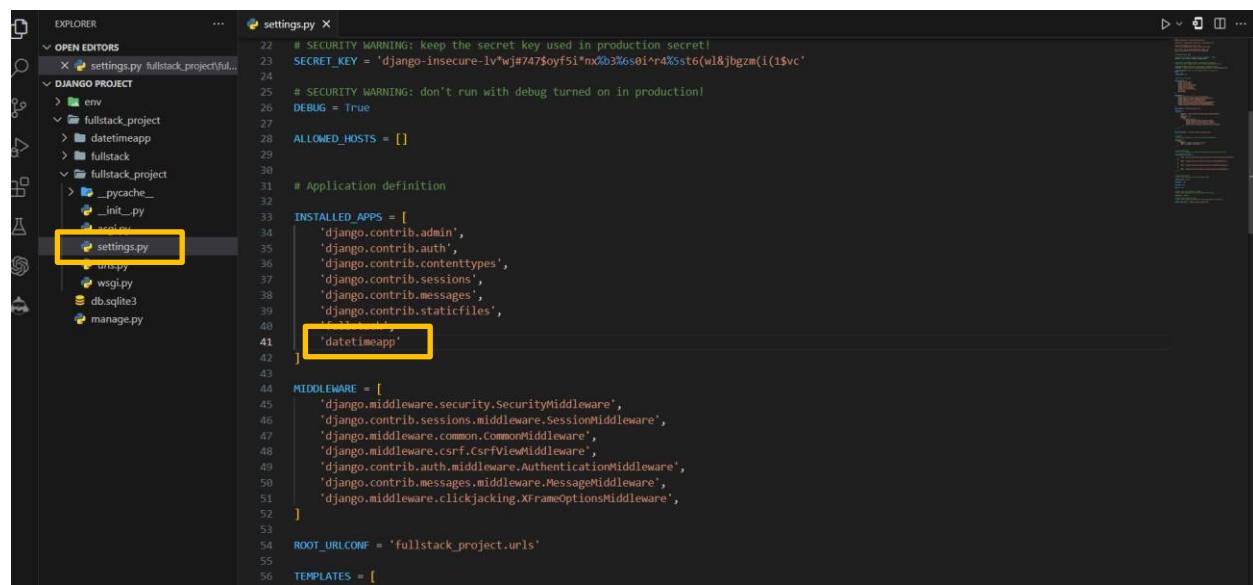
**Step-01:** This app will be created in the Django project we made earlier.

```
(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).

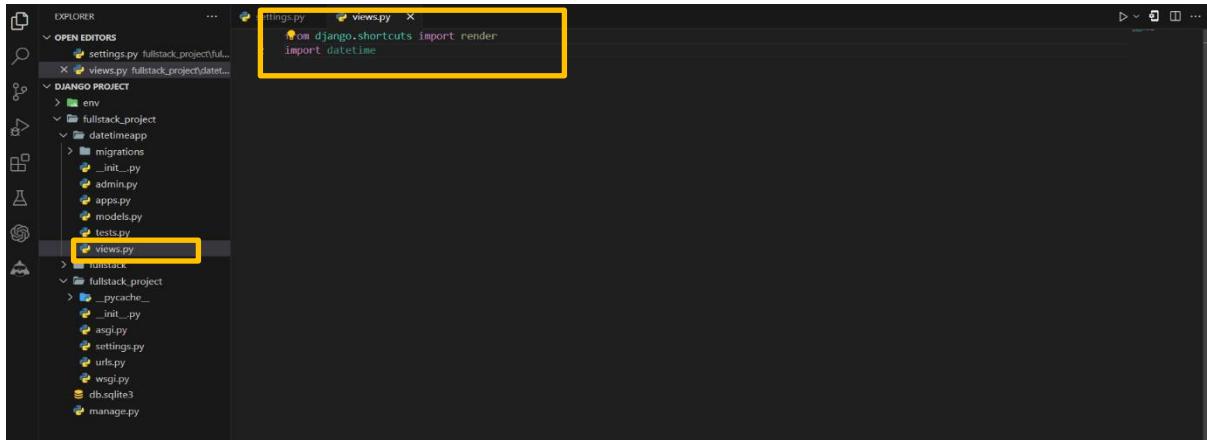
Locate the **INSTALLED\_APPS** list and add the name of your new app to the list:



```
22 # SECURITY WARNING: keep the secret key used in production secret!
23 SECRET_KEY = 'django-insecure-lv*wj#7$oyf5!nx@b3g6$0!r4Xst6(wl&jbgzm(i1$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_project',
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



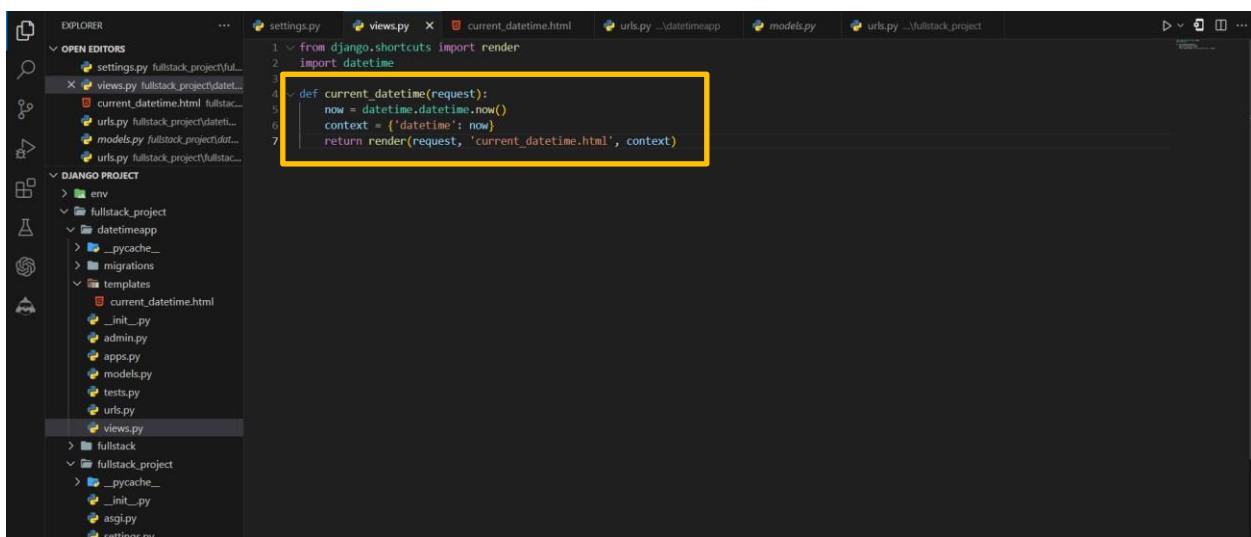
- Create a new **view function** that will handle the request and render the date and time:

```
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>
```

The screenshot shows a code editor with a dark theme. On the left is the Explorer sidebar showing a Django project structure with files like settings.py, views.py, urls.py, models.py, and migrations. A folder named 'templates' is selected and highlighted with a yellow box. In the main editor area, the 'current\_datetime.html' file is open and also highlighted with a yellow box. The code in the file is:

```
<!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
```

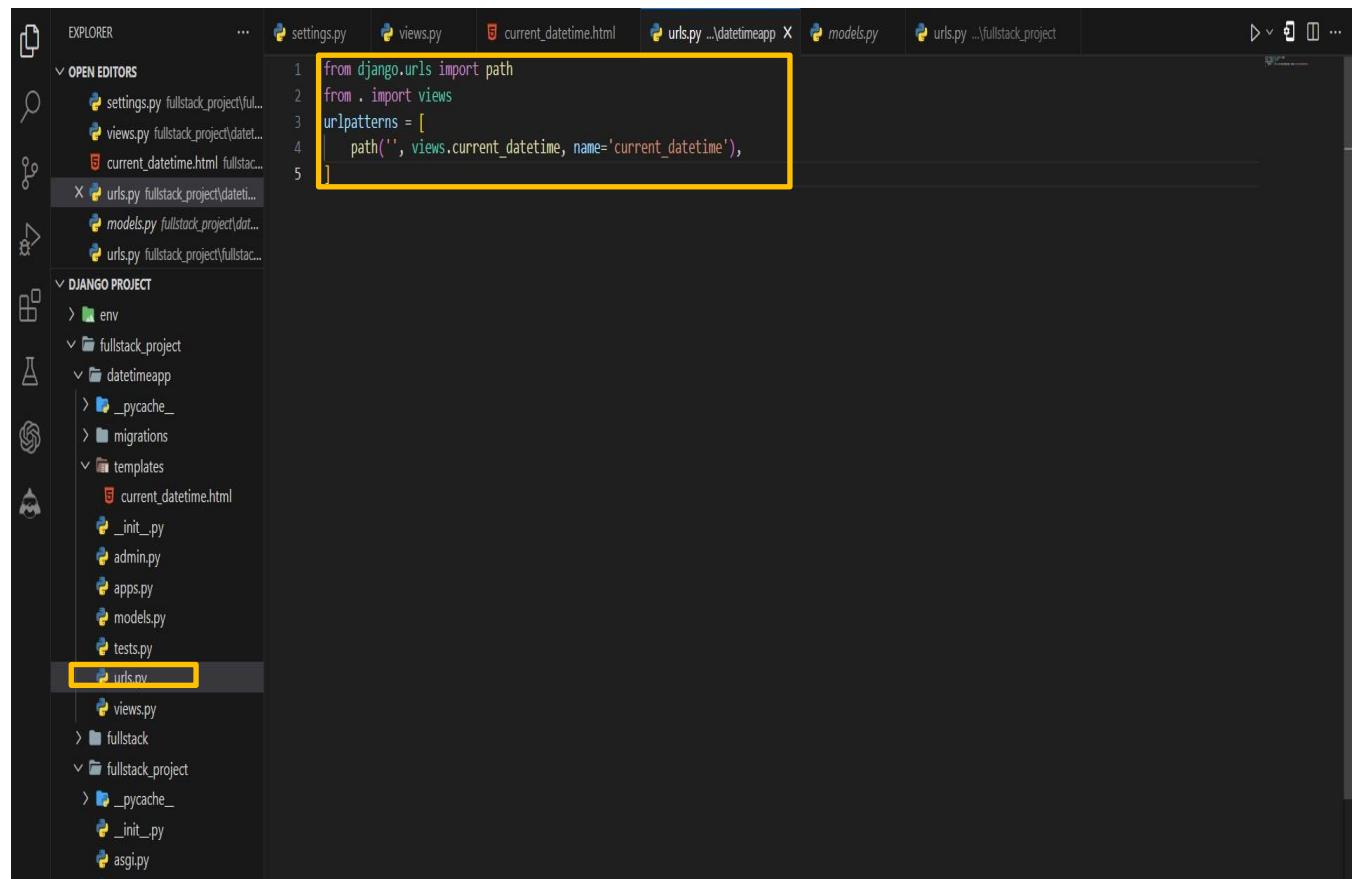
```
from . import views
```

```
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'),
]
```

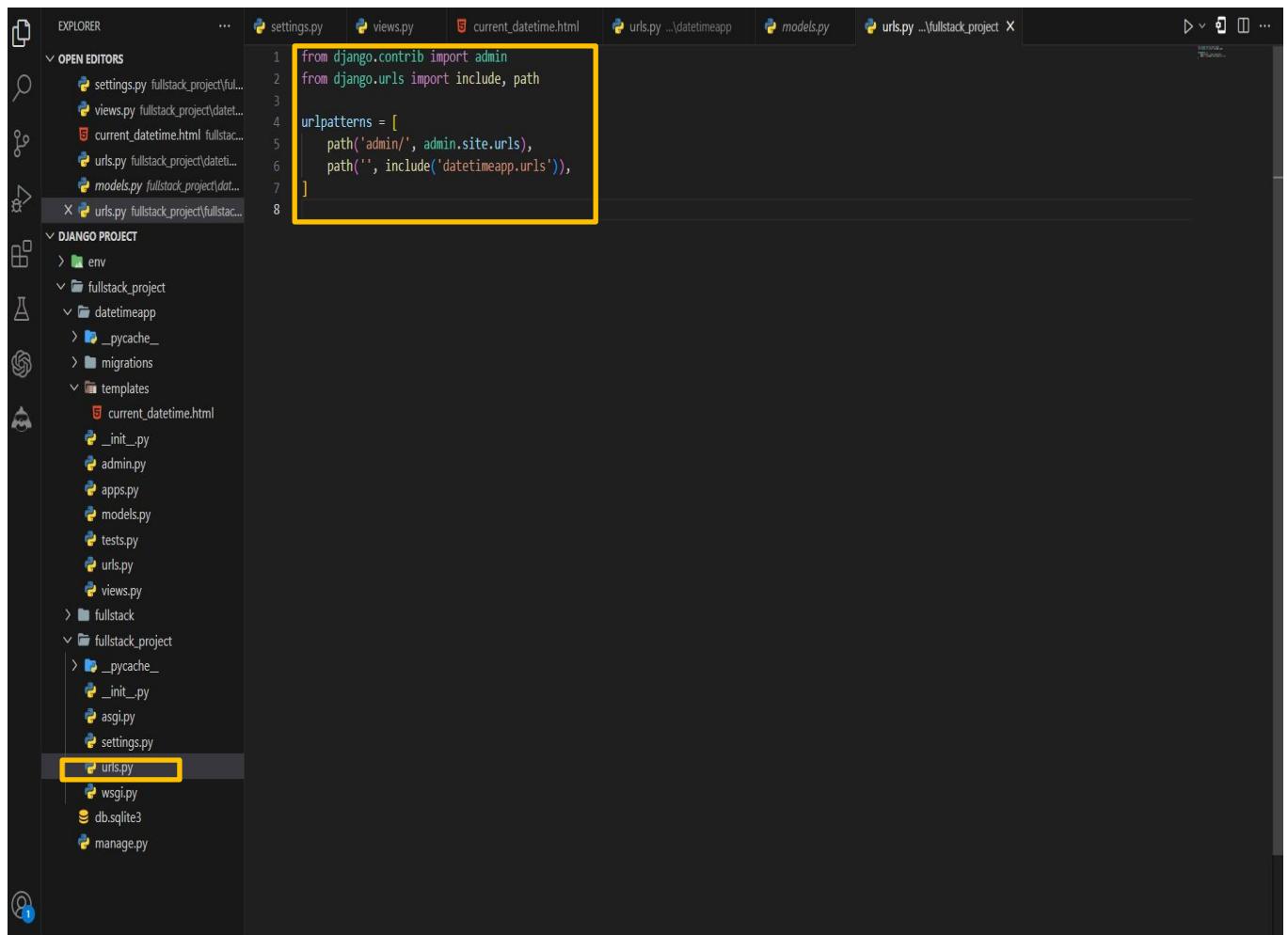
The screenshot shows the `urls.py` file for the `datetimeapp` directory in a code editor. The file contains the code shown above. The entire code block is highlighted with a yellow rectangle. The file path in the sidebar is `fullstack_project\datetimeapp\urls.py`. The sidebar also lists other files like `settings.py`, `views.py`, and `current_datetime.html`.

## 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

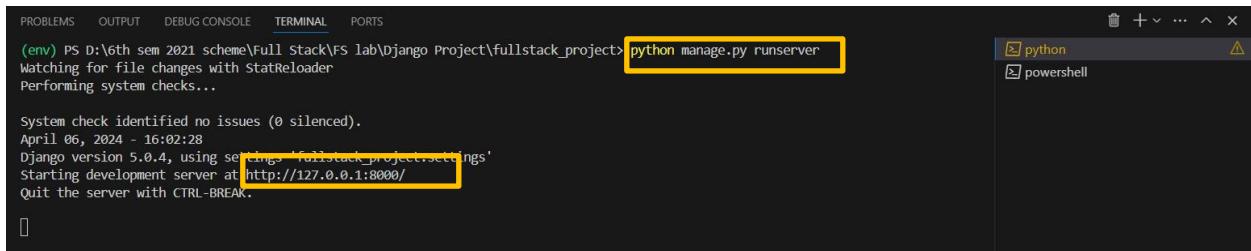
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('datetimeapp.urls')),
]
```

## 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.

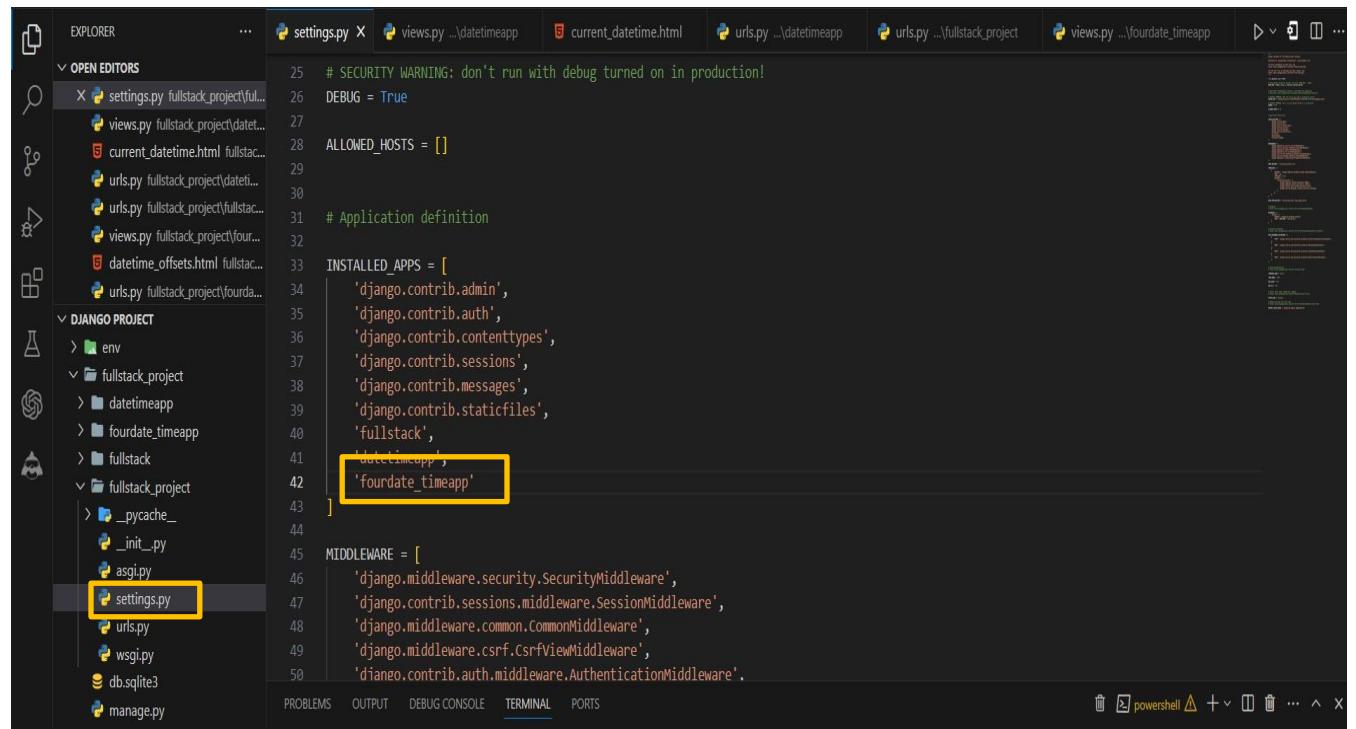
**Step-01:** This app will be created in the Django project we made earlier.

```
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
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     '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,

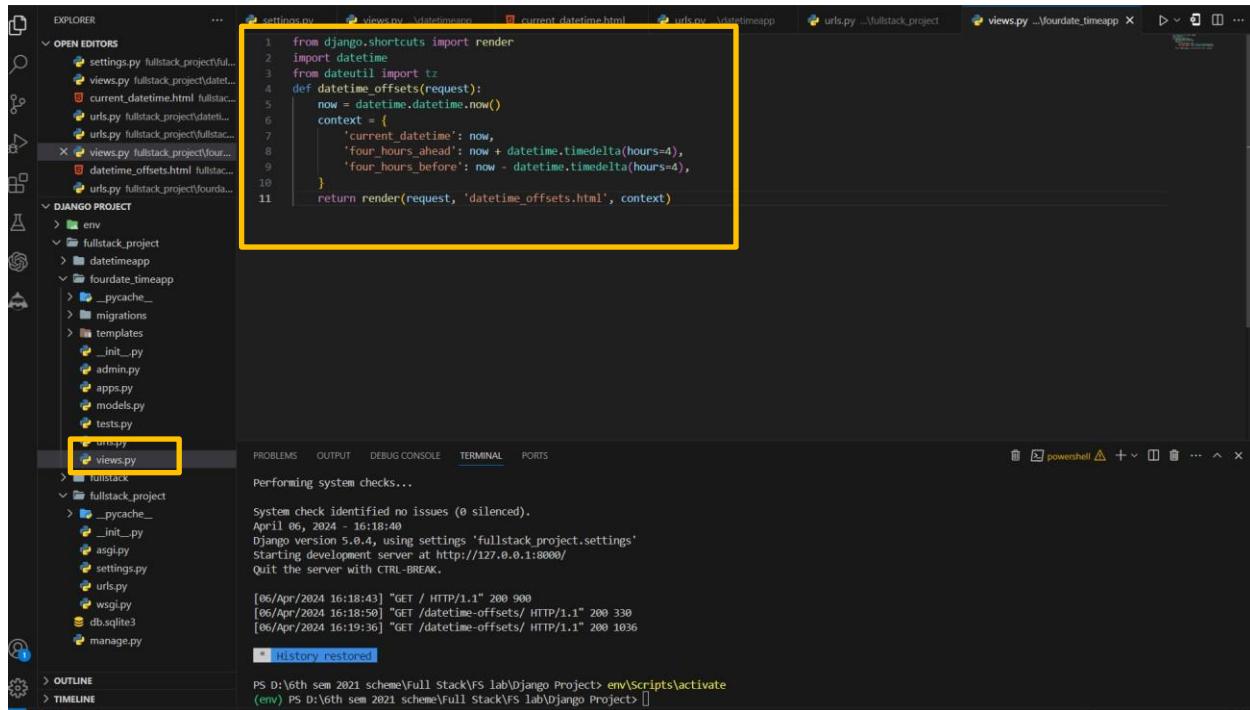
        'four_hours_ahead': now + datetime.timedelta(hours=4),

        '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
12     return render(request, 'datetime_offsets.html', context)

```

The screenshot shows the VS Code interface with the Django project structure in the Explorer sidebar. The file `views.py` is open in the editor, containing the provided Python code. A yellow box highlights the entire code block. The terminal tab shows system checks and Django server startup logs. The status bar indicates the current path is `D:\6th sem 2021 scheme\Full Stack\FS lab\ Django Project>`.

## Step-04: Create a new template

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

```

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

```

```
<title>Date and Time Offsets</title>

<!-- Bootstrap CSS -->

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css"
rel="stylesheet">

<style>

/* Center content vertically */

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 VS Code interface with the following details:

- Explorer View:** Shows the project structure under "OPEN EDITORS". The "templates" folder is expanded, and "datetime\_offsets.html" is selected and highlighted with a yellow box.
- Code Editor:** Displays the content of "datetime\_offsets.html". The code includes HTML for displaying current date/time and four-hour offsets, and a Bootstrap CSS link. It also includes the Bootstrap and Popper JS script imports at the bottom.

```
<!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'),
```

```
]
```

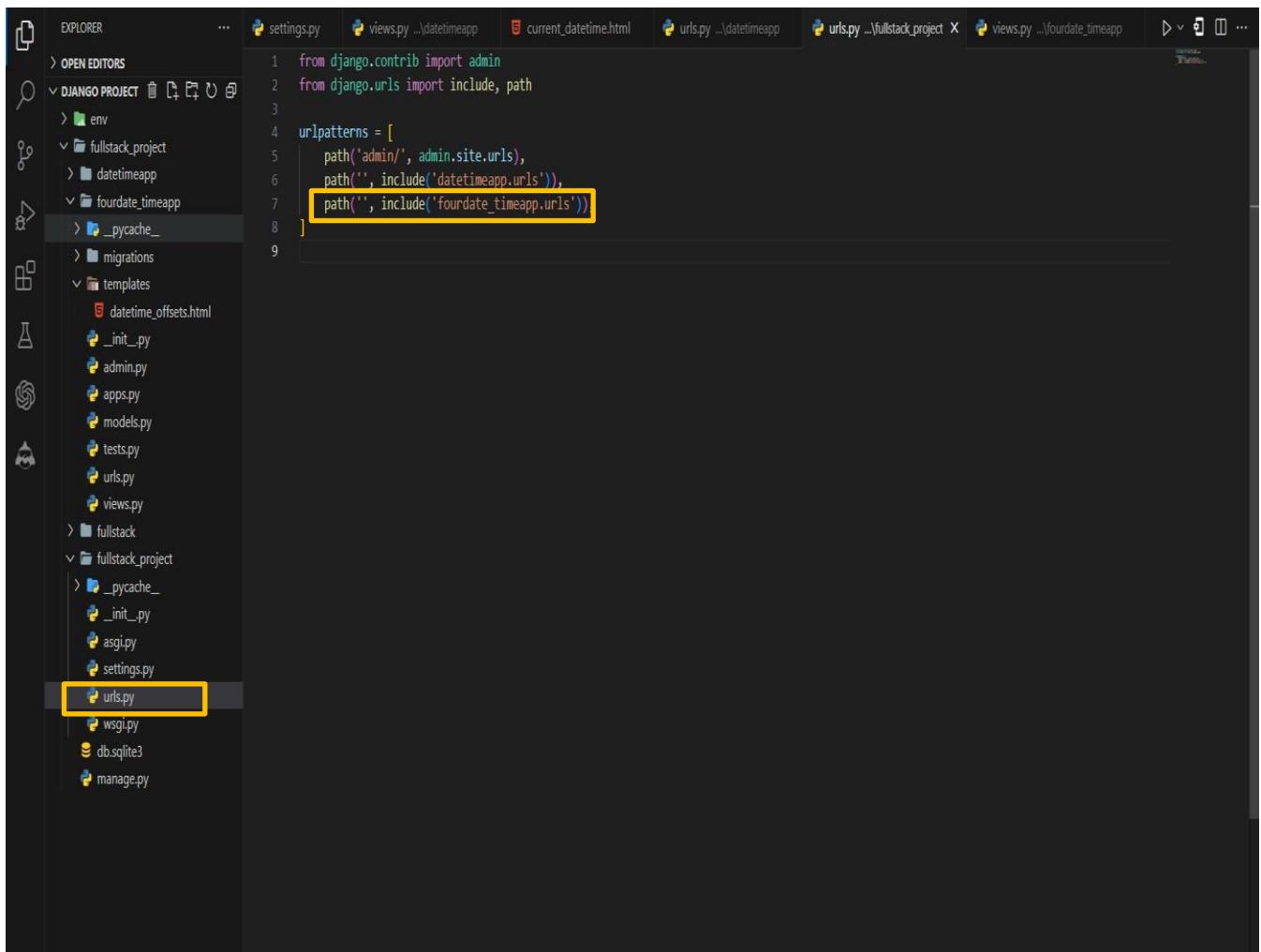
```
from django.urls import path
from . import views
urlpatterns = [
    path('datetime-offsets/', views.datetime_offsets, name='datetime_offsets'),
]
```

## 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

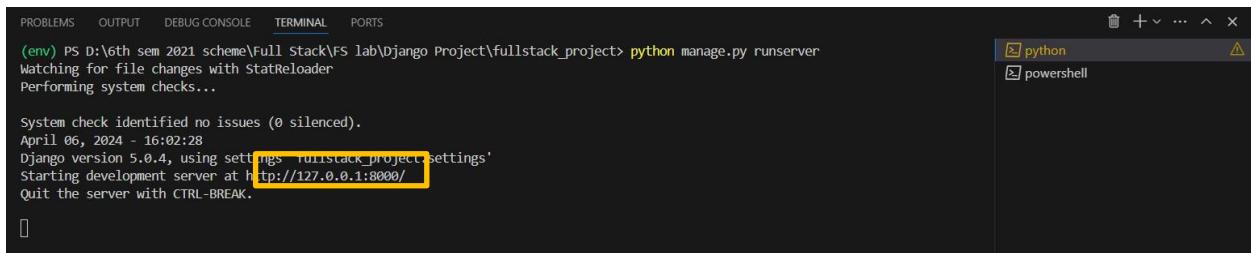
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('datetimeapp.urls')),
    path('', include('fourdate_timeapp.urls'))
]
```

## 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/>



```
(env) PS D:\6th sem 2021 scheme\Full Stack\F5 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`).

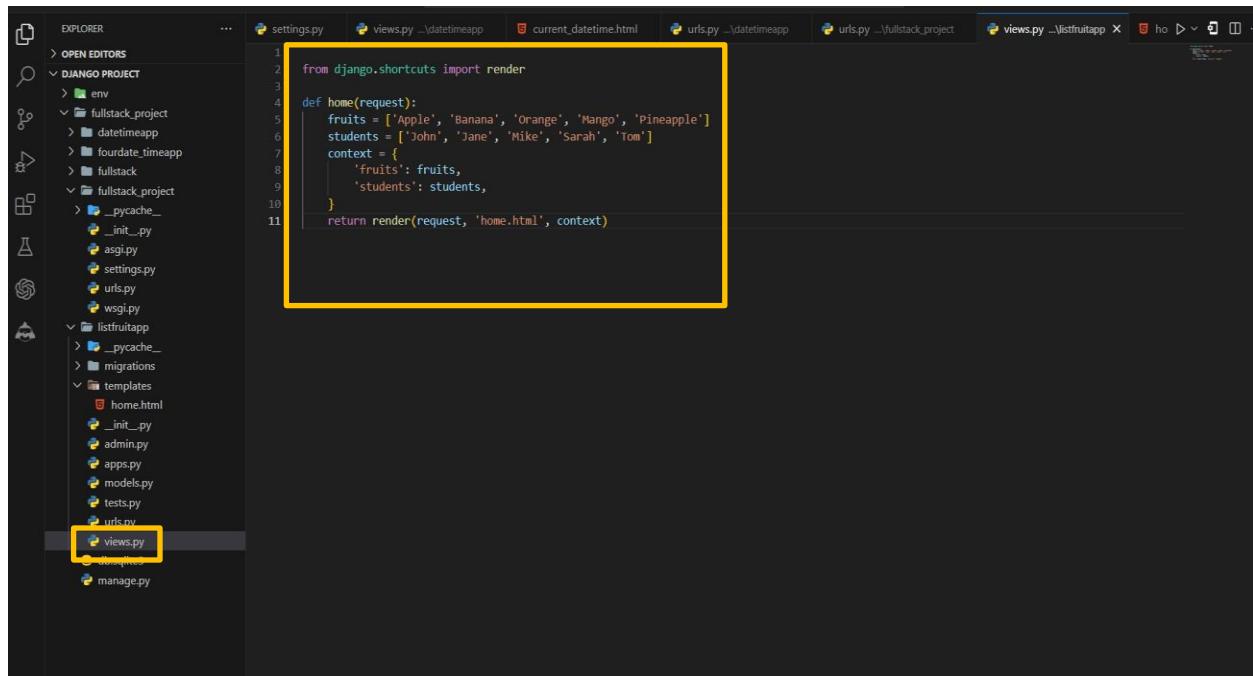
Locate the **INSTALLED\_APPS** list and add the name of your new app to the list:

```
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     ]
72
73     WSGI_APPLICATION = 'listfruitapp.wsgi.application'
74
75     AUTHENTICATION_BACKENDS = [
76         'django.contrib.auth.backends.ModelBackend',
77     ]
78
79     AUTH_PASSWORD_VALIDATORS = [
80         {
81             'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
82         },
83         {
84             'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
85         },
86         {
87             'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
88         },
89         {
90             'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
91         },
92     ]
93
94     LANGUAGE_CODE = 'en-us'
95     TIME_ZONE = 'UTC'
96     USE_I18N = True
97     USE_L10N = True
98     USE_TZ = True
99
100    STATIC_URL = '/static/'
```

## 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** sidebar: Shows the project structure with folders like 'fullstack\_project', 'listfruitapp', and files like 'settings.py', 'views.py', 'urls.py', etc.
- OPEN EDITORS**: A tab bar at the top with several tabs, one of which is 'views.py'.
- Code Editor Area**: The main area displays Python code. The code is highlighted with a yellow box around the entire function definition. The code is as follows:

```
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 }}</li>
```

```
                { % endfor % }
```

```
            </ul>
```

```
</div>

<div class="col">

    <h1>Selected Students</h1>

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

        { % for student in students % }

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

        { % endfor % }

    </ol>

</div>

</div>

</div>

<!-- Bootstrap JS (optional) -->

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

<script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js"></script>

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

</body>

</html>
```

The screenshot shows a code editor interface with a dark theme. On the left is the Explorer sidebar showing a Django project structure with files like `views.py`, `urls.py`, `models.py`, etc. In the center, a code editor window displays `home.html`. The code uses Django's template language to render lists of fruits and students. A yellow box highlights the `home.html` file in the Explorer and the code editor area.

```

<!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 }}</li>
                    {% endfor %}
                </ul>
            </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

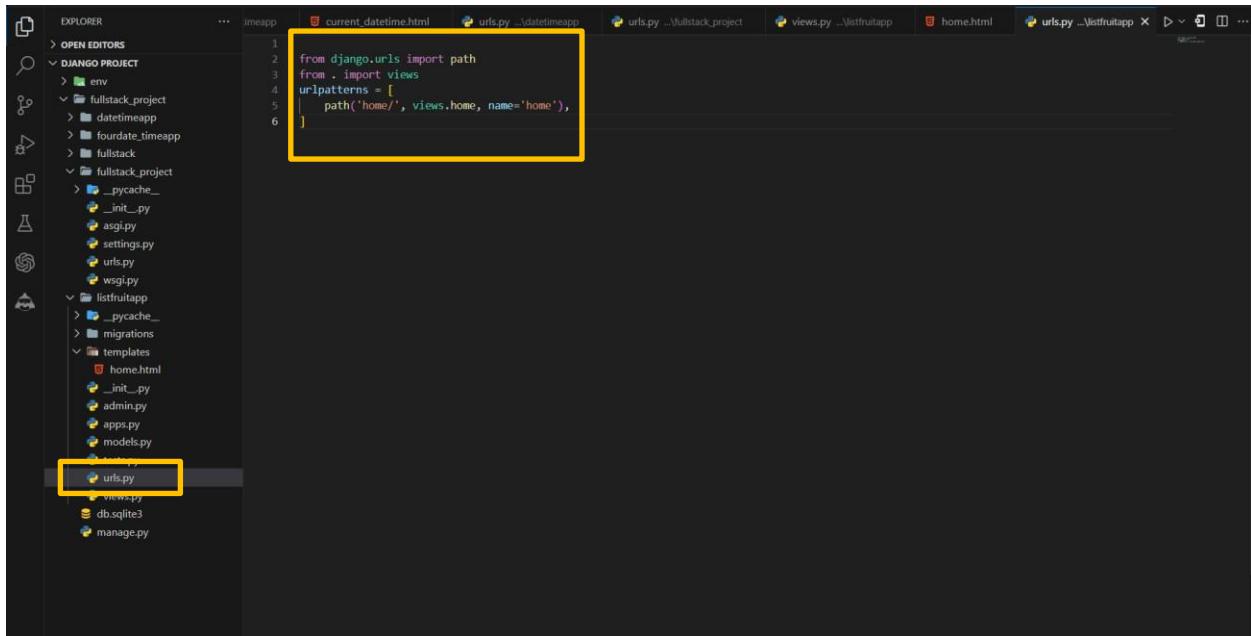
```
from django.urls import path
```

```
from . import views
```

```
urlpatterns = [
```

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

```
]
```



The screenshot shows the VS Code interface with the project structure in the Explorer sidebar. The current file is `urls.py` in the `datetimeapp` directory. The code in the editor is:

```

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

```

The `urls.py` file in the project root is also visible in the Explorer sidebar, highlighted with a yellow box.

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

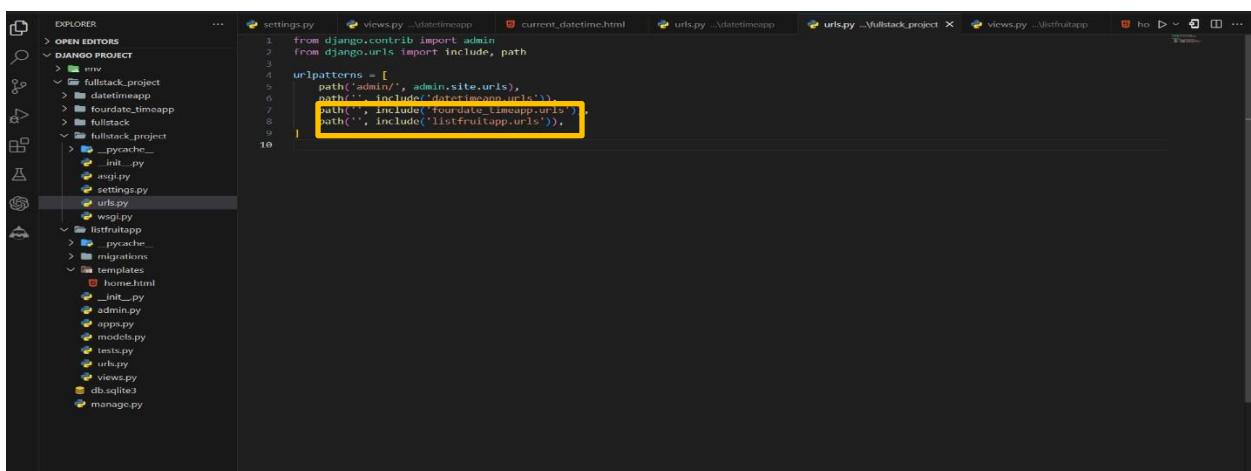
- Open the `urls.py` file in your project's directory (e.g., `fullstack_project/urls.py`).
- Import the `include` function from `django.urls` and the `path` function from `django.urls`:

`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 `settings.py` file in the project root. The `urlpatterns` list now includes the URLs from the `listfruitapp` app:

```

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('datetimeapp.urls')),
7     path('', include('fourdate_timeapp.urls')),
8     path('', include('listfruitapp.urls')),
9 ]

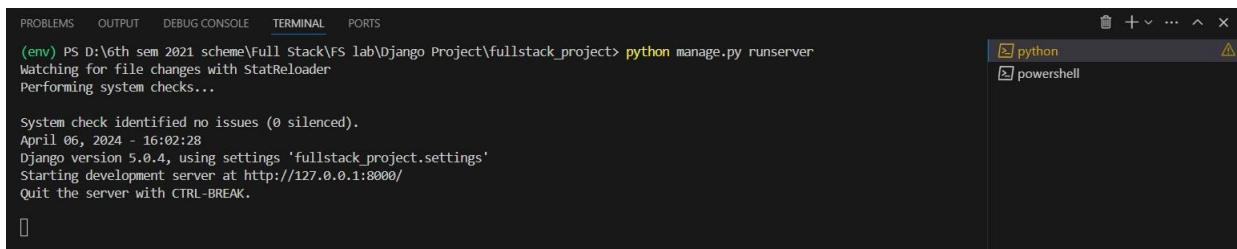
```

## 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/>.

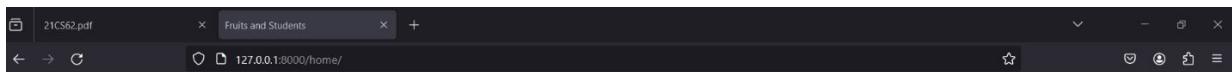


```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
(env) PS D:\6th sem 2021 scheme\Full Stack\FS lab\Django Project\fullstack_project> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

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

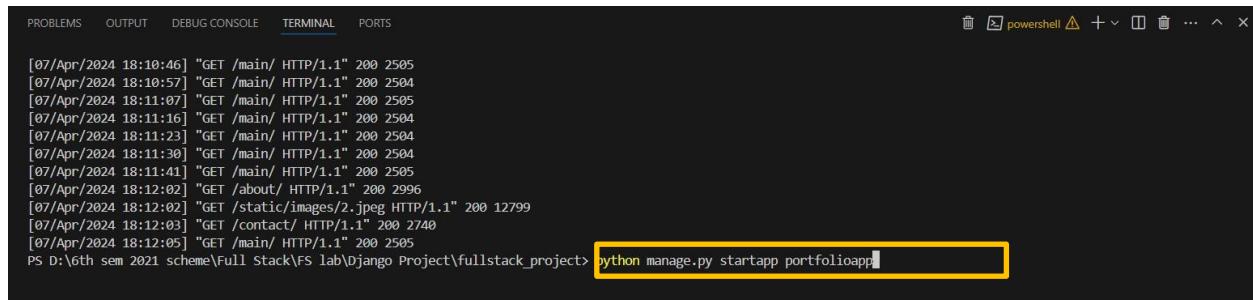


Fruits	Selected Students
Apple	John
Banana	Jane
Orange	Mike
Mango	Sarah
Pineapple	Tom

## 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.**

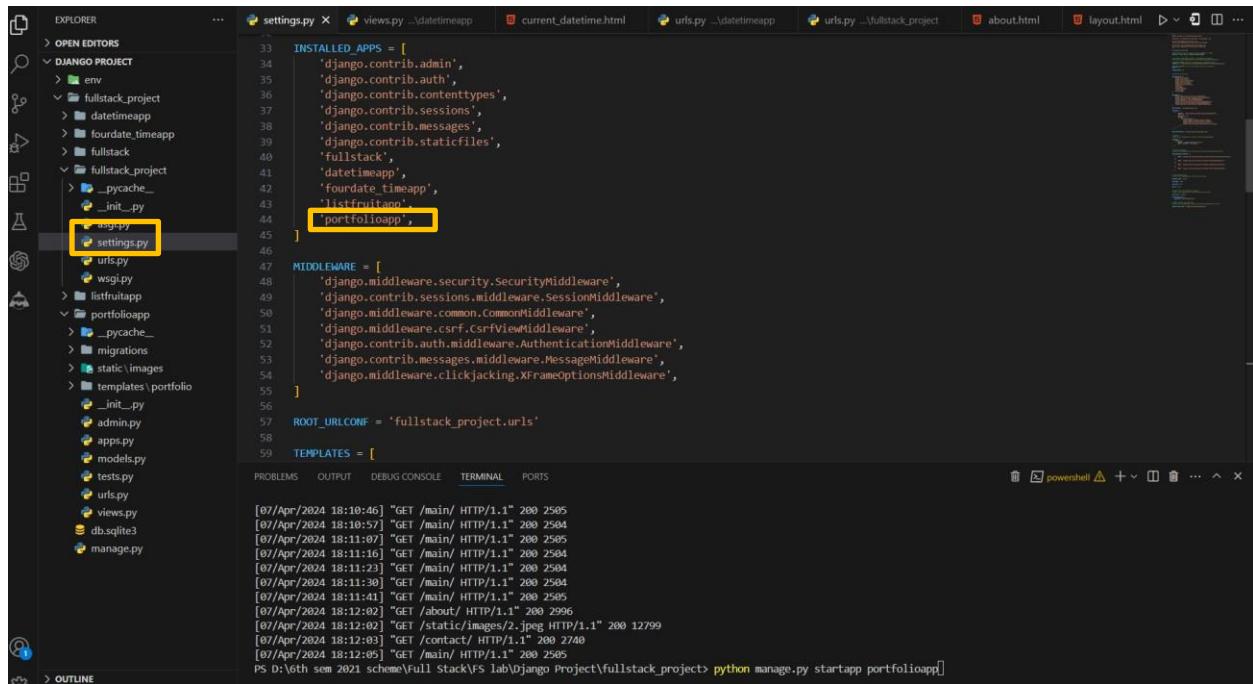


```
[07/Apr/2024 18:10:46] "GET /main/ HTTP/1.1" 200 2505
[07/Apr/2024 18:10:57] "GET /main/ HTTP/1.1" 200 2504
[07/Apr/2024 18:11:07] "GET /main/ HTTP/1.1" 200 2505
[07/Apr/2024 18:11:16] "GET /main/ HTTP/1.1" 200 2504
[07/Apr/2024 18:11:23] "GET /main/ HTTP/1.1" 200 2504
[07/Apr/2024 18:11:30] "GET /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\FS 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',
    'listfrutapp',
    'portfolioapp',
    'portfolioapp',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'fullstack_project.urls'

TEMPLATES = [
```

```
[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\FS lab\ Django Project\fullstack_project> python manage.py startapp 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>

<ul>

<li><a href="{% url 'main' %}" style="text-decoration:none;">Home</a></li>

<li><a href="{% url 'about' %}" style="text-decoration:none;">About Us</a></li>

<li><a href="{% url 'contact' %}" style="text-decoration:none;">Contact Us</a></li>

</ul>
```

```
</nav>

</header>

<div class="content-wrapper">

    {% block content %}

    {% endblock %}

</div>

<footer>

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

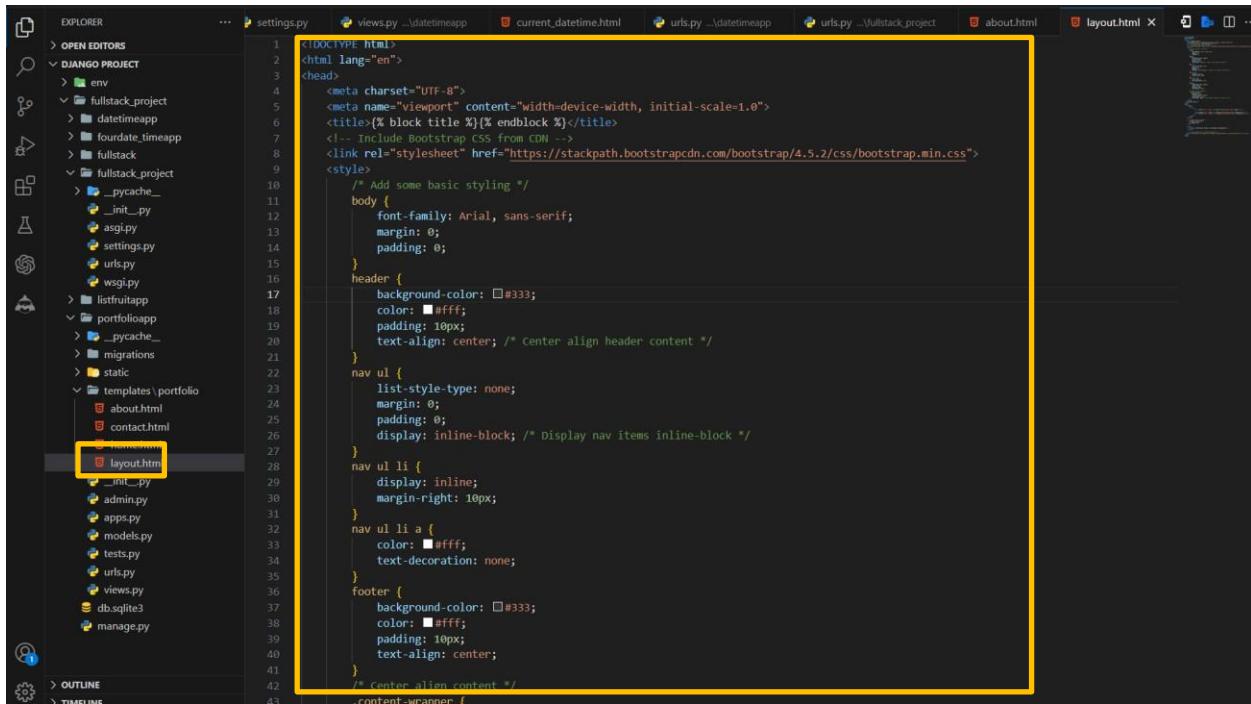
</footer>

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

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

</body>

</html>
```



```
<!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 %}
```

The screenshot shows the VS Code interface with the Django project structure in the Explorer sidebar. The file `home.html` is selected and its content is displayed in the main editor area. The code uses the `block` and `endblock` tags to define sections, includes Bootstrap CSS and JS from CDNs, and centers the content on the page.

```
1  {% extends 'portfolio/layout.html' %} 
2  3  {% block title %}Home{% endblock %}
4  5  {% block extra_css %}
5  6      <!-- Include Bootstrap CSS from CDN -->
6  7      <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
7  8      <!-- Custom CSS to center align content --
8  9          center-content {
9 10             display: flex;
10             justify-content: center;
11             align-items: center;
12             height: 80vh; /* set height to viewport height for full page centering */
13         }
14         center-content > div {
15             text-align: center;
16         }
17     </style>
18     {% endblock %}
19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38
```

## 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>

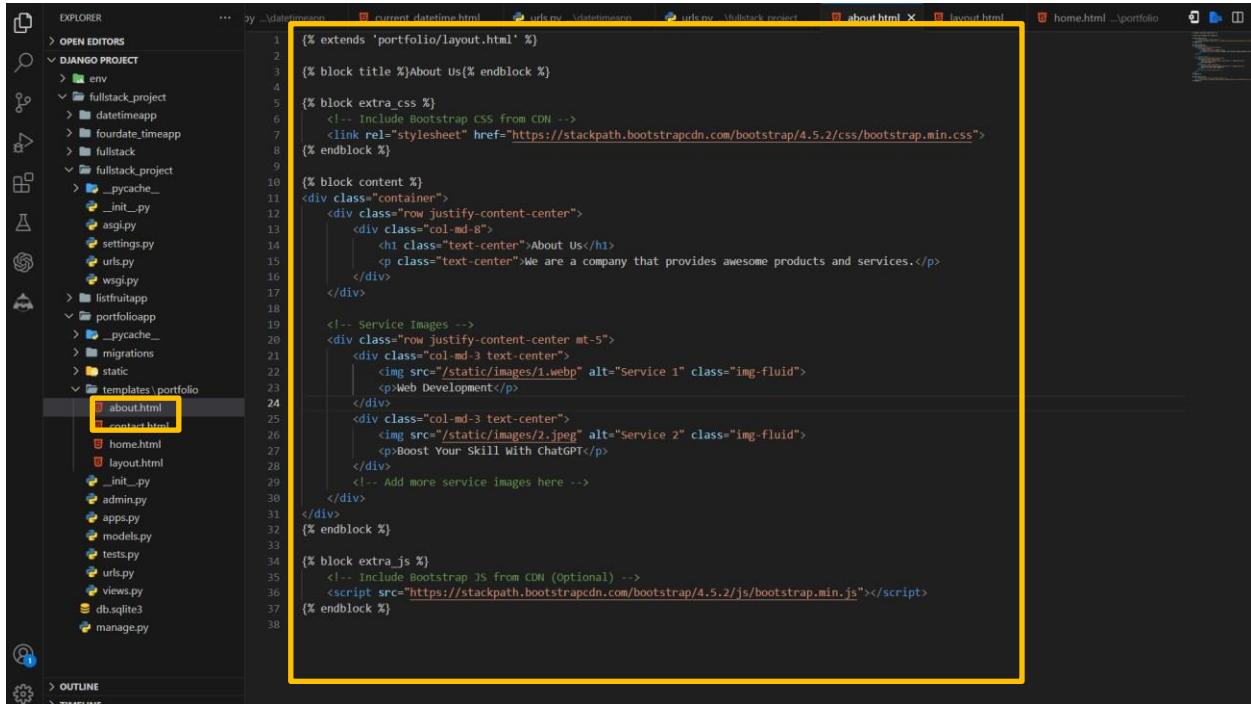
{ % endblock % }

{ % 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>
{% endblock %}

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

## contact.html

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

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

```
{% block extra_css %}
```

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

```
<link rel="stylesheet"
```

```
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
```

```
{% endblock %}
```

```
{% block content %}
```

```
<div class="container">

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

<div class="col-md-6">

    <h1 class="text-center">Contact Us</h1>

    <p class="text-center">You can reach us at:</p>

    <ul class="list-unstyled">

        <li class="text-center">Email: contact@mywebsite.com</li>

        <li class="text-center">Phone: 123-456-7890</li>

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

    </ul>

</div>

</div>

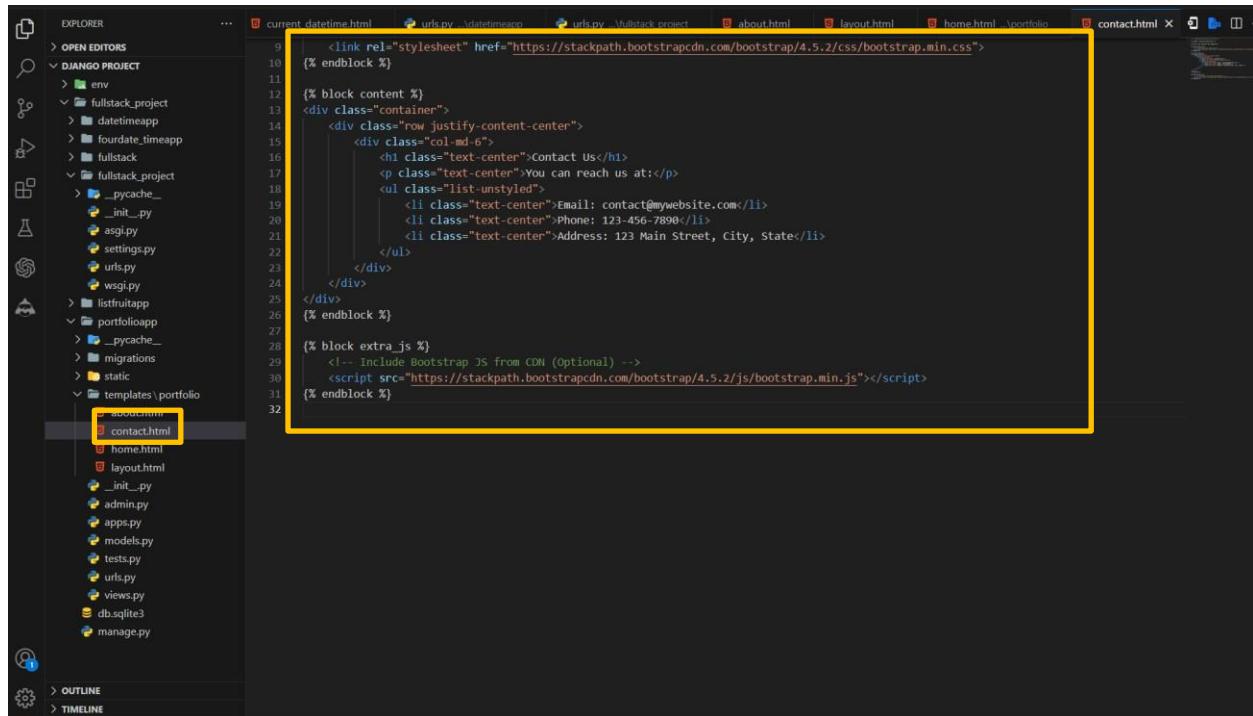
{ % endblock % }

{ % 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 % }
```



```
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
{%
    block content %}
<div class="container">
    <div class="row justify-content-center">
        <div class="col-md-6">
            <h1 class="text-center">Contact Us</h1>
            <p class="text-center">You can reach us at:</p>
            <ul class="list-unstyled">
                <li class="text-center">Email: contact@mywebsite.com</li>
                <li class="text-center">Phone: 123-456-7890</li>
                <li class="text-center">Address: 123 Main Street, City, State</li>
            </ul>
        </div>
    </div>
{%
    block extra_js %}
    <!-- Include Bootstrap JS from CDN (Optional) -->
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
{%
    endblock %}
```

## Step-04: Create a view function

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

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

```
def main(request):
```

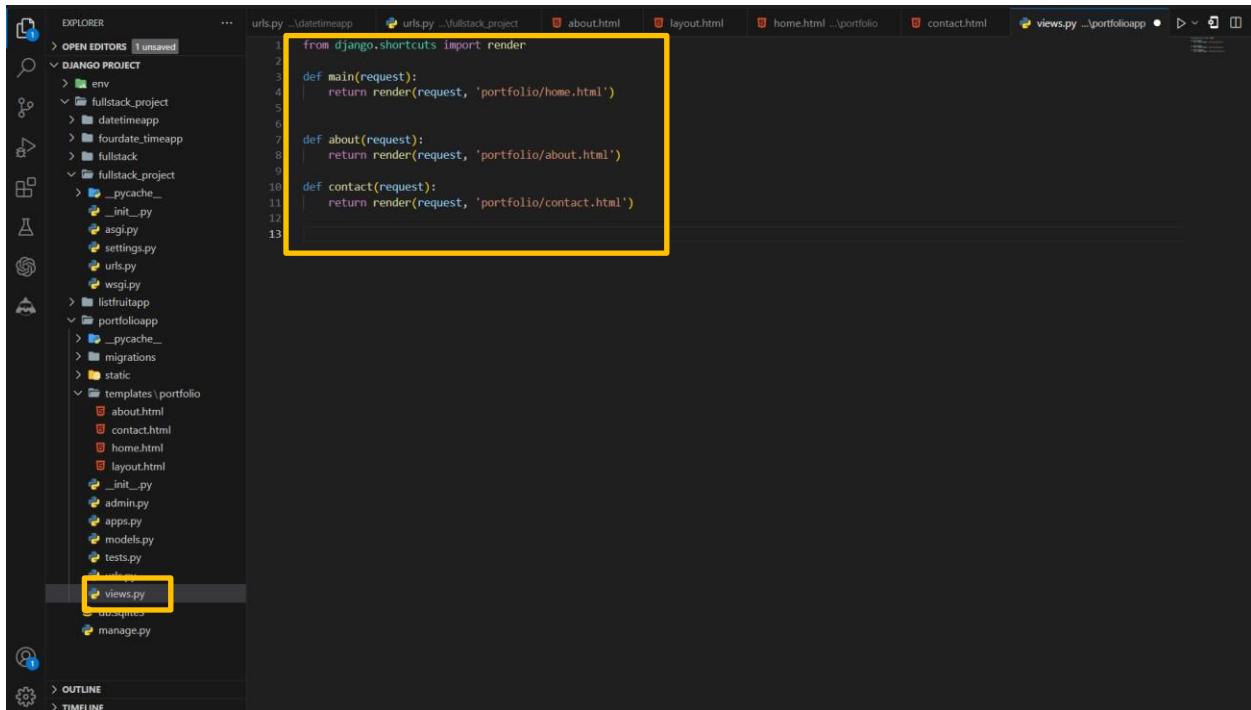
```
    return render(request, 'portfolio/home.html')
```

```
def about(request):
```

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

```
def contact(request):
```

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



```
from django.shortcuts import render

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

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

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

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

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

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

```
from . import views
```

```
urlpatterns = [
    path('main/', views.main, name='main'),
    path('about/', views.about, name='about'),
    path('contact/', views.contact, name='contact'),
]
```

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

- EXPLORER:** Shows the project structure under "DJANGO PROJECT". It includes a "fullstack\_project" folder containing "env", "fullstack", "fullstack\_timeapp", "listfruitapp", and "portfolioapp". The "portfolioapp" folder contains "\_\_pycache\_\_", "migrations", "static", and "templates\portfolio". Inside "templates\portfolio", there is a "urls.py" file which is highlighted with a yellow box.
- EDITOR:** The "urls.py" file is open in the editor. The code is as follows:

```

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'),
]

```
- STATUS BAR:** Shows "11" at the bottom left, indicating the current line number.

## 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 a "fullstack\_project" folder containing "env", "fullstack", "fullstack\_timeapp", and "listfruitapp". The "listfruitapp" folder contains "\_\_pycache\_\_", "migrations", "static", and "templates". Inside "templates", there is a "urls.py" file which is highlighted with a yellow box.
- EDITOR:** The "urls.py" file is open in the editor. The code is as follows:

```

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

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

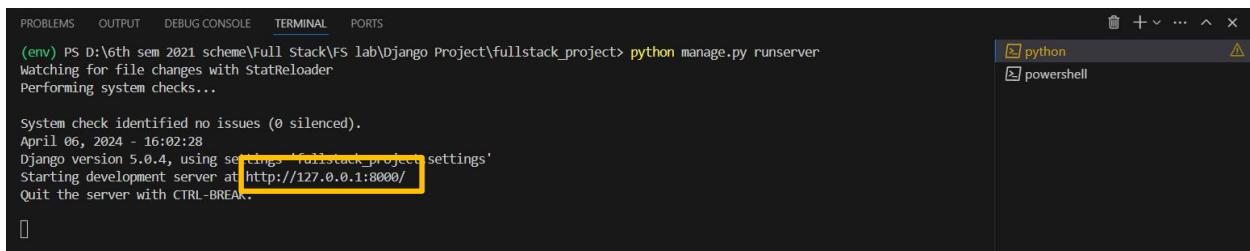
```
- STATUS BAR:** Shows "11" at the bottom left, indicating the current line number.

## 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/main/>

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

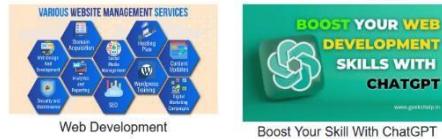


**Fig: Home Page Screen**

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.

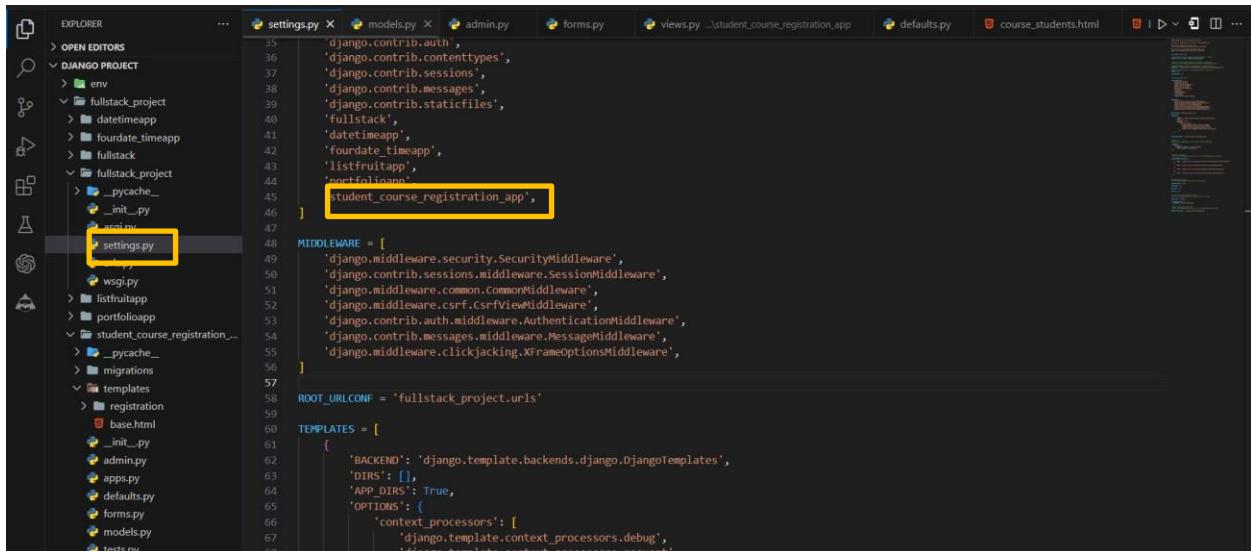
**Step-01: This app will be created in the Django project we made earlier.**

```
[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
> datetimapp
> foundate_timeapp
> fullstack
fullstack_project
> __pycache__
> __init__.py
> asgi.py
> settings.py
> listruitapp
> portfolioapp
student_course_registration_
> __pycache__
> migrations
> templates
> registration
> base.html
> __init__.py
> admin.py
> apps.py
> forms.py
> models.py
> tests.py

35     django.contrib.auth,
36     'django.contrib.contenttypes',
37     'django.contrib.sessions',
38     'django.contrib.messages',
39     'django.contrib.staticfiles',
40     'fullstack',
41     'datetimapp',
42     'foundate_timeapp',
43     'listruitapp',
44     'portfolioapp',
45     'student_course_registration_app',
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.contrib.auth.middleware.AuthenticationMiddleware',
54     'django.contrib.messages.middleware.MessageMiddleware',
55     'django.middleware.clickjacking.XFrameOptionsMiddleware',
56 ]
57
58 ROOT_URLCONF = 'fullstack_project.urls'
59
60 TEMPLATES = [
61     {
62         'BACKEND': 'django.template.backends.django.DjangoTemplates',
63         'DIRS': [],
64         'APP_DIRS': True,
65         'OPTIONS': {
66             'context_processors': [
67                 '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)

```

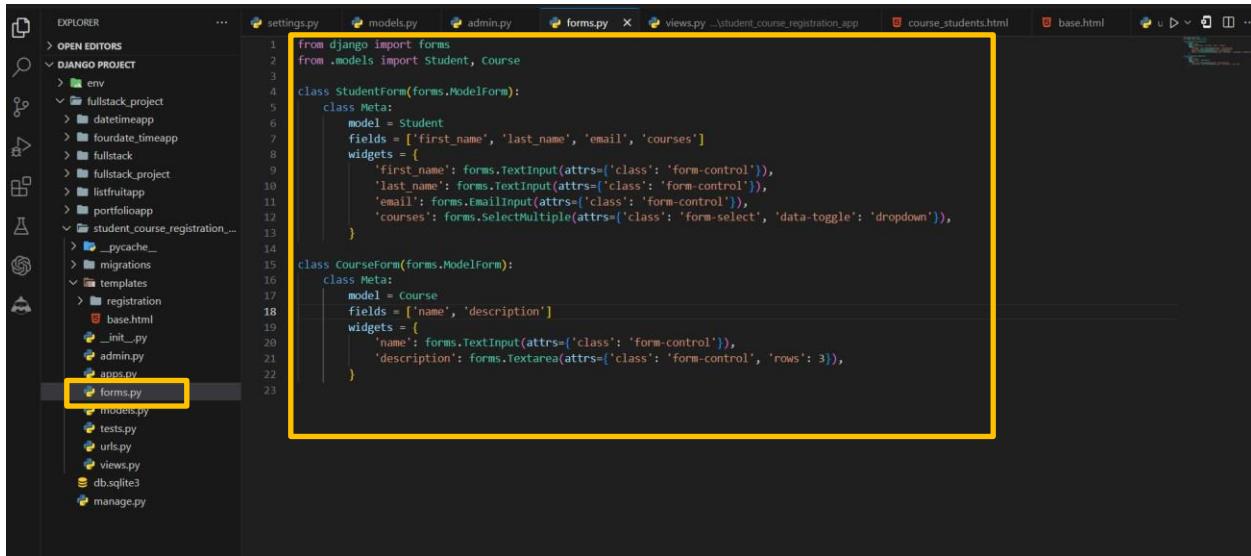
### Step-05: Create forms

- 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:
```

```
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})
```

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

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

def register_student(request):
    if request.method == 'POST':
        form = StudentForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('index')
    else:
        form = StudentForm()
    return render(request, 'registration/register_student.html', {'form': form})

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

def student_list(request, course_id):
    course = Course.objects.get(id=course_id)
    students = course.students.all()
    return render(request, 'registration/student_list.html', {'students': students, 'course': course})
```

## Step-07: Create templates

- Create a new directory called templates inside your student\_course\_registration\_app, and create the following template files:

### index.html

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

{% block content %}

<div class="container">

<h1 class="mt-5">Courses</h1>

<ul class="list-group mt-3">

{% for course in courses %}

<li class="list-group-item"><a href="{% url 'student_list' course.id %}">{{ course.name }}</a></li>

{% endfor %}

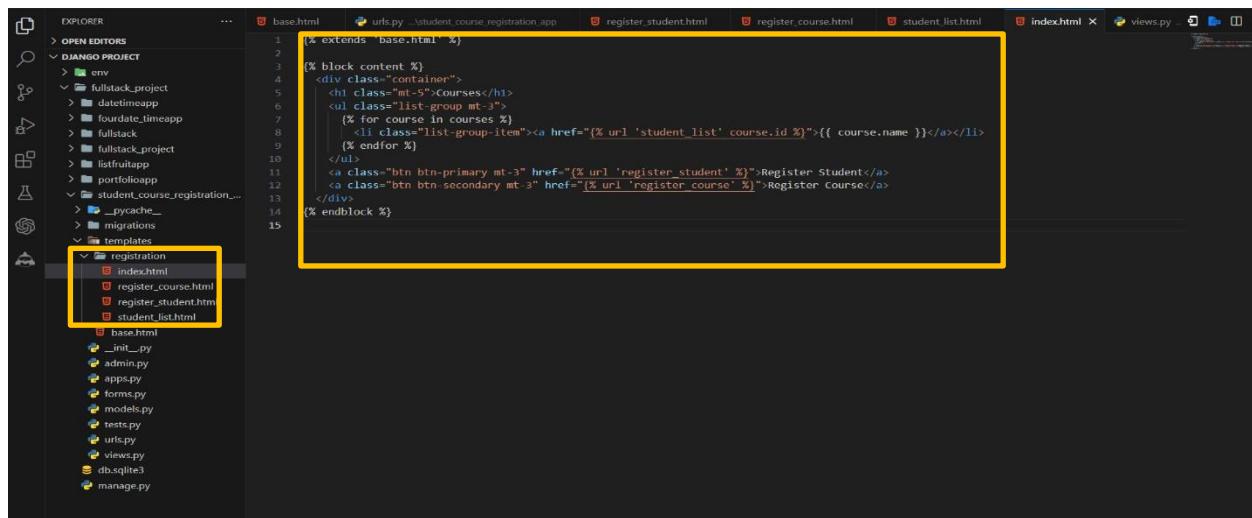
</ul>

<a class="btn btn-primary mt-3" href="{% url 'register_student' %}">Register Student</a>

<a class="btn btn-secondary mt-3" href="{% url 'register_course' %}">Register Course</a>

</div>

{% endblock %}
```



```
1  {% extends 'base.html' %}           base.html  urls.py -\student_course_registration_app  register_student.html  register_course.html  student_list.html  index.html  views.py  ...
2
3  {% block content %}               ...
4  <div class="container">           ...
5  <h1 class="mt-5">Courses</h1>       ...
6  <ul class="list-group mt-3">         ...
7  {% for course in courses %}       ...
8  <li class="list-group-item"><a href="{% url 'student_list' course.id %}">{{ course.name }}</a></li>
9  {% endfor %}                     ...
10 </ul>                           ...
11 <a class="btn btn-primary mt-3" href="{% url 'register_student' %}">Register Student</a>
12 <a class="btn btn-secondary mt-3" href="{% url 'register_course' %}">Register Course</a>
13 </div>                          ...
14
15
```

### 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>

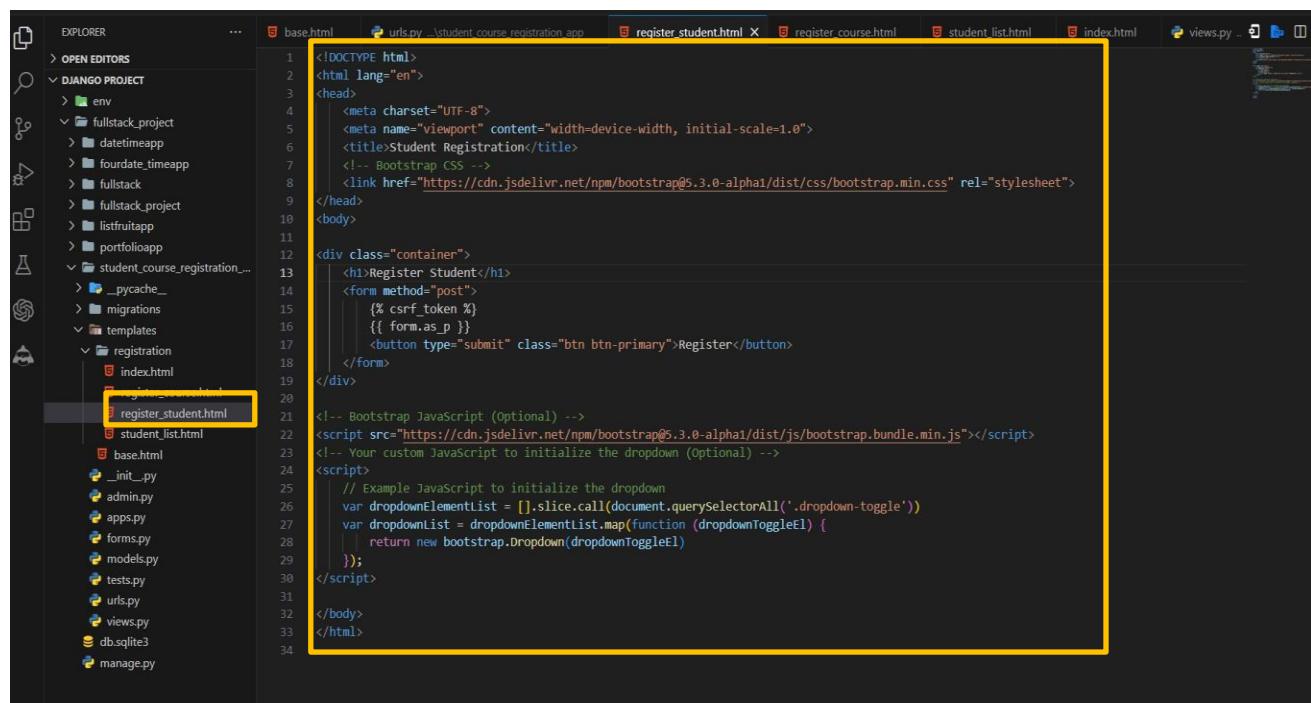
    <!-- Your custom JavaScript to initialize the dropdown (Optional) -->

    <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>

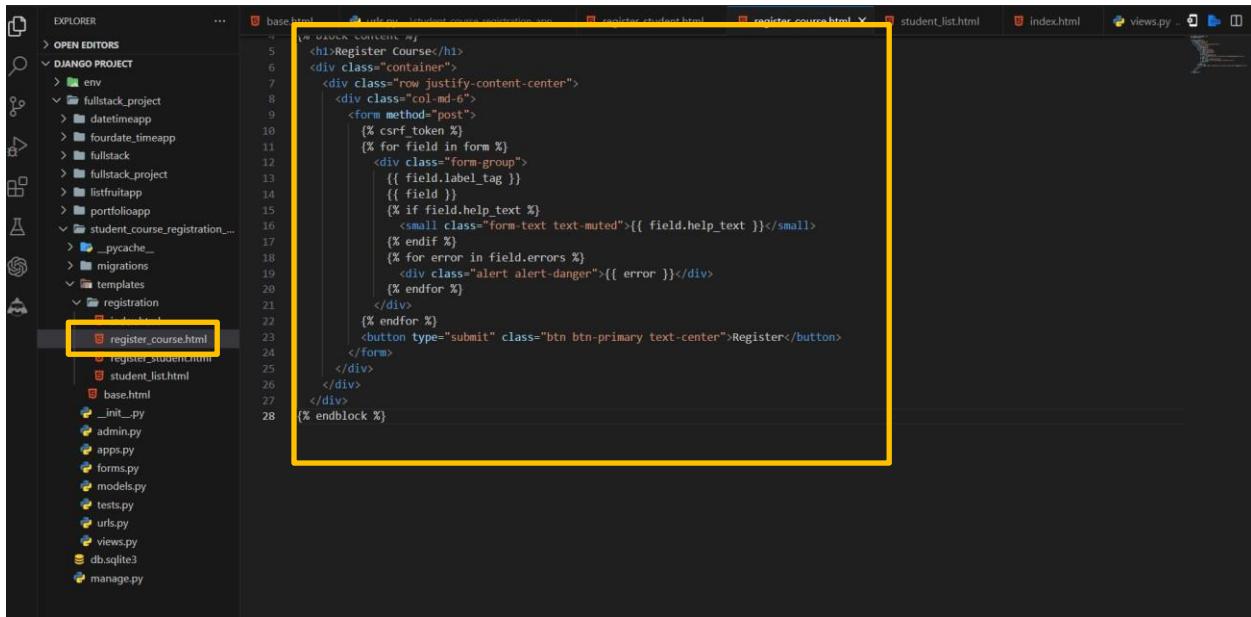
</form>

</div>

</div>

</div>

    {% endblock %}
```



The screenshot shows a code editor interface with a dark theme. On the left is the 'EXPLORER' sidebar, which lists a 'DJANGO PROJECT' folder containing various apps like 'env', 'fullstack\_project', 'datetimeapp', 'foudrate\_timeapp', 'fullstack', 'listfruitapp', 'portfolioapp', and 'student\_course\_registration...'. Inside this last folder, there are 'migrations', 'templates' (containing 'registration' and 'student\_list.html'), and other files like 'base.html', 'register\_course.html' (which is highlighted with a yellow box), 'student\_list.html', 'views.py', etc. On the right is the main editor area showing the content of 'register\_course.html'. The code is a Django template with HTML and Python logic:

```
5 <h1>Register Course</h1>
6 <div class="container">
7   <div class="row justify-content-center">
8     <div class="col-md-6">
9       <form method="post">
10      {% csrf_token %}
11      {% for field in form %}
12        <div class="form-group">
13          {{ field.label_tag }}
14          {{ field }}
15          {% if field.help_text %}
16            <small class="form-text text-muted">{{ field.help_text }}</small>
17          {% endif %}
18          {% for error in field.errors %}
19            <div class="alert alert-danger">{{ error }}</div>
20          {% endfor %}
21        </div>
22      {% endfor %}
23      <button type="submit" class="btn btn-primary text-center">Register</button>
24    </form>
25  </div>
26 </div>
27 </div>
28 {% endblock %}
```

## student\_list.html

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

{% block content %}

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

<ul>

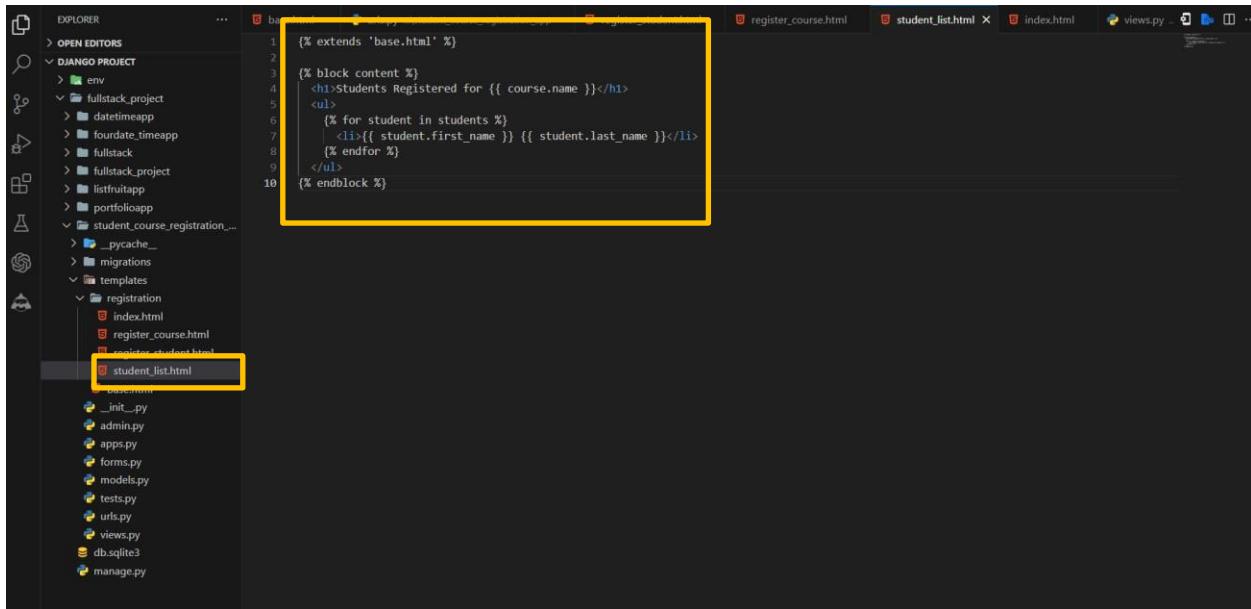
  {% for student in students %}

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

  {% endfor %}

</ul>

{% endblock %}
```



The screenshot shows a code editor interface with a sidebar labeled 'EXPLORER'. The sidebar lists a 'DJANGO PROJECT' folder containing several apps: 'fullstack\_project', 'datetimeapp', 'foudrate\_timeapp', 'fullstack', 'listfruitapp', 'portfolioapp', and 'student\_course\_registration...'. Inside 'student\_course\_registration...', there are files like '\_pycache\_...', 'migrations', and 'templates'. The 'templates' folder contains 'registration' and 'student' subfolders. In the 'student' folder, 'index.html', 'register\_course.html', and 'student.list.html' are listed. The 'student.list.html' file is highlighted with a yellow box. The main editor area shows the content of 'student.list.html':

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

{% block content %}
<h1>Students Registered for {{ course.name }}</h1>
<ul>
    {% for student in students %}
        <li>{{ student.first_name }} {{ student.last_name }}</li>
    {% endfor %}
</ul>
{% endblock %}
```

## Step-08: Create a base template

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

### base.html

```
<!DOCTYPE html>

<html>
    <head>
        <title>Student Registration</title>
    </head>
    <body>
        {% block content %}

        {% endblock %}
    </body>
</html>
```

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

- Left Sidebar:** Shows the "EXPLORER" view with a tree structure of the Django project. The "student\_course\_registration\_app" folder is expanded, showing its contents: \_\_pycache\_\_, migrations, templates, registration, and static files for index.html, register.course.html, and register.student.html.
- Bottom Left:** A list of files in the current workspace:
  - base.html
  - \_\_init\_\_.py
  - admin.py
  - apps.py
  - forms.py
  - models.py
  - tests.py
  - urls.py
  - views.py
  - db.sqlite3
  - manage.py
- Central Area:** The "base.html" file is open in the editor. It contains the following code:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Student Registration</title>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" integrity="sha384-JcKbqoWwD+rqUgkXhPjPZy7JzC0Ddycq4u4QZuOOGPmFy38WbFnE1ibyG61z" data-bbox="270 100 970 200"/>
    <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-DfxzPH0lSs5nCtpu/jy4C0GpmFvY38WbFnE1ibyG61z" data-bbox="270 210 970 310"/>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js" integrity="sha384-9/reFTGAW83EW2RDu250KaIzap3H6G1Z" data-bbox="270 320 970 420"/>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js" integrity="sha384-B4gt1jrGC7h4AgTPsDUtOBvf08shuf" data-bbox="270 430 970 530"/>
  </head>
  <body>
    {# block content %}
    {# endblock %}

    <!-- Bootstrap JavaScript (Optional, if you need Bootstrap JavaScript features) -->
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-alpha1/dist/js/bootstrap.bundle.min.js" integrity="sha384-ygbV56/W8NvPiYx+Za+LlW4fK37W8oXgq+Q7RrQXb0QX+MzJXG8DgE0" data-bbox="270 540 970 640"/>
  </body>
</html>
```
- Top Bar:** Shows tabs for "base.html", "urls.py", "register\_student.html", "register\_course.html", "student\_list.html", "index.html", and "views.py".

## Step-09: Configure URLs

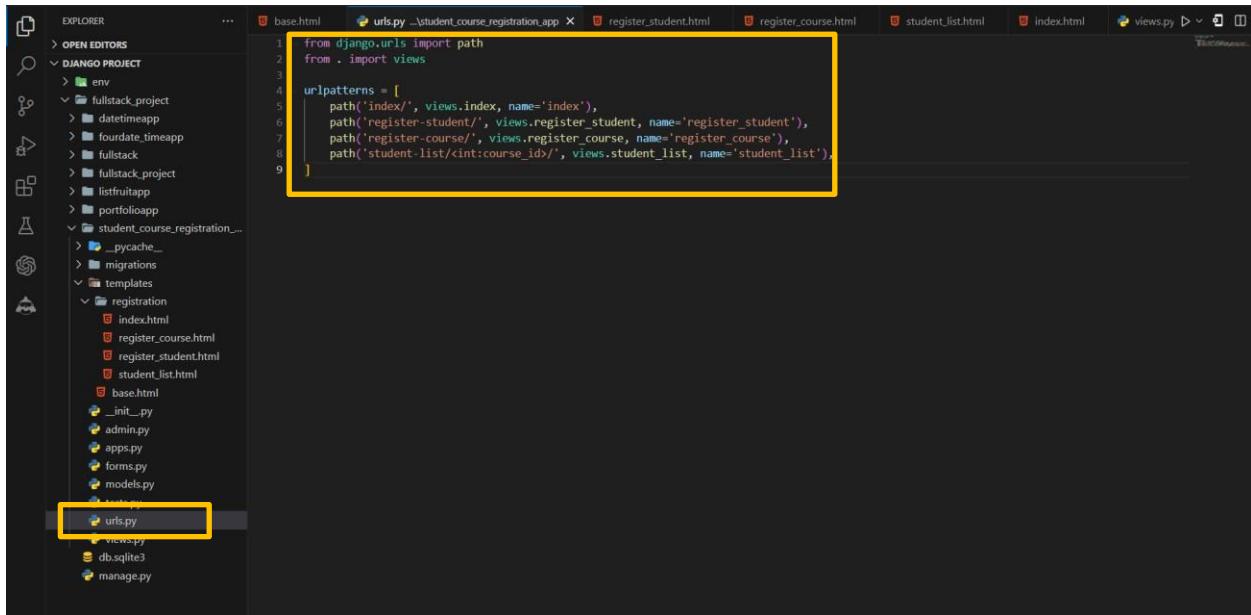
- Open the urls.py file inside your student\_course\_registration\_app and define your URLs:

```
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'),  
]
```



```

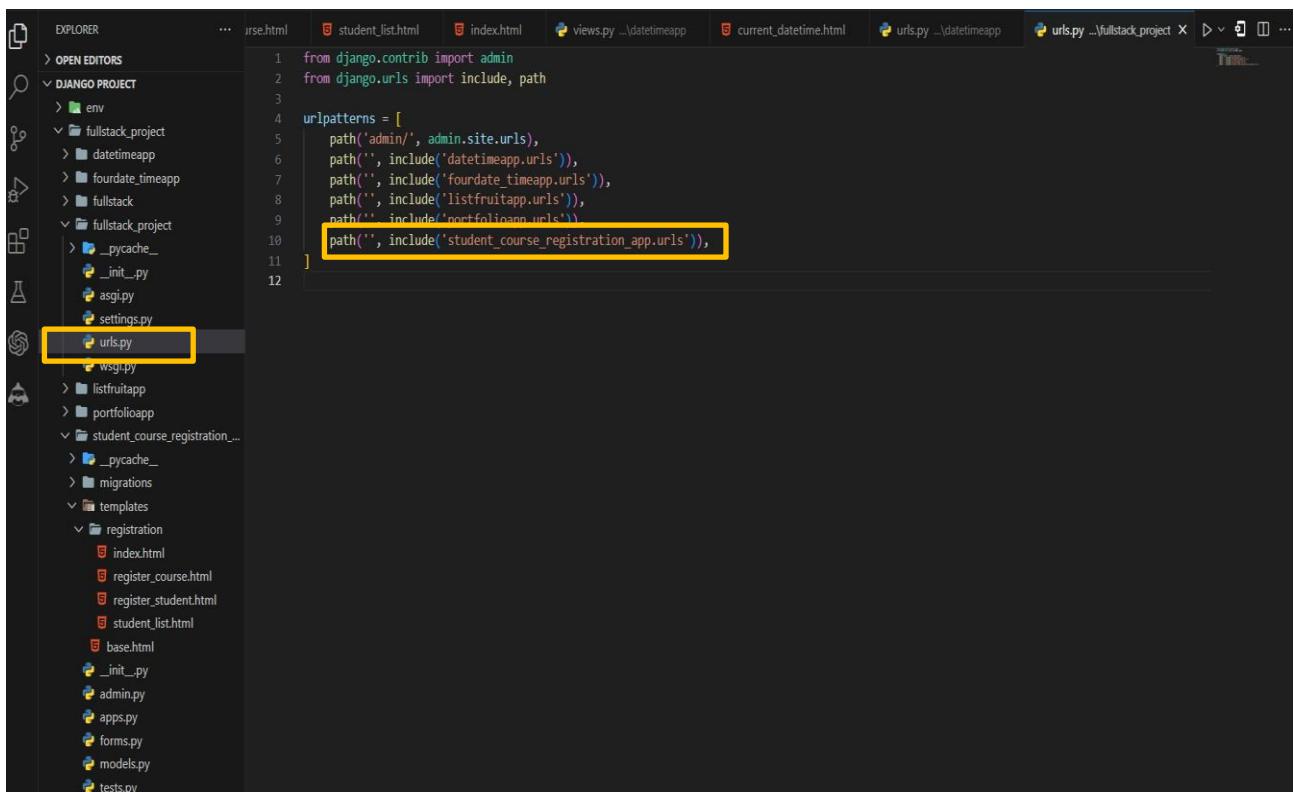
from django.urls import path
from . import views

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('foudrante_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

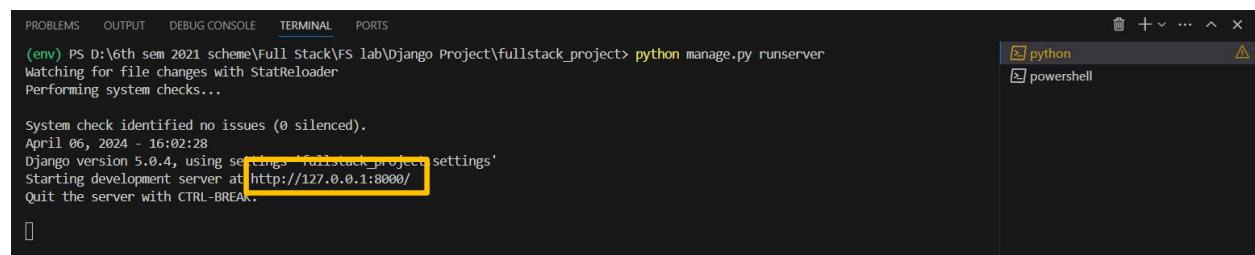
```
PS D:\6th sem 2021 scheme\Full Stack\FS lab\ Django Project\fullstack_project> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions, student_course_registration_app
Running migrations:
  Applying student_course_registration_app.0002_remove_student_name_student_courses_and_more... 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 statfileloader
Performing system checks...
System check identified no issues (0 silenced).
April 06, 2024 - 16:02:28
Django version 5.0.4, using settings 'fullstack_project.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

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

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

The screenshot shows a web browser window with the URL 127.0.0.1:8000/index/. The title bar says "Courses". Below it is a table with two rows: "DBMS" and "CN". At the bottom are two buttons: "Register Student" (blue) and "Register Course" (grey).

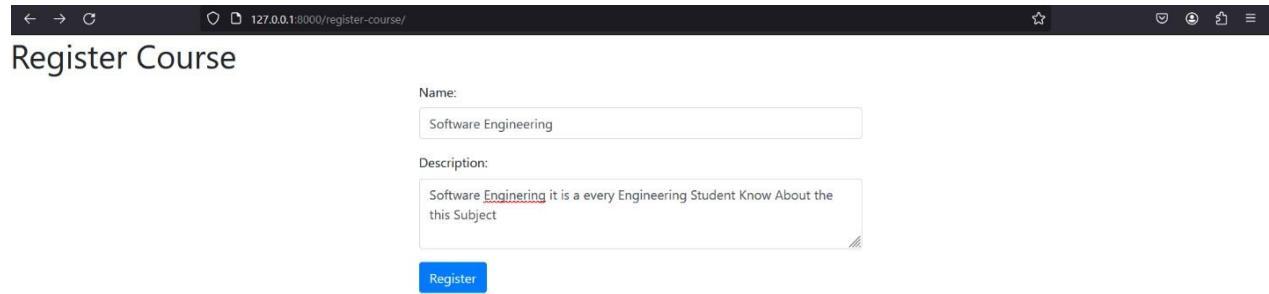
**Fig: Index Screen**

The screenshot shows a web browser window with the URL 127.0.0.1:8000/register-student/. The title bar says "Register Student". It contains four input fields: "First name" (Hanumanthu), "Last name" (Hanu), "Email" (hanu@gmail.com), and a "Courses" dropdown menu with options "DBMS" and "CN". A "Register" button is at the bottom.

**Fig: Student Register Screen**

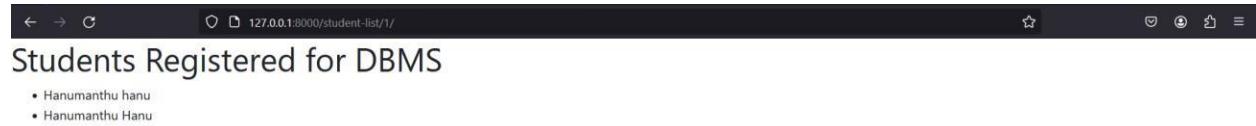
## 21CS62 | Full Stack Django Development|

---



A screenshot of a web browser showing a form to register a course. 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 students registered 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):
```

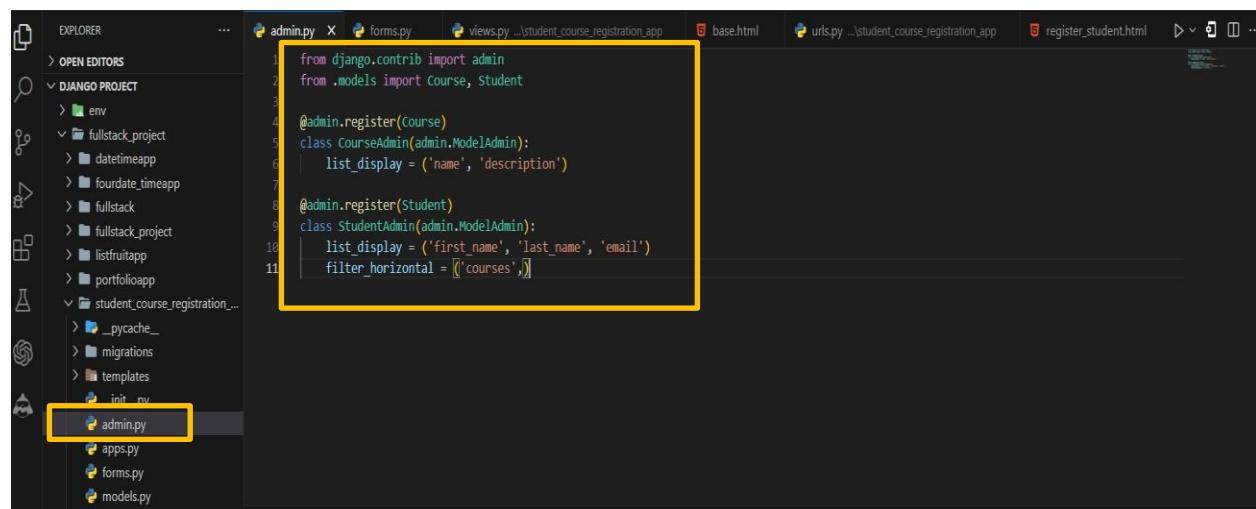
```
    list_display = ('name', 'description')
```

```
@admin.register(Student)
```

```
class StudentAdmin(admin.ModelAdmin):
```

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

```
    filter_horizontal = ('courses',)
```



The screenshot shows the PyCharm IDE interface with the 'admin.py' file open in the center editor tab. The code in the file registers the 'Course' and 'Student' models with the Django admin site, defining their list displays and filter horizontal fields. A yellow box highlights the registration code for 'Course' and 'Student'. The left sidebar shows the project structure with 'admin.py' highlighted. Other files like 'forms.py', 'views.py', 'urls.py', and 'register\_student.html' are visible in the background.

```
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**

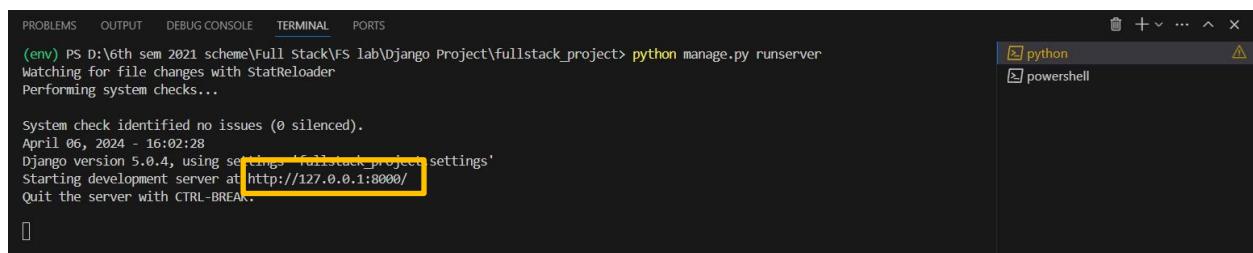
**Password:1234 [For Simply but your wish to create your own username and password]**

## 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/>.

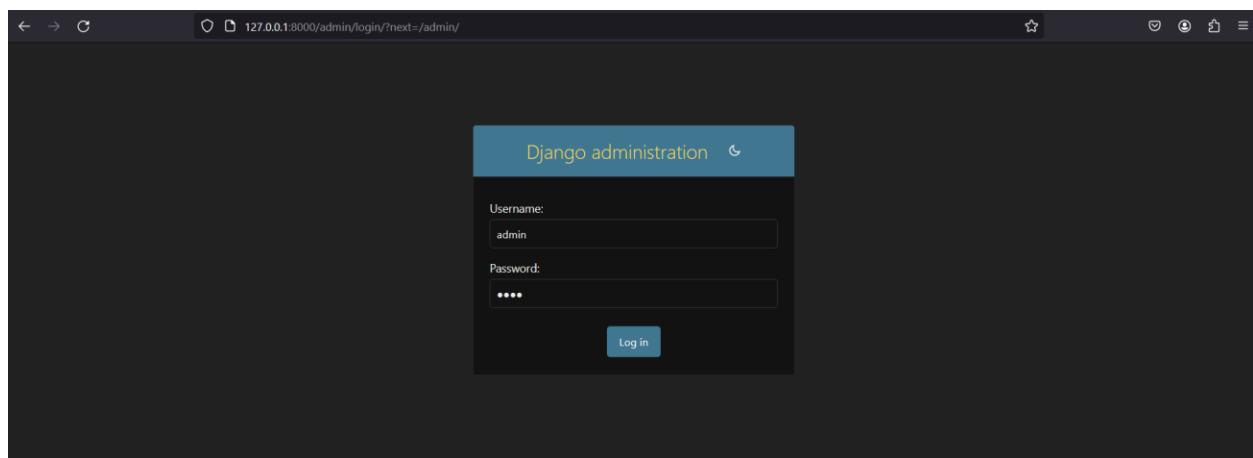


```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
(env) PS D:\6th sem 2021 scheme\Full Stack\FS lab\ Django Project\fullstack_project> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...
System check identified no issues (0 silenced).
April 06, 2024 - 16:02:28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

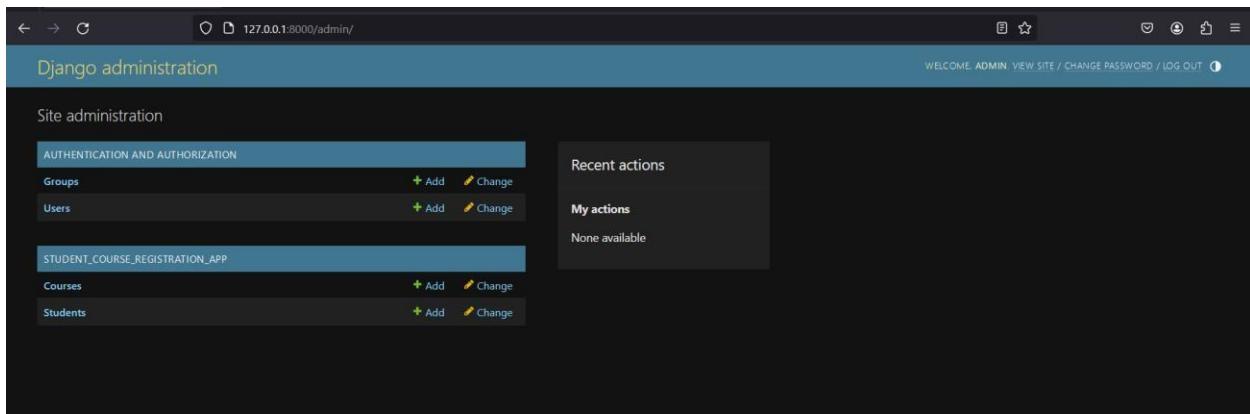
Next, click on "Students" in the left sidebar to add a new student.

- 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**

# 21CS62 | Full Stack Django Development|

The screenshot shows the Django Admin interface for the 'Courses' model. The left sidebar has sections for AUTHENTICATION AND AUTHORIZATION (Groups, Users) and STUDENT COURSE REGISTRATION APP (Courses, Students). The main area title is 'Select course to change'. It shows four courses: Software Engineering (description: 'Software Engineering it is a every Engineering Student Know About the this Subject'), Software Engineering (description: 'Software Engineering it is a every Engineering Student Know About the this Subject'), CN (description: 'nice'), and DBMS (description: 'It is Nice Code Version'). A button 'ADD COURSE +' is at the top right.

**Fig: Admin Course Screen**

The screenshot shows the Django Admin interface for the 'Students' model. The left sidebar has sections for AUTHENTICATION AND AUTHORIZATION (Groups, Users) and STUDENT COURSE REGISTRATION APP (Courses, Students). The main area title is 'Select student to change'. It shows three students: Hanumanthu (last name Hanu, email hanu@gmail.com), asfd (last name dsfad, email shivu@gmail.com), and Hanumanthu (last name hanu, email abc@gmail.com). A button 'ADD STUDENT +' is 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):
```

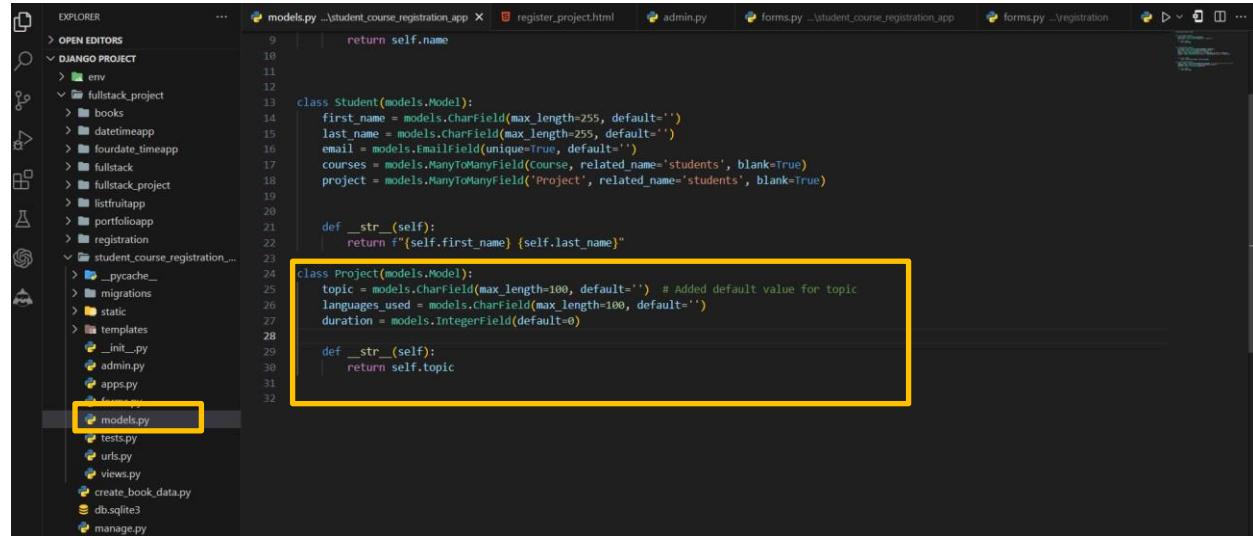
```
topic = models.CharField(max_length=100, default="") # Added default value for topic
```

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

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

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

```
    return self.topic
```



```
class 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 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'}),
```

```
        }
```

```

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})
```

```

    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,`

`)`

```

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 .models import Course, Student, Project
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 % }
```

```
<!-- 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">

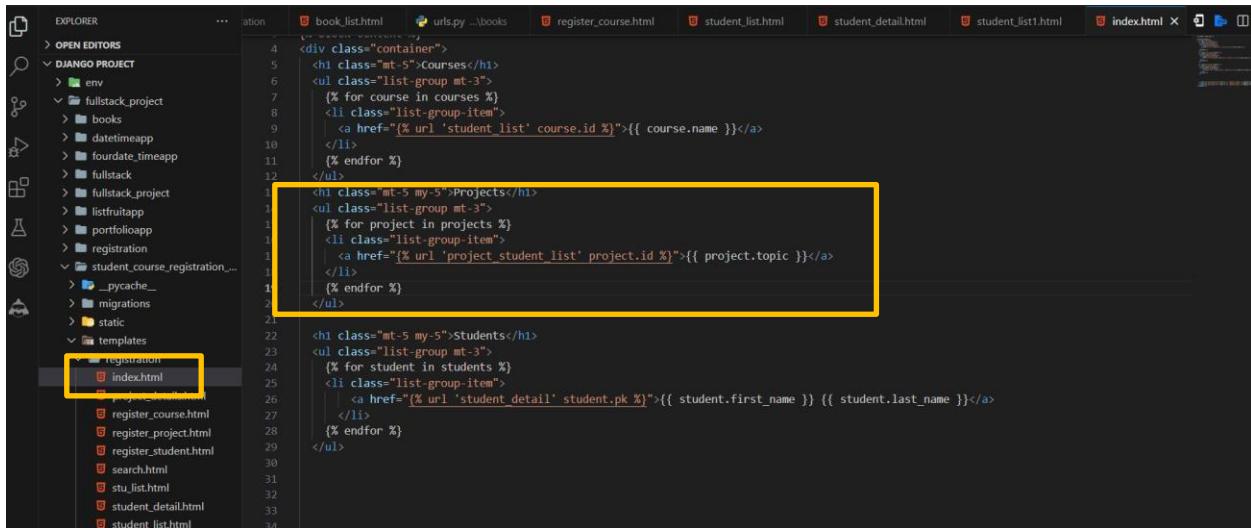
  {% for project in projects %}

    <li class="list-group-item">
      <a href="{% url 'project_student_list' project.id %}">{{ project.topic }}</a>
    </li>

  {% endfor %}

</ul>
```

# 21CS62 | Full Stack Django Development



```
4 <div class="container">
5   <h1 class="mt-5">Courses</h1>
6   <ul class="list-group mt-3">
7     {% for course in courses %}
8       <li class="list-group-item">
9         <a href="{% url 'student_list' course.id %}">{{ course.name }}</a>
10      {% endfor %}
11    </ul>
12
13   <h1 class="mt-5 my-5">Projects</h1>
14   <ul class="list-group mt-3">
15     {% for project in projects %}
16       <li class="list-group-item">
17         <a href="{% url 'project_student_list' project.id %}">{{ project.topic }}</a>
18       {% endfor %}
19     </ul>
20
21   <h1 class="mt-5 my-5">Students</h1>
22   <ul class="list-group mt-3">
23     {% for student in students %}
24       <li class="list-group-item">
25         <a href="{% url 'student_detail' student.pk %}">{{ student.first_name }} {{ student.last_name }}</a>
26       {% endfor %}
27     </ul>
28
29
30
31
32
33
34
```

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

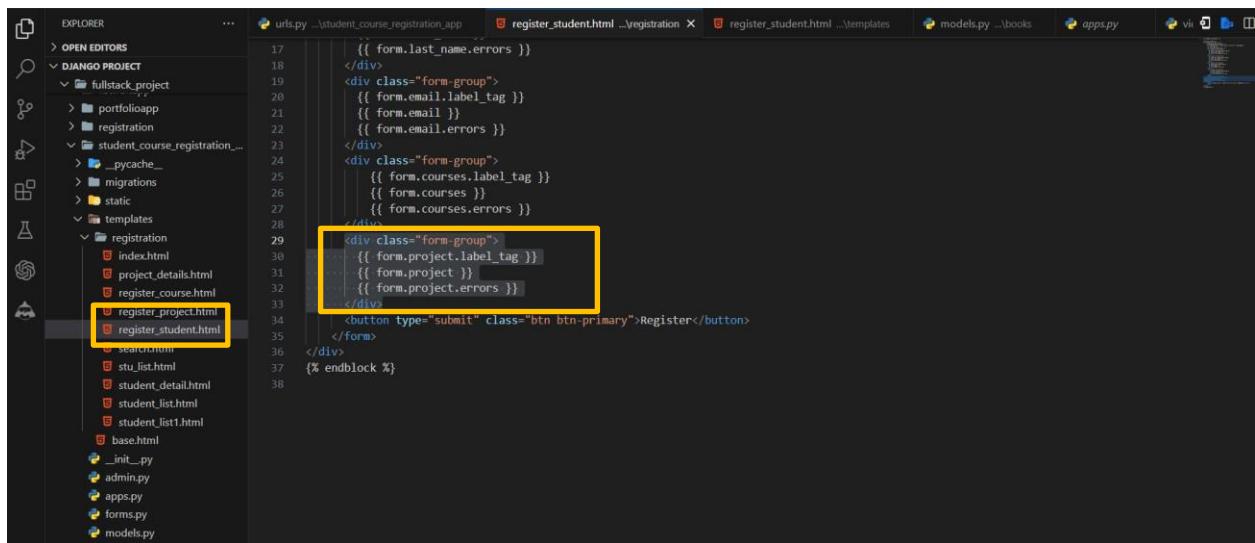
```
<div class="form-group">
```

```
  {{ form.project.label_tag }}
```

```
  {{ form.project }}
```

```
  {{ form.project.errors }}
```

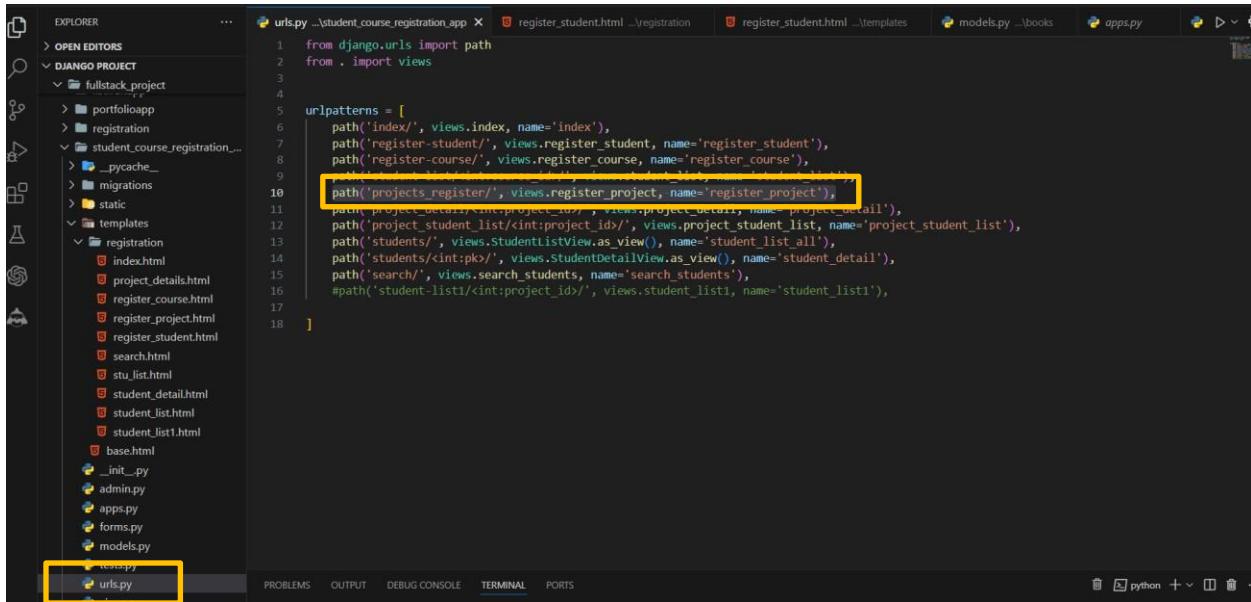
```
</div>
```



```
17   {{ form.last_name.errors }}
18 </div>
19 <div class="form-group">
20   {{ form.email.label_tag }}
21   {{ form.email }}
22   {{ form.email.errors }}
23 </div>
24 <div class="form-group">
25   {{ form.courses.label_tag }}
26   {{ form.courses }}
27   {{ form.courses.errors }}
28 </div>
29 <div class="form-group">
30   {{ form.project.label_tag }}
31   {{ form.project }}
32   {{ form.project.errors }}
33 </div>
34 <button type="submit" class="btn btn-primary">Register</button>
35 </form>
36 </div>
37 {% 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

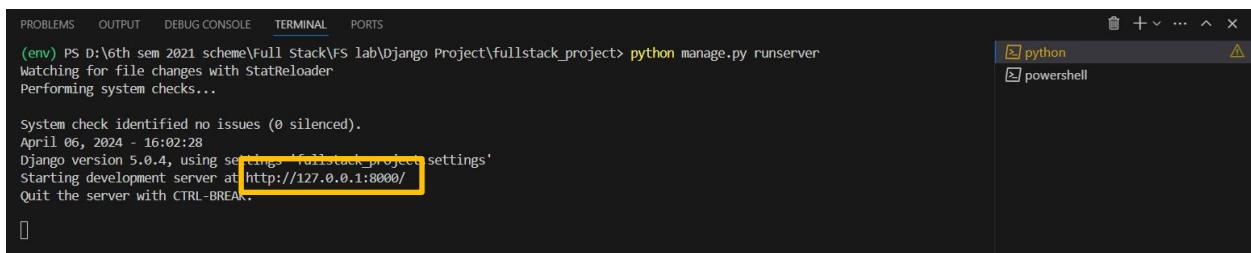
urlpatterns = [
    path('index/', views.index, name='index'),
    path('register-student/', views.register_student, name='register_student'),
    path('register-course/', views.register_course, name='register_course'),
    path('projects_register/', views.register_project, name='register_project'),
    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/>.



```
(env) PS D:\6th sem 2021 scheme\Full Stack\FS lab\ Django Project\fullstack_project> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...
System check identified no issues (0 silenced).
April 06, 2024 - 16:02:28
Django version 5.0.4, using settings 'fullstack_project.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

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

## Final Output By Clicking Register 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:

Languages Used:

Duration:

**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**

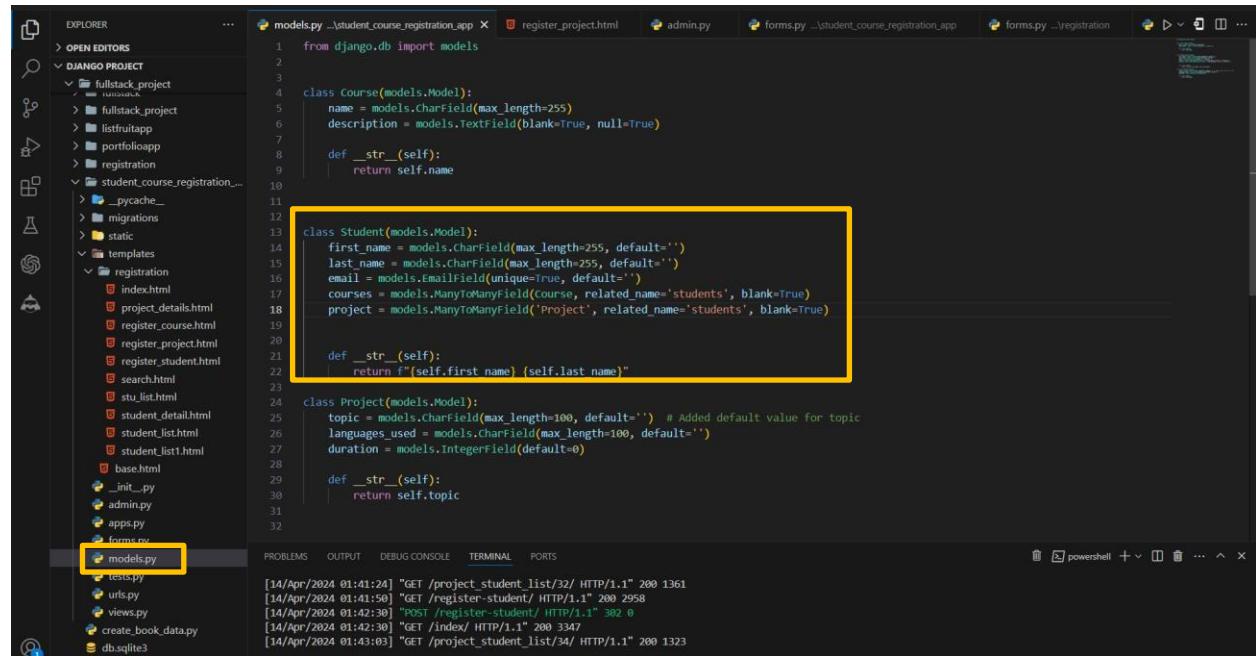
## Experiment-10

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)
```



```
models.py  ... \student_course_registration_app x register_project.html admin.py forms.py \student_course_registration_app forms.py \registration ...
OPEN EDITORS
DJANGO PROJECT
fullstack_project
  student_course_registration...
    models.py
EXPLORER
  _pycache_
  migrations
  static
  templates
    registration
      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
    create_book_data.py
    db.sqlite3
```

```
1  from django.db import models
2
3
4  class Course(models.Model):
5      name = models.CharField(max_length=255)
6      description = models.TextField(blank=True, null=True)
7
8      def __str__(self):
9          return self.name
10
11
12
13  class Student(models.Model):
14      first_name = models.CharField(max_length=255, default='')
15      last_name = models.CharField(max_length=255, default='')
16      email = models.EmailField(unique=True, default='')
17      courses = models.ManyToManyField(Course, related_name='students', blank=True)
18      project = models.ManyToManyField('Project', related_name='students', blank=True)
19
20
21      def __str__(self):
22          return f'{self.first_name} {self.last_name}'
23
24
25  class Project(models.Model):
26      topic = models.CharField(max_length=100, default='') # Added default value for topic
27      languages_used = models.CharField(max_length=100, default='')
28      duration = models.IntegerField(default=0)
29
30      def __str__(self):
31          return self.topic
32
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
[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-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 several tabs open. The tabs include: admin.py, forms.py (student\_course\_registration\_app), forms.py (registration), urls.py (registration), and views.py (student\_course\_registration\_app). The views.py tab is active and contains the code for the StudentListView and StudentDetailView classes. The code editor has a yellow box highlighting the class definitions and their methods. The bottom of the screen shows a terminal window with a log of HTTP requests.

```
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})

def student_list_new(request):
    students = Student.objects.all()
    return render(request, 'registration/stu_list.html', {'students': students})

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'

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:
            results.append({'id': student.id, 'name': student.name})
        return JsonResponse(results)
```

- In the StudentListView, we've overridden the get\_queryset method to filter the students based on the course\_id parameter in the URL. If the course\_id is

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

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

#### stu\_list.html

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

{% block content %}

<h1>Students</h1>

<ul>

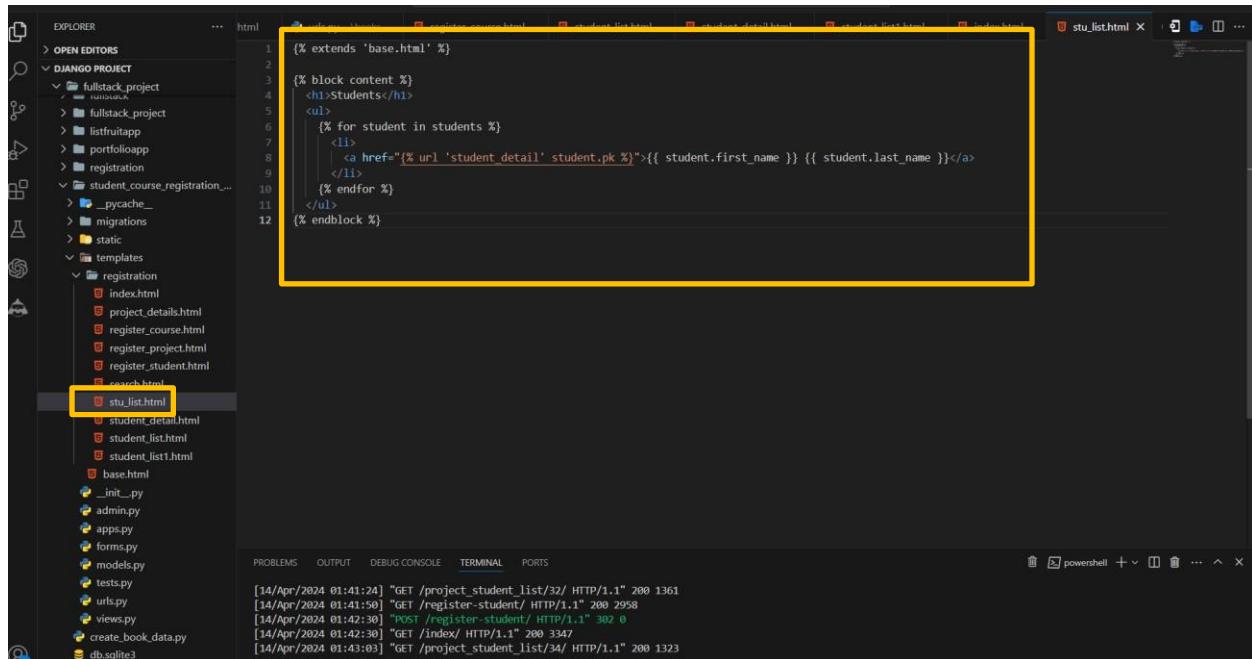
  {% for student in students %}

    <li>
      <a href="{% url 'student_detail' student.pk %}">{{ student.first_name }} {{ student.last_name }}</a>
    </li>

  {% endfor %}

</ul>

{% endblock %}
```



```

1  {% extends 'base.html' %} 
2
3  {% block content %} 
4      <h1>Students</h1>
5      <ul>
6          {% for student in students %}
7              <li>
8                  <a href="{% url 'student_detail' student.pk %}">{{ student.first_name }} {{ student.last_name }}</a>
9              </li>
10         {% endfor %}
11     </ul>
12  {% endblock %}

```

The screenshot shows the VS Code interface with the 'stu\_list.html' file open in the editor. The file contains Django template code. A yellow box highlights the section from line 12 to line 12, which is the closing '{% endblock %}' tag. The left sidebar shows the project structure under 'DJANGO PROJECT'. The bottom status bar shows terminal logs.

## 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 %}
```

A screenshot of a code editor interface, likely Visual Studio Code, showing a Django project structure in the Explorer sidebar. The current file being edited is 'student\_detail.html' located in the 'registration' directory under 'templates'. The code in the editor is:

```
1  {% extends 'base.html' %}  
2  
3  {% block content %}  
4      <div class="container my-5">  
5          <div class="row">  
6              <div class="col-md-6 offset-md-3">  
7                  <div class="card">  
8                      <div class="card-body">  
9                          <h1 class="card-title"> Name: {{ student.first_name }} {{ student.last_name }}</h1>  
10                         <h1 class="card-title">Email: {{ student.email }}</h1>  
11                     </div>  
12                 </div>  
13             </div>  
14         </div>  
15     </div>  
16  {% endblock %}
```

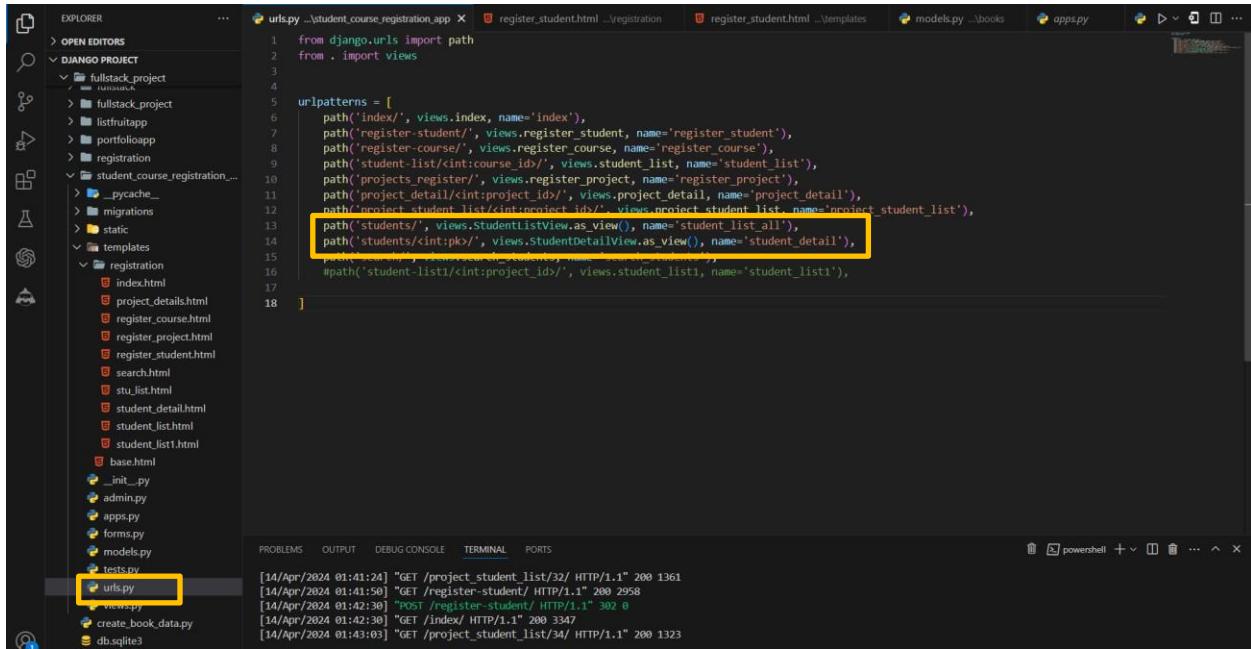
The code area is highlighted with a yellow rectangle. The bottom right corner of the editor shows a terminal window with the following log output:

```
[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'),
```



The screenshot shows the VS Code interface with the 'urls.py' file open in the center editor tab. The file contains Python code defining URL patterns for a Django application. A yellow box highlights the following code block:

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

The left sidebar shows the project structure with several apps like 'fullstack\_project', 'listfrutapp', 'portfolioapp', and 'student\_course\_registration\_app'. The bottom status bar indicates the file is saved and shows the path 'C:\Users\user\PycharmProjects\fullstack\student\_course\_registration\_app\urls.py'.

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

### Index.html

```
<h1 class="mt-5 my-5">Students</h1>

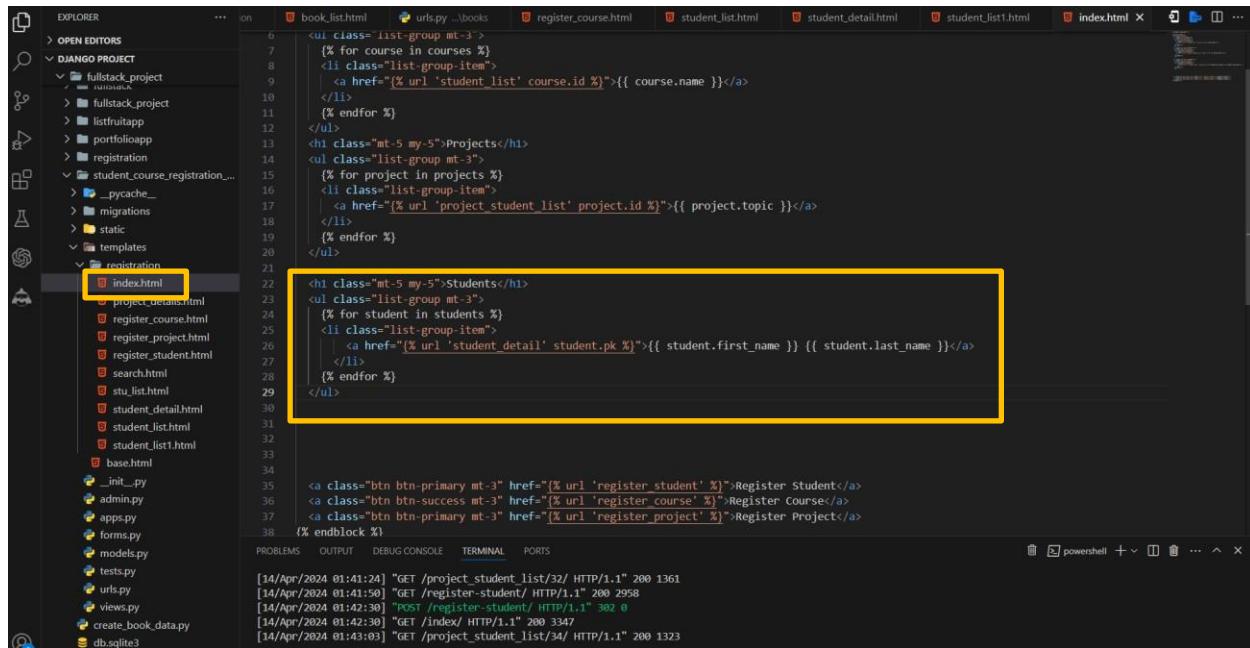
<ul class="list-group mt-3">

    {% for student in students %}

        <li class="list-group-item">
            <a href="{% url 'student_detail' student.pk %}">{{ student.first_name }} {{ student.last_name }}</a>
        </li>

    {% endfor %}
```

</ul>



```

<ul class="list-group mt-3">
    {% for course in courses %}
        <li class="list-group-item">
            <a href="{% url 'student_list' course.id %}">{{ course.name }}</a>
        </li>
    {% endfor %}
</ul>
<h1 class="mt-5 my-5">Projects</h1>
<ul class="list-group mt-3">
    {% for project in projects %}
        <li class="list-group-item">
            <a href="{% url 'project_student_list' project.id %}">{{ project.topic }}</a>
        </li>
    {% endfor %}
</ul>
<h1 class="mt-5 my-5">Students</h1>
<ul class="list-group mt-3">
    {% for student in students %}
        <li class="list-group-item">
            <a href="{% url 'student_detail' student.pk %}">{{ student.first_name }} {{ student.last_name }}</a>
        </li>
    {% endfor %}
</ul>

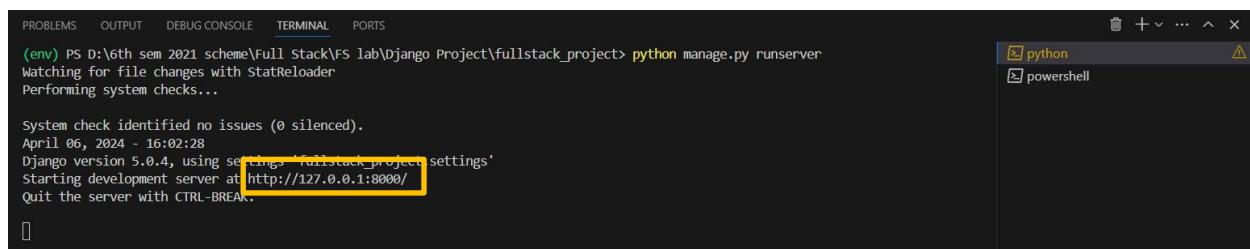
```

## Step-6: Run the development server

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

**python manage.py runserver**

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



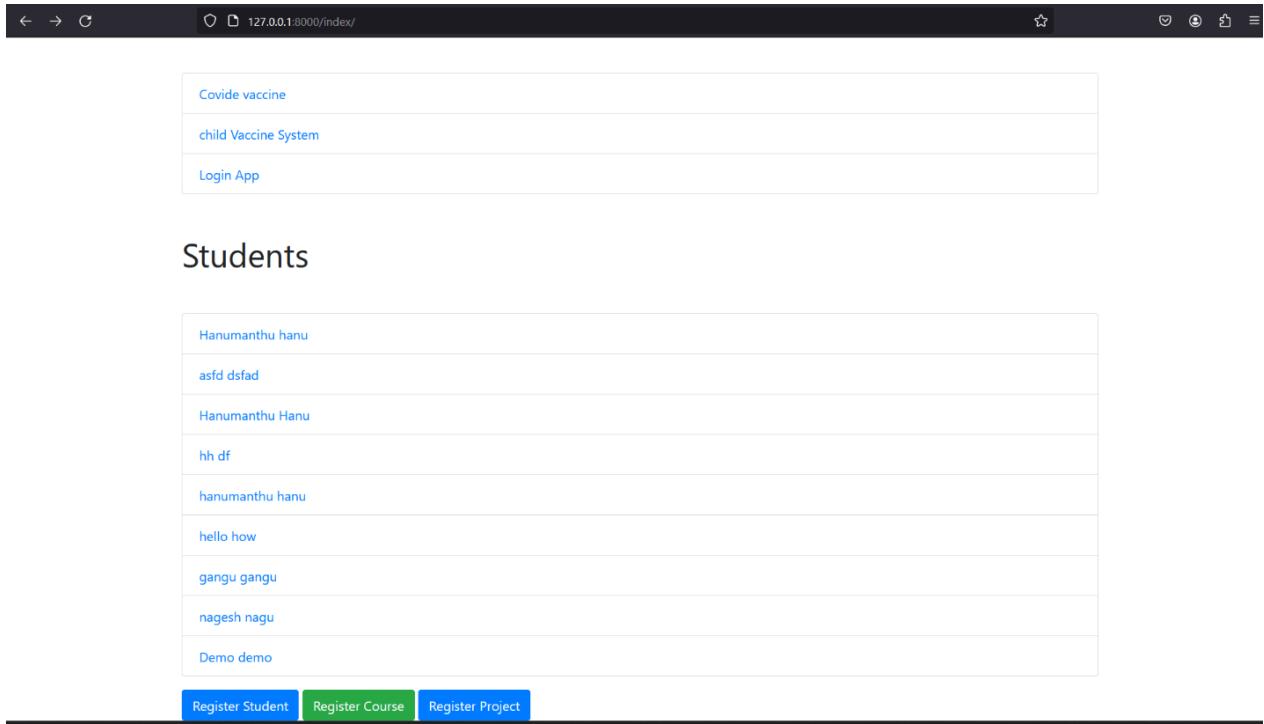
```

PS D:\6th sem 2021 scheme\Full Stack FS lab\ Django Project\fullstack_project> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...
System check identified no issues (0 silenced).
April 06, 2024 - 16:02:28
Django version 5.0.4, using settings 'fullstack_project.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

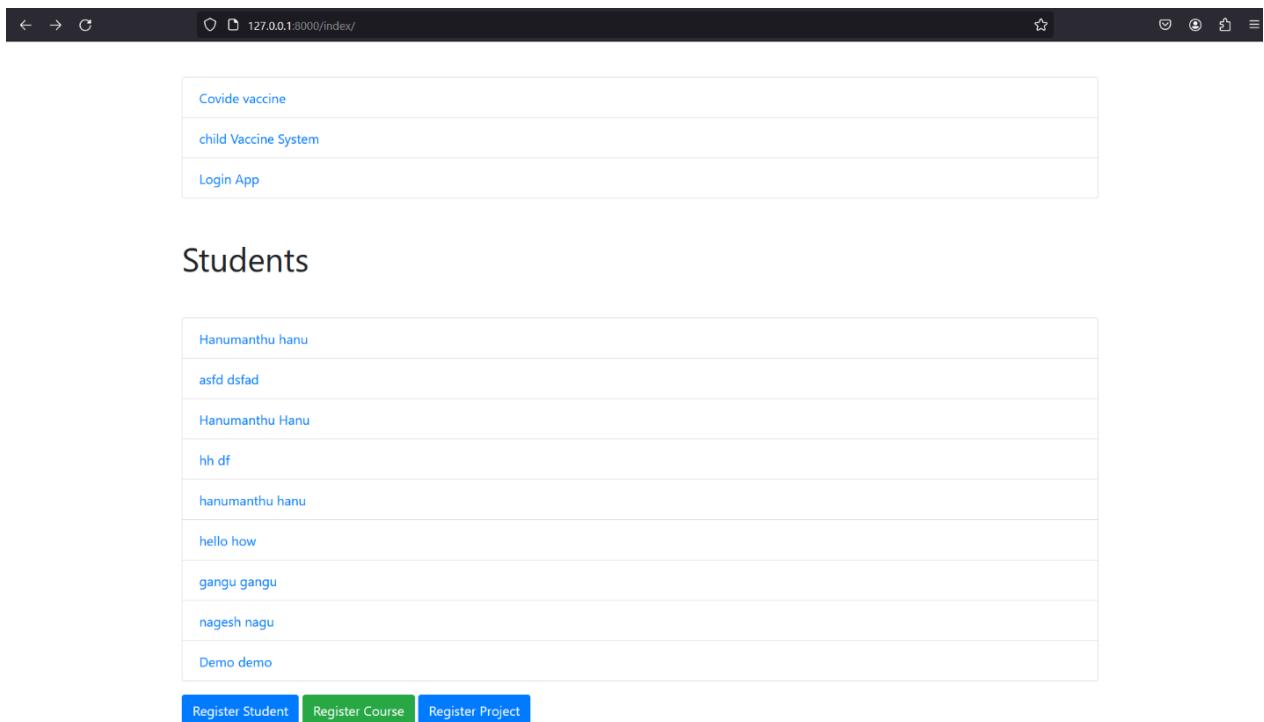
```

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

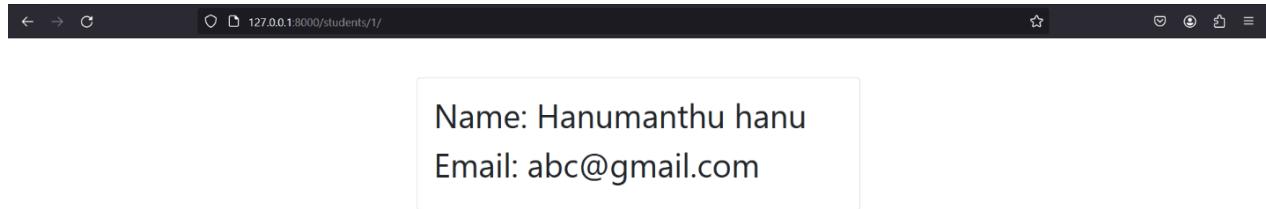
## Final Output By Clicking Register Project and Register Student



**Fig: Main Screen**



**Fig: List View of Students Screen**



**Fig: Detailed View of Student Screen**

## Experiment-11

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

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

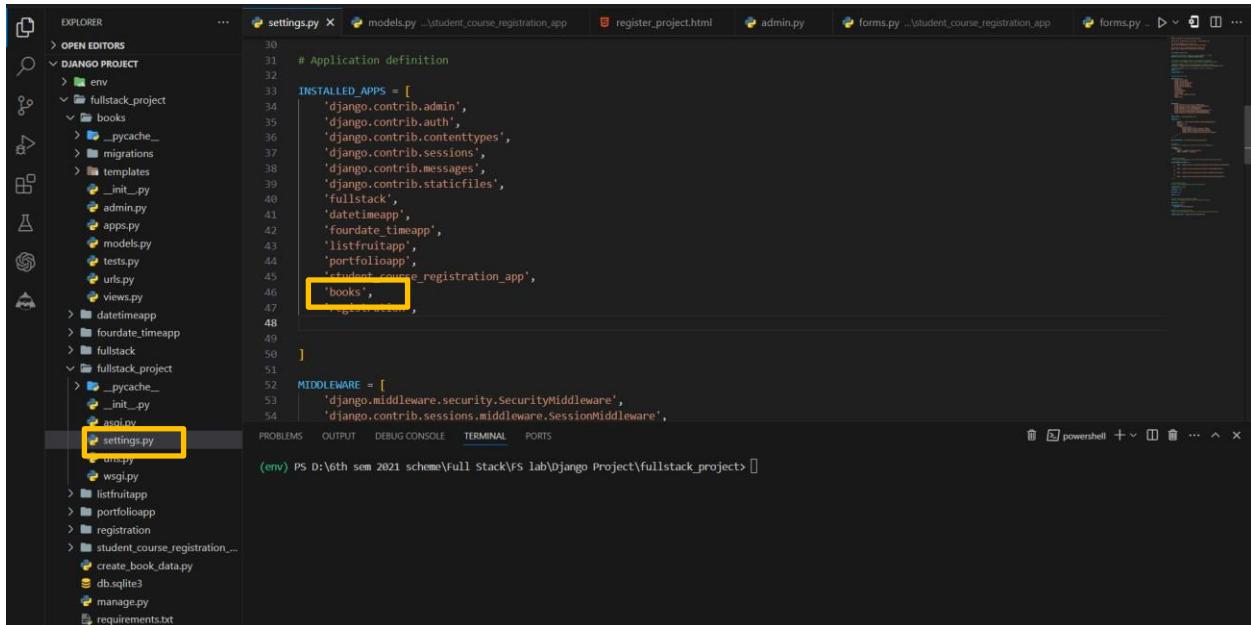


```
(env) PS D:\6th sem 2021 scheme\Full Stack\FS lab\Django Project\fullstack_project> django-admin startapp books
```

**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:



```
settings.py
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     'portfolioapp',
45     'student_course_registration_app',
46     'books',
47     ...
48 ]
49
50 MIDDLEWARE = [
51     'django.middleware.security.SecurityMiddleware',
52     'django.contrib.sessions.middleware.SessionMiddleware',
53     ...
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):
```

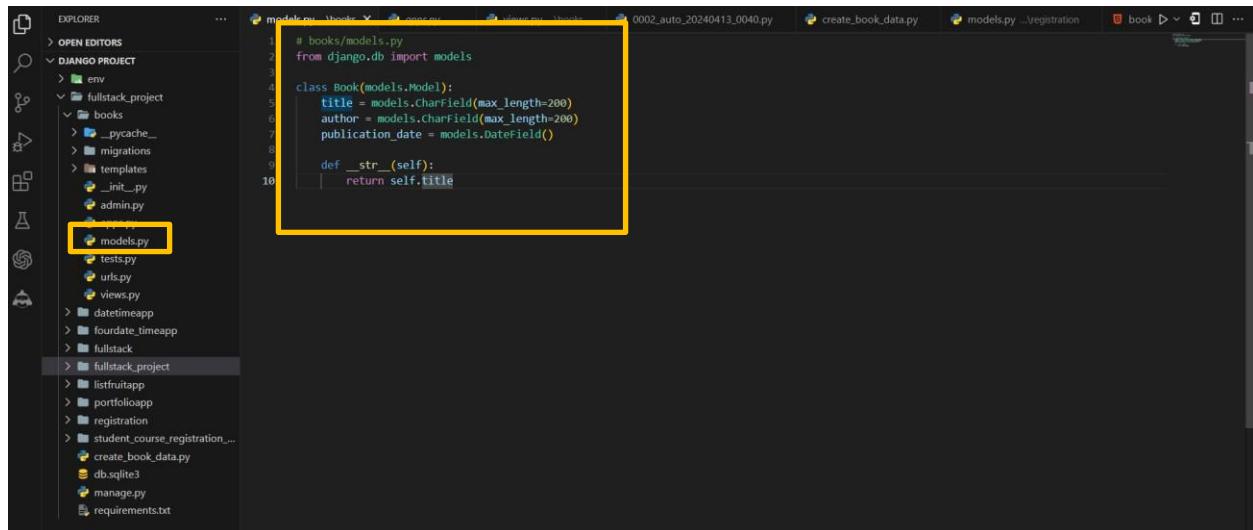
```
    title = models.CharField(max_length=200)
```

```
    author = models.CharField(max_length=200)
```

```
    publication_date = models.DateField()
```

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

```
        return self.title
```



```
# books/models.py
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)

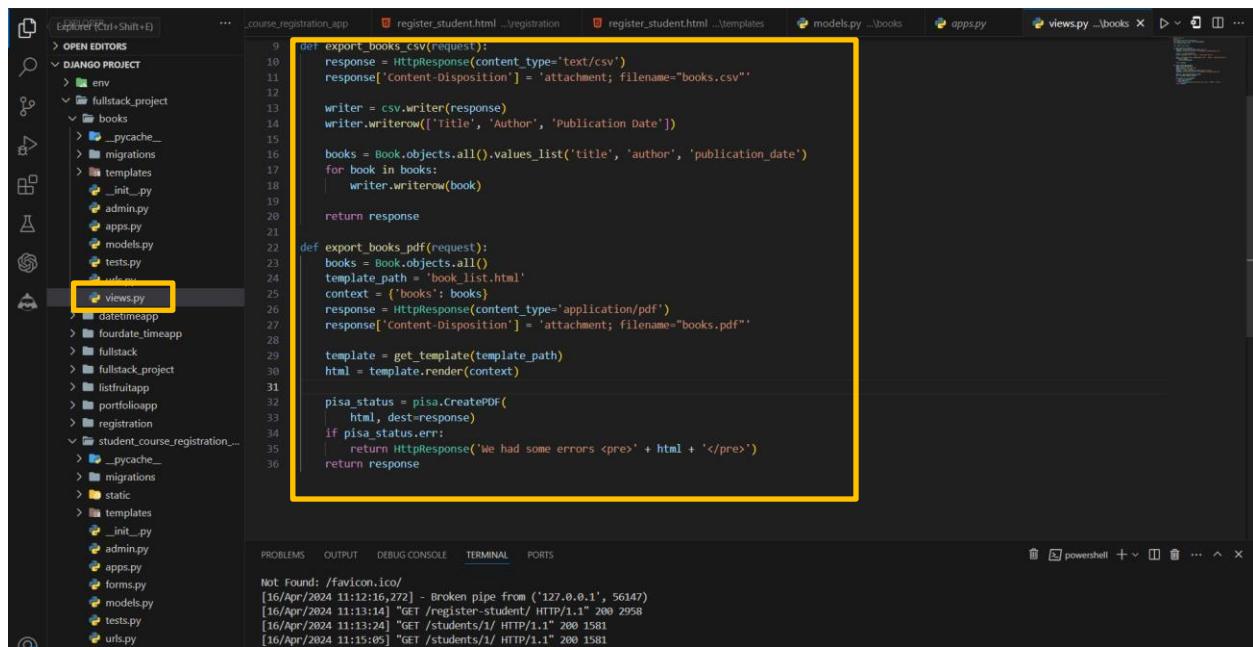
pisa_status = pisa.CreatePDF(

    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 several tabs open. The main tab contains Python code for generating CSV and PDF files. A yellow box highlights the section of code from line 9 to line 36, which defines two functions: `export_books_csv` and `export_books_pdf`. The `export_books_csv` function uses `HttpResponse` with `content_type='text/csv'` and sets `Content-Disposition` to attachment with filename "books.csv". It uses a CSV writer to iterate over `Book.objects.all()` and write rows for title, author, and publication date. The `export_books_pdf` function uses `HttpResponse` with `content_type='application/pdf'` and sets `Content-Disposition` to attachment with filename "books.pdf". It uses a template named "book\_list.html" and `pisa.CreatePDF` to render the HTML into a PDF. If there are errors during the PDF creation, it returns an error message. Otherwise, it returns the generated 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>

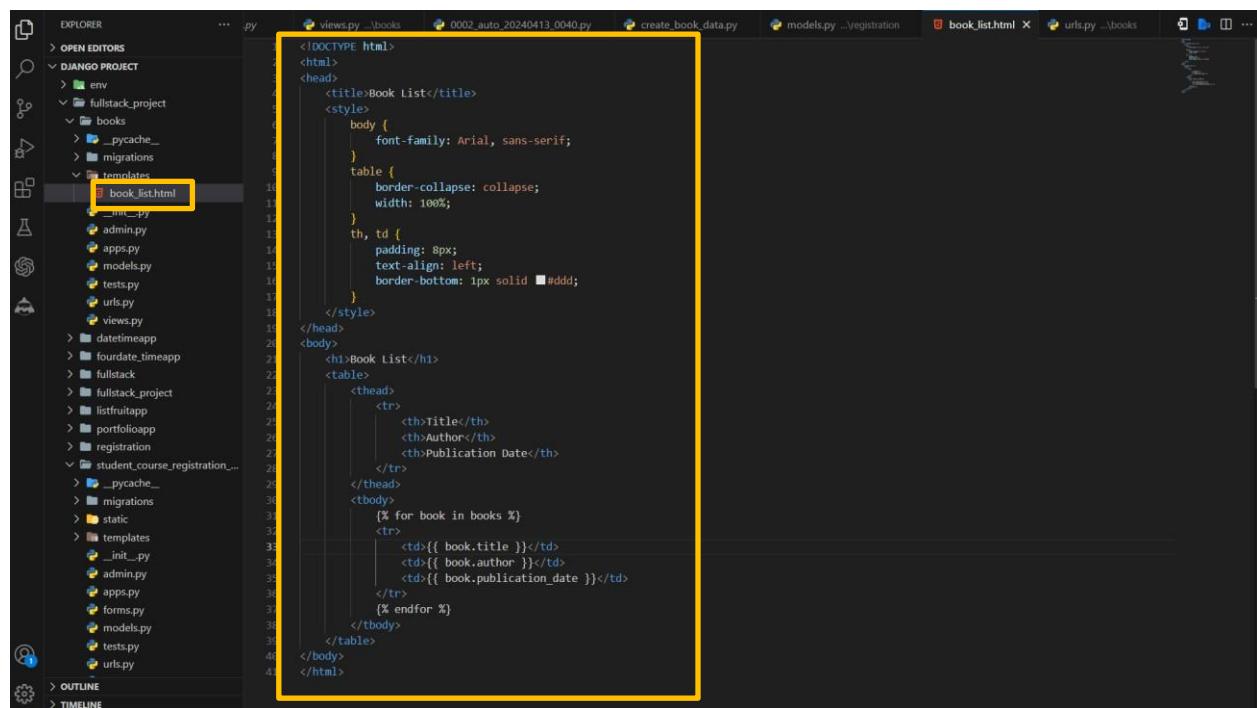
{ % endfor % }

</tbody>

</table>

</body>

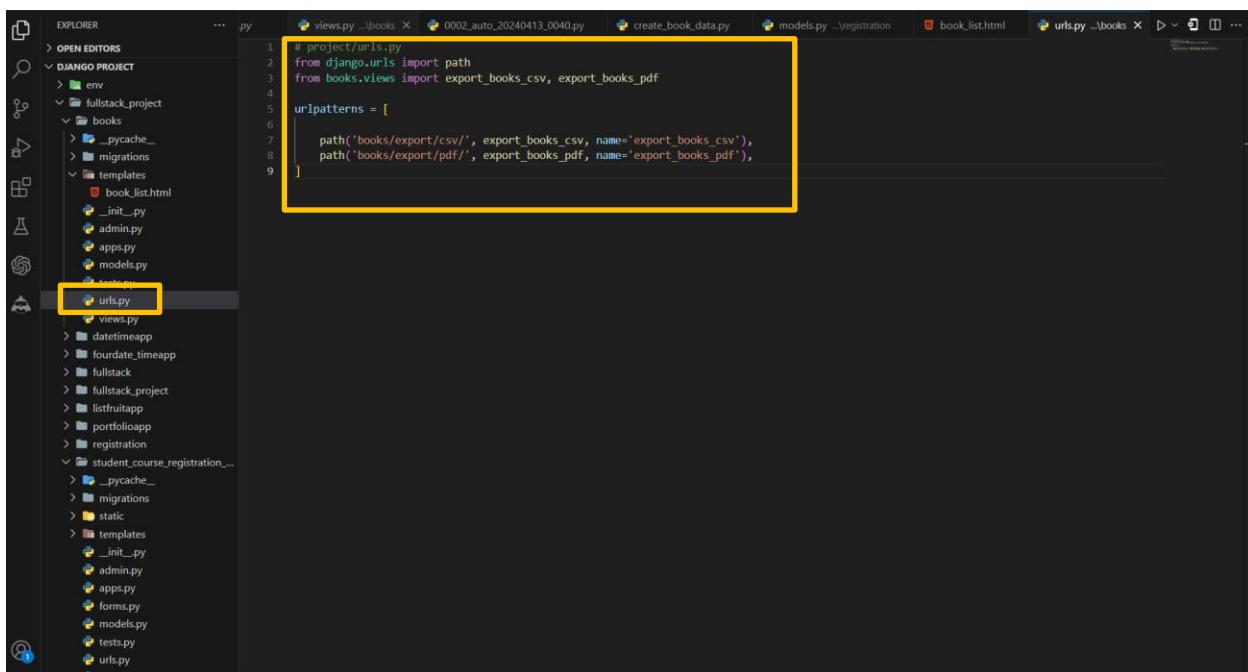
</html>
```



```
<!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:

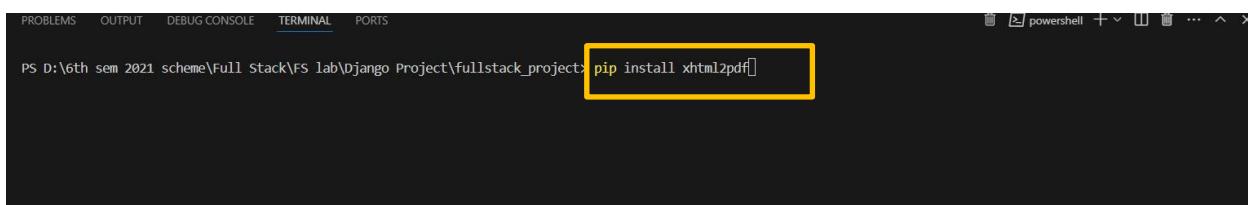
```
from django.urls import path  
  
from books.views import export_books_csv, export_books_pdf  
  
urlpatterns = [  
  
    path('books/export/csv/', export_books_csv, name='export_books_csv'),  
  
    path('books/export/pdf/', export_books_pdf, name='export_books_pdf'),  
  
]
```



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

```
pip install xhtml2pdf
```



```
PS D:\6th sem 2021 scheme\Full Stack\FS Lab\ Django Project\fullstack_project> pip install xhtml2pdf
```

## 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
```

The screenshot shows a Django project structure in the Explorer panel. A migration file named '0002\_auto\_20240413\_0040.py' is open in the editor. The code defines a 'create\_book\_data' function and a 'Migration' class. The 'operations' list contains a single item: 'migrations.RunPython(create\_book\_data)'. The terminal at the bottom shows the command 'python manage.py makemigrations books --empty' being run.

```
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),
    ]
```

- 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'),
])
```

The screenshot shows the VS Code interface with the 'create\_book\_data.py' file open in the editor. The code defines a function that creates five books in the 'books' database. The file is located in the 'fullstack\_project' directory under the 'fullstack' app. The 'create\_book\_data.py' file is highlighted with a yellow box.

```

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`

The screenshot shows the terminal window in VS Code with the command 'python create\_book\_data.py' entered. The terminal output shows the script running successfully.

```

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:

The screenshot shows the 'urls.py' file in the 'fullstack' app. The 'books.urls' inclusion is highlighted with a yellow box. The 'urls.py' file is located in the 'fullstack' directory under the 'fullstack' 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.urls')),
]

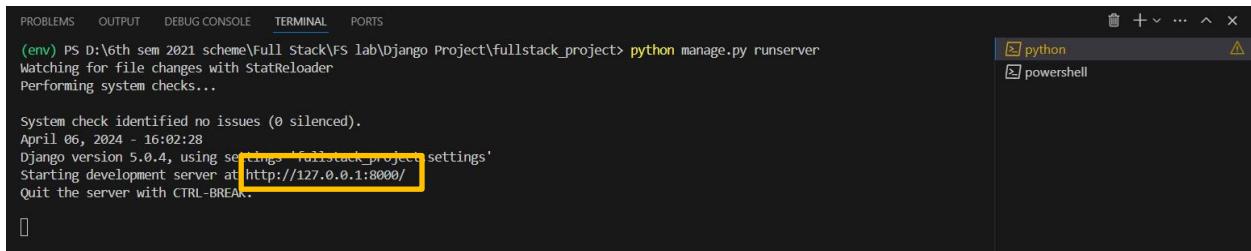
```

## 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.
```

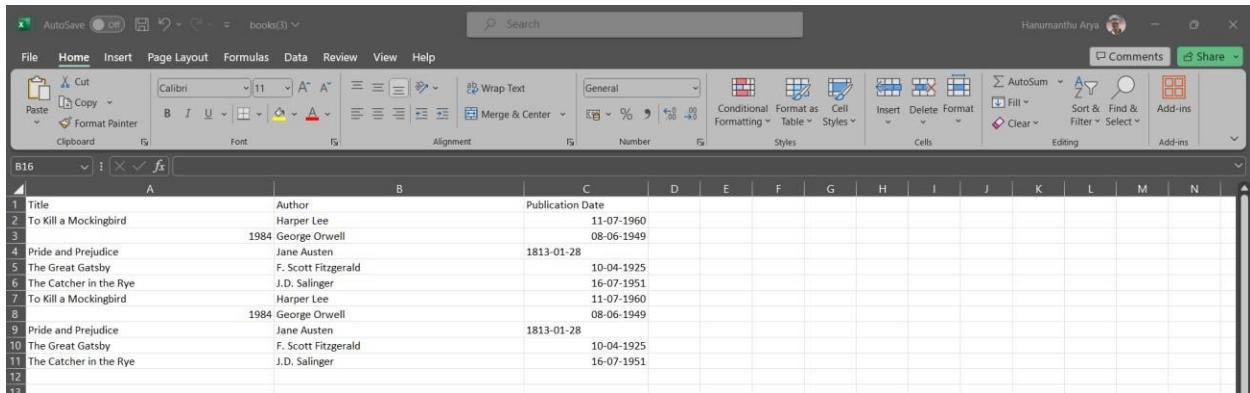
- Type or copy this <http://127.0.0.1:8000/books/export/csv/>

## Final Output By Clicking Register Project and Register Student



**Fig: CSV File Download Screen**

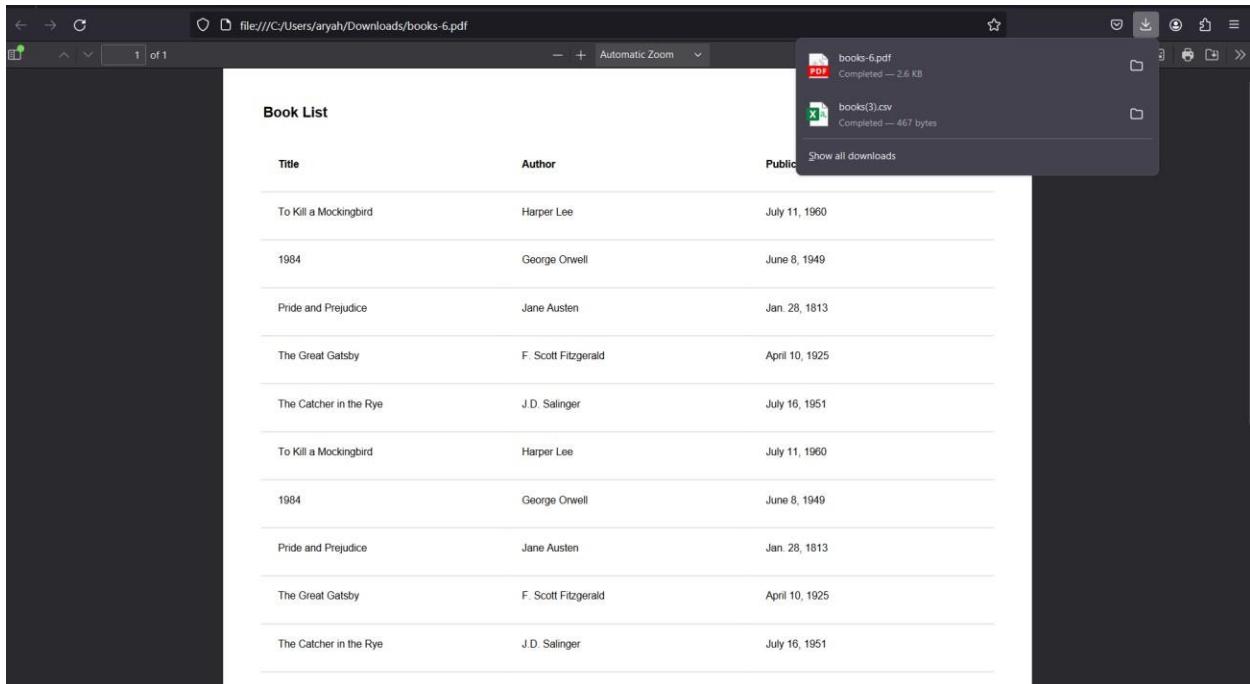
# 21CS62 | Full Stack Django Development



A screenshot of Microsoft Excel showing a table of book data. The table has columns for Title, Author, Publication Date, and other details. The data includes books like "To Kill a Mockingbird", "Pride and Prejudice", and "The Great Gatsby".

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

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



A screenshot of a browser window showing a PDF file download dialog. The dialog shows two files: "books-6.pdf" (Completed — 2.6 KB) and "books(3).csv" (Completed — 467 bytes). The "books-6.pdf" file is selected.

Book List

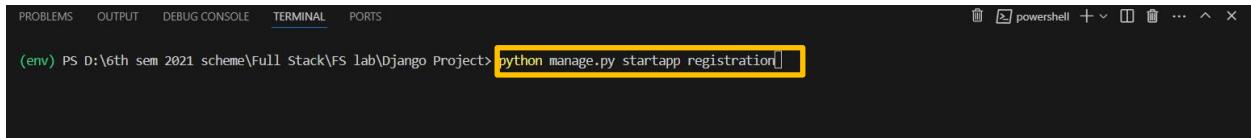
Title	Author	Publication Date
To Kill a Mockingbird	Harper Lee	July 11, 1960
1984	George Orwell	June 8, 1949
Pride and Prejudice	Jane Austen	Jan. 28, 1813
The Great Gatsby	F. Scott Fitzgerald	April 10, 1925
The Catcher in the Rye	J.D. Salinger	July 16, 1951
To Kill a Mockingbird	Harper Lee	July 11, 1960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

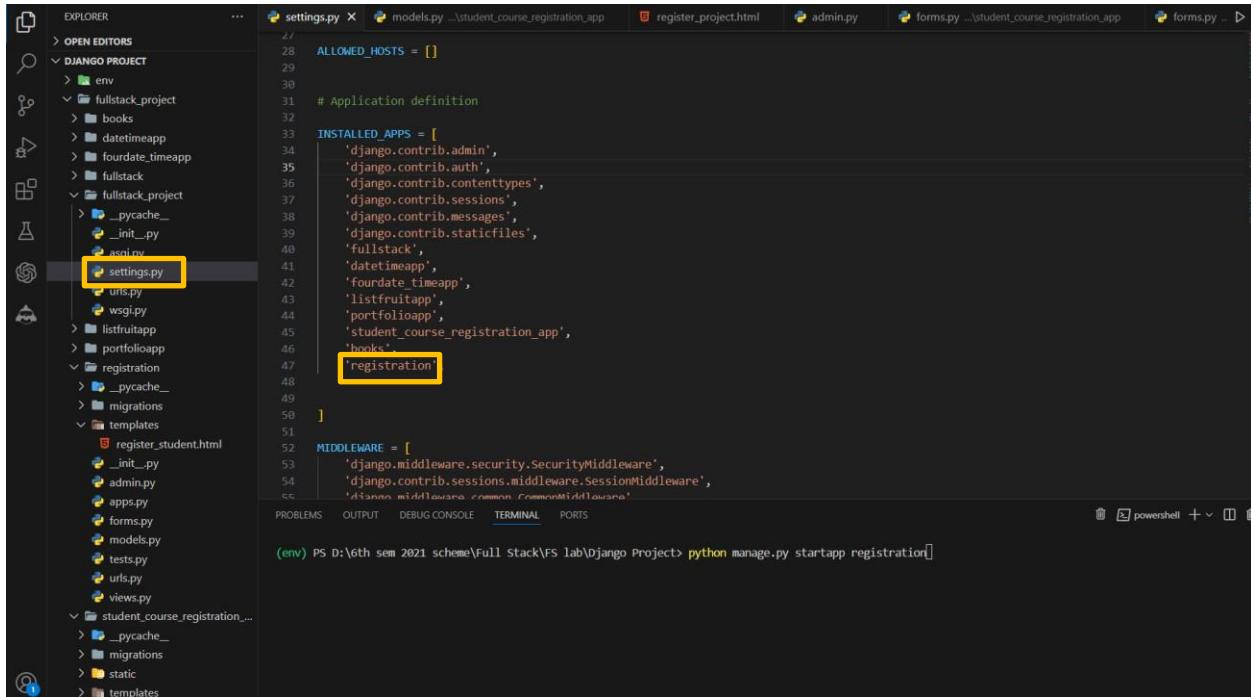


```
(env) PS D:\6th sem 2021 scheme\Full Stack\FS lab\ Django Project> python manage.py startapp registration
```

**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
...
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',
    'student_course_registration_app',
    'books',
    'registration'
]
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware'
```

## Step-03: Create models

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

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

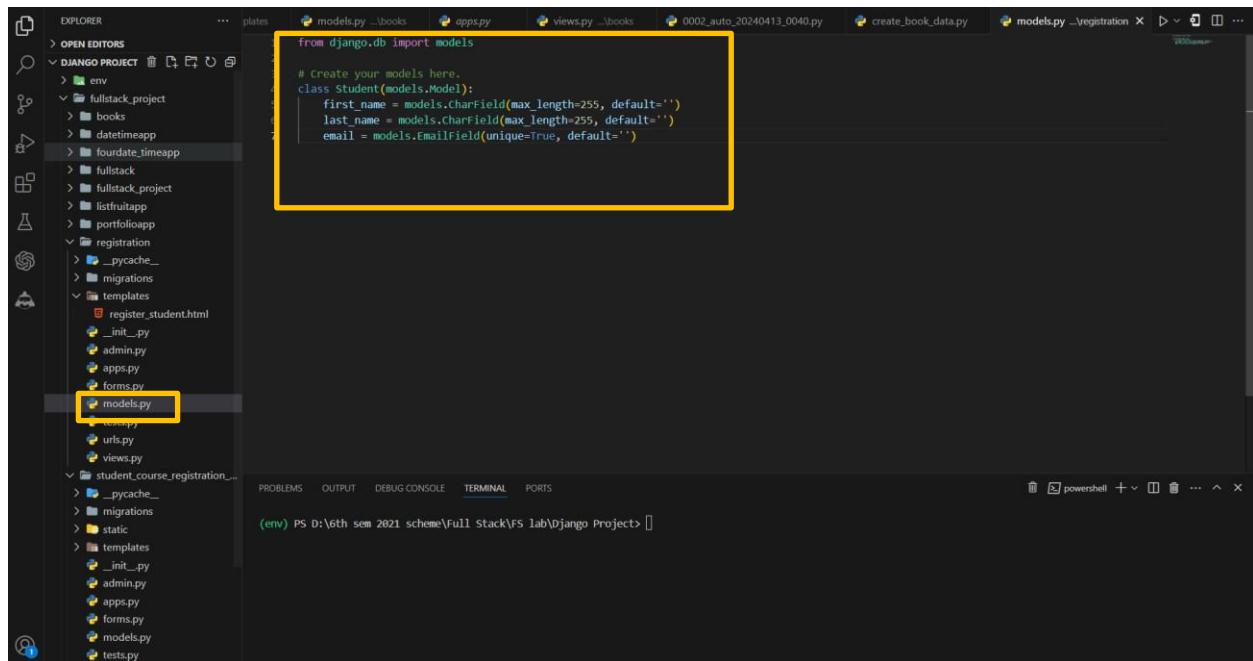
```
# Create your models here.
```

```
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='')
```



```
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()
```

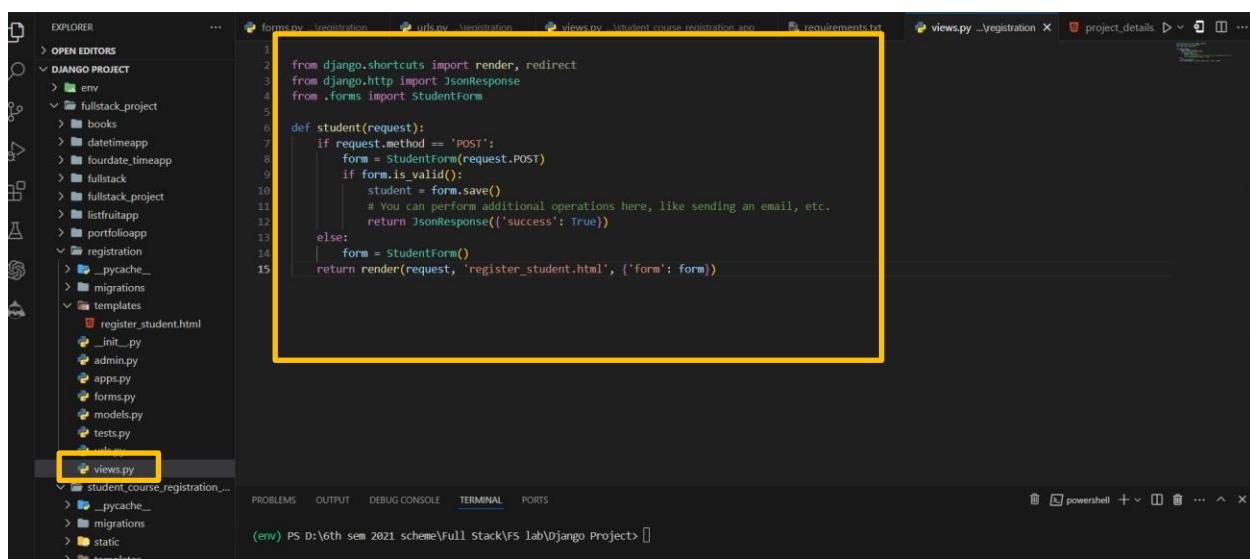
```
# You can perform additional operations here, like sending an email, etc.
```

```
        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']
```

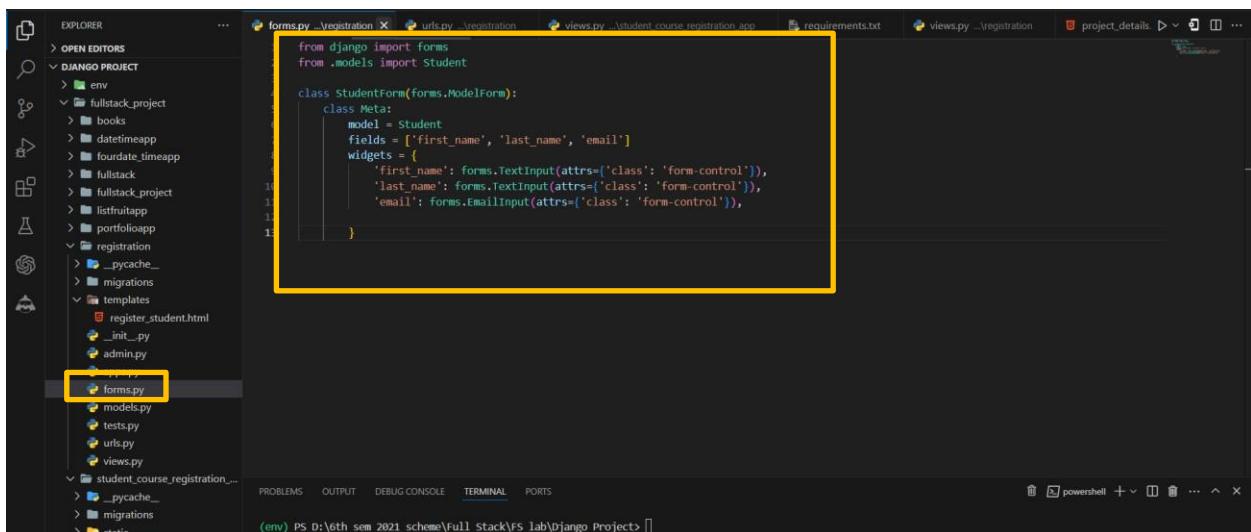
```
        widgets = {
```

```
            '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>
</html>
```

```
{% 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>');

                }

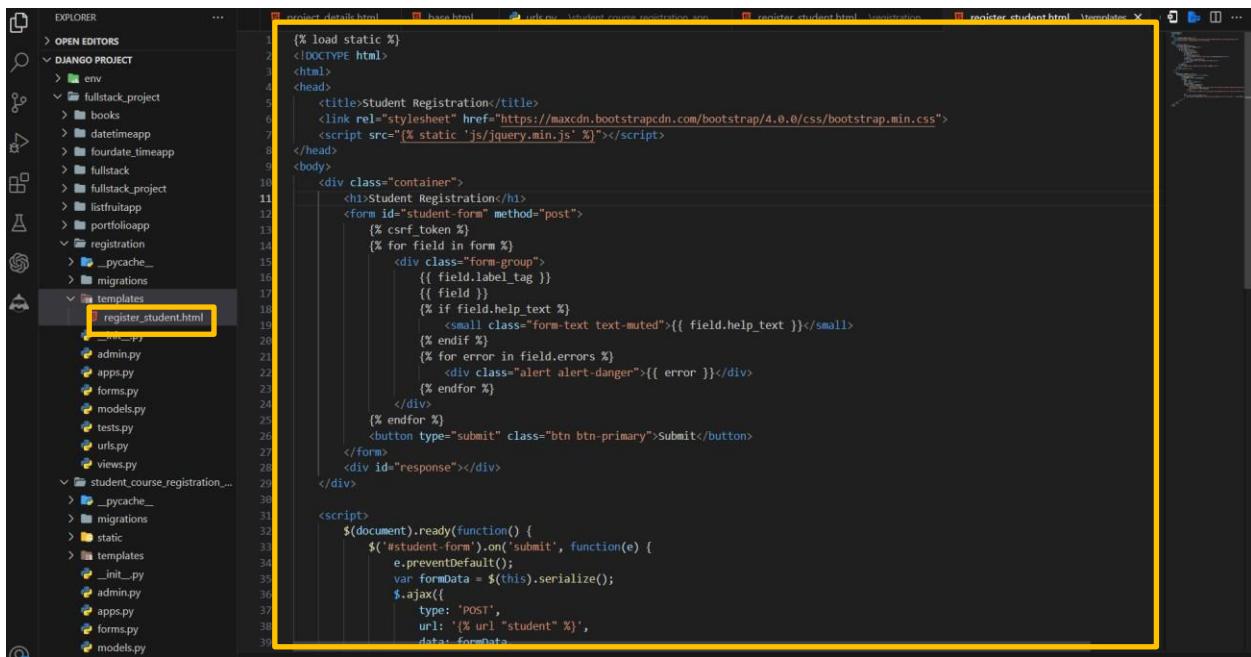
            },

        });

    });

});
```

```
error: function(xhr, errmsg, err) {  
  
    $('#response').html('<div class="alert alert-danger">An error occurred: ' +  
xhr.status + ' ' + xhr.responseText + '</div>');  
  
}  
  
});  
  
});  
  
</script>  
  
</body>  
  
</html>
```



The screenshot shows a code editor interface with a dark theme. On the left is the 'EXPLORER' sidebar, which lists a 'Django Project' structure. Inside 'templates', the file 'register\_student.html' is selected and highlighted with a yellow box. The main editor area displays the content of 'register\_student.html'. The code includes HTML for a student registration form, a script to handle form submission via AJAX, and a CSS class 'form-text text-muted' for small text elements.

```
{% 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);  
        });  
    });  
});
```

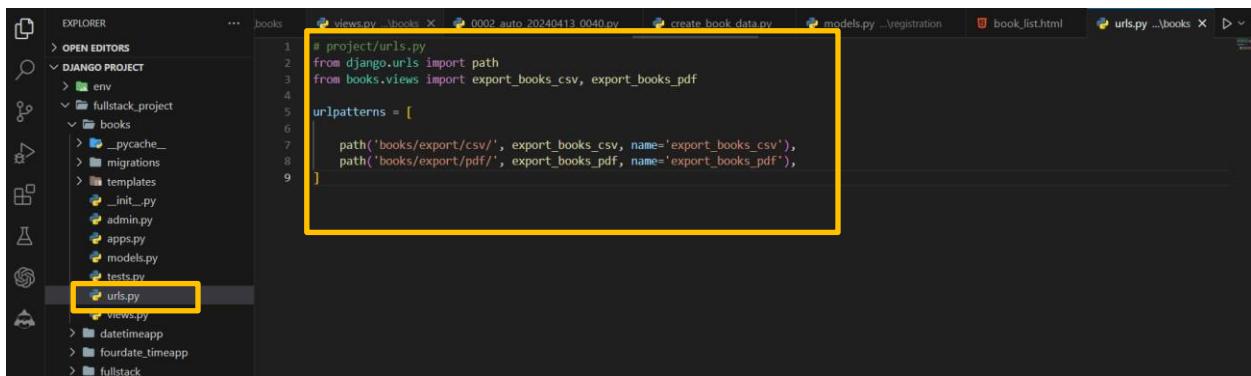
## Step-07: Update app URLs

```
# project/urls.py

from django.urls import path

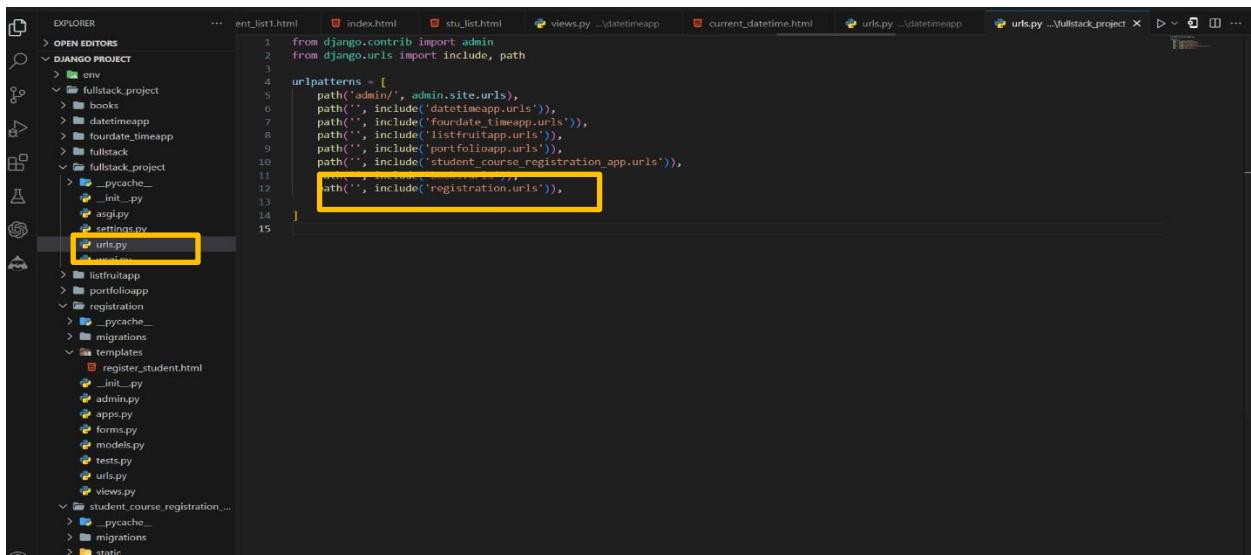
from books.views import export_books_csv, export_books_pdf

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:

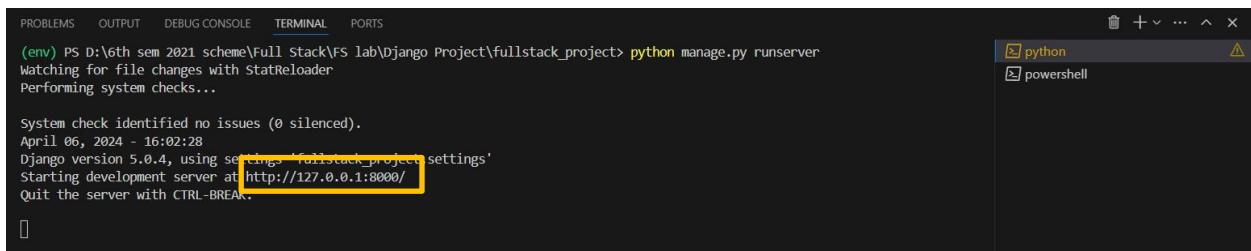


## Step-09: Run the development server

- 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/>



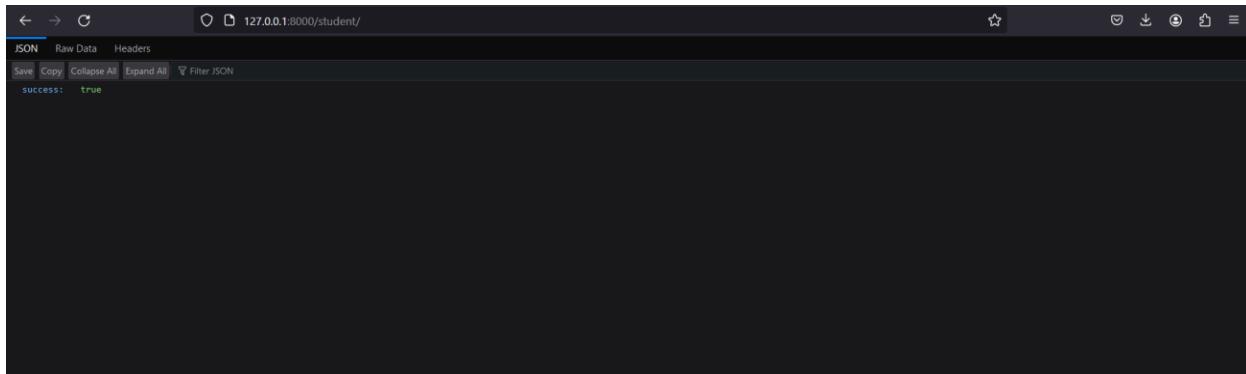
Student Registration

First name:

Last name:

Email:

**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', '')

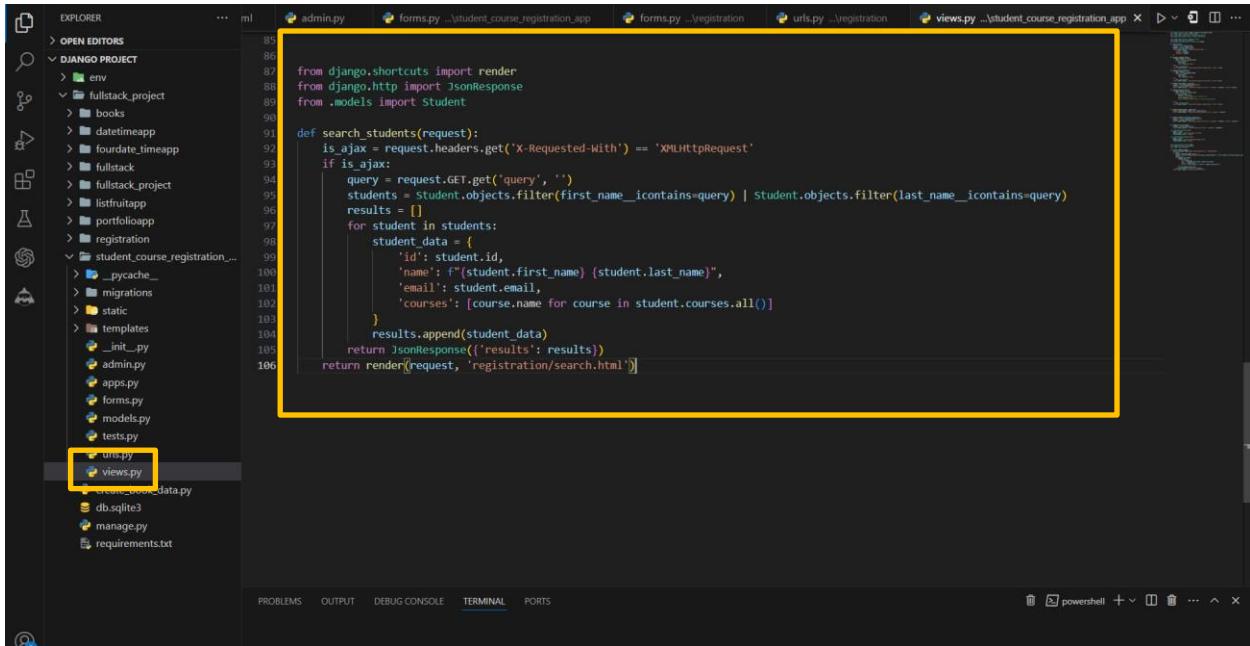
        students = Student.objects.filter(first_name__icontains=query) |
        Student.objects.filter(last_name__icontains=query)

        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><strong>Email:</strong> ' + student.email + '</p>';  
                        html += '<p><strong>Courses:</strong> ' + student.courses.join(',') + '</p>';  
                        html += '</li>';  
                    }  
                    html += '</ul>';  
                }  
                $('#search-results').html(html);  
            }  
        });  
    });  
});
```

```
        }

        html += '</ul>';

    } else {

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

    }

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

}

});

});

});

</script>

</head>

<body>

{% csrf_token %}

<div class="container">

<h1>Search Students</h1>

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

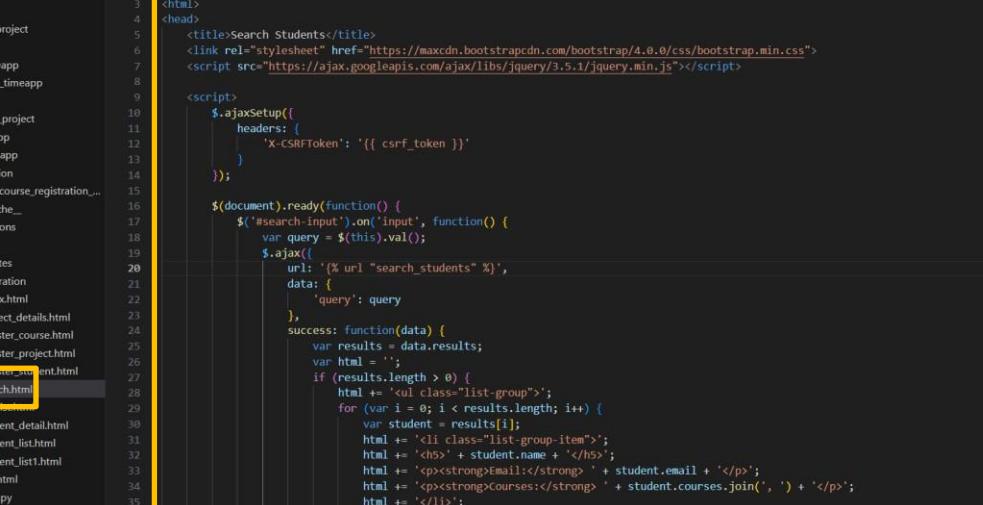
<div id="search-results"></div>

</div>

</body>

</html>
```

**21CS62 | Full Stack Django Development|**



```
index.html stu_list.html views.py \datetime.html current_datetime.html urls.py \datetime.html urls.py Multitask project search.html

EXPLORER
> OPEN EDITORS
> DJANGO PROJECT
  > env
  & fullstack_project
    > books
    > datetimeapp
    > foudrate_timeapp
    > fullstack
    & fullstack_project
      > listviewapp
      > portfolioapp
      > registration
      & student_course_registration...
        > _pycache_ ...
        & migrations
        > static
      & templates
        & registration
          index.html
          project_details.html
          register_course.html
          register_project.html
          register_student.html
          search.html
        & student
          student_detail.html
          student_list.html
          student_list1.html
        base.html
        __init__.py
        admin.py
        apps.py
        forms.py
        models.py

1  <% load static %>
2  <!DOCTYPE html>
3  <html>
4  <head>
5    <title>Search Students</title>
6    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
7    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
8
9
10   <script>
11     $.ajaxSetup({
12       headers: {
13         'X-CSRFToken': '{{ csrf_token }}'
14       }
15     });
16
17   $(document).ready(function() {
18     $('#search-input').on('input', function() {
19       var query = $(this).val();
20       $.ajax({
21         url: '{% url "search_students" %}',
22         data: {
23           'query': query
24         },
25         success: function(data) {
26           var results = data.results;
27           var html = '';
28           if (results.length > 0) {
29             html += '<ul class="list-group">';
30             for (var i = 0; i < results.length; i++) {
31               var student = results[i];
32               html += '<li class="list-group-item">';
33               html += '<h3>' + student.name + '</h3>';
34               html += '<p><strong>Email:</strong> ' + student.email + '</p>';
35               html += '<p><strong>Courses:</strong> ' + student.courses.join(', ') + '</p>';
36               html += '</li>';
37             }
38           }
39         })
40       })
41     )
42   )
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
```

- 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 several tabs open. The left sidebar displays the project structure under 'EXPLORER'.

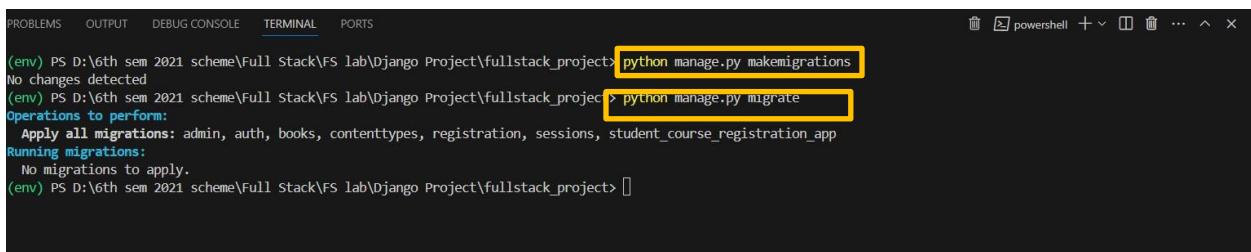
- OPEN EDITORS**
- DIANGO PROJECT**
- fullstack\_project**
  - portfolioapp
  - registration
  - student\_course\_registration...
    - \_\_pycache\_\_
    - migrations
    - static
    - templates
      - registration
        - index.html
        - project\_details.html
        - register\_course.html
        - 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

The main editor area shows the `urls.py` file for the `student_course_registration` application. A yellow box highlights the following code block:

```
path('search/', views.search_students, name='search_students'),  
path('student_list1/', views.StudentList1View.as_view(), name='student_list1'),
```

The bottom navigation bar includes 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', and 'PORTS'. The status bar at the bottom right shows 'powershell' and other icons.

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



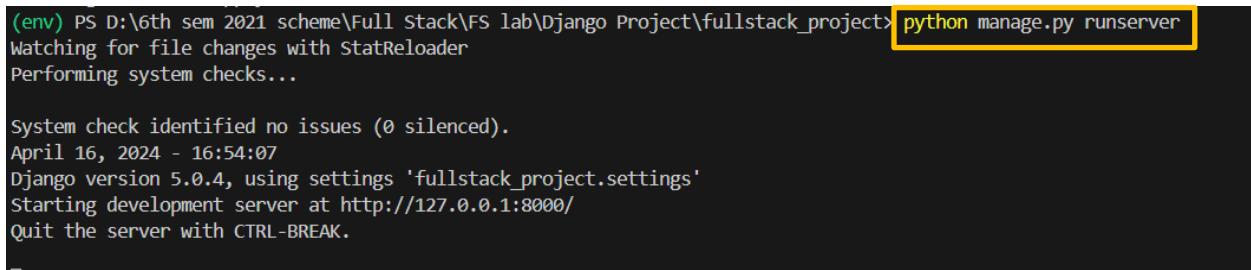
A screenshot of a VS Code terminal window. The title bar shows tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is selected), and PORTS. The terminal content shows a PowerShell session in an environment named '(env)'. The user runs 'python manage.py makemigrations' and 'python manage.py migrate'. The output indicates no changes detected and no migrations to apply. The terminal ends with '(env) PS D:\6th sem 2021 scheme\Full Stack\FS lab\Django Project\fullstack\_project> []'.

**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

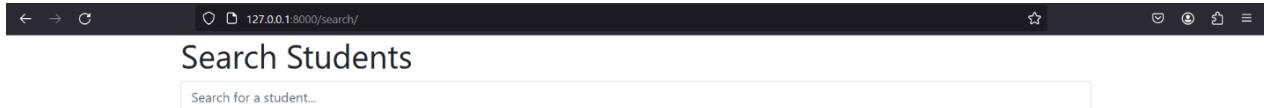
**python manage.py runserver**

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

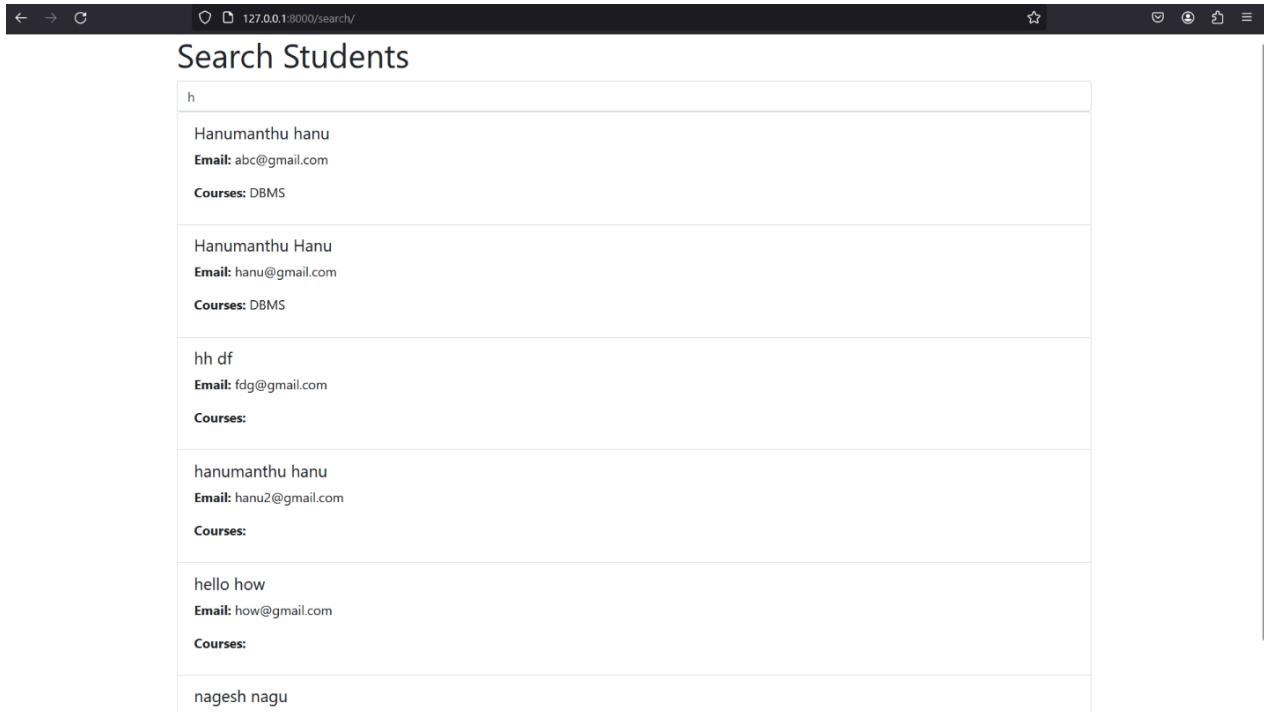


A screenshot of a VS Code terminal window showing the output of the 'python manage.py runserver' command. The terminal shows the command being run, followed by system checks, and the start of the development server at 'http://127.0.0.1:8000/'. It also includes a note to quit the server with CTRL-BREAK.

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



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



**Fig: Search Screen**