

Pre-Requisite:

- Install Python
- Install VSCode

Steps

Program 5: *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.*

1. Setting Up the Project

First, create a new Django project and app.

```
python -m django startproject project5
```

```
cd .\project5\
```

```
python manage.py startapp program5
```

2. Defining Models

We need to define models for `Student`, `Course`, and an intermediary model `Enrollment` for the many-to-many relationship.

Edit `program5/models.py` as follows:

```
from django.db import models

class Student(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
    email = models.EmailField(unique=True)

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

class Course(models.Model):
    name = models.CharField(max_length=100)
    description = models.TextField()

    def __str__(self):
```

```

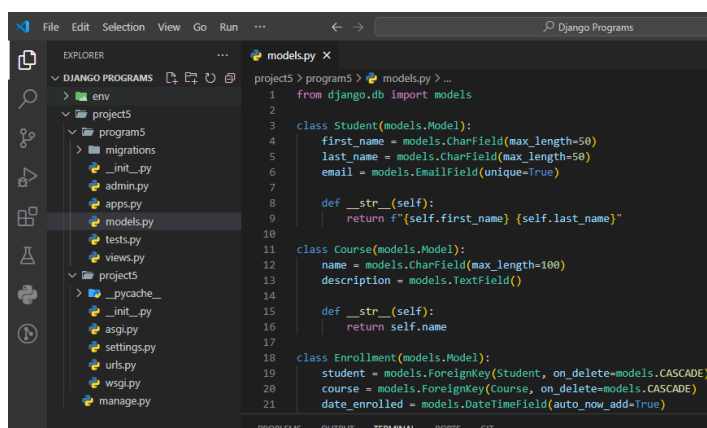
        return self.name

class Enrollment(models.Model):
    student = models.ForeignKey(Student, on_delete=models.CASCADE)
    course = models.ForeignKey(Course, on_delete=models.CASCADE)
    date_enrolled = models.DateTimeField(auto_now_add=True)

    class Meta:
        unique_together = ('student', 'course')

    def __str__(self):
        return f"{self.student} enrolled in {self.course}"

```



3. Creating and Applying Migrations

using command:

```

python manage.py makemigrations
python manage.py migrate

```

4. Registering Models in Admin

To manage the models via the Django admin interface, register them in program5/admin.py:

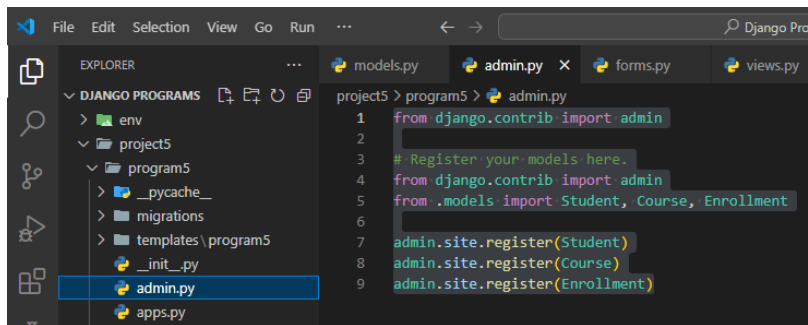
```

from django.contrib import admin

# Register your models here.
from django.contrib import admin
from .models import Student, Course, Enrollment

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

```



5. Creating Views and Forms

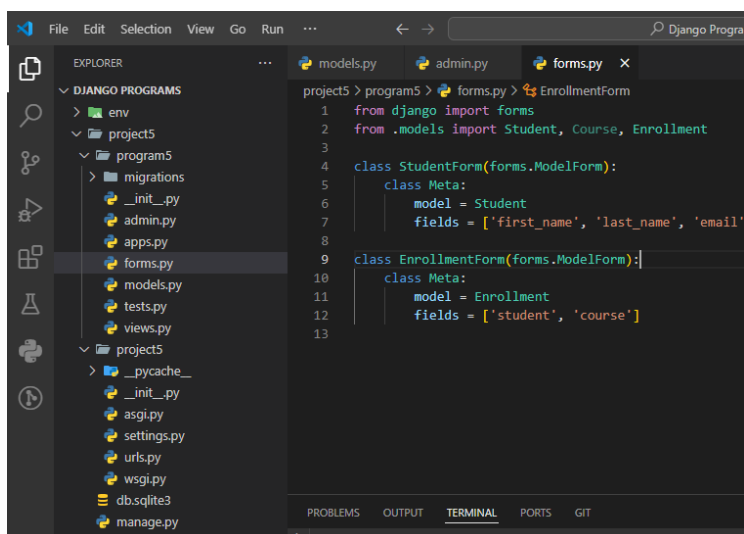
Now, let's create views and forms for student registration and displaying the list of students enrolled in a course.

Create a `forms.py` file in the **program5** app:

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

class StudentForm(forms.ModelForm):
    class Meta:
        model = Student
        fields = ['first_name', 'last_name', 'email']

class EnrollmentForm(forms.ModelForm):
    class Meta:
        model = Enrollment
        fields = ['student', 'course']
```



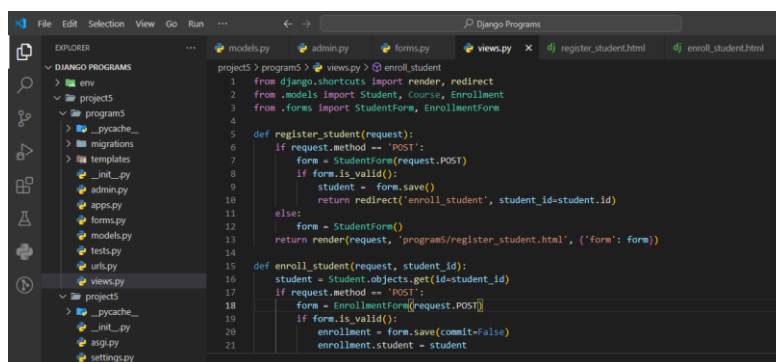
Update `views.py` to handle registration and listing:

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

def register_student(request):
    if request.method == 'POST':
        form = StudentForm(request.POST)
        if form.is_valid():
            student = form.save()
            return redirect('enroll_student', student_id=student.id)
    else:
        form = StudentForm()
    return render(request, 'program5/register_student.html', {'form': form})

def enroll_student(request, student_id):
    student = Student.objects.get(id=student_id)
    if request.method == 'POST':
        form = EnrollmentForm(request.POST)
        if form.is_valid():
            enrollment = form.save(commit=False)
            enrollment.student = student
            enrollment.save()
            return redirect('enrollment_list')
    else:
        form = EnrollmentForm(initial={'student': student})
    return render(request, 'program5/enroll_student.html', {'form': form,
'student': student})

def enrollment_list(request):
    enrollments = Enrollment.objects.all()
    return render(request, 'program5/enrollment_list.html', {'enrollments':
enrollments})
```



5. Creating Templates and subfolder program5 inside which we must have 3 html templates:

- **register_student.html**

```
<!DOCTYPE html>

<html>

<head>

    <title>Register Student</title>

</head>

<body>

    <h1>Register Student</h1>

    <form method="post">

        { % csrf_token % }

        {{ form.as_p }}

        <button type="submit">Register</button>

    </form>

</body>

</html>
```

- **enroll_student.htm**

```
• <!DOCTYPE html>
• <html>
• <head>
•     <title>Enroll Student</title>
• </head>
• <body>
•     <h1>Enroll Student in a Course</h1>
•     <form method="post">
•         { % csrf_token % }
•         {{ form.as_p }}
•         <button type="submit">Enroll</button>
•     </form>
• </body>
• </html>
•
```

- **enrollment_list.html**

```
• <!DOCTYPE html>
• <html>
• <head>
•     <title>Enrollment List</title>
• </head>
• <body>
•     <h1>Enrollment List</h1>
•     <ul>
•         {% for enrollment in enrollments %}
•             <li>{{ enrollment.student }} enrolled in {{
enrollment.course }} on {{ enrollment.date_enrolled }}</li>
•             {% endfor %}
•     </ul>
• </body>
• </html>
```

6. Configuring URLs

In main project url do these changes **project5/urls.py**

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('register_student/', views.register_student,
name='register_student'),
    path('enroll_student/<int:student_id>/', views.enroll_student,
name='enroll_student'),
    path('enrollment_list/', views.enrollment_list, name='enrollment_list'),
]
```

7. Add the application in settings installed apps

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'program5'
]
```

8. Ensure Courses Exist in the Database

in terminal execute this command : **python manage.py shell**
and then paste this script

```
from program5.models import Course
Course.objects.create(name='Mathematics',
description='An introduction to mathematics')
```

```
Course.objects.create(name='Science', description='An introduction to science')
```

9. Run Server

python manage.py runserver

10. Testing

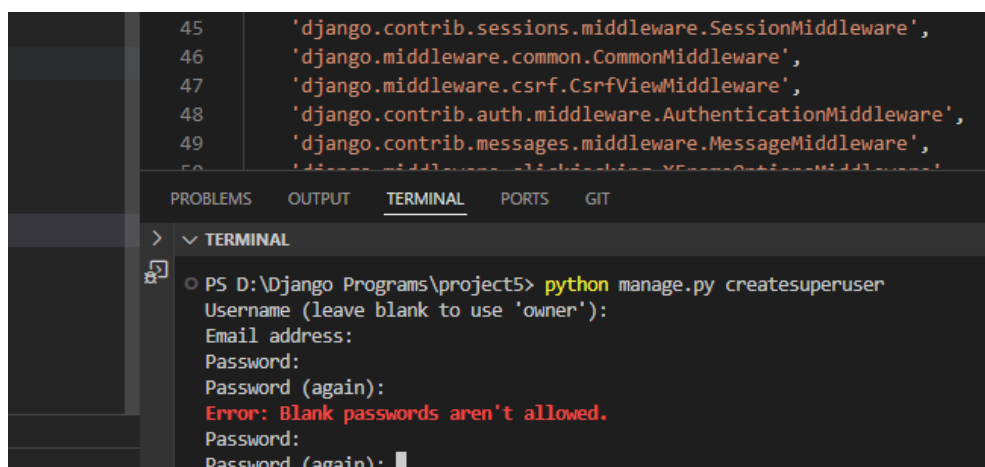
1. **Register a Student:** Visit http://127.0.0.1:8000/register_student/ and register a new student.
2. **Enroll a Student:** After registering, you should be redirected to http://127.0.0.1:8000/enroll_student/<student_id>/ where you can select a course from the dropdown list and enroll the student.
3. **View Enrollments:** Visit http://127.0.0.1:8000/enrollment_list/ to see the list of enrollments.

This should ensure that courses are listed in the enrollment form, and students can be enrolled in selected courses.

In another web browser open

TO check the sqlite database

In the terminal create super user : **python manage.py createsuperuser**
Give a username and password



```
45 'django.contrib.sessions.middleware.SessionMiddleware',
46 'django.middleware.common.CommonMiddleware',
47 'django.middleware.csrf.CsrfViewMiddleware',
48 'django.contrib.auth.middleware.AuthenticationMiddleware',
49 'django.contrib.messages.middleware.MessageMiddleware',
50 'django.middleware.clickjacking.XFrameOptionsMiddleware',
```

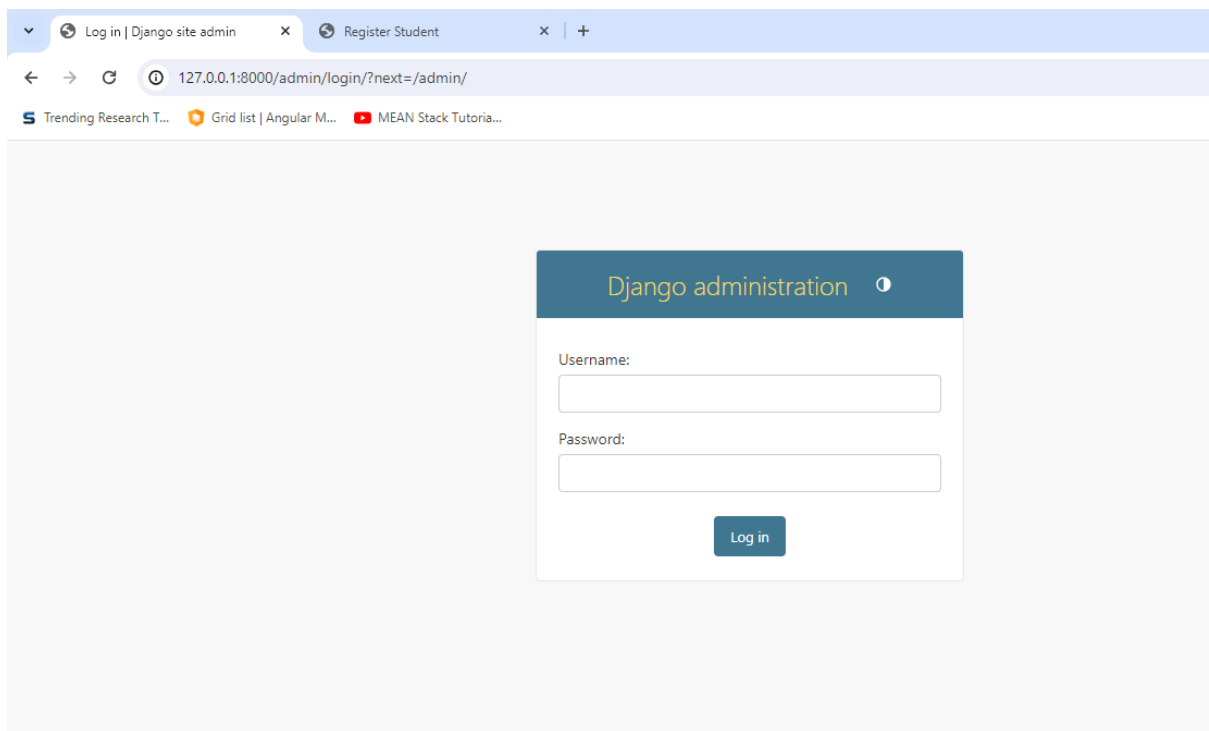
PROBLEMS OUTPUT **TERMINAL** PORTS GIT

> **TERMINAL**

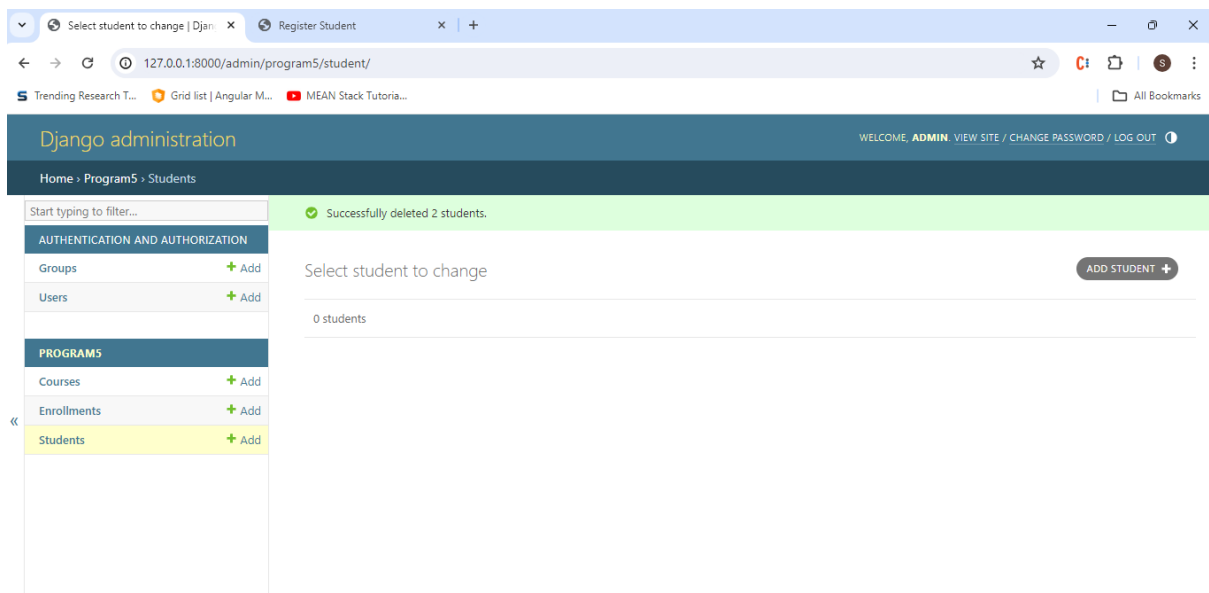
```
PS D:\Django Programs\project5> python manage.py createsuperuser
Username (leave blank to use 'owner'):
Email address:
Password:
Password (again):
Error: Blank passwords aren't allowed.
Password:
Password (again):
```

Run the application **python manage.py runserver**

Now in new browser open admin page and login with the same username and password

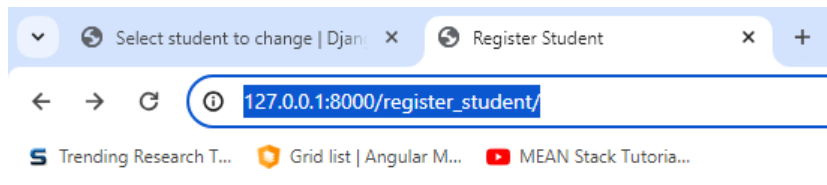


Give the username and password you created and you can see the students registered and enrolled.



You can see that model is created for Courses, Enrollments and Students.

Now register a user: in http://127.0.0.1:8000/register_student/



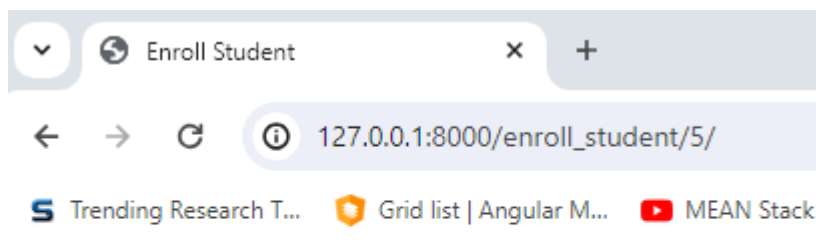
Register Student

First name:

Last name:

- Student with this Email already exists.

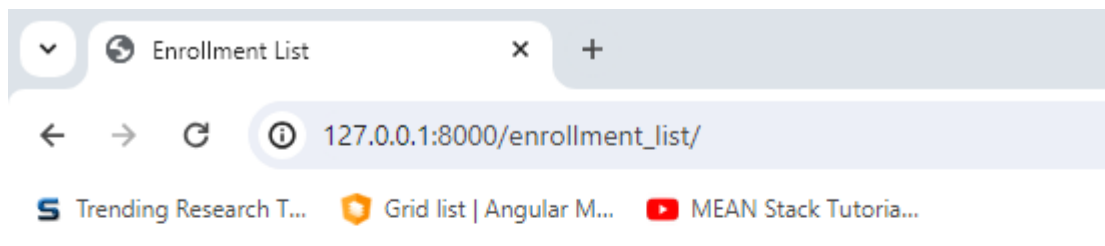
Email:



Enroll Student in a Course

Student:

Course:



Enrollment List

- adam smith enrolled in Science on June 10, 2024, 1:29 p.m.

Program6: *For student and course models created in Lab experiment for Module2, register admin interfaces, perform migrations and illustrate data entry through admin forms.*

To register the `Student` and `Course` models with the Django admin interface, perform migrations, and illustrate data entry through the admin forms, follow these steps:

1. Register Models with the Admin Interface

First, register the `Student` and `Course` models in the Django admin interface. To do this, you need to edit the `admin.py` file in your `program5` app.

Open `program5/admin.py` and add the following code:

```
from django.contrib import admin
```

```
from .models import Student, Course, Enrollment
```

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

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

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

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

```
@admin.register(Course)
```

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

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

```
    search_fields = ('name',)
```

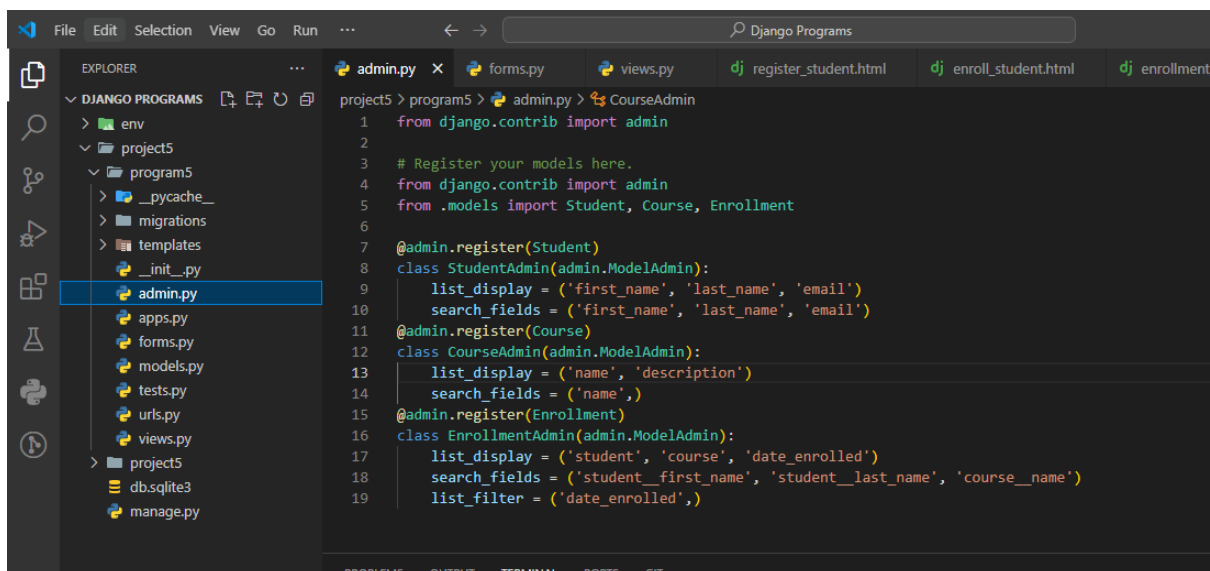
```
@admin.register(Enrollment)
```

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

```
    list_display = ('student', 'course', 'date_enrolled')
```

```
    search_fields = ('student__first_name',  
'student__last_name', 'course__name')
```

```
    list_filter = ('date_enrolled',)
```



2. Perform Migrations

To create the necessary database tables for your models, you need to make and apply migrations.

Run the following commands:

```
python manage.py makemigrations
```

```
python manage.py migrate
```

3. Create a Superuser

To access the Django admin interface, you'll need a superuser account. If you don't already have one, create it using the following command:

```
python manage.py createsuperuser
```

4. Start the Development Server

Start the development server to access the admin interface:

5. Access the Admin Interface

Open your web browser and navigate to `http://127.0.0.1:8000/admin/`. Log in using the superuser account you created.

6. Illustrate Data Entry Through Admin Forms

1. Add Students:

- Click on the "Students" link in the admin interface.
- Click "Add Student" in the top right corner.
- Fill in the student's first name, last name, and email, then click "Save."

2. Add Courses:

- Click on the "Courses" link in the admin interface.
- Click "Add Course" in the top right corner.
- Fill in the course name and description, then click "Save."

3. Add Enrollments:

- Click on the "Enrollments" link in the admin interface.
 - Click "Add Enrollment" in the top right corner.
 - Select a student and a course from the dropdown lists, then click "Save."
-

Program7: *Develop a Model form for student that contains his topic chosen for project, languages used and duration with a model called project.*

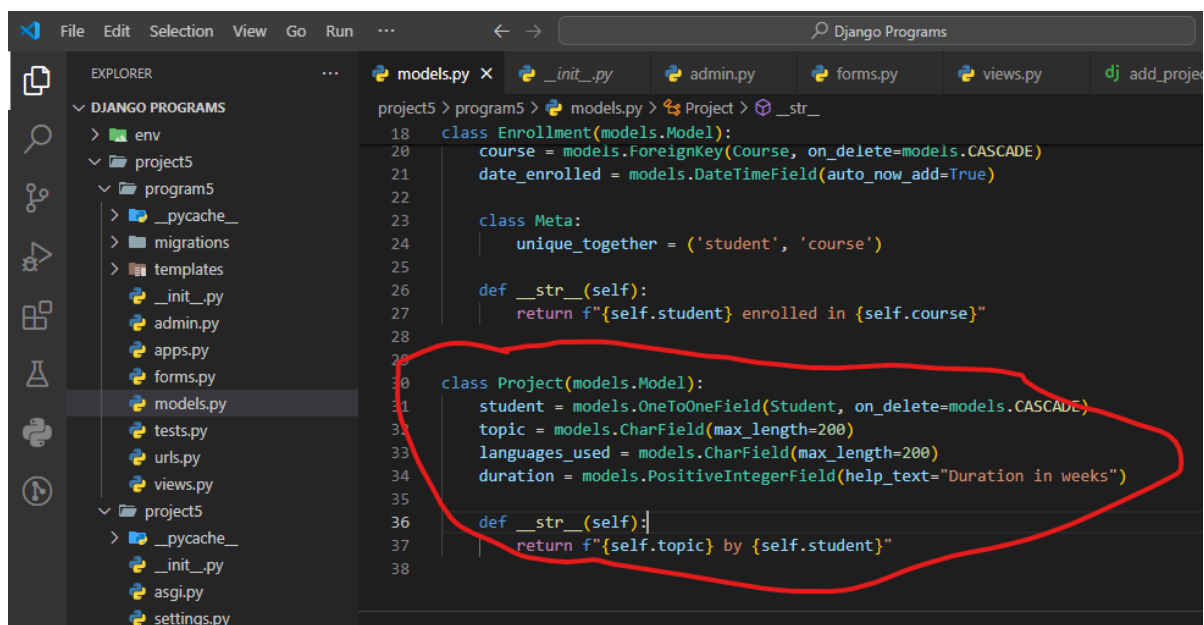
To create a ModelForm for a student that includes fields for their project topic, languages used, and duration, you will need to set up a new `Project` model that has a relationship with the `Student` model. Then, you'll create a form that allows for the entry of these details. Here are the steps:

1. Update Models

First, update your `models.py` file to include the `Project` model and establish a relationship with the `Student` model.

```
class Project(models.Model):
    student = models.OneToOneField(Student, on_delete=models.CASCADE)
    topic = models.CharField(max_length=200)
    languages_used = models.CharField(max_length=200)
    duration = models.PositiveIntegerField(help_text="Duration in weeks")

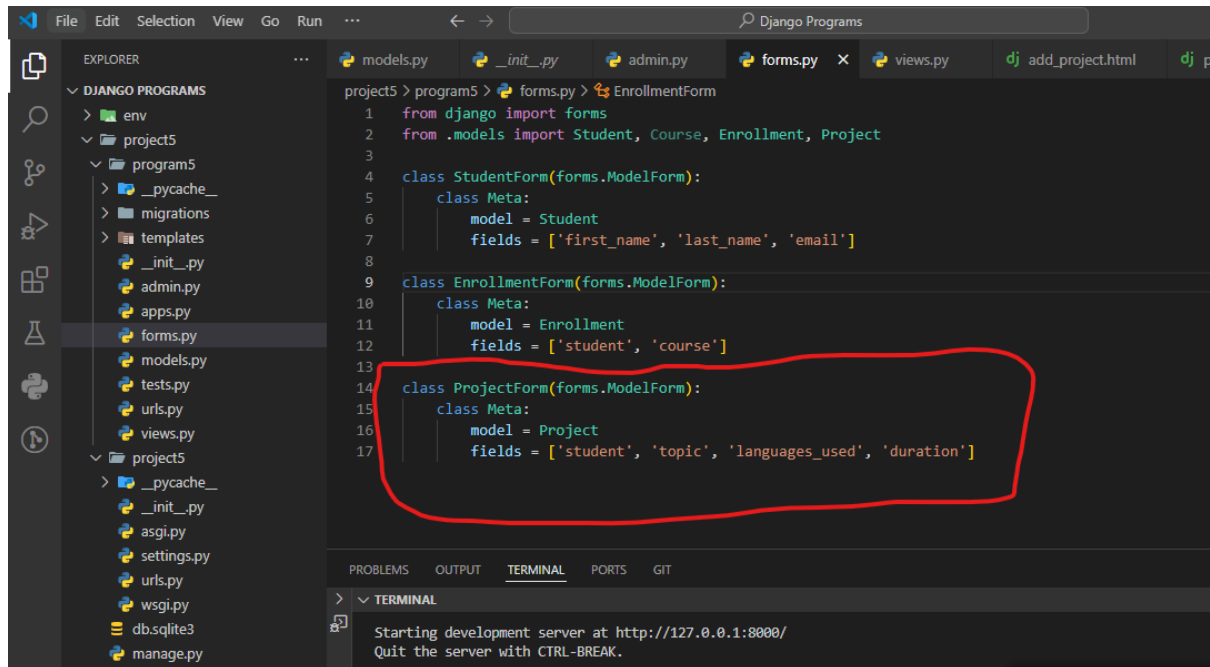
    def __str__(self):
        return f"{self.topic} by {self.student}"
```



2. Create Model Form

Next, create a ModelForm for the `Project` model. Add this to a new file `forms.py` if it doesn't already exist, or update it if it does.

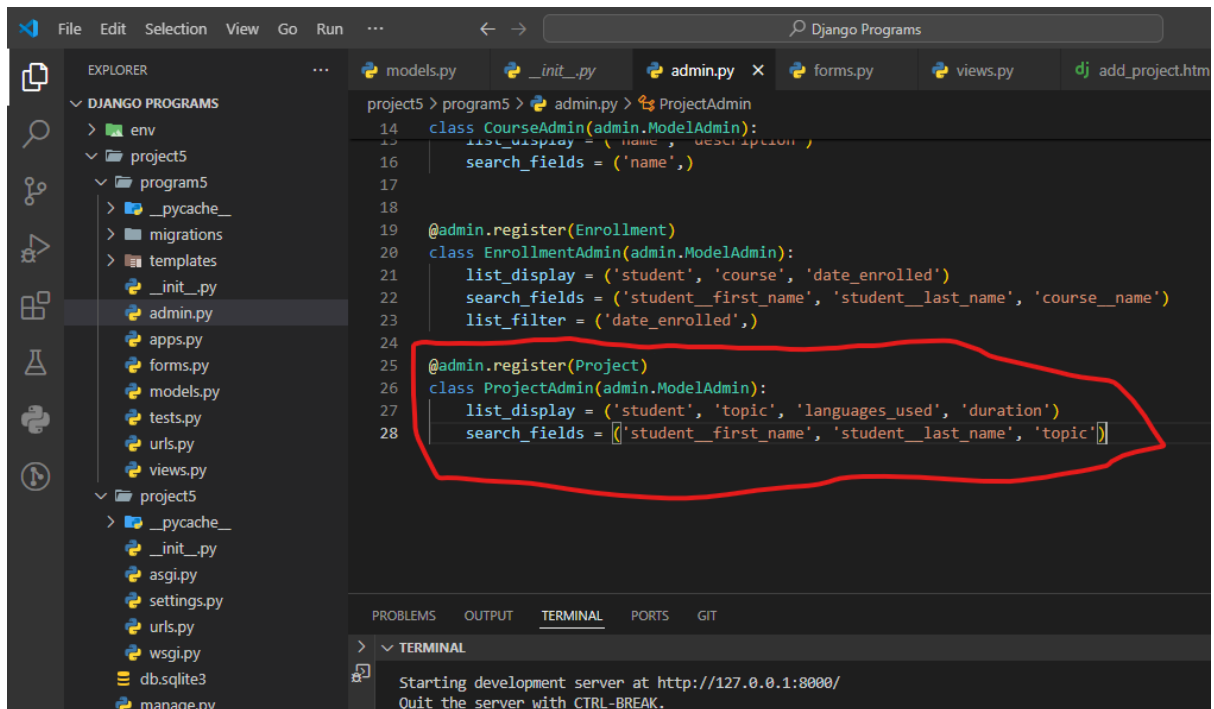
```
class ProjectForm(forms.ModelForm):
    class Meta:
        model = Project
        fields = ['student', 'topic', 'languages_used', 'duration']
```



3. Update Admin Interface

Register the Project model in the admin interface. Update admin.py:

```
@admin.register(Project)
class ProjectAdmin(admin.ModelAdmin):
    list_display = ('student', 'topic', 'languages_used', 'duration')
    search_fields = ('student__first_name', 'student__last_name', 'topic')
```

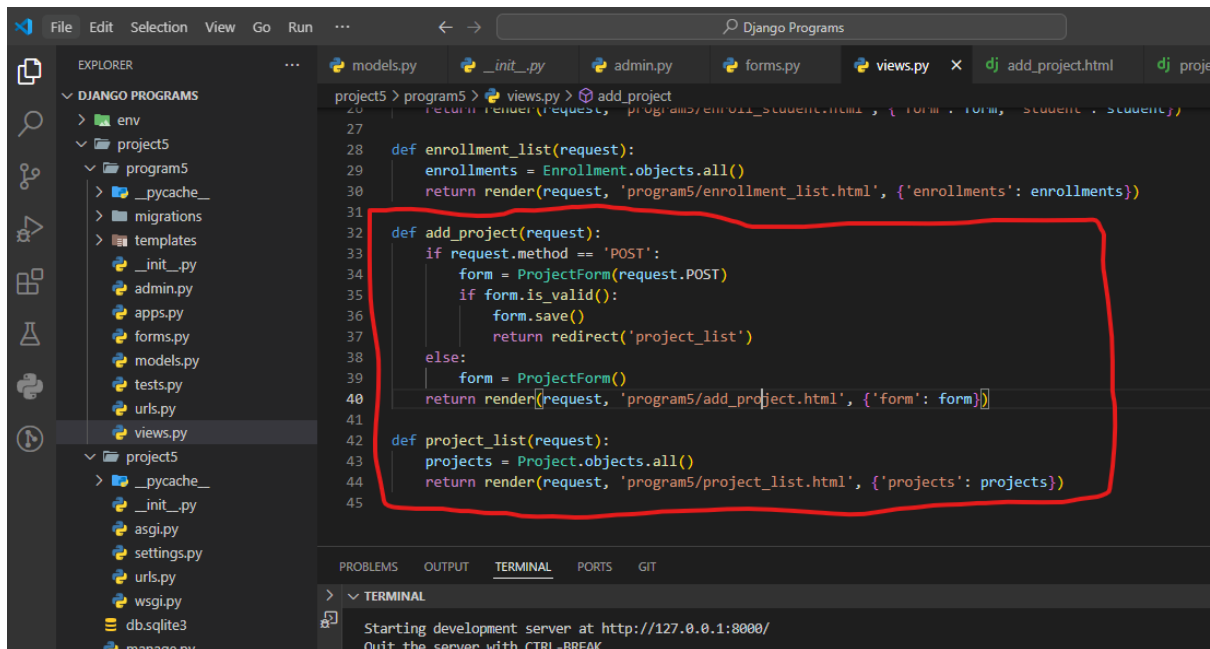


4. Create Views for Project Form

Create a view to handle the creation and update of `Project` instances. Update `views.py`:

```
def add_project(request):
    if request.method == 'POST':
        form = ProjectForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('project_list')
    else:
        form = ProjectForm()
    return render(request, 'program5/add_project.html', {'form': form})

def project_list(request):
    projects = Project.objects.all()
    return render(request, 'program5/project_list.html', {'projects':
projects})
```



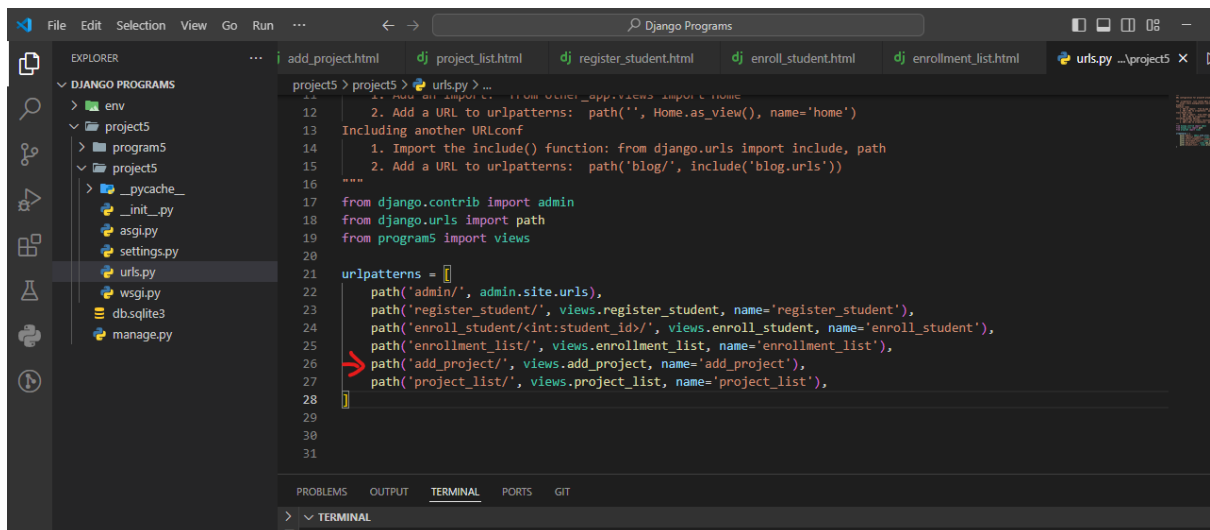
5. Update URLs

Ensure that the URLs are defined for the new views. Update `urls.py`:

```

path('add_project/', views.add_project, name='add_project'),
path('project_list/', views.project_list, name='project_list'),

```



6. Create Templates

Create the necessary templates for adding projects and viewing the project list.

templates/program5/add_project.html:

```

<!DOCTYPE html>
<html>
<head>

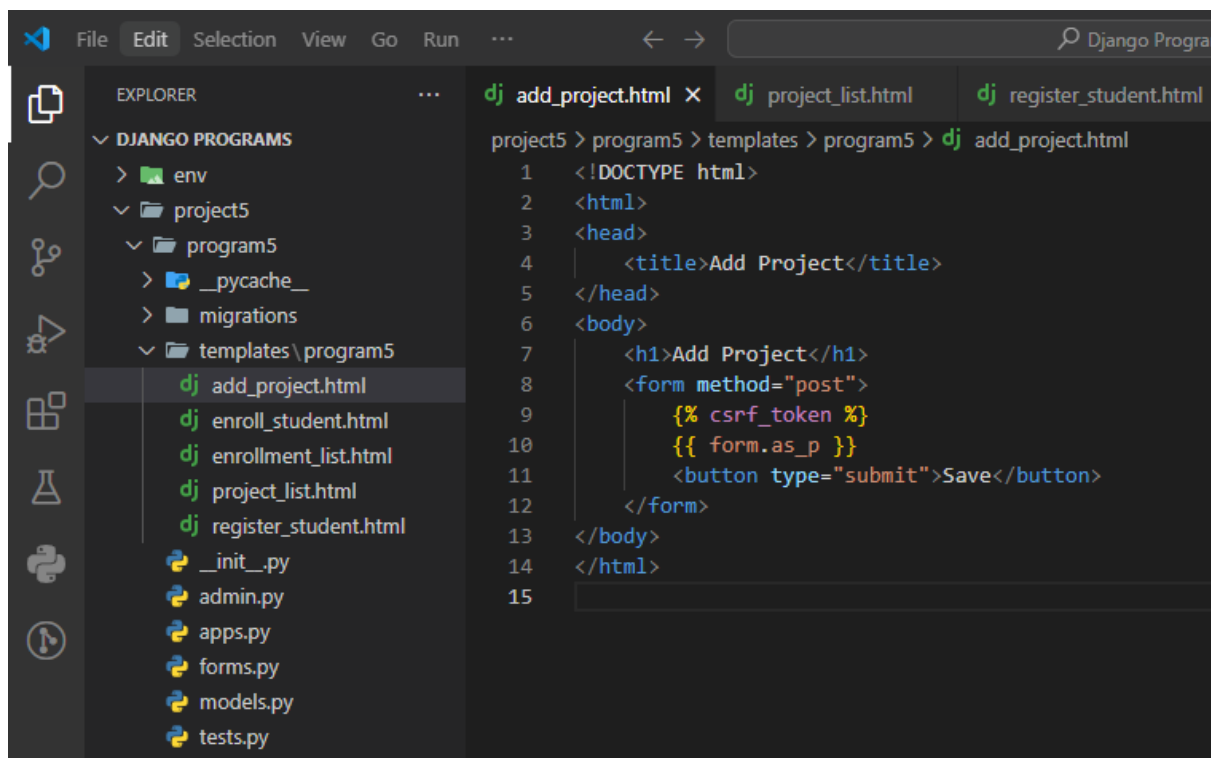
```



```

    <title>Add Project</title>
</head>
<body>
    <h1>Add Project</h1>
    <form method="post">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit">Save</button>
    </form>
</body>
</html>

```



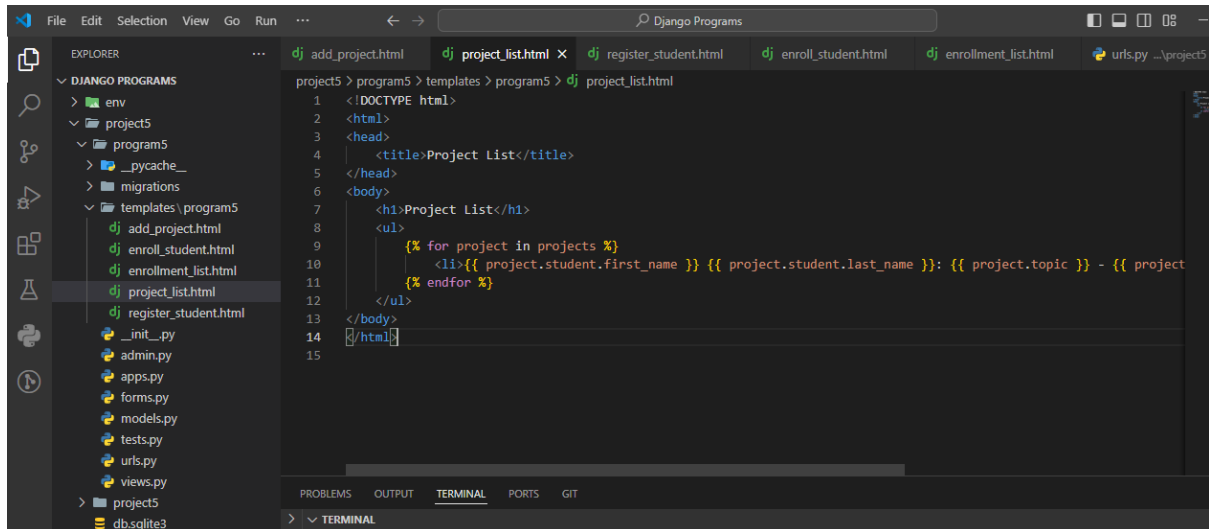
templates/program5/project_list.html:

```

<!DOCTYPE html>
<html>
<head>
    <title>Project List</title>
</head>
<body>
    <h1>Project List</h1>
    <ul>
        {% for project in projects %}
            <li>{{ project.student.first_name }} {{ project.student.last_name }}: {{ project.topic }} - {{ project.languages_used }} ({{ project.duration }} weeks)</li>
        {% endfor %}
    </ul>

```

```
</ul>
</body>
</html>
```



7. Apply Migrations

Make and apply migrations to include the new `Project` model in your database:

```
python manage.py makemigrations
```

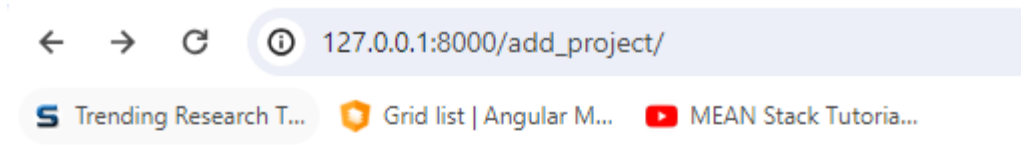
```
python manage.py migrate
```

8. Test the Implementation

1. **Start the development server:**

```
python manage.py runserver
```

2. **Access the admin interface** at `http://127.0.0.1:8000/admin/` and log in with your superuser account.
3. **Add some students** and courses via the admin interface.
4. **Visit** `http://127.0.0.1:8000/add_project/` to add projects for students.
5. **Visit** `http://127.0.0.1:8000/project_list/` to see the list of projects.



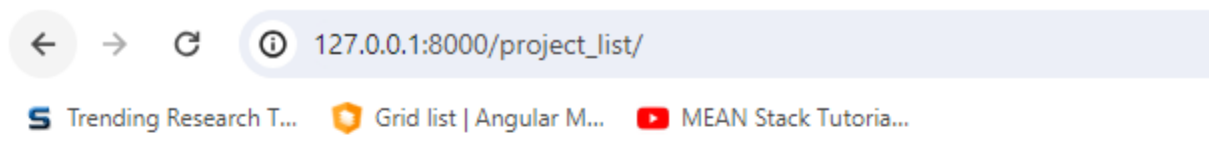
Add Project

Student:

Topic:

Languages used:

Duration: Duration in weeks



Project List

- Hello Page: biology - c (5 weeks)

Lets apply some styles to make page look beautiful

1. Create a CSS File

First, create a CSS file in your static directory. If you don't already have a `static` directory in your app, create one:

`program5/static/program5/`

```
body {
```

```
font-family: Arial, sans-serif;
background-color: #f8f9fa;
margin: 0;
padding: 0;
display: flex;
justify-content: center;
align-items: center;
height: 100vh;
}

.container {
  background-color: #fff;
  padding: 20px;
  border-radius: 8px;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
  width: 500px;
  max-width: 100%;
}

h1 {
  font-size: 24px;
  margin-bottom: 20px;
}

form {
  display: flex;
  flex-direction: column;
}

form .form-group {
  margin-bottom: 15px;
}

form .form-group label {
  margin-bottom: 5px;
  font-weight: bold;
}

form .form-group input,
form .form-group select {
  padding: 8px;
  border: 1px solid #ced4da;
  border-radius: 4px;
  width: 100%;
  box-sizing: border-box;
}

form .form-group input:focus,
```

```

form .form-group select:focus {
    border-color: #80bdff;
    outline: none;
}

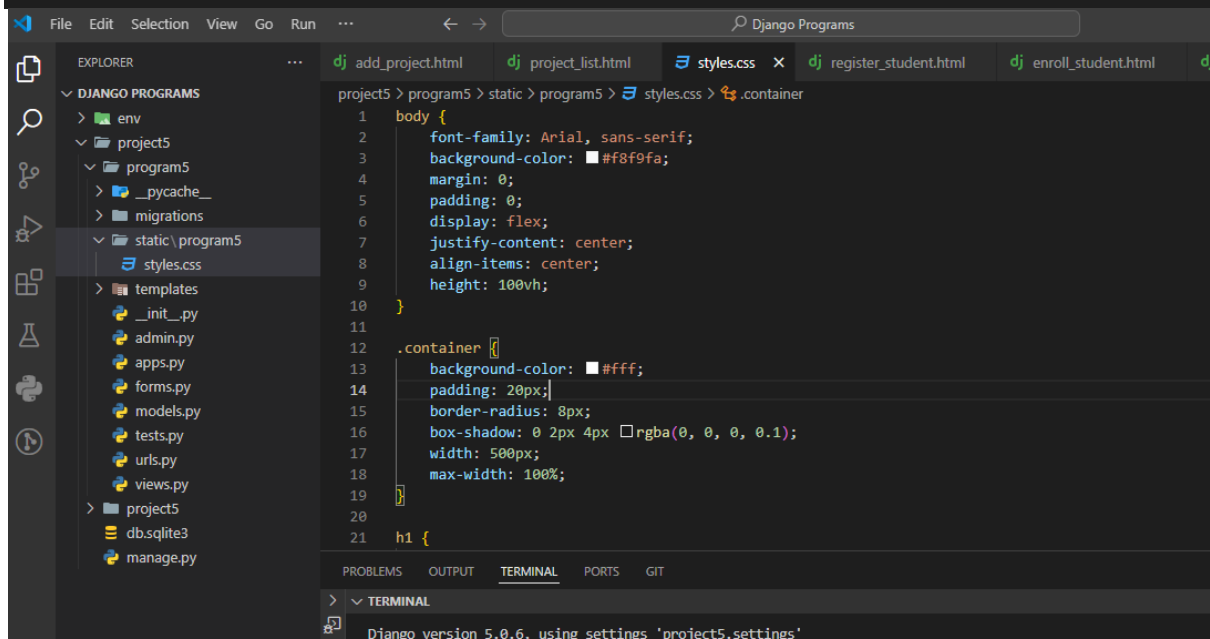
button {
    padding: 10px 15px;
    border: none;
    border-radius: 4px;
    background-color: #007bff;
    color: #fff;
    font-size: 16px;
    cursor: pointer;
}

button:hover {
    background-color: #0056b3;
}

ul {
    list-style-type: none;
    padding: 0;
}

ul li {
    background-color: #f8f9fa;
    margin-bottom: 10px;
    padding: 10px;
    border: 1px solid #ced4da;
    border-radius: 4px;
}

```



2. Update HTML Templates

Next, update your HTML templates to include the CSS file. You can do this by adding the `{% load static %}` template tag and linking to the CSS file in the `<head>` section of each template.

templates/program5/register_student.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>Register Student</title>
    {% load static %}
    <link rel="stylesheet" type="text/css" href="{% static
'program5/styles.css' %}">
</head>
<body>
    <div class="container">
        <h1>Register Student</h1>
        <form method="post">
            {% csrf_token %}
            <div class="form-group">
                {{ form.as_p }}
            </div>
            <button type="submit">Register</button>
        </form>
    </div>
</body>
</html>
```

templates/program5/enroll_student.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>Enroll Student</title>
    {% load static %}
    <link rel="stylesheet" type="text/css" href="{% static
'program5/styles.css' %}">
</head>
<body>
    <div class="container">
        <h1>Enroll {{ student.first_name }} {{ student.last_name }} in a
Course</h1>
        <form method="post">
            {% csrf_token %}
            <div class="form-group">
```

```

        {{ form.as_p }}
    </div>
    <button type="submit">Enroll</button>
</form>
</div>
</body>
</html>

```

templates/program5/enrollment_list.html:

```

<!DOCTYPE html>
<html>
<head>
    <title>Enrollment List</title>
    {% load static %}
    <link rel="stylesheet" type="text/css" href="{% static
'program5/styles.css' %}">
</head>
<body>
    <div class="container">
        <h1>Enrollment List</h1>
        <ul>
            {% for enrollment in enrollments %}
                <li>{{ enrollment.student.first_name }} {{
enrollment.student.last_name }}: {{ enrollment.course.name }}</li>
            {% endfor %}
        </ul>
    </div>
</body>
</html>

```

templates/program5/add_project.html:

```

<!DOCTYPE html>
<html>
<head>
    <title>Add Project</title>
    {% load static %}
    <link rel="stylesheet" type="text/css" href="{% static
'program5/styles.css' %}">
</head>
<body>
    <div class="container">
        <h1>Add Project</h1>
        <form method="post">

```

```

        {% csrf_token %}
        <div class="form-group">
            {{ form.as_p }}
        </div>
        <button type="submit">Save</button>
    </form>
</div>
</body>
</html>

```

templates/program5/project_list.html:

```

<!DOCTYPE html>
<html>
<head>
    <title>Project List</title>
    {% load static %}
    <link rel="stylesheet" type="text/css" href="{% static
'program5/styles.css' %}">
</head>
<body>
    <div class="container">
        <h1>Project List</h1>
        <ul>
            {% for project in projects %}
                <li>{{ project.student.first_name }} {{
project.student.last_name }}: {{ project.topic }} - {{ project.languages_used
}} ({{ project.duration }} weeks)</li>
            {% endfor %}
        </ul>
    </div>
</body>
</html>

```

4. Restart the Development Server

Restart your development server to apply the changes:

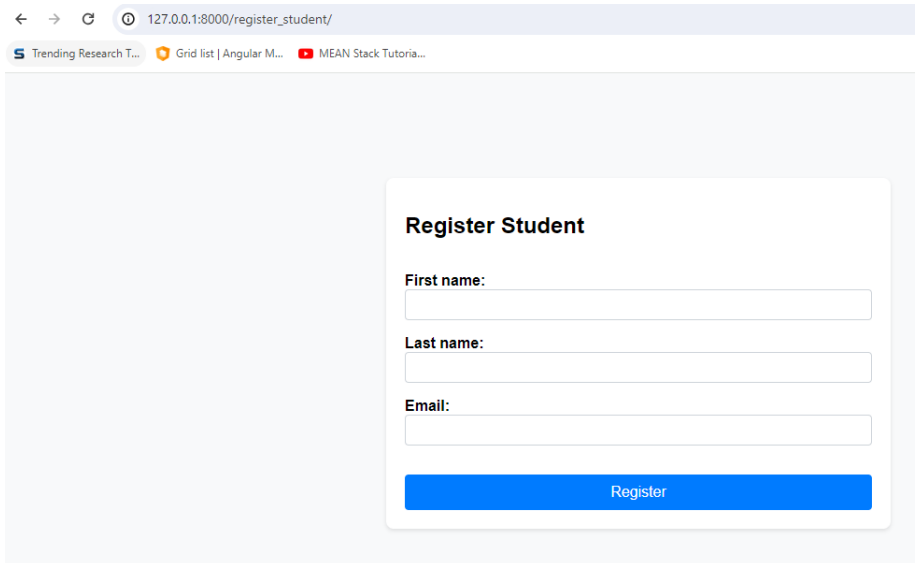
```
python manage.py runserver
```

5. View the Styled Pages

Visit your web pages to see the applied styles:

- Register Student: http://127.0.0.1:8000/register_student/

- Enroll Student: http://127.0.0.1:8000/enroll_student/<student_id>/
- Enrollment List: http://127.0.0.1:8000/enrollment_list/
- Add Project: http://127.0.0.1:8000/add_project/
- Project List: http://127.0.0.1:8000/project_list/



The screenshot shows a web browser window with the address bar displaying `127.0.0.1:8000/register_student/`. The browser tabs include 'Trending Research T...', 'Grid list | Angular M...', and 'MEAN Stack Tutoria...'. The main content area displays a 'Register Student' form with the following fields:

- First name:**
- Last name:**
- Email:**
- Register** (blue button)

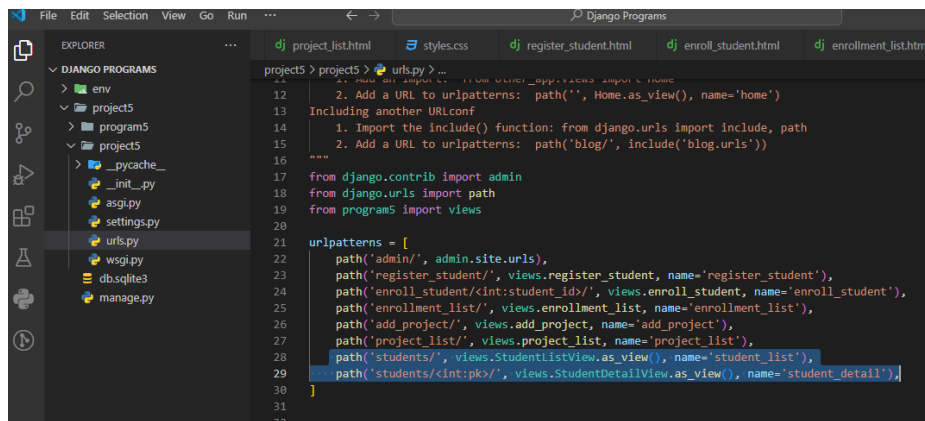
Program 8 : For students enrolment developed in Module 2, create a generic class view which displays list of students and detailview that displays student details for any selected student in the list.

1. Update URLs

First, update your `urls.py` to include paths for the student list and detail views.

`urls.py`:

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



2. Create Generic Class Views

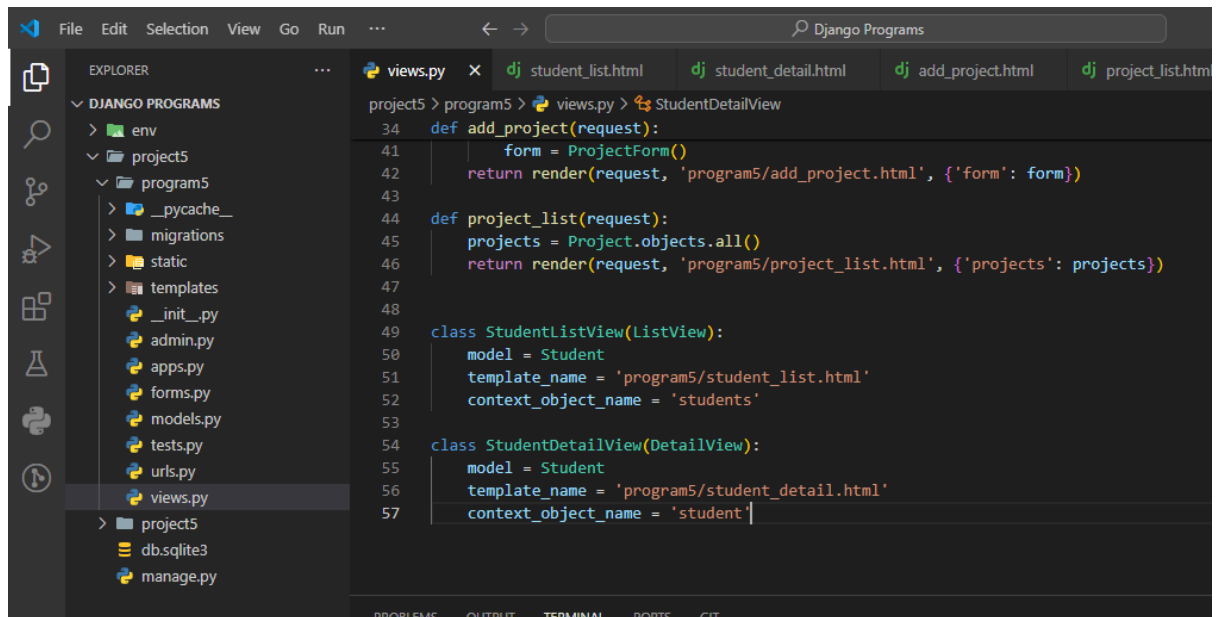
Next, create the `ListView` and `DetailView` for the `Student` model. Add these to your `views.py` file.

views.py:

```
from django.shortcuts import render, redirect
from django.views.generic import ListView, DetailView
from django.shortcuts import render, redirect
from .models import Project, Student, Course, Enrollment
from .forms import ProjectForm, StudentForm, EnrollmentForm

class StudentListView(ListView):
    model = Student
    template_name = 'program5/student_list.html'
    context_object_name = 'students'

class StudentDetailView(DetailView):
    model = Student
    template_name = 'program5/student_detail.html'
    context_object_name = 'student'
```

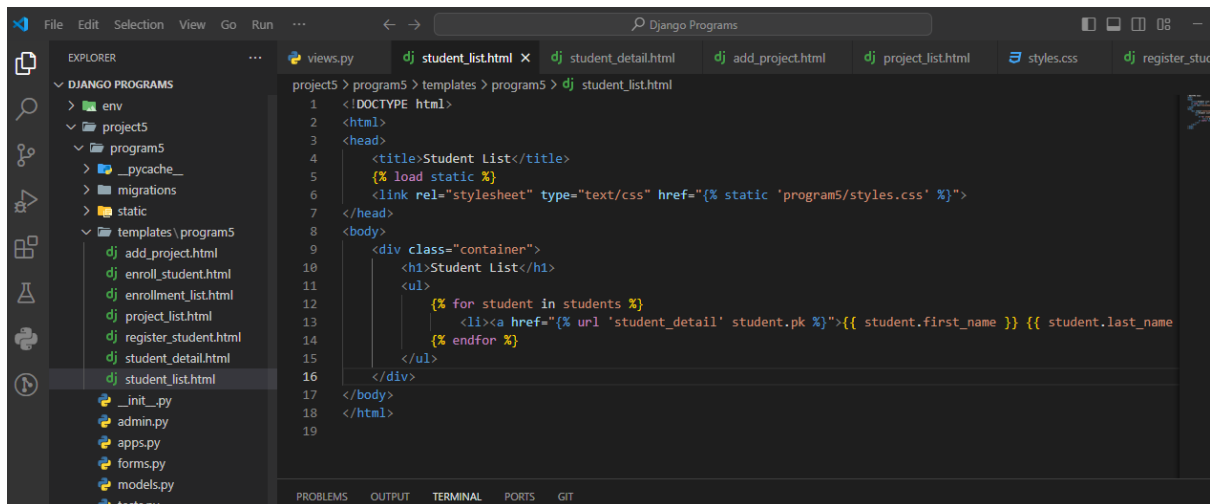


3. Create Templates

Create the templates for the student list and detail views.

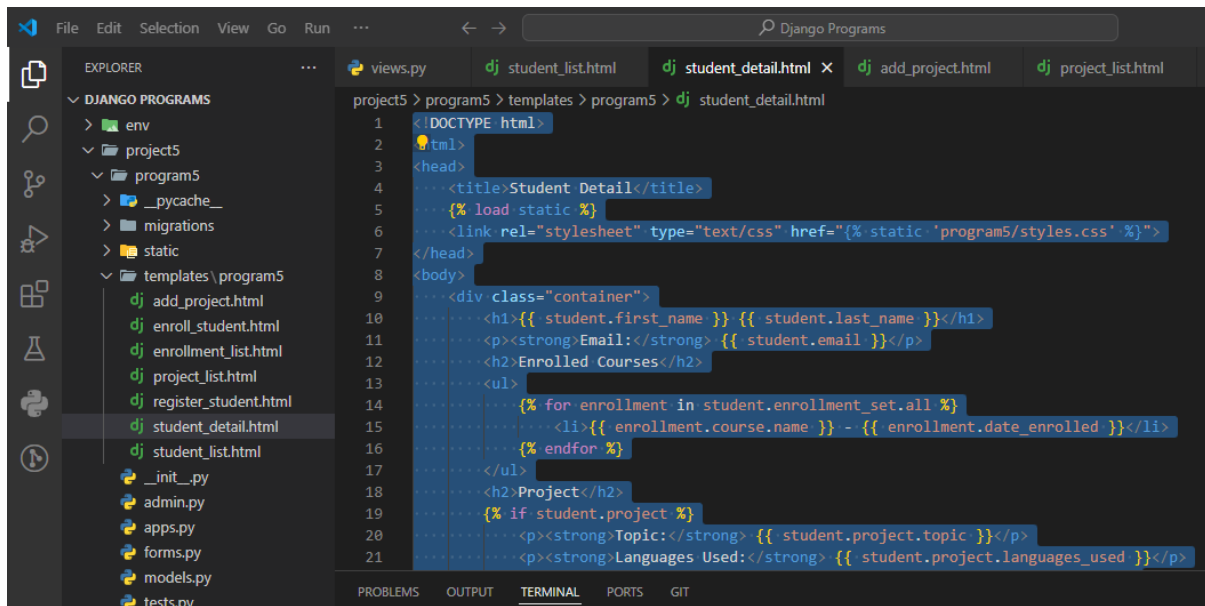
templates/program5/student_list.html:

```
<!DOCTYPE html>
<html>
<head>
  <title>Student List</title>
  {% load static %}
  <link rel="stylesheet" type="text/css" href="{% static
'program5/styles.css' %}">
</head>
<body>
  <div class="container">
    <h1>Student List</h1>
    <ul>
      {% for student in students %}
        <li><a href="{% url 'student_detail' student.pk %}">{{
student.first_name }} {{ student.last_name }}</a></li>
      {% endfor %}
    </ul>
  </div>
</body>
</html>
```



templates/program5/student_detail.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>Student Detail</title>
    {% load static %}
    <link rel="stylesheet" type="text/css" href="{% static
'program5/styles.css' %}">
</head>
<body>
    <div class="container">
        <h1>{{ student.first_name }} {{ student.last_name }}</h1>
        <p><strong>Email:</strong> {{ student.email }}</p>
        <h2>Enrolled Courses</h2>
        <ul>
            {% for enrollment in student.enrollment_set.all %}
                <li>{{ enrollment.course.name }} - {{ enrollment.date_enrolled
}}</li>
            {% endfor %}
        </ul>
        <h2>Project</h2>
        {% if student.project %}
            <p><strong>Topic:</strong> {{ student.project.topic }}</p>
            <p><strong>Languages Used:</strong> {{
student.project.languages_used }}</p>
            <p><strong>Duration:</strong> {{ student.project.duration }}
weeks</p>
        {% else %}
            <p>No project assigned.</p>
        {% endif %}
    </div>
</body>
</html>
```



```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>Student Detail</title>
5 {% load static %}
6 <link rel="stylesheet" type="text/css" href="{% static 'program5/styles.css' %}">
7 </head>
8 <body>
9 <div class="container">
10 <h1>{{ student.first_name }} {{ student.last_name }}</h1>
11 <p><strong>Email:</strong> {{ student.email }}</p>
12 <h2>Enrolled Courses</h2>
13 <ul>
14 {% for enrollment in student.enrollment_set.all %}
15 <li>{{ enrollment.course.name }} - {{ enrollment.date_enrolled }}</li>
16 {% endfor %}
17 </ul>
18 <h2>Project</h2>
19 {% if student.project %}
20 <p><strong>Topic:</strong> {{ student.project.topic }}</p>
21 <p><strong>Languages Used:</strong> {{ student.project.languages_used }}</p>
```

4. Testing the Implementation

1. **Start the development server:** `python manage.py runserver`
2. **Visit the student list page** at <http://127.0.0.1:8000/students/> to see the list of students.
3. **Click on a student's name** to view their details, which will be shown on a page like http://127.0.0.1:8000/students/<student_id>/.

Program 9: Develop example Django app that performs CSV and PDF generation for any models created in previous laboratory component

To create a Django app that can generate CSV and PDF files for the models we've developed (Student, Course, Enrollment, and Project), you'll need to use Python libraries such as `csv` for CSV generation and `reportlab` for PDF generation.

1. Install Required Packages

First, ensure you have the necessary packages installed:

```
pip install reportlab
```

```
28 </html>
29

PROBLEMS OUTPUT TERMINAL PORTS GIT
> ▼ TERMINAL
(env) PS D:\Django Programs\project5> pip install reportlab
```

```
> ▼ TERMINAL
Downloading chardet-5.2.0-py3-none-any.whl (199 kB)
100% |#####| 199.4/199.4 kB 3.0 MB/s eta 0:00:00
Installing collected packages: pillow, chardet, reportlab
Successfully installed chardet-5.2.0 pillow-10.3.0 reportlab-4.2.0

[notice] A new release of pip available: 22.3 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip
(env) PS D:\Django Programs\project5>
```

2. Update Views for CSV and PDF Generation

Add views to generate CSV and PDF files for the `Student` model.

views.py:

```
import csv
from django.http import HttpResponse
from reportlab.lib.pagesizes import letter
from reportlab.pdfgen import canvas
# CSV generation view
def export_students_csv(request):
    response = HttpResponse(content_type='text/csv')
    response['Content-Disposition'] = 'attachment; filename="students.csv"'

    writer = csv.writer(response)
    writer.writerow(['First Name', 'Last Name', 'Email'])

    students = Student.objects.all()
    for student in students:
        writer.writerow([student.first_name, student.last_name,
student.email])

    return response

# PDF generation view
def export_students_pdf(request):
    response = HttpResponse(content_type='application/pdf')
```

```

response['Content-Disposition'] = 'attachment; filename="students.pdf"'

buffer = BytesIO()
p = canvas.Canvas(buffer, pagesize=letter)

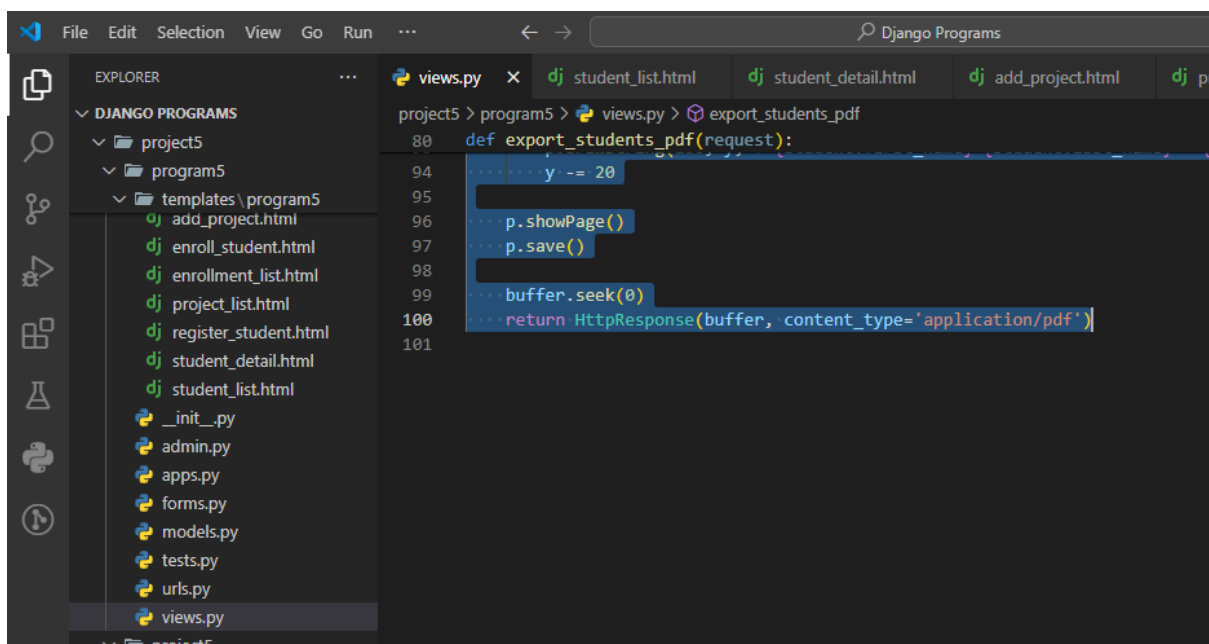
p.drawString(100, 750, "Students List")
p.drawString(100, 730, "=====")

students = Student.objects.all()
y = 710
for student in students:
    p.drawString(100, y, f"{student.first_name} {student.last_name} - {student.email}")
    y -= 20

p.showPage()
p.save()

buffer.seek(0)
return HttpResponse(buffer, content_type='application/pdf')

```



3. Update URLs

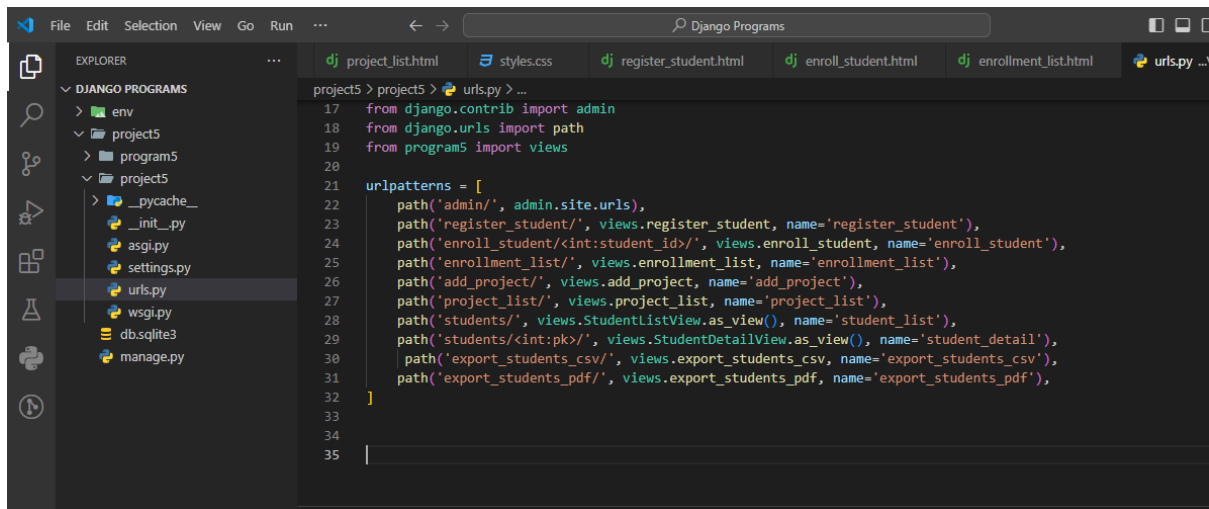
Add the URLs for the CSV and PDF generation views.

urls.py:

```

path('export_students_csv/', views.export_students_csv,
name='export_students_csv'),
path('export_students_pdf/', views.export_students_pdf,
name='export_students_pdf'),

```



4. Update Templates

Add links to the export functionality in the student list template.

templates/program5/student_list.html:

```
<!DOCTYPE html>
<html>
<head>
  <title>Student List</title>
  {% load static %}
  <link rel="stylesheet" type="text/css" href="{% static
'program5/styles.css' %}">
</head>
<body>
  <div class="container">
    <h1>Student List</h1>
    <a href="{% url 'export_students_csv' %}">Export as CSV</a> |
    <a href="{% url 'export_students_pdf' %}">Export as PDF</a>
    <ul>
      {% for student in students %}
        <li><a href="{% url 'student_detail' student.pk %}">{{
student.first_name }} {{ student.last_name }}</a></li>
      {% endfor %}
    </ul>
  </div>
</body>
</html>
```

5. Testing the Implementation

1. **Start the development server:** python manage.py runserver
2. **Visit the student list page** at <http://127.0.0.1:8000/students/>.
3. **Click on the "Export as CSV"** link to download the CSV file.

4. **Click on the "Export as PDF"** link to download the PDF file.