Student Management System Setup Report

GNG5300-A

Jiayiwei

300122703

https://github.com/jiayiwei2/student\_management.git

Project Setup

The Student Management System was built using Django, a web framework for Python. Here is a summary of how I did the setup and the challenges I faced.

Environment Setup

1. Virtual Environment Creation: A virtual environment was created to keep project dependencies separate using the command:

python -m venv env

2. Package Installation: The required packages were installed using pip from the requirements.txt file:

pip install -r requirements.txt

3. Django App Initialization: The project was started using Django's startproject command, and the student management features were added using the startapp command to create the students app.

Challenges Encountered

1. TypeError: 'module' object is not iterable:

This error happened due to incorrect imports. It was fixed by ensuring that the correct data types were being used in the code.

2. Issue with startapp Command:

There were problems when creating the students app, which were resolved by checking the syntax and making sure the virtual environment was activated.

1. Webpage Not Running Correctly:

The webpage did not display properly due to incorrect URL settings in urls.py and issues with template paths. I started over again and fixed this question.

4. GitHub Account Suspension:

The GitHub account was suspended unexpectedly, which interrupted progress. I created a GitHub account and the repository was re-uploaded to continue using GitHub Copilot.

5. Virtual Environment Setup Issues:

Several issues were caused by an incorrect virtual environment setup. Once the environment was correctly activated, the problems were resolved.

Project Structure

The Student Management System has the following structure:

- student\_management: The main project folder with core settings and configuration files.

- settings.py: Contains project settings, like installed apps and database configurations.

- urls.py: Defines the main project routes.

- students: The app folder for managing student-related features.

- models.py: Defines the Student model, which stores student information.

- views.py: Handles the logic for listing, adding, editing, and searching students.

- forms.py: Contains forms for adding and editing student data with validation.

- templates/students: Contains HTML files used for rendering the webpages.

- student\_list.html: Shows the list of students.

- student\_form.html: Provides the form for adding or editing student details.

- student\_detail.html: Displays a single student's details.

- login.html: Handles user login.

Feature Implementation

1. Student Management:

The student records were managed using Django's Object-Relational Mapping (ORM). Below is the Student model from models.py:

class Student(models.Model):

first\_name = models.CharField(max\_length=50)

last\_name = models.CharField(max\_length=50)

email = models.EmailField(unique=True)

date\_of\_birth = models.DateField()

enrollment\_date = models.DateField()

grade = models.IntegerField()

def \_\_str\_\_(self):

return f"{self.first\_name} {self.last\_name}"

2. Forms and Validation:

Forms were created to handle student data input. Below is an example from forms.py:

class StudentForm(forms.ModelForm):

class Meta:

model = Student

fields = ['first\_name', 'last\_name', 'email', 'date\_of\_birth', 'enrollment\_date', 'grade']

def clean\_grade(self):

grade = self.cleaned\_data.get('grade')

if grade < 1 or grade > 12:

raise forms.ValidationError("Grade must be between 1 and 12.")

return grade

3. Authentication:

Authentication was added to ensure that only authorized users could manage student records. The login.html template was used for user login.

4. Templates and Views:

Templates were used to display HTML pages. student\_list.html showed all students, and student\_form.html allowed for adding or editing students. Below is an example of the student list view from views.py:

def student\_list(request):

students = Student.objects.all()

return render(request, 'students/student\_list.html', {'students': students})

5. Search Functionality:

A search bar was added to let users search for students by name. Below is how it was implemented in views.py:

def student\_list(request):

query = request.GET.get('q')

if query:

students = Student.objects.filter(first\_name\_\_icontains=query) | Student.objects.filter(last\_name\_\_icontains=query)

else:

students = Student.objects.all()

return render(request, 'students/student\_list.html', {'students': students})

6. Pagination:

Pagination was added to limit the number of students displayed per page:

from django.core.paginator import Paginator

def student\_list(request):

students = Student.objects.all()

paginator = Paginator(students, 10) # Show 10 students per page

page\_number = request.GET.get('page')

page\_obj = paginator.get\_page(page\_number)

return render(request, 'students/student\_list.html', {'page\_obj': page\_obj})

7. Error Handling and Validation:

Proper error handling was added to manage missing student records or incorrect data inputs. For example, handling a student ID that does not exist:

from django.shortcuts import get\_object\_or\_404

def student\_detail(request, pk):

student = get\_object\_or\_404(Student, pk=pk)

return render(request, 'students/student\_detail.html', {'student': student})

Screenshots and Examples

Below are screenshots of some issues faced during development:

-Missing Installed App Error:

The screenshot below shows the error "No installed app with label 'students'". This happened because the students app was not added correctly in INSTALLED\_APPS in settings.py.

Conclusion

This project faced challenges, mainly with setting up the environment and issues with GitHub. These were resolved with persistent troubleshooting. This experience highlighted the importance of properly setting up the virtual environment in Django projects.

