**Student Management System Report**

Xinye Zhu (300367809)

**Introduction**

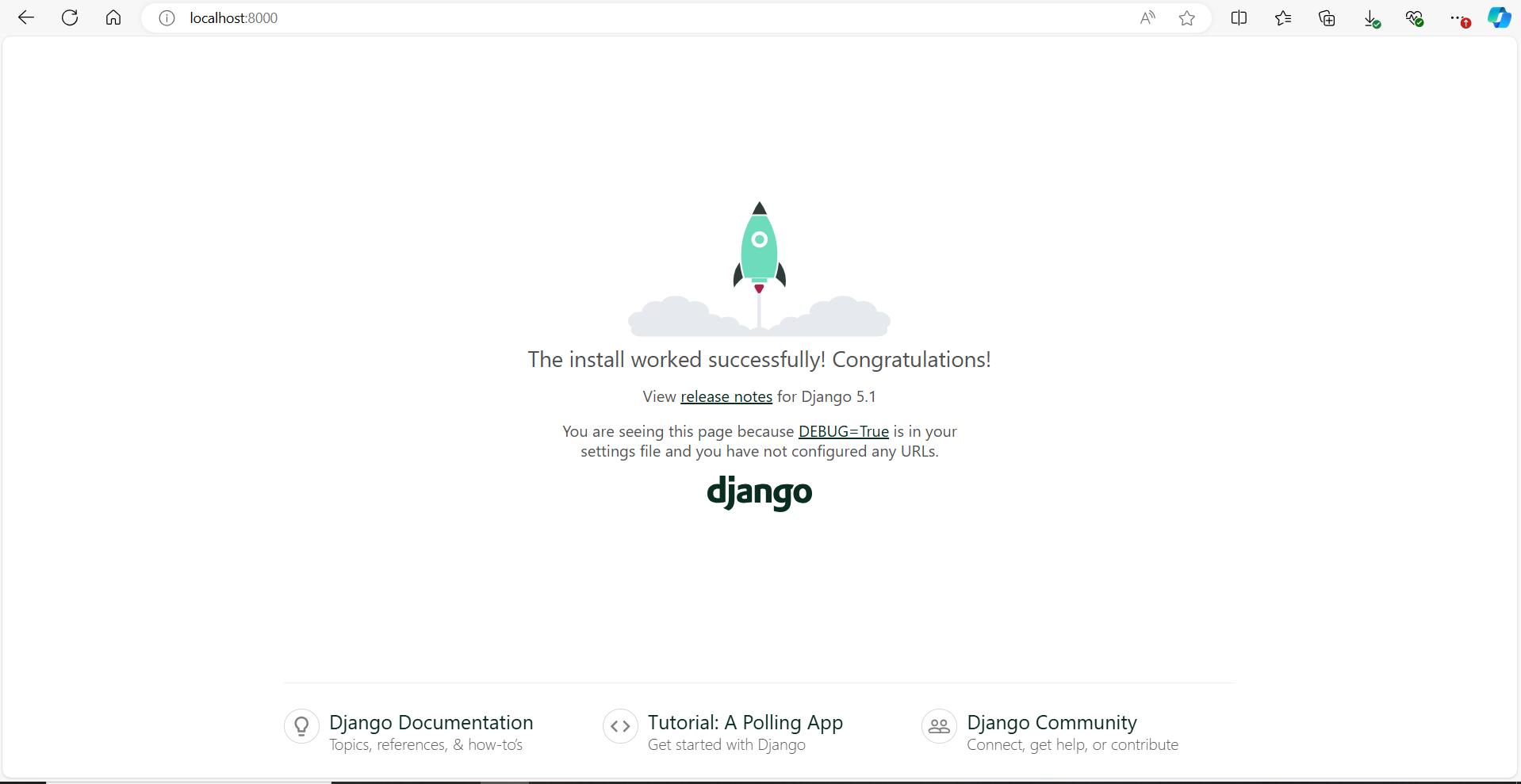
Student management system is a web application with functionalities such as display student list and student detail, add and edit student, and user authentication. This project is built with Django web framework.

**Building Steps**

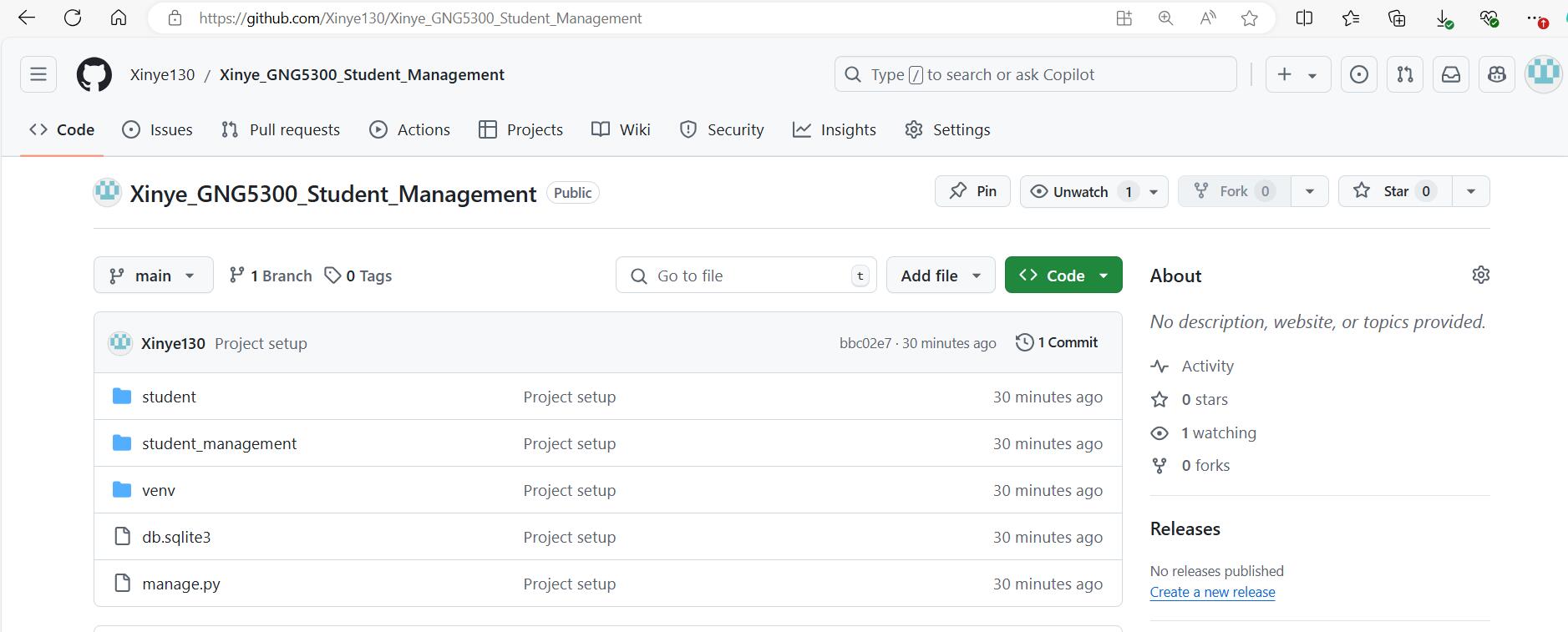
**1. Project Setup**

Firstly, I created the directory **Xinye\_GNG5300\_Student\_Management** for the project. Then I set up a virtual environment with command **python -m venv venv** and activated it using **source venv/Scripts/activate** in the git bash terminal in VSCode. I installed Django using **python -m pip install Django** in the virtual environment.

In the project directory, I created the project student\_management: **django-admin startproject student\_management .** , and the app student: **python manage.py startapp student**. I also installed the app in student\_management/settings.py under INSTALLED\_APPS. Then, I run python **manage.py runserver** to check my site in http://localhost:8000, which is shown in the image below.

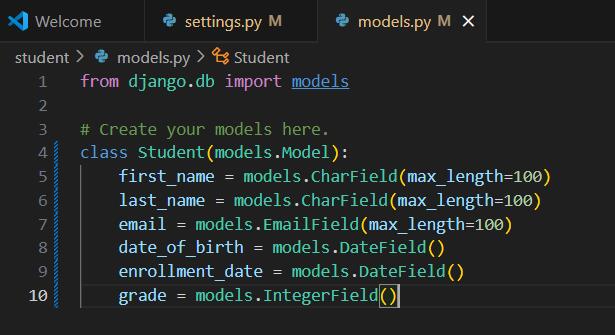


Here’s the project structure in my first commit and push to GitHub:



**2. Student Model Creation**

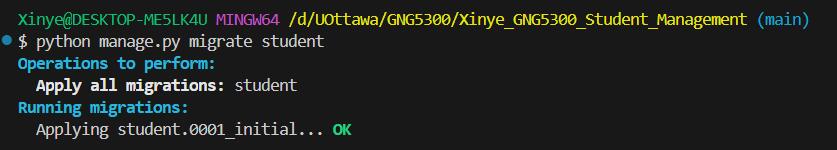
In my student app, I created a models.py file and implemented the Student model:



Then I created a migration, and create my data base using migrate command:

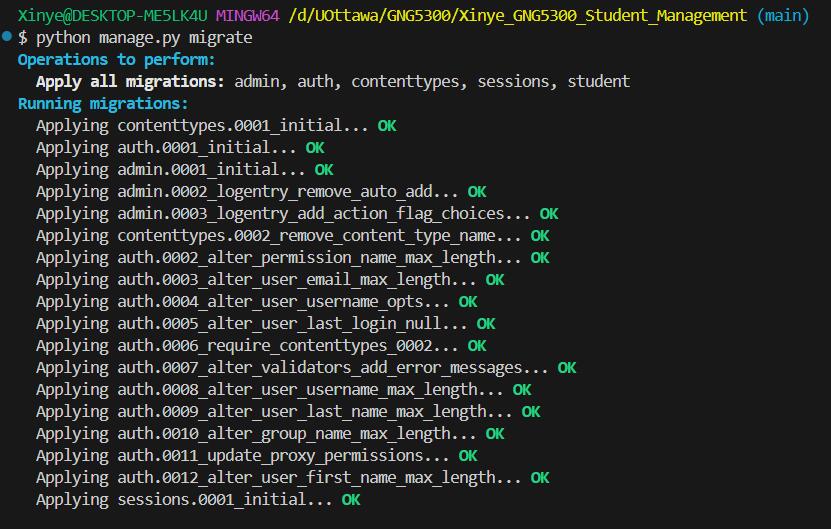
A screen shot of a computer

Description automatically generated

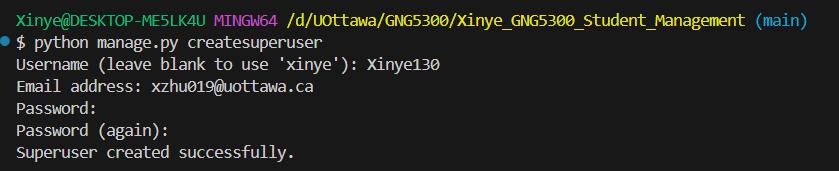


**3. Admin Interface**

Firstly, I applied the existing migrations Django provides:



Then I added myself as a superuser and logged into http://localhost:8000/admin.



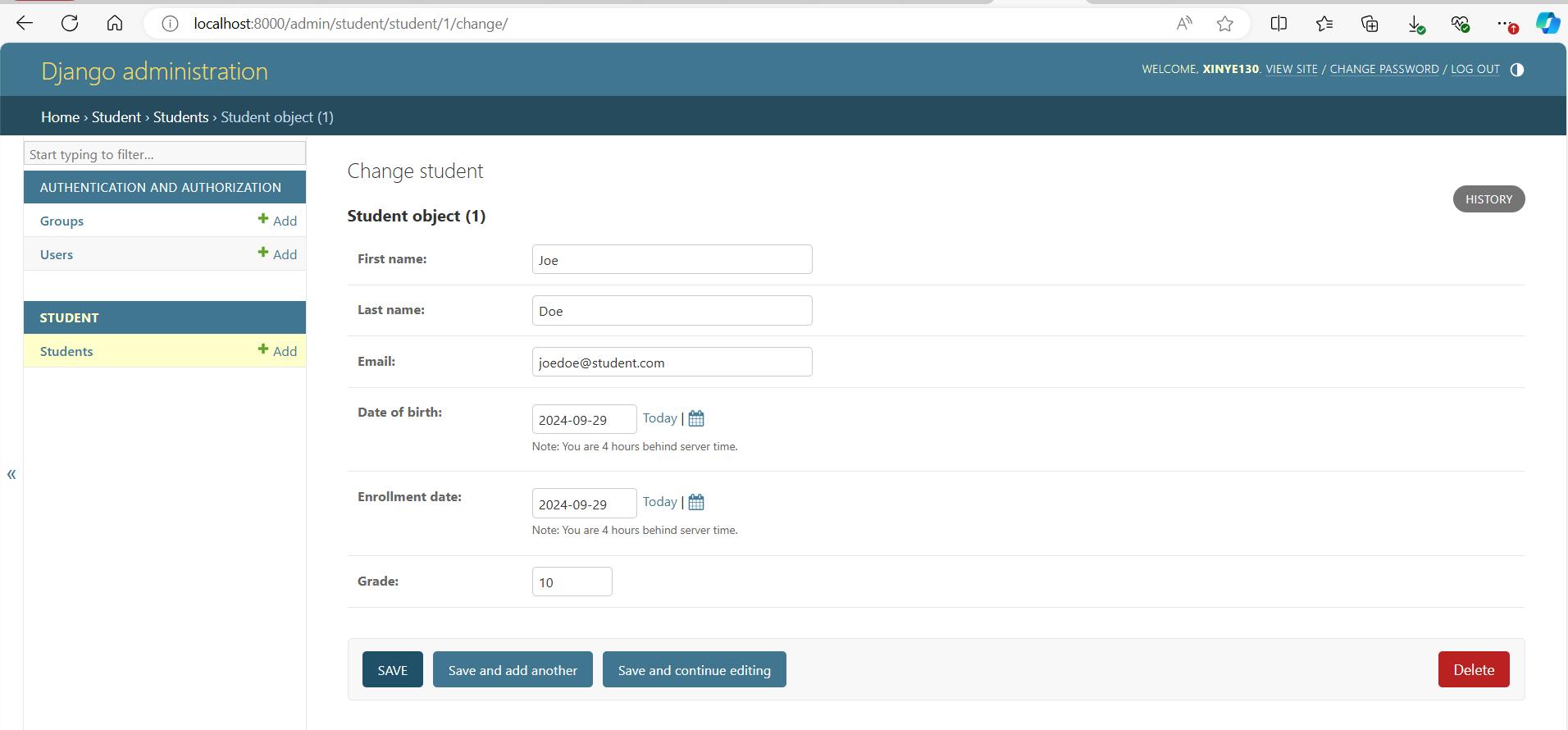
In student/admin.py file, I added code to import and register my Student model. Registered model is shown in the second image below.

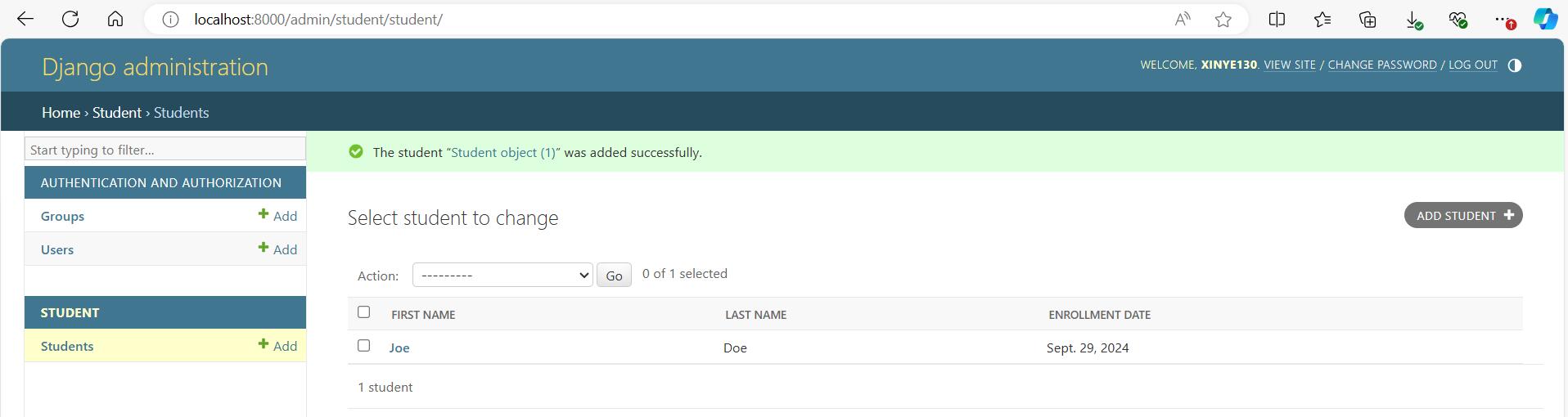


A screenshot of a computer

Description automatically generated

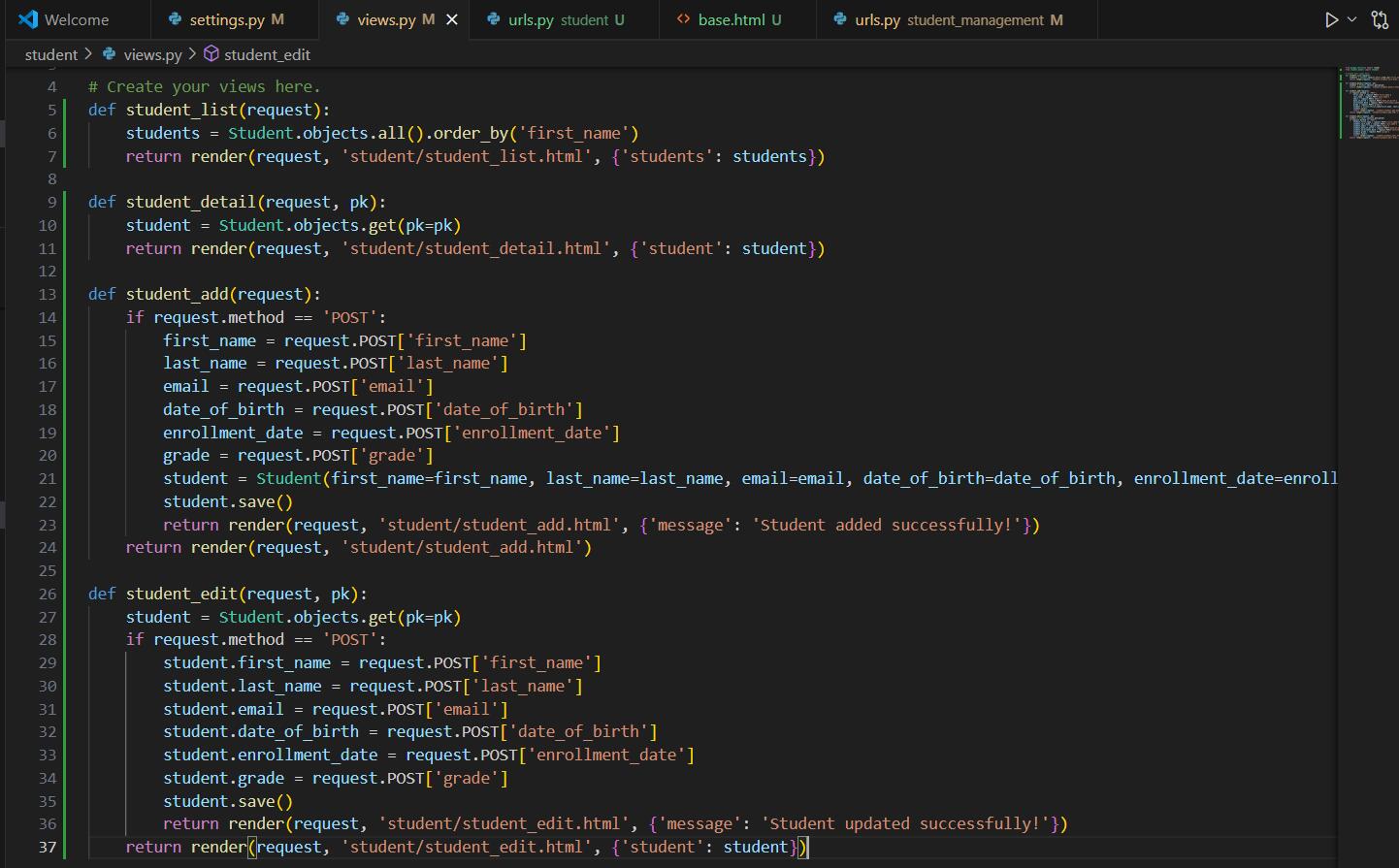
In previous code, I also implemented customized student list display to show the first name, last name, and enrollment date. Then I tested this feature by adding a student in the admin.



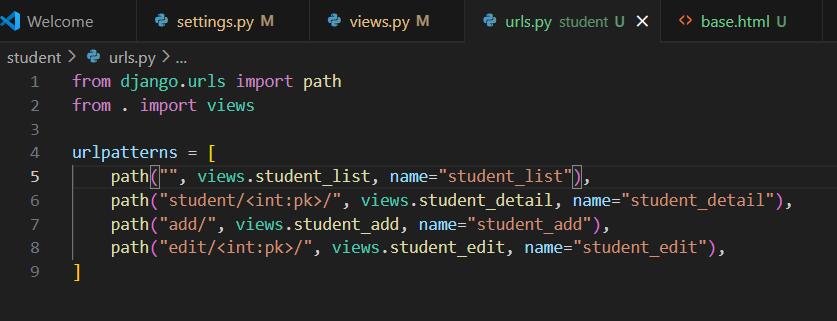


**4. Views and Templates**

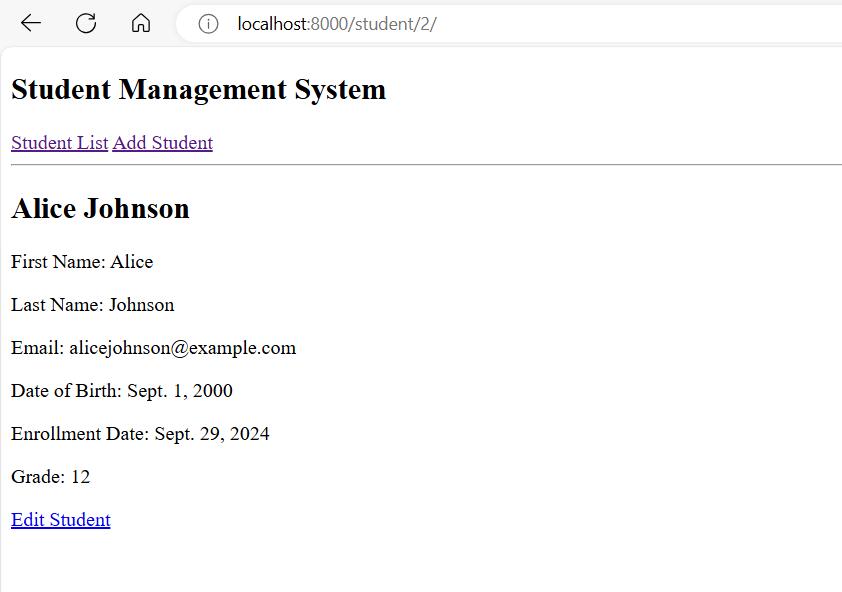
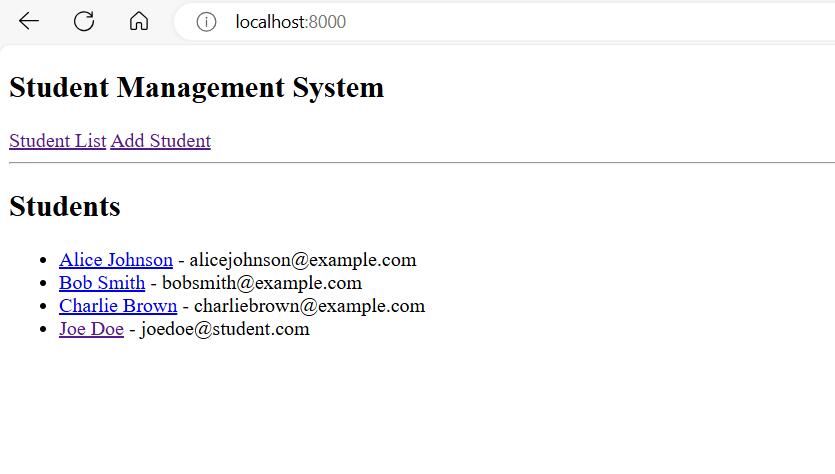
In the view.py file in the student/ directory, I created the functions for display the whole list of students, display details of a single student, add student and edit student. Then I built templates for these functionalities in student/templates/student/ directory. Currently, student\_add and student\_edit views and templates just hold placeholders, and they will be implemented in step 5.

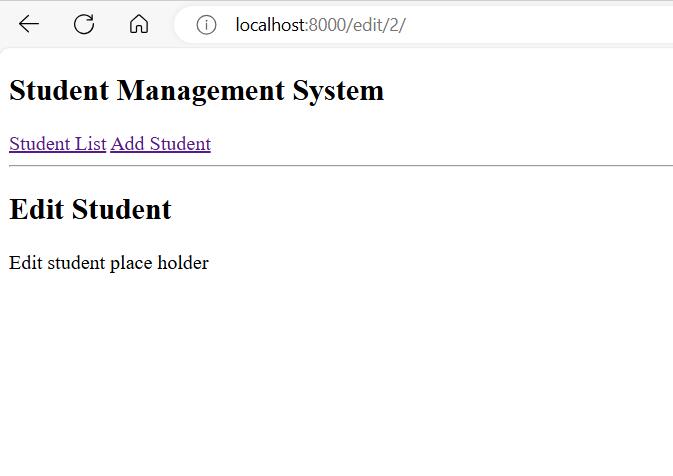
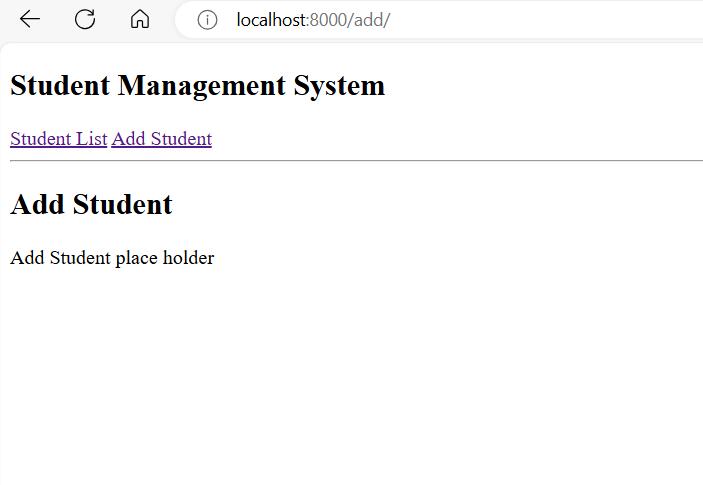


I created an urls.py file inside blog/ and add the URLs for the four views. I also added them to the project’s URL configuration in student\_management/urls.py using **path("", include("student.urls"))**.



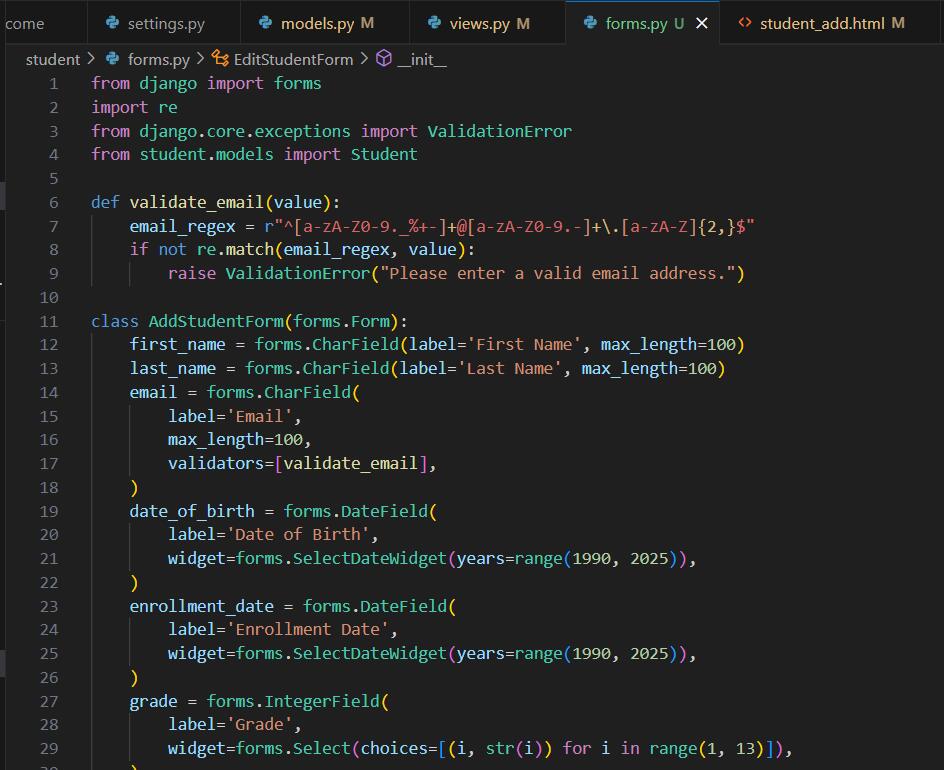
I also created a base template in templates/base.html, which will show a “Student Management System” headline and a navigation bar containing links to student list and add student functionality. I then added some more students in the admin for demonstration. Following images show current pages for the four functionalities. Students (later renamed to be “Student List”) page displays all students alphabetically by first name. For edit student functionality, user should access it through the “Edit Student” link in corresponding student detail page.



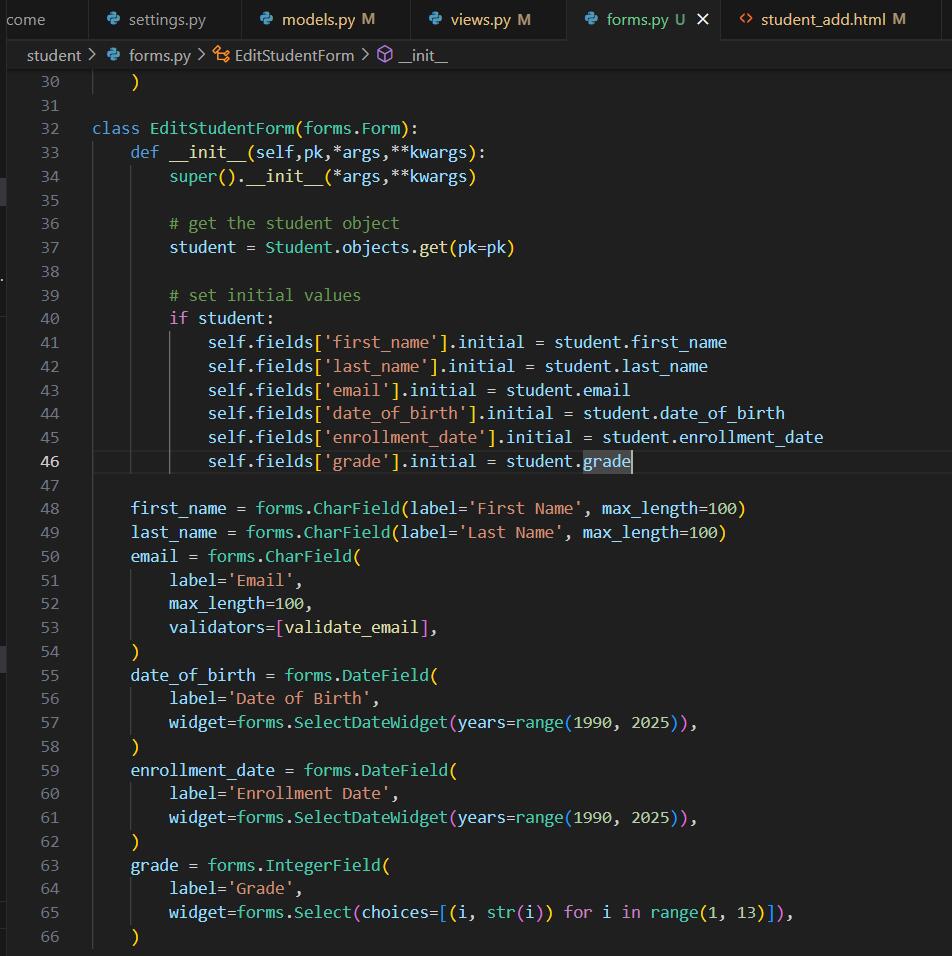


**5. Forms**

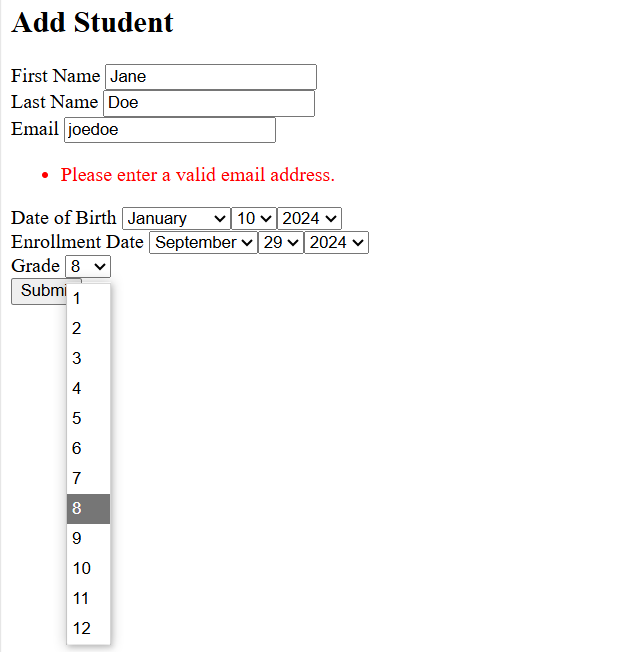
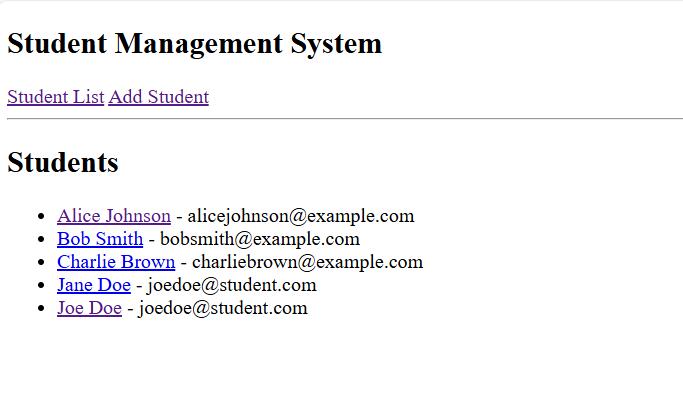
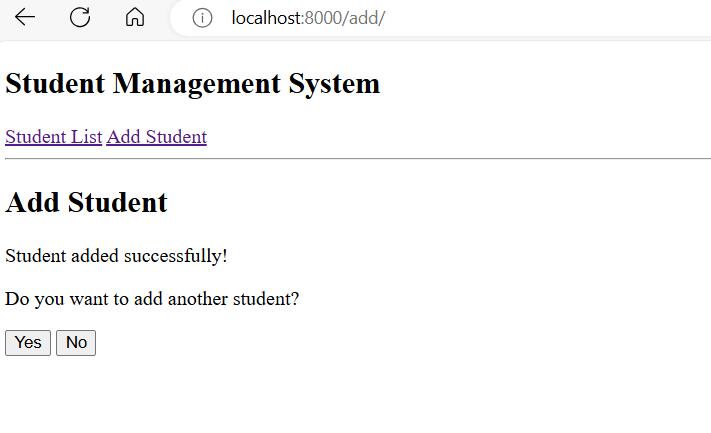
In directory student/, I created form.py to implement the forms for adding and editing students. For AddStudentForm, CharField is used for first name and last name. Django performs automatic check for empty fields, if nothing is entered a message would appear that prompts for an input, if only space is entered the form won’t be submitted. I encountered a challenge when implementing email validation. Although Django EmailField checks for email format, it only outputs instructions for emails without @ sign or emails that don’t have text after the @ sign. For example, for “test@example”, although it won’t allow the form to be submitted, it doesn’t display proper error message which I wish to be there. Therefore, I used CharField for the email and implemented a validator validate\_email that uses regex to validate an email address and define the error message to be displayed in the webpage. For date of birth and enrollment date, I used DateField so the users can only select from a range of valid dates. To ensure grade to be between 1 to 12, I used IntegerField and set the expected range.



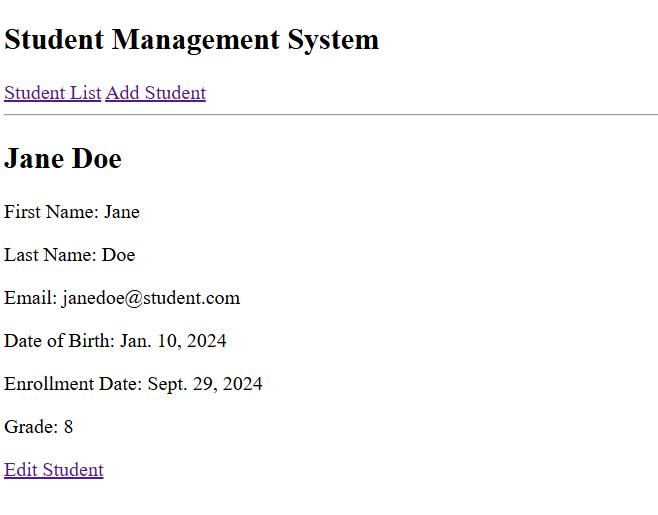
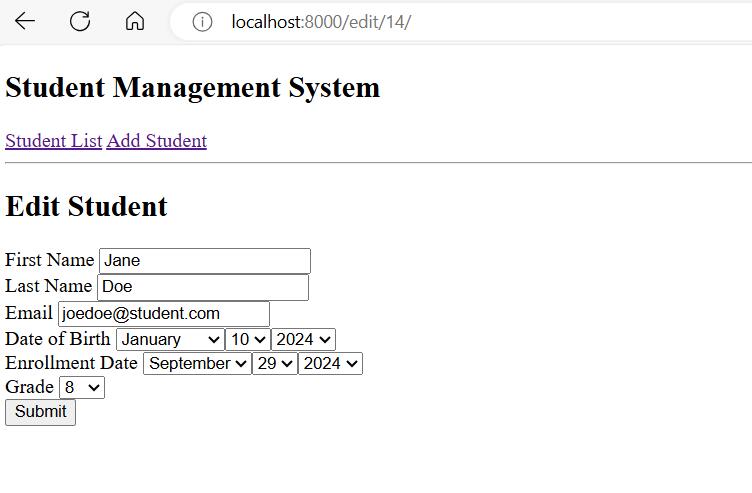
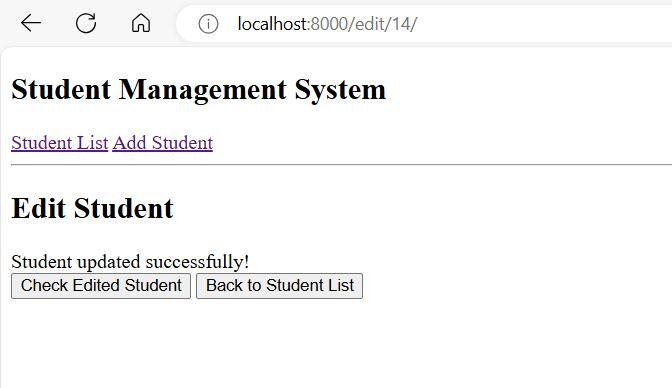
For EditStudentForm, the configuration for each field is the same as above. However, I also wish when users come to edit student page through student detail, each field should be filled with that student’s origin information, so the users only need to make slight changes. I conducted searches online and solved the problem by implementing a \_\_init\_\_ function for the form. The function will get the student to be edited using its pk, and then set the initial values for each form field.



Then I updated view functions and templates for adding and editing students. Below is a demonstration of adding a student “Jane Joe”:



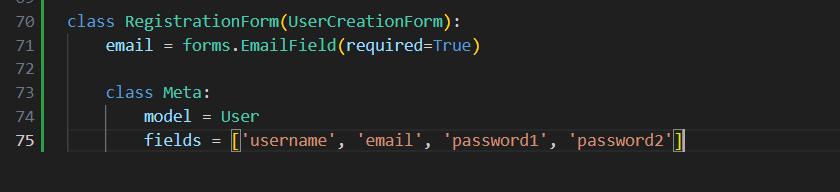
Below is a demonstration of editing a student:



**6. User Authentication**

I chose Django build-in authentication to implement user authentication functionalities. Firstly, I set up auth views and URLs on the main project’s urls.py file using this code: **path('accounts/', include('django.contrib.auth.urls'))**. This would add a group of user account authentication URLs, for example **accounts/login/** and **accounts/logout**, and they would require templates within root’s **templates/registration/** directory. I implemented login.html and logout.html following directions in [Django Tutorial Part 8: User authentication and permissions - Learn web development | MDN (mozilla.org)](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Authentication).

Then I implemented the form, view function, and template for user registration. I also added the URL for it into the app’s URLs.



A screenshot of a computer screen

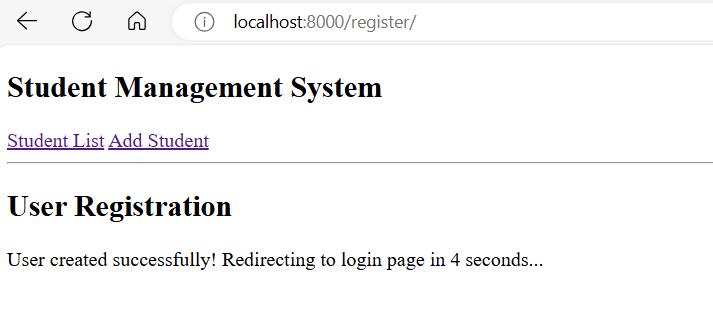
Description automatically generated

Following images demonstrate the registration page. With Django built-in functionalities, it will validate every input field.

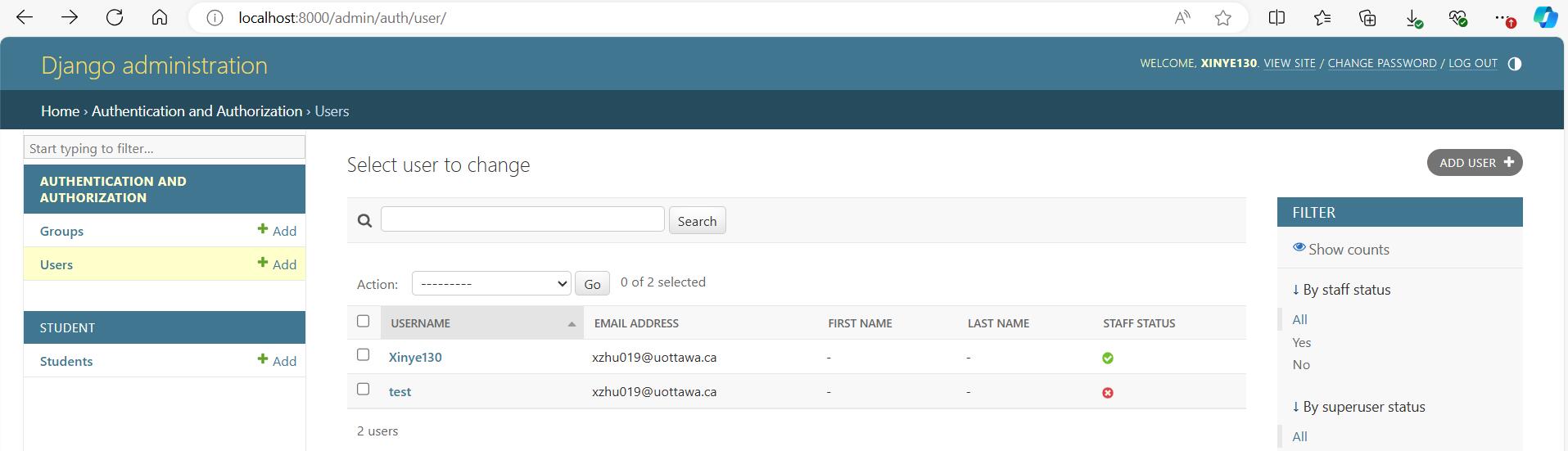
A screenshot of a computer screen

Description automatically generated

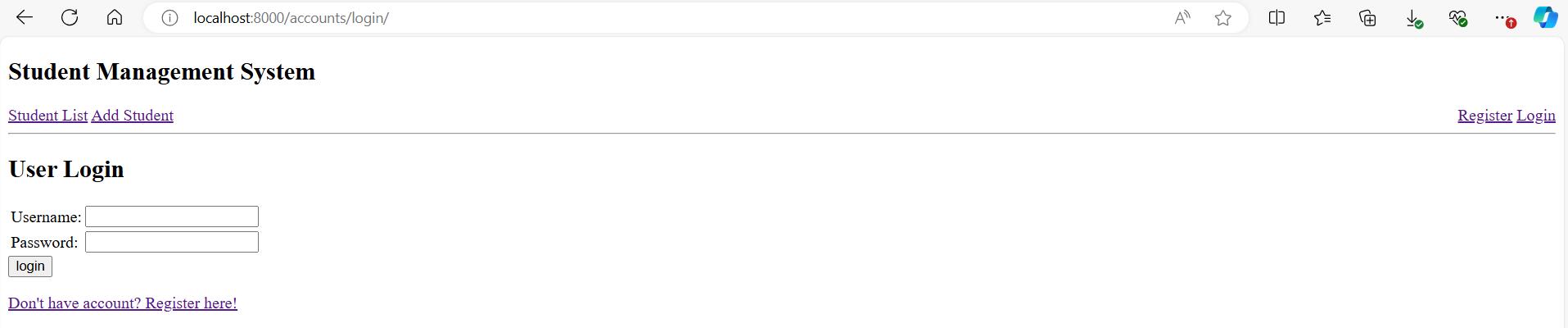
After registration completes, the page will display a successful message and a countdown. After the 5 second countdown, the user will be automatically redirected to the login page.



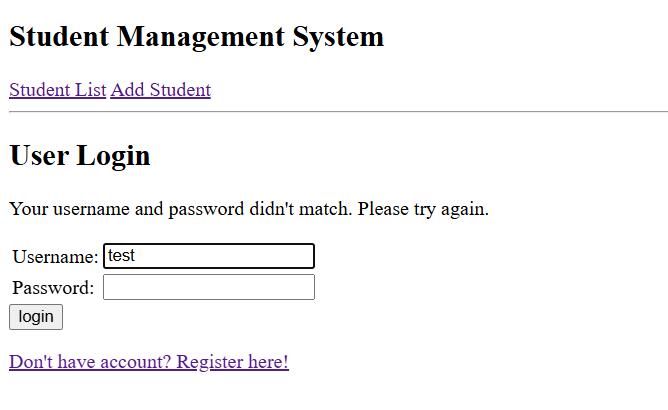
We can check the user “test” just added through the admin.



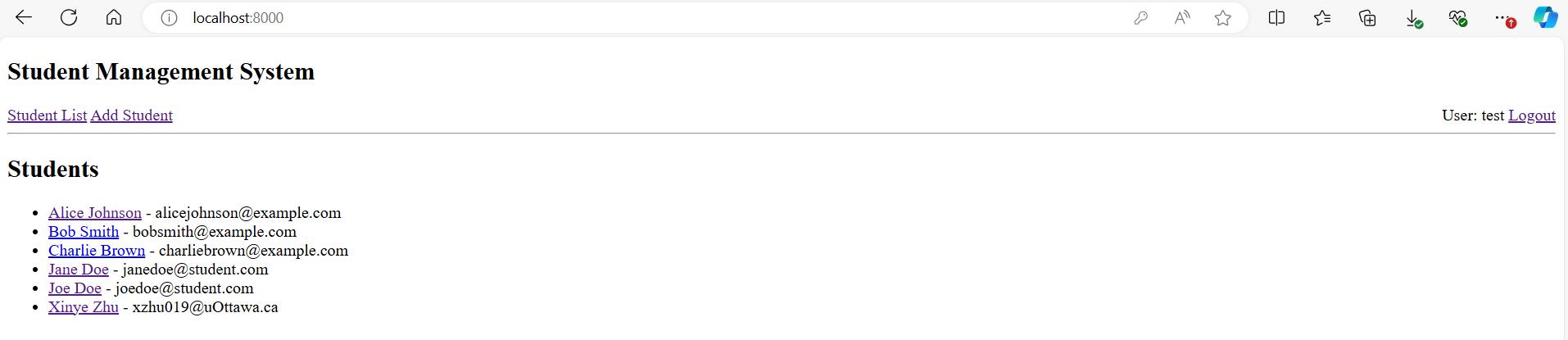
Below are demonstrations of login and logout pages. I modified the navigation bar so that when the user is not logged in, it will show links to register and login page; after the user logging in, it will display the username and logout button.



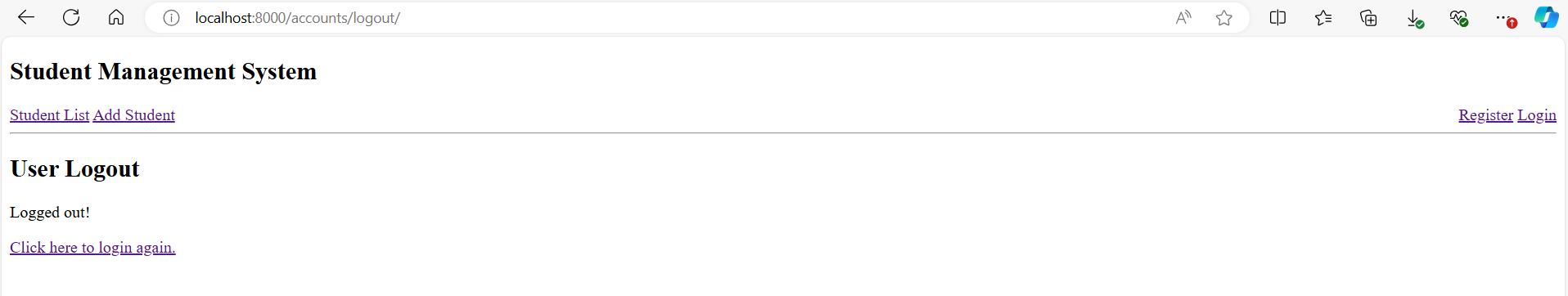
Login page automatically validates user information:



After logging in, the user will be redirected to the student list page. Logged in page:

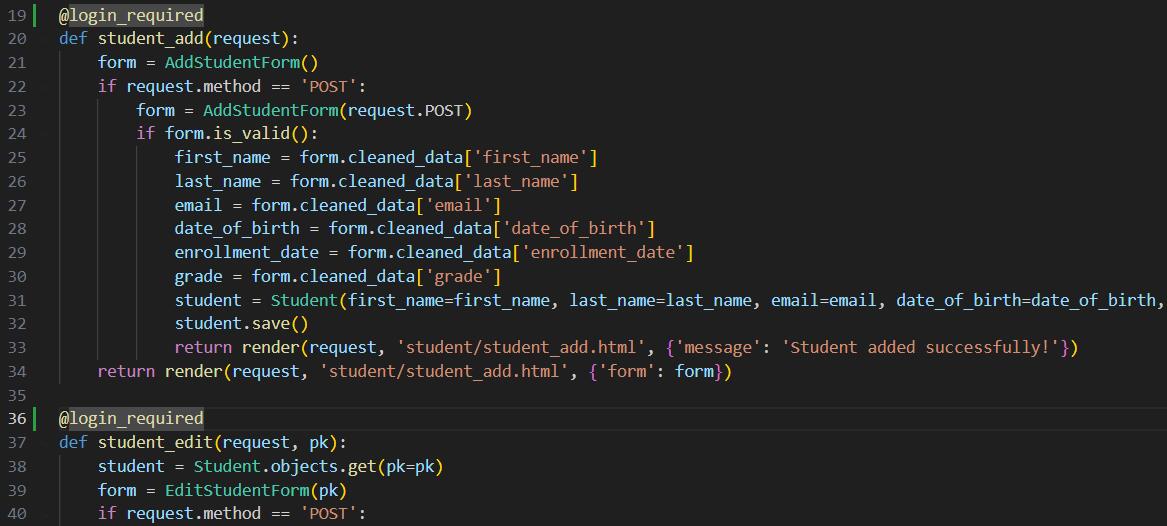


Logout page:



Lastly, I implemented restrict access to add and edit student functionalities by applying the login\_required decorator to their view functions, as shown below. If the user is logged in then the view code will execute as normal. If the user is not logged in, this will redirect to the login URL, passing the current absolute path as the next URL parameter. If the user succeeds in logging in then they will be returned back to this page.

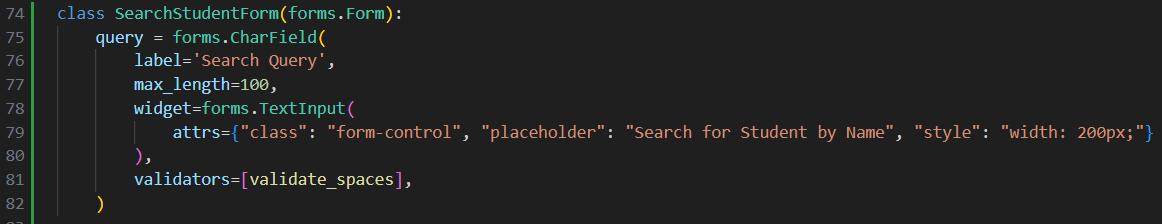
A computer screen with text

Description automatically generated

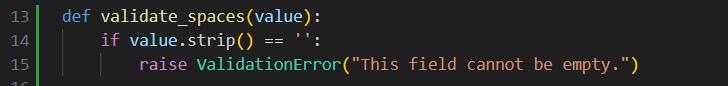
Moreover, only my superuser account can access the admin page to modify (add, delete, edit) users and students in the system. Accounts created by the users won’t have these permissions.

**7. Search Functionality**

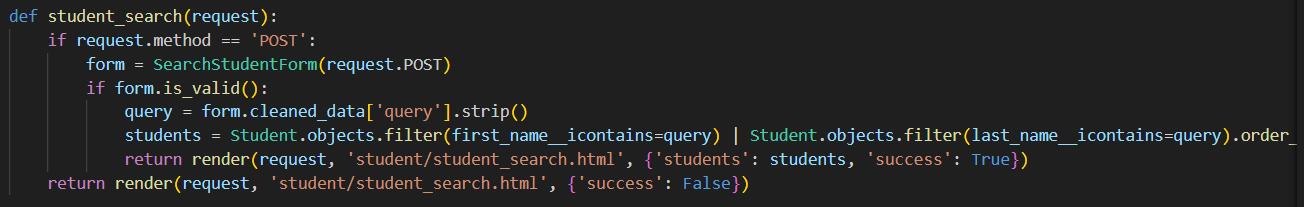
I first created a form for the search bar. I used forms.TextInput widget for the query field, so Django will load it as an HTML text input element in the templates.



When testing the system, I found input spaces (“ “) into the search bar will cause errors, so I added a space validator validate\_space to the query field.



I then implemented the view for the search functionality. If the user enters a valid query into the search field (any string other than only spaces), the system will get a list of students whose names match the query, and then the list and a bool indicates success indicator will be added into the context. Partial match is supported in this system. Otherwise, only a bool indicates failure will be added into the context. I also added the URL for the search functionality into student app’s URLs.

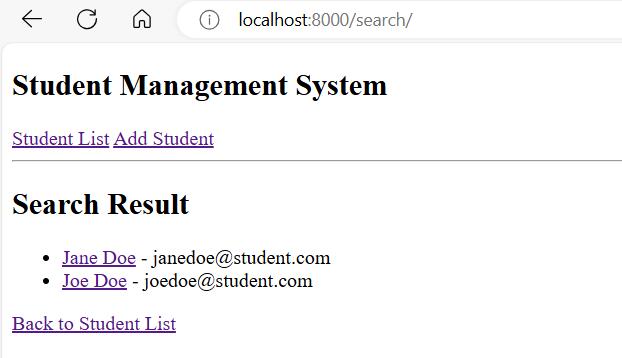


In the student\_list template, I add the search bar form to be displayed under the page title. After entering the query, the user can hit “enter” or click the Search button to submit the query. Below is the updated student list page:

A screenshot of a web page

Description automatically generated

In the student\_search template for the search result page, if a searching is performed successfully with a valid query input, it will check for the list of search results. If the list is not empty, it will display the students. Lower left image is a demonstration of searching with query “doe”. Otherwise, it will display a message indicating no student matches the query, which is demonstrated in the lower left image.

A screenshot of a search engine

Description automatically generated

If the user enters nothing in the search bar, a Django default warning message will appear that prompts the user to fill out the field. If the user inputs only spaces, they will only be redirected to the student list page.

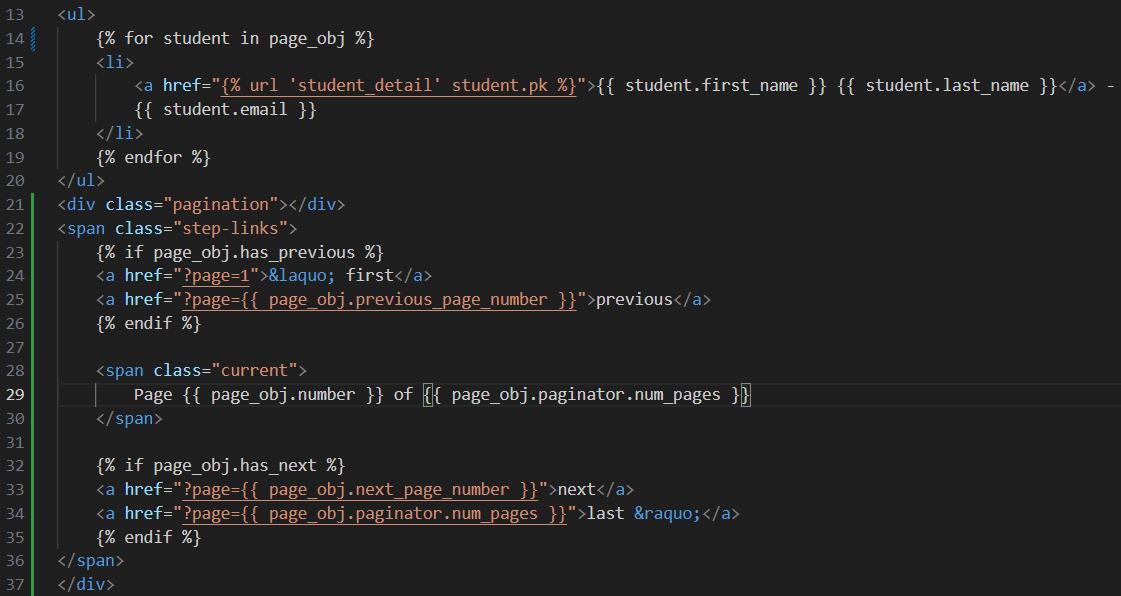
**8. Pagination**

I used Django Paginator in the student\_list view function to paginate the display of student list. I set each page to display 5 students using the code below.

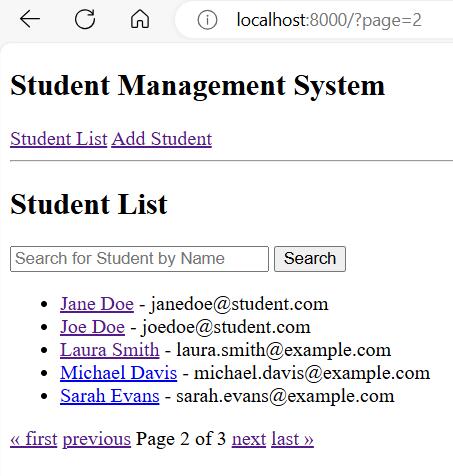
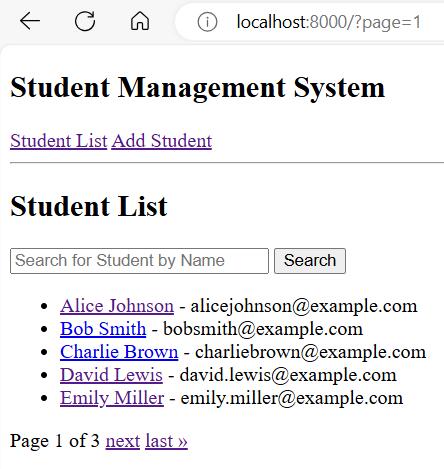
A computer code on a black background

Description automatically generated

This code adds a paginator and page\_obj to the context. I then added into student\_list template the code below, which add links to the next, previous, first, and last page.



The users can now navigate between pages on the student list page. The outcome is displayed below for 11 students in 3 pages:

A screenshot of a computer

Description automatically generated

**9. Error Handling and Validation**

**9.1 Input Field Validation**

As mentioned in previous parts, for add and edit student forms, Django CharField checks for empty field, DateField restricts date input, and IntegerField restricts grade input. I’ve implemented validator for email field. Django also performs validations for register and login forms with its built-in functionalities. Finally, I’ve implemented input validation for the search bar.

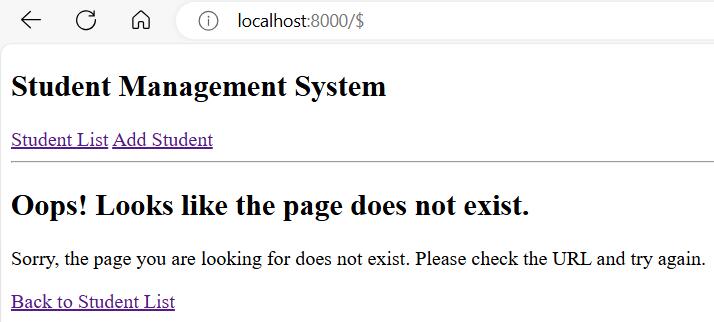
**9.2 Non-existent Student Record and Invalid URLs Handling**

For non-existent student record, I wish to handle invalid URLs such as **http://localhost:8000/student/[index\_does\_not\_exist]/** for student detail page and **http://localhost:8000/edit/[index\_does\_not\_exist]/** for edit student page. I also wish to handle any other invalid URLs such as **http://localhost:8000/[random\_text]** by displaying a customized 404 page.

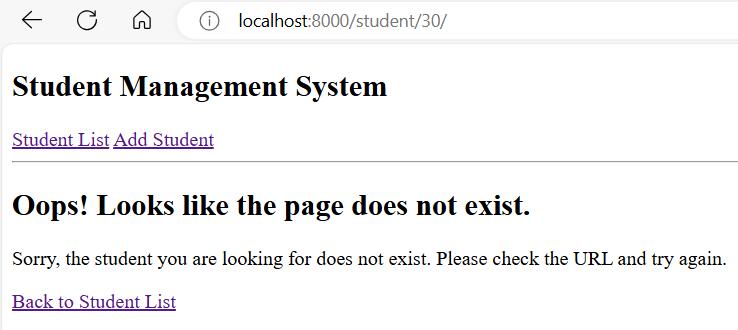
I created a customized 404.html template in templates/ directory that shows my website’s navigation bar and customized page does not exist message. To display this page instead of the Django default one, I need to set DEBUG = False and add ALLOWED\_HOSTS in student\_management/settings.py. Noticeable, with DEBUG set to False, Django admin page’s default CSS won’t load. Below is a demonstration of invalid URLs handling.

A black background with orange text

Description automatically generated



Then, in view functions for adding and editing student, I added a check that if the student record of user required pk id doesn’t exist, the customized 404 page will be rendered with a message indicating student does not exist. Below is a demonstration of non-existent student record handling.



**9.3 Duplicated Student Validation**

Students with the same first name, last name, and email combination are considered to be the same student. In the view function student\_add, I added a check after getting input data from the form. If a student with the same first name, last name, and email already exists, they won’t be added and a 'Student already exists!' message will be added to the context to be rendered onto the Add Student page.

In the view function student\_edit, after getting input data from the form, if nor first name, last name, or email is modified, we won’t perform the duplication check. Elsewise, if a student with edited name and email already exists, the edit won’t be processed and a 'Student already exists!' message will be added to the context to be rendered onto the Edit Student page.