MINGZHAO YU_300435050_Student Management Report

Repo Link

https://github.com/ivanivan999/student_management

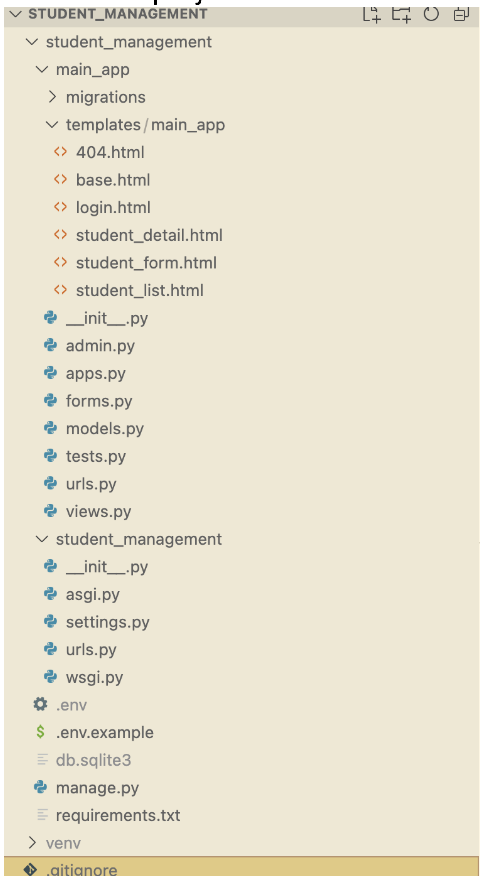
Overview

The Student Management System is a Django web app for efficiently managing student records, offering features for listing, searching, viewing, creating, and updating records. It includes Bootstrap styling and a custom 404 error page for handling missing records.

Project Setup

First, set up a Django project with a virtual environment to ensure consistency. Then, create a student management app called "main." After activating the environment, install the necessary requirements, set your secret key, apply migrations, and run the server. More details on README documentation.

Here is the project structure for the student management app:



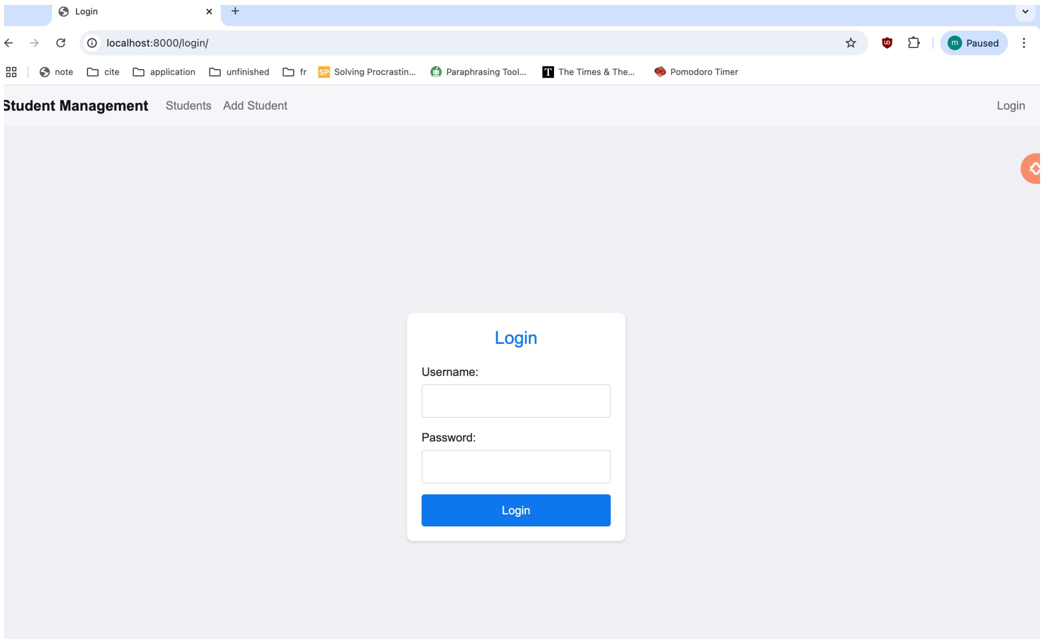
```
STUDENT_MANAGEMENT
  student_management
    main_app
      migrations
      templates / main_app
        404.html
        base.html
        login.html
        student_detail.html
        student_form.html
        student_list.html
      __init__.py
      admin.py
      apps.py
      forms.py
      models.py
      tests.py
      urls.py
      views.py
    student_management
      __init__.py
      asgi.py
      settings.py
      urls.py
      wsgi.py
    .env
    .env.example
    db.sqlite3
    manage.py
    requirements.txt
  venv
  .gitignore
```

Challenges Encountered

1. Unfamiliar with Django's project structure: I spent time understanding the relationship between projects and apps.
2. Integrating Bootstrap for styling: find difficulties on pagination to display record counts and allow users to choose the number of records per page.
3. Environment Variable Configuration: Faced issues with reading the SECRET_KEY from the .env file, which I resolved by correcting the formatting—removing spaces around the = sign.
4. Custom 404 Page: To properly display the custom 404 page, I set DEBUG=False and configured handler404 in urls.py.

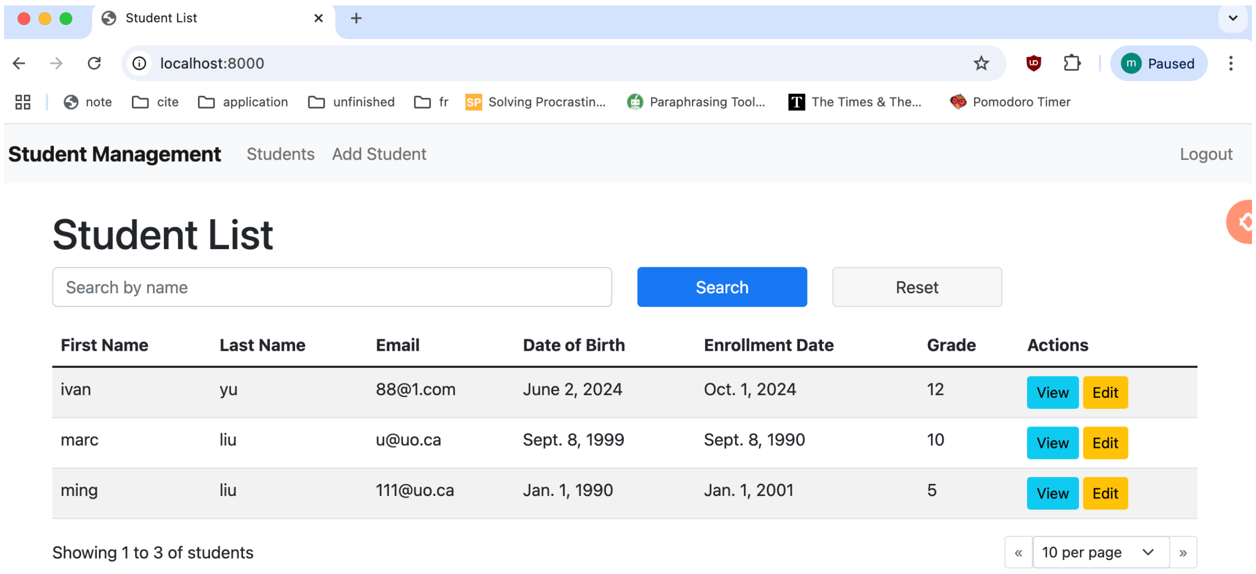
Features

Login Page



Student List

This page lists student records with search functionality. Users can view, edit, and search students by name. It displays details like first name, last name, email, birth date, enrollment date, and grade. Pagination allows choosing the number of records per page, and buttons provide easy access to view or edit student details.



Student List

localhost:8000

Student Management Students Add Student Logout

Student List

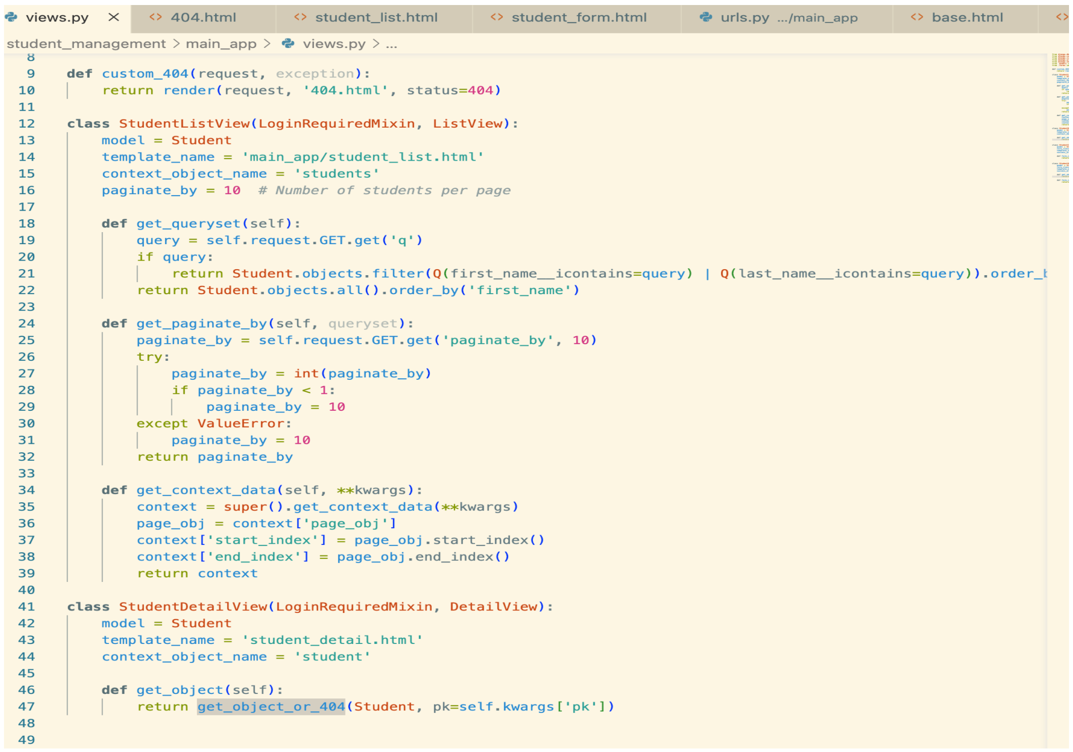
Search by name | Search | Reset

First Name	Last Name	Email	Date of Birth	Enrollment Date	Grade	Actions
ivan	yu	88@1.com	June 2, 2024	Oct. 1, 2024	12	View Edit
marc	liu	u@uo.ca	Sept. 8, 1999	Sept. 8, 1990	10	View Edit
ming	liu	111@uo.ca	Jan. 1, 1990	Jan. 1, 2001	5	View Edit

Showing 1 to 3 of students

« 10 per page »

Key code

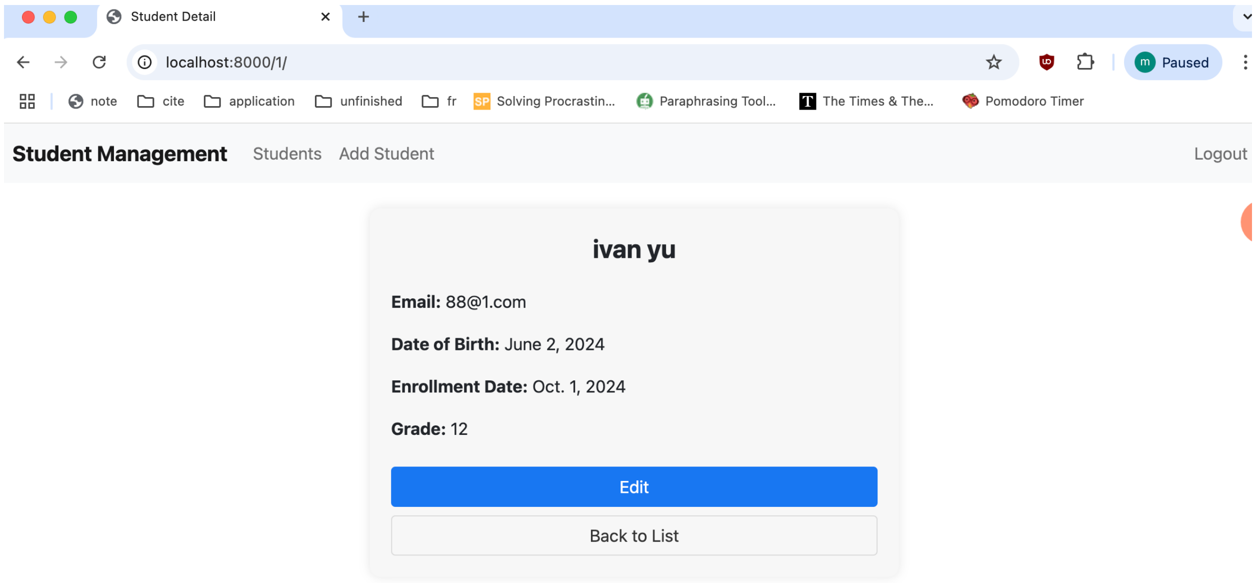


views.py | 404.html | student_list.html | student_form.html | urls.py .../main_app | base.html

student_management > main_app > views.py > ...

```python

def custom_404(request, exception):
    return render(request, '404.html', status=404)

class StudentListView(LoginRequiredMixin, ListView):
    model = Student
    template_name = 'main_app/student_list.html'
    context_object_name = 'students'
    paginate_by = 10  # Number of students per page

    def get_queryset(self):
        query = self.request.GET.get('q')
        if query:
            return Student.objects.filter(Q(first_name__icontains=query) | Q(last_name__icontains=query)).order_
        return Student.objects.all().order_by('first_name')

    def get_paginate_by(self, queryset):
        paginate_by = self.request.GET.get('paginate_by', 10)
        try:
            paginate_by = int(paginate_by)
            if paginate_by < 1:
                paginate_by = 10
        except ValueError:
            paginate_by = 10
        return paginate_by

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        page_obj = context['page_obj']
        context['start_index'] = page_obj.start_index()
        context['end_index'] = page_obj.end_index()
        return context

class StudentDetailView(LoginRequiredMixin, DetailView):
    model = Student
    template_name = 'student_detail.html'
    context_object_name = 'student'

    def get_object(self):
        return get_object_or_404(Student, pk=self.kwargs['pk'])

```

View



Student Detail

localhost:8000/1/

note cite application unfinished fr Solving Procrastin... Paraphrasing Tool... The Times & The... Pomodoro Timer

Student Management Students Add Student Logout

ivan yu

Email: 88@1.com

Date of Birth: June 2, 2024

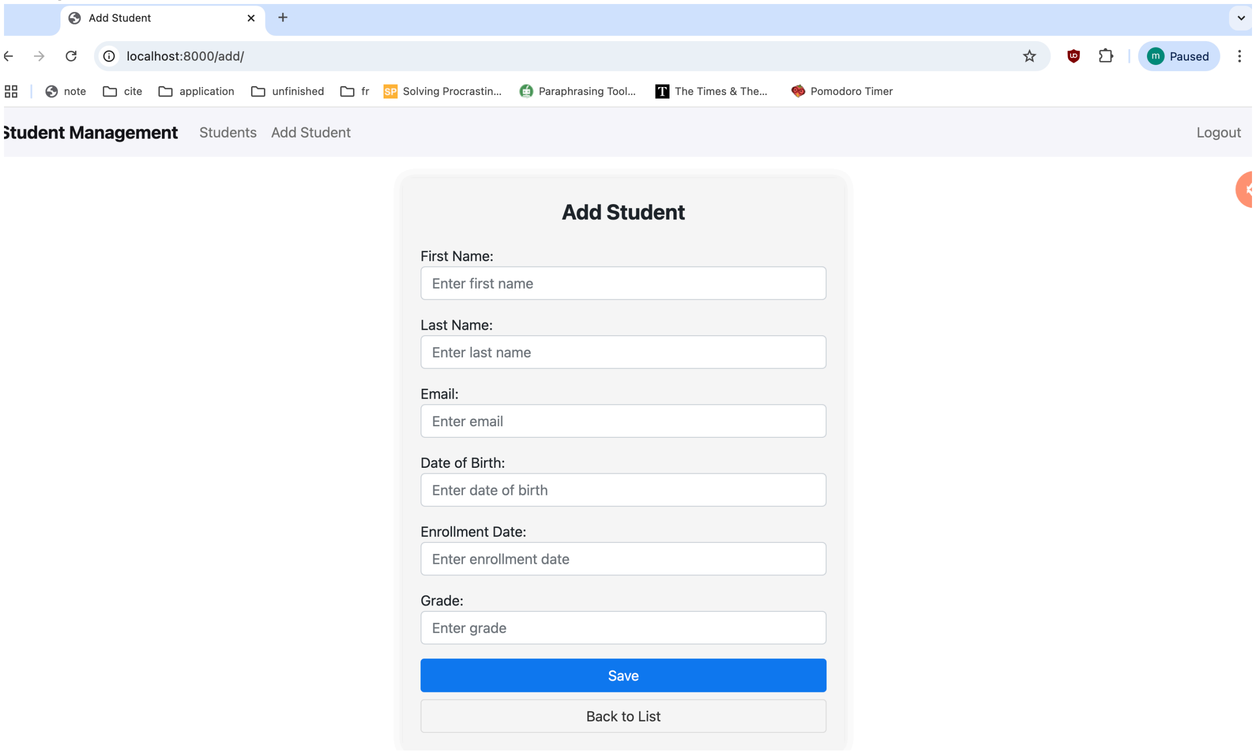
Enrollment Date: Oct. 1, 2024

Grade: 12

Edit

Back to List

Add/Edit



Add Student

localhost:8000/add/

note cite application unfinished fr Solving Procrastin... Paraphrasing Tool... The Times & The... Pomodoro Timer

Student Management Students Add Student Logout

Add Student

First Name: Enter first name

Last Name: Enter last name

Email: Enter email

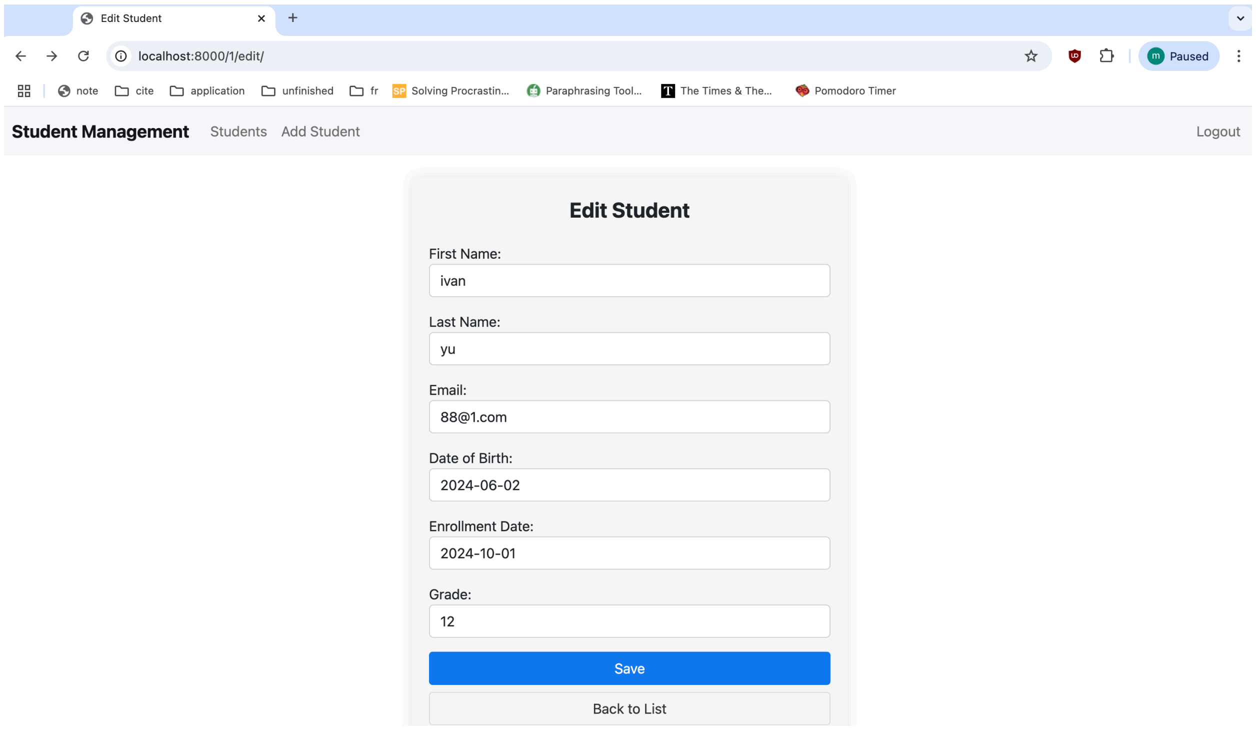
Date of Birth: Enter date of birth

Enrollment Date: Enter enrollment date

Grade: Enter grade

Save

Back to List


Edit Student

localhost:8000/1/edit/

Student Management Students Add Student Logout

Edit Student

First Name: ivan

Last Name: yu

Email: 88@1.com

Date of Birth: 2024-06-02

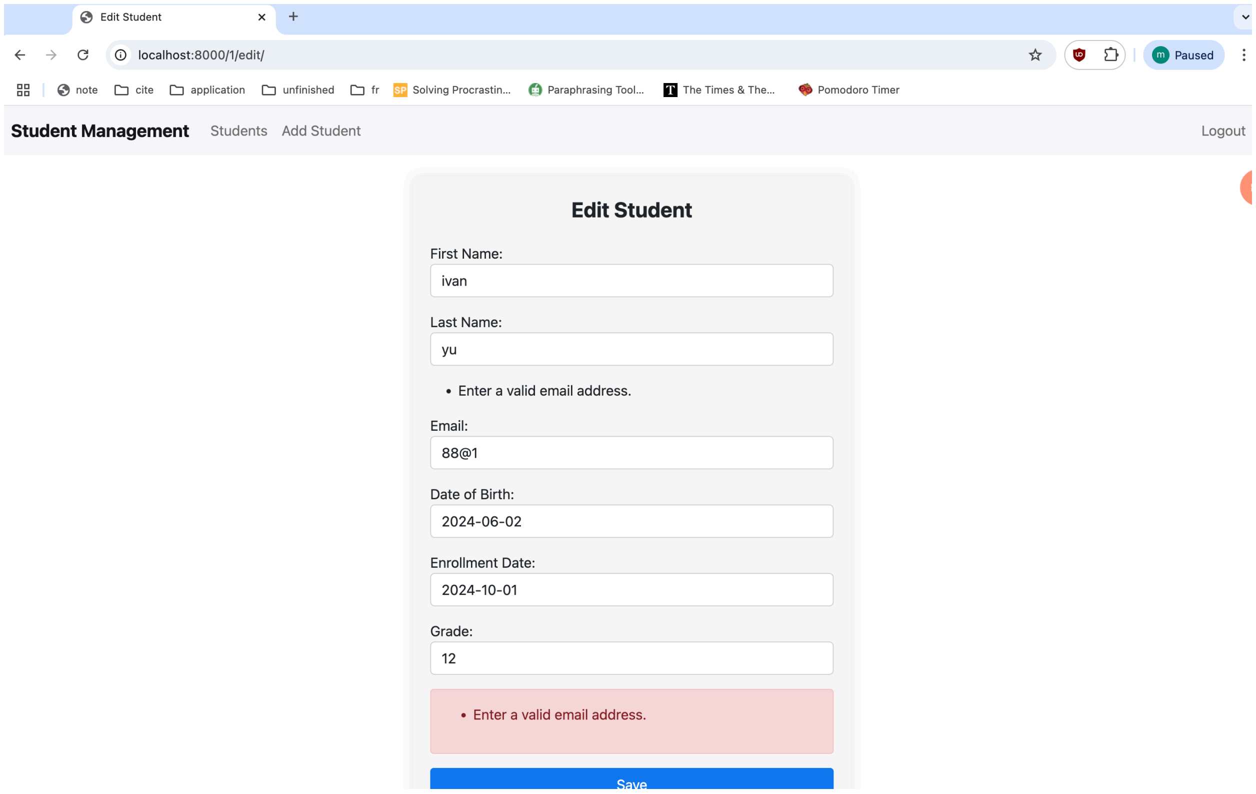
Enrollment Date: 2024-10-01

Grade: 12

Save

Back to List

Validation


Edit Student

localhost:8000/1/edit/

Student Management Students Add Student Logout

Edit Student

First Name: ivan

Last Name: yu

- Enter a valid email address.

Email: 88@1

Date of Birth: 2024-06-02

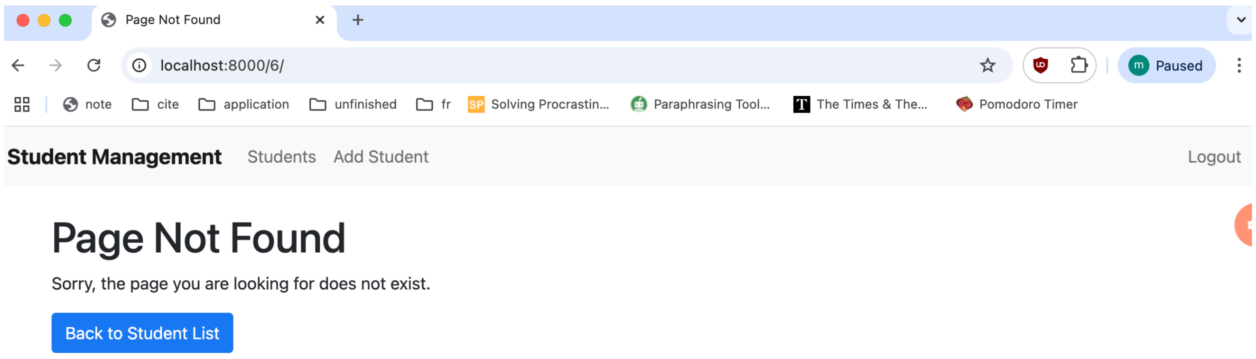
Enrollment Date: 2024-10-01

Grade: 12

- Enter a valid email address.

Save

Page Not Found



Page Not Found

Sorry, the page you are looking for does not exist.

Implementation

1. Search Bar: Filters students by name using `request.GET.get('q')`.
2. Search/Reset Buttons: Search submits the form; Reset clears filters.
3. Student Table: Displays student data with a Bootstrap table.
4. View/Edit Buttons: Links generated using `student.id` for viewing/editing.
5. Pagination: Uses Django's `Paginator` with Bootstrap navigation.
6. Record Count: Shows the current student range on the page.
7. Records Per Page: Dropdown adjusts records per page and updates the paginator.
8. Validation: Django's form validation ensures data integrity when adding or editing student records (e.g., date format, email format).
9. Page Not Found (404): A custom 404 page is triggered by setting `DEBUG=False` and defining `handler404` in `urls.py`. This page displays a user-friendly error message when a non-existent record is requested.