Building a Student Management System using Django

GITHUB LINK:

https://github.com/F-saiyed/Full_stack/tree/main/Assignment_2_final

Overview: -

A web-based program created with Django, the **Student Management System** is intended to effectively handle student records. Setting up a Django environment, creating a `Student` model with fields for name, email, date of birth, and grade, and putting functionalities like adding, editing, and displaying student information into practice are all part of the project. Only authorized users can alter records due to the system's user authentication feature. Users can filter students by name using a search feature, and pagination was included to limit the number of students shown on each page. Email address uniqueness and a grade between 1 and 12 are guaranteed by validation. The project offers a strong and intuitive interface for managing student information by integrating error handling to handle erroneous data input and non-existent records.

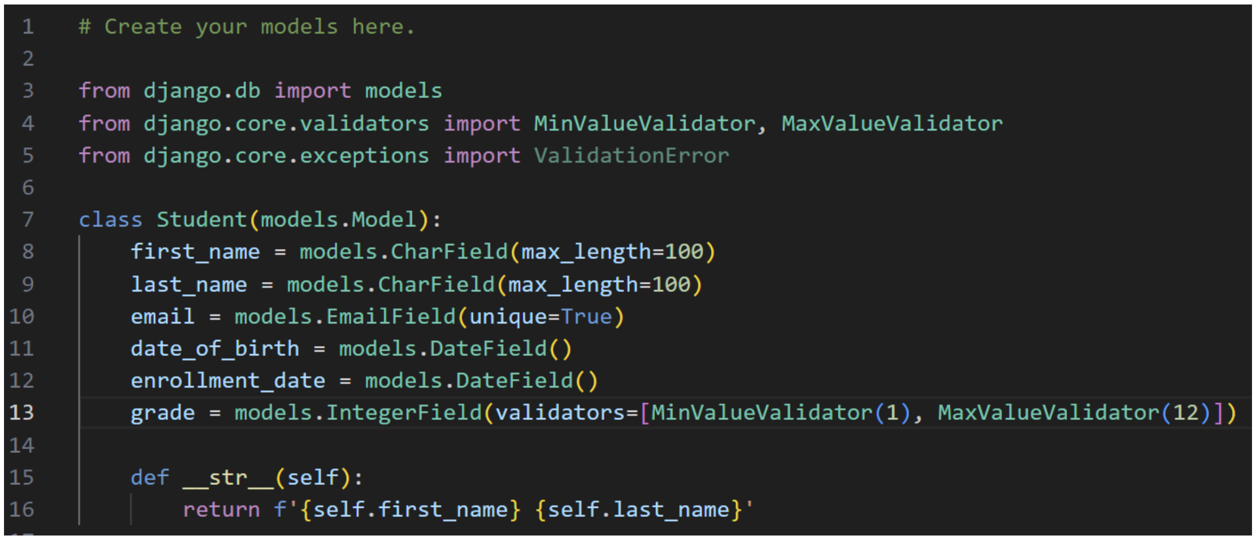
STEP 1: - Set Up

Created a new Django project called student_management and a new app called students for managing student-related functionality.

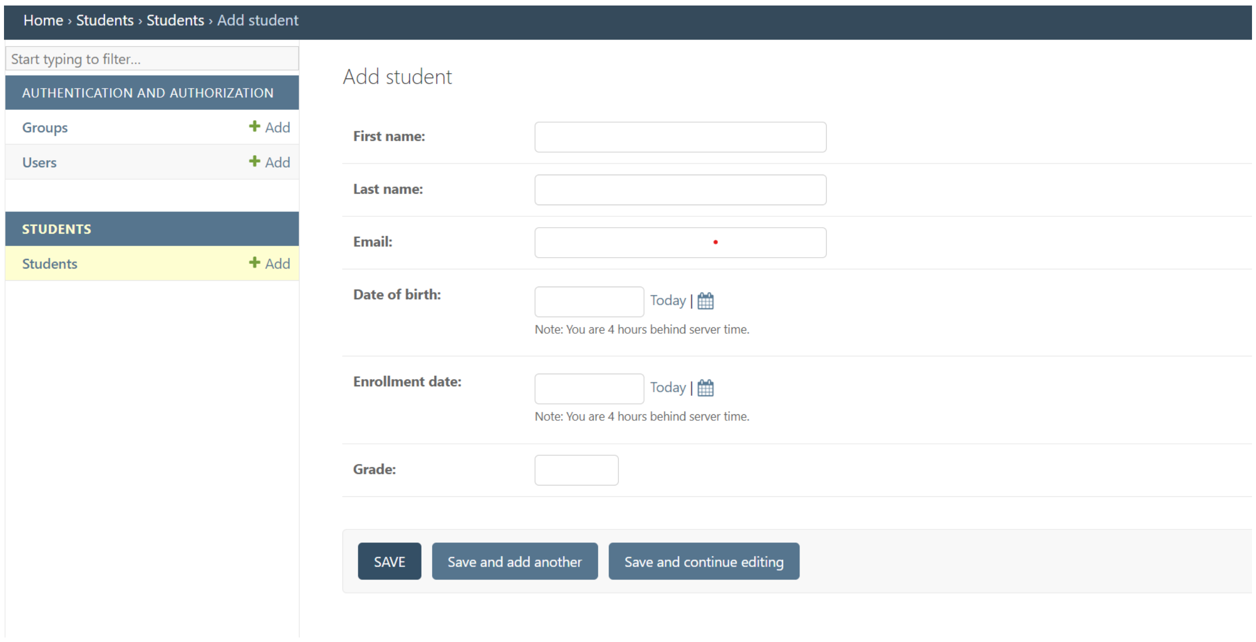
Virtual Environment: Set up a virtual environment using python -m venv env to manage dependencies, and installed Django using pip install django.

STEP 2: - Student Model

Defined a student model in students/models.py with fields like first_name, last_name, email, date_of_birth, enrollment_date, and grade.



```python
# Create your models here.

from django.db import models
from django.core.validators import MinValueValidator, MaxValueValidator
from django.core.exceptions import ValidationError

class Student(models.Model):
    first_name = models.CharField(max_length=100)
    last_name = models.CharField(max_length=100)
    email = models.EmailField(unique=True)
    date_of_birth = models.DateField()
    enrollment_date = models.DateField()
    grade = models.IntegerField(validators=[MinValueValidator(1), MaxValueValidator(12)])

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


Home › Students › Students › Add student

Start typing to filter...

AUTHENTICATION AND AUTHORIZATION

Groups + Add

Users + Add

STUDENTS

Students + Add

Add student

First name:

Last name:

Email:

Date of birth: Today |
Note: You are 4 hours behind server time.

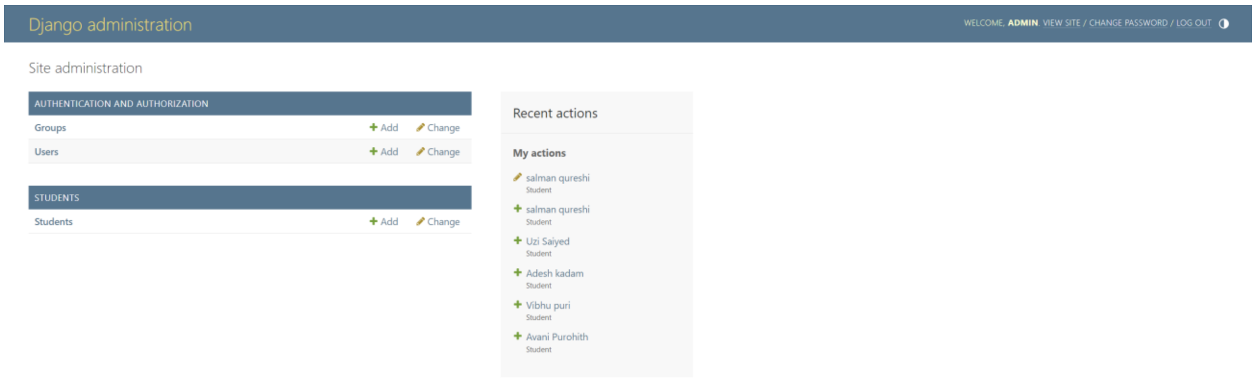
Enrollment date: Today |
Note: You are 4 hours behind server time.

Grade:

SAVE | Save and add another | Save and continue editing

STEP 3: - Admin Interface

Registered the Student model in students/admin.py to make it manageable via Django's admin interface. Customized the list display to show first_name, last_name, and enrollment_date.


Django administration — WELCOME, ADMIN. VIEW SITE / CHANGE PASSWORD / LOG OUT

Site administration

AUTHENTICATION AND AUTHORIZATION

Groups + Add ✎ Change

Users + Add ✎ Change

STUDENTS

Students + Add ✎ Change

Recent actions

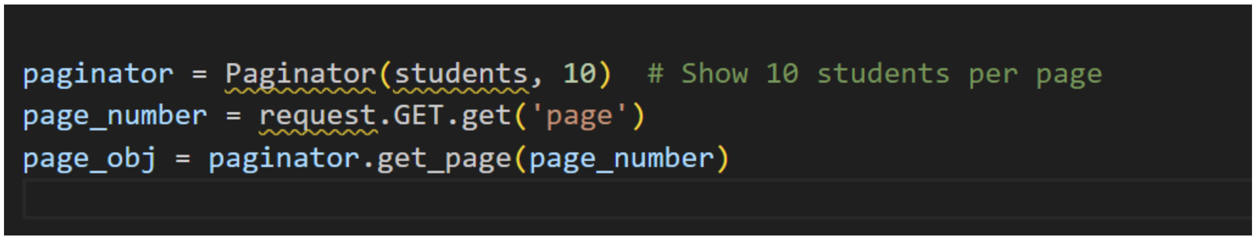
My actions

- ✎ salman qureshi
  Student
- + salman qureshi
  Student
- + Uzi Saiyed
  Student
- + Adesh kadam
  Student
- + Vibhu puri
  Student
- + Avani Purohith
  Student

STEP 4: - Views and Templates

Views and Templates were implemented for listing students, viewing student details, and editing/adding new students.

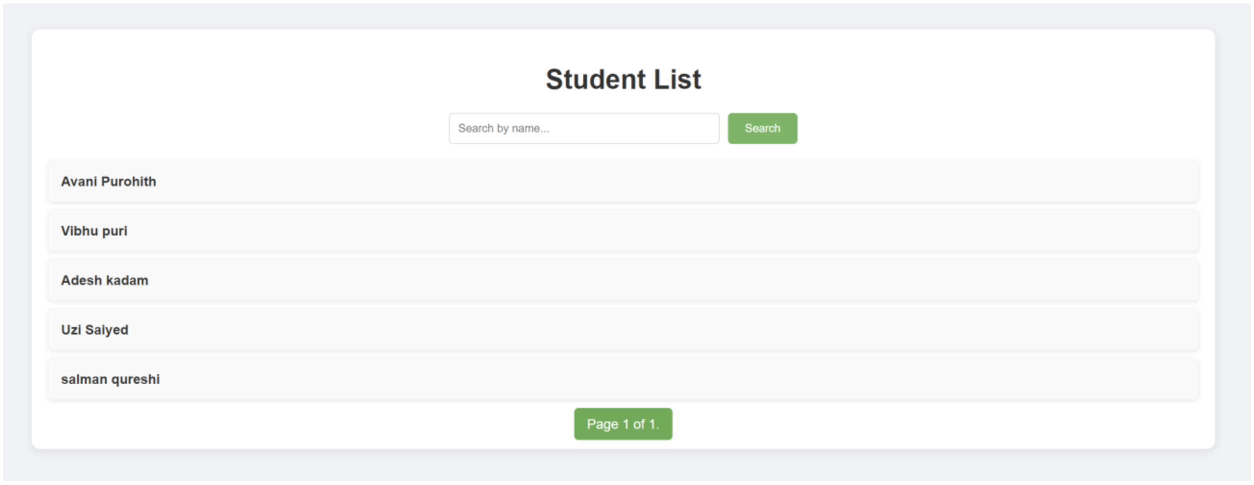
The list view utilized pagination to limit the number of students displayed per page.



```
paginator = Paginator(students, 10)  # Show 10 students per page
page_number = request.GET.get('page')
page_obj = paginator.get_page(page_number)
```

Features Implemented:

- student_list to display a list of students with pagination.



Student List

Avani Purohith

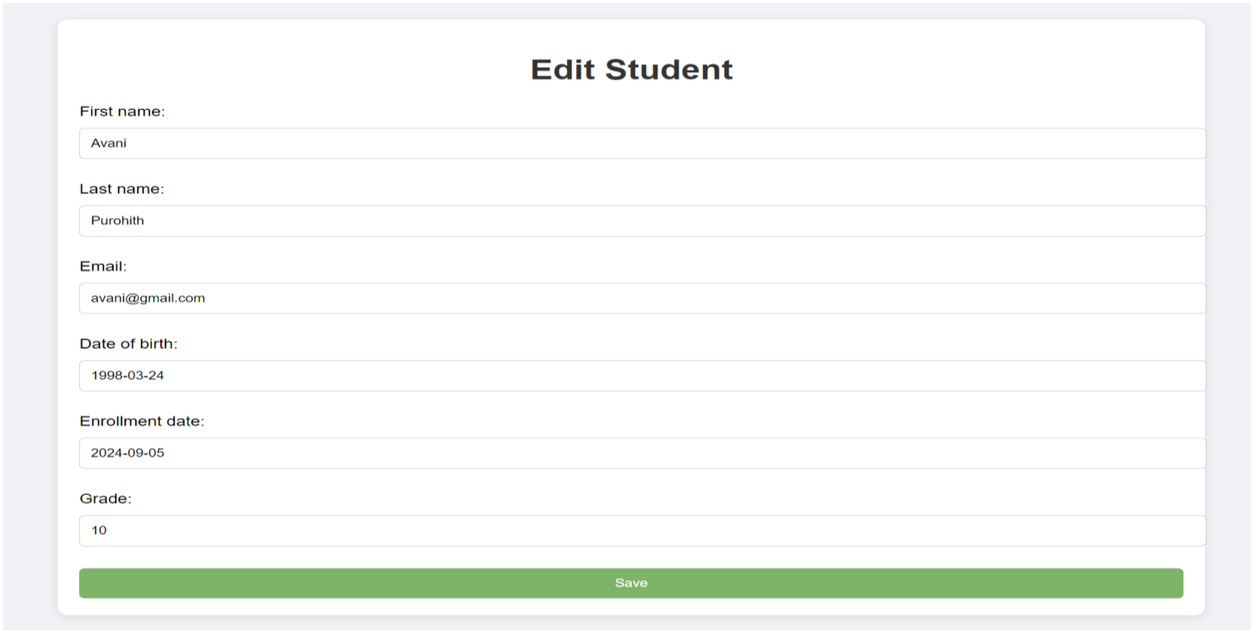
Vibhu puri

Adesh kadam

Uzi Saiyed

salman qureshi

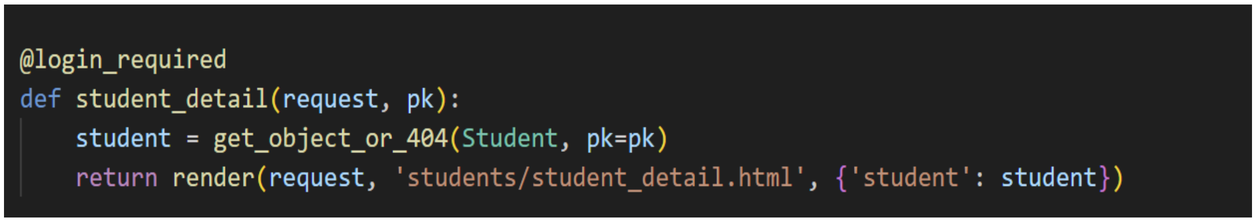
- student_detail to show individual student details.
- add_student and edit_student views for adding and updating student information.



Edit Student

First name:	Avani
Last name:	Purohith
Email:	avani@gmail.com
Date of birth:	1998-03-24
Enrollment date:	2024-09-05
Grade:	10

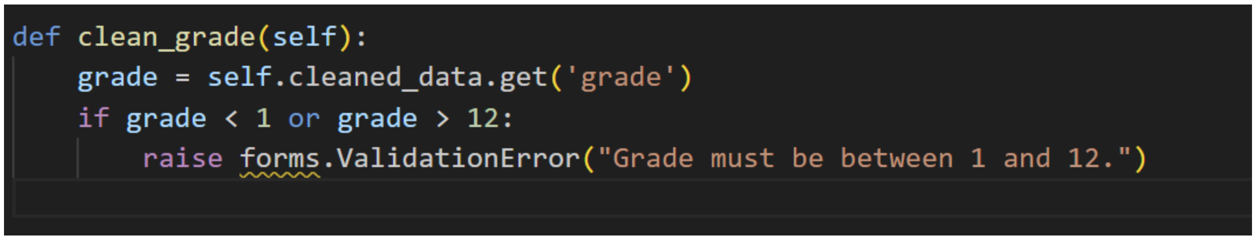
Implemented a view to display detailed information about each student. This view also includes a link to edit the student's information.



```python
@login_required
def student_detail(request, pk):
    student = get_object_or_404(Student, pk=pk)
    return render(request, 'students/student_detail.html', {'student': student})
```

STEP 5: - Forms with Validation

A Django form was created for adding and editing students. The form handled validation for fields like email (ensuring uniqueness) and grade (ensuring it's between 1 and 12).

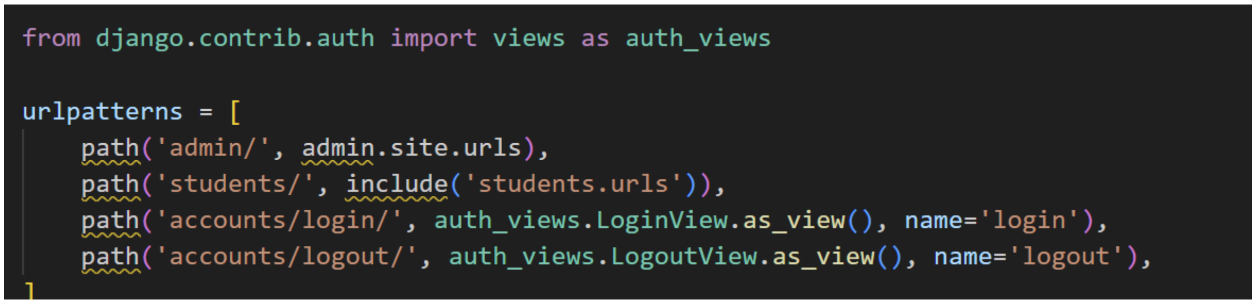


```python
def clean_grade(self):
    grade = self.cleaned_data.get('grade')
    if grade < 1 or grade > 12:
        raise forms.ValidationError("Grade must be between 1 and 12.")
```

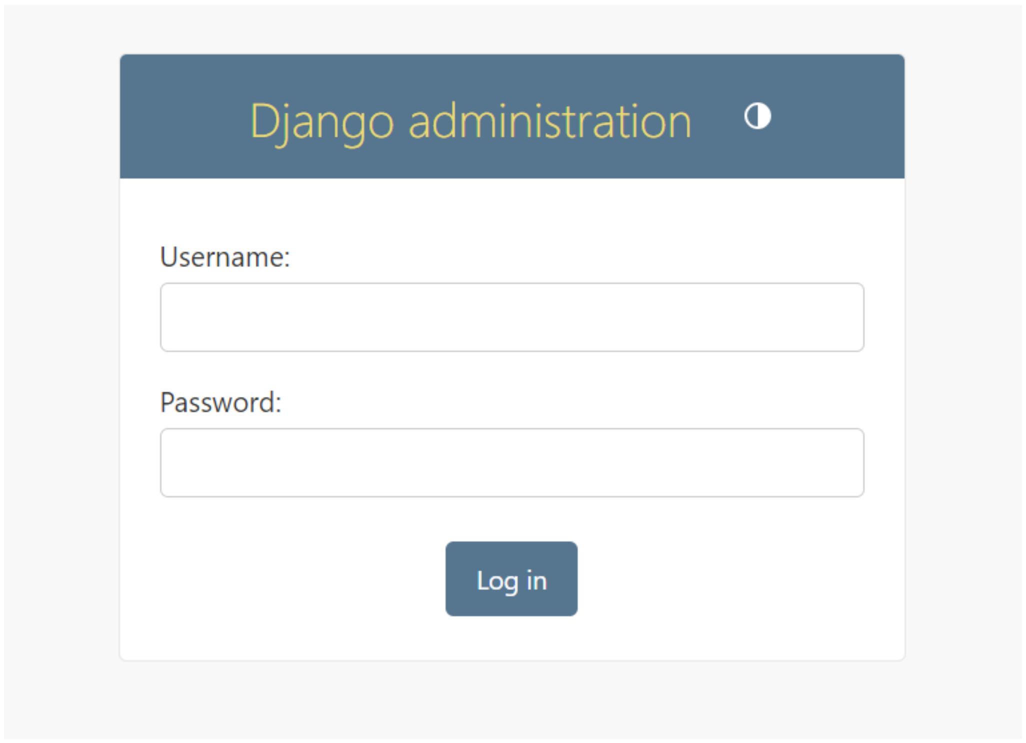
STEP 6: - Authentication

Django's built-in authentication system was used to manage user login and logout. This was chosen for its simplicity and security features.

Configured user authentication by adding @login_required decorators to views that should be restricted to logged-in users.

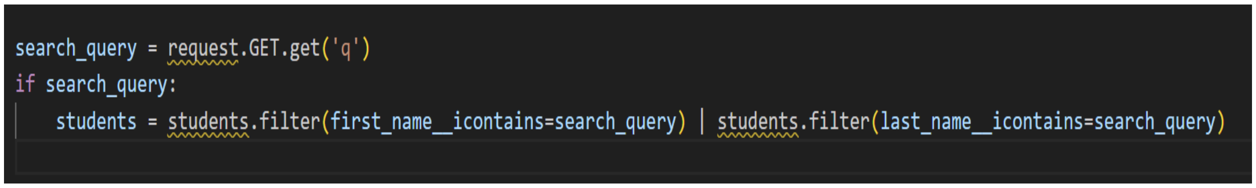


```python
from django.contrib.auth import views as auth_views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('students/', include('students.urls')),
    path('accounts/login/', auth_views.LoginView.as_view(), name='login'),
    path('accounts/logout/', auth_views.LogoutView.as_view(), name='logout'),
]
```

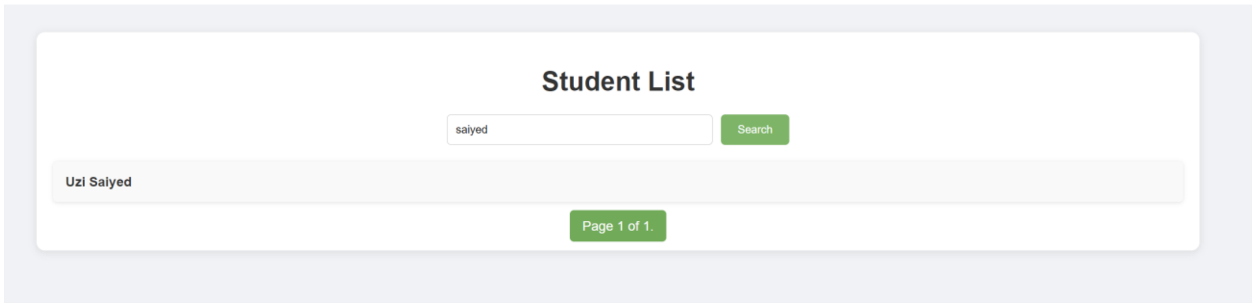


STEP 7: - Search Functionality

A search bar was implemented in the student list view to allow filtering of students by their first or last names.



```
search_query = request.GET.get('q')
if search_query:
    students = students.filter(first_name__icontains=search_query) | students.filter(last_name__icontains=search_query)
```

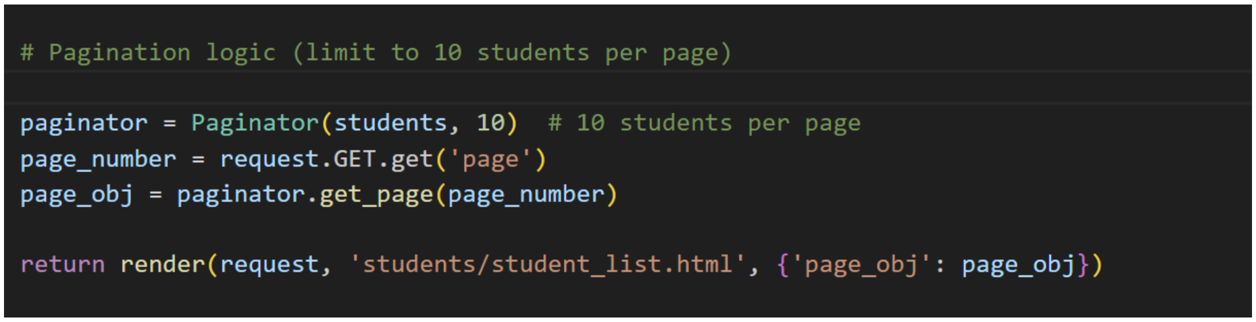


Uzi Saiyed

Page 1 of 1.

STEP 8: - Pagination

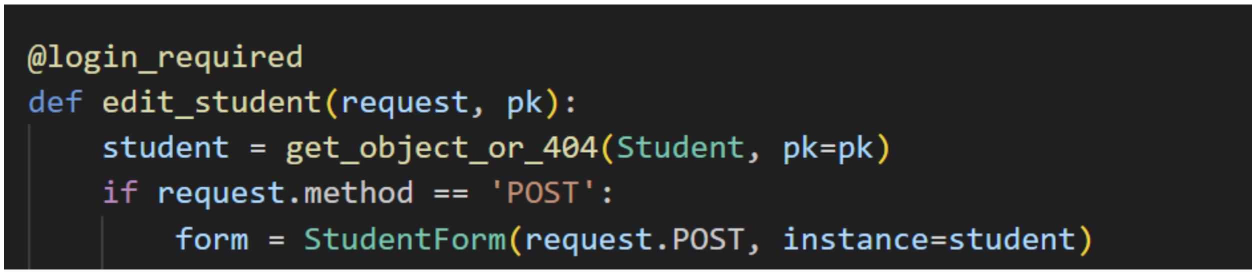
Pagination was implemented to limit the number of students displayed per page (10 per page) using Django’s Paginator class. The template includes controls for navigating between pages (First, Previous, Next, Last).



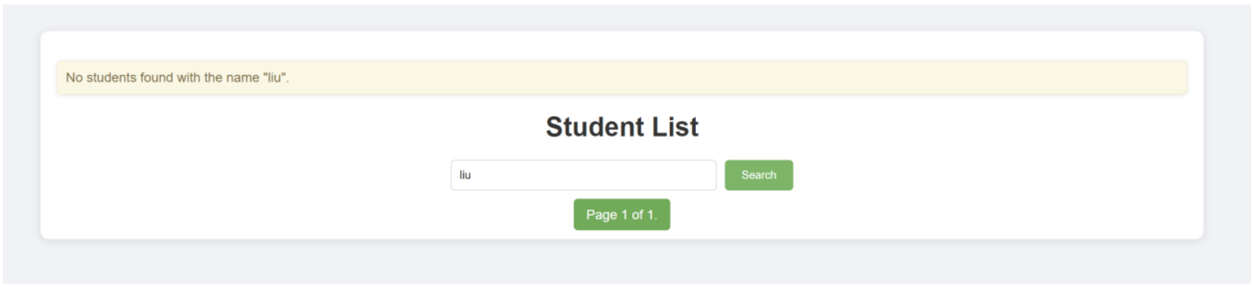
```
# Pagination logic (limit to 10 students per page)

paginator = Paginator(students, 10)  # 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})
```

STEP 9: - Error Handling and Validation

Django's get_object_or_404 was used to handle the case where a non-existent student record is requested, automatically rendering a 404 page if the record doesn't exist

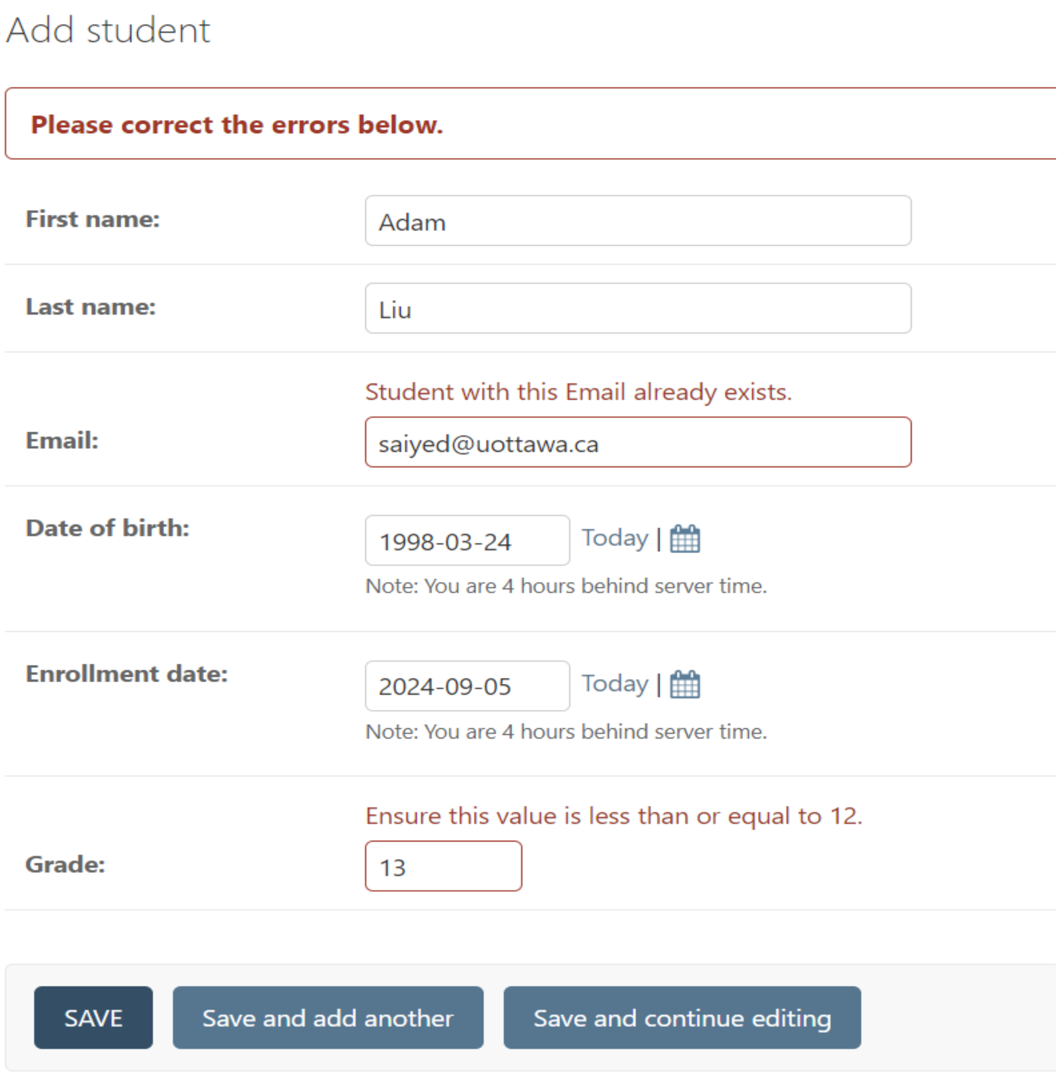


```
@login_required
def edit_student(request, pk):
    student = get_object_or_404(Student, pk=pk)
    if request.method == 'POST':
        form = StudentForm(request.POST, instance=student)
```



No students found with the name "liu".

Student List


Add student

Please correct the errors below.

First name:	Adam
Last name:	Liu
Email:	Student with this Email already exists. saiyed@uottawa.ca
Date of birth:	1998-03-24 Today | Note: You are 4 hours behind server time.
Enrollment date:	2024-09-05 Today | Note: You are 4 hours behind server time.
Grade:	Ensure this value is less than or equal to 12. 13

Challenges Encountered:

1. Email Uniqueness: Although Django has built-in validation for email fields, it was still difficult to make sure that every student's email was distinct in the database and wouldn't cause a database error.
2. Correctly displaying pagination in the template: It was challenging to make sure the pagination buttons showed up and functioned properly in the user interface, particularly when testing with different student counts.
3. Ensuring Proper Redirection: At first, it was confusing to send non-authenticated users to the login page when they attempted to access restricted views. It was essential to comprehend how Django handles redirection using the LOGIN_URL option and the @login_required decorator.

4. Form Validation: It was difficult to make sure the form handled validation issues correctly, such as missing required fields, invalid grades (beyond 1 to 12), or invalid email formats. Incorrect data could still be put into the system if it is not handled properly.

Conclusion:

There were some difficulties in putting the Student Management System into use, particularly regarding login, pagination, search capabilities, and validation. Utilizing Django's integrated tools, these difficulties were resolved. Despite the challenges faced, Django's robust templating and form validation tools, as well as its adaptability in managing authentication, made development easy.

By carefully configuring Django's features and conducting regular testing to make sure the system operated as intended in various scenarios, each difficulty was overcome.