**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.

A screen shot of a computer program

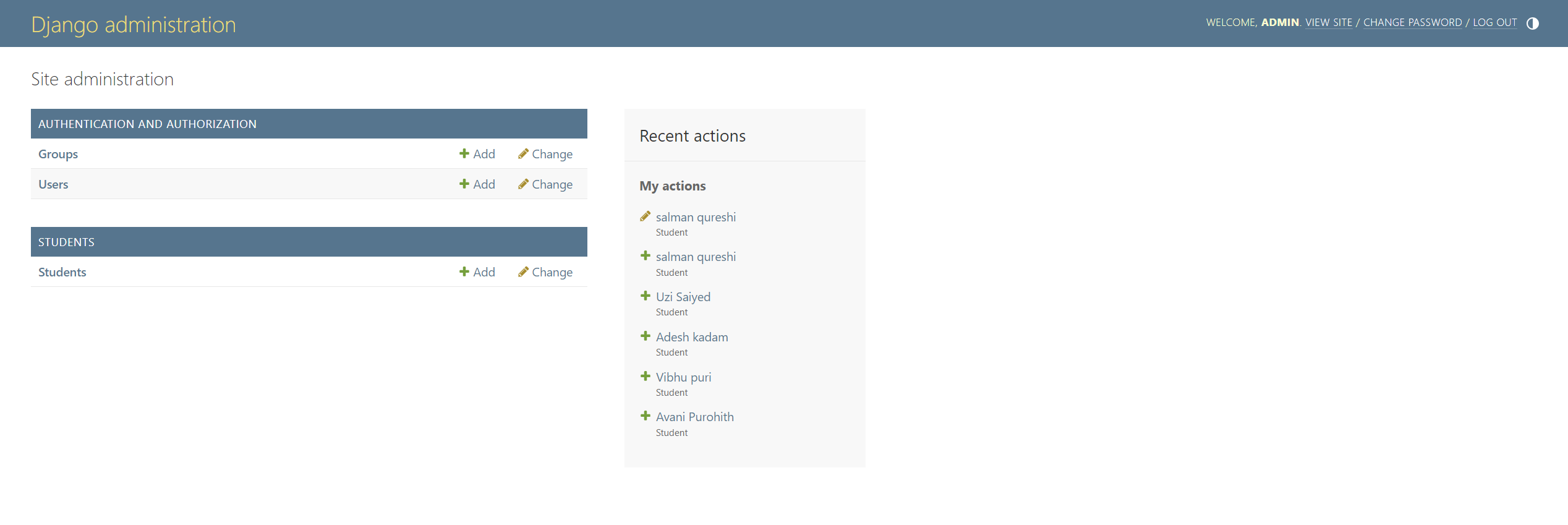
Description automatically generated

A screenshot of a computer

Description automatically generated

**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.



**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.

A screenshot of a computer code

Description automatically generated

**Features Implemented:**

* student\_list to display a list of students with pagination.

A screenshot of a computer

Description automatically generated

* student\_detail to show individual student details.
* add\_student and edit\_student views for adding and updating student information.

A screenshot of a computer

Description automatically generated

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

A black screen with colorful text

Description automatically generated

**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).

A screenshot of a computer program

Description automatically generated

**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.

A screen shot of a computer code

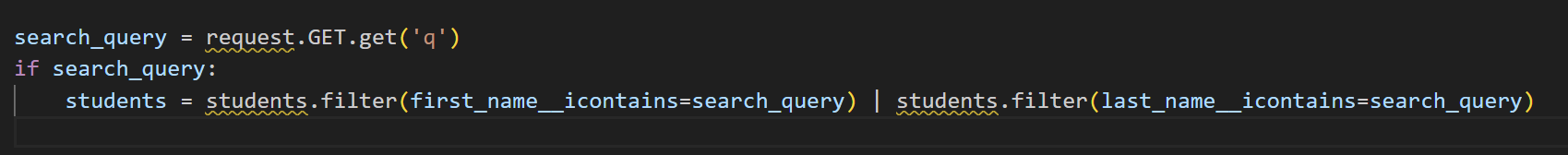
Description automatically generated

A screenshot of a login form

Description automatically generated

**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.



A screenshot of a computer

Description automatically generated

**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).

A computer code with text on it

Description automatically generated

**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

**A computer code on a black background

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**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.