**Building a Student Management System using Django**

1. **Project overview**

Designed to properly handle student information, the Student Management System is a web application built with Django. Viewing a list of students, doing a name search for a student, and panning through the list are all possible. New student additions and record updates are supported by the application. To further limit access to particular features, the program has user authentication built in.  
  
**Among this project's primary goals were:**  
Use the Django framework to create a simple student management system.  
For the purpose of handling student records, provide an intuitive interface.  
Include supplementary features like pagination, user authentication, and search functionality.

**2. Project Setup**

**2.1 Setting Up the Django Project**

* Created a Django project named student\_management.
* Set up a virtual environment using venv and installed Django.
* Initialized a Git repository and made the first commit with the basic project structure.

**2.2 Commands Used**

* Set up the virtual environment and installed Django:

bash

Copy code

python -m venv env

source env/bin/activate # On Windows: env\Scripts\activate

pip install django

* Created a new Django project and app:

bash

Copy code

django-admin startproject student\_management .

python manage.py startapp students

* Created initial migrations and ran the server:

bash

Copy code

python manage.py makemigrations

python manage.py migrate

python manage.py runserver

**2.3 Git Version Control**

* Initialized Git in the project directory and pushed the code to GitHub using your username:

bash

Copy code

git init

git add .

git commit -m "Initial project setup"

git remote add origin https://github.com/upadhyaymohit28/student\_management.git

git push -u origin main

**2.4 Challenges Encountered**

* Setting up the project structure and configuring the initial settings was challenging but was resolved by referring to the Django documentation.

**3. Student Model**

**3.1 Implementation**

Created a Django model for students in students/models.py with the following fields:

* first\_name: CharField to store the first name of the student.
* last\_name: CharField to store the last name of the student.
* email: EmailField to store a unique email address.
* date\_of\_birth: DateField to store the date of birth.
* enrollment\_date: DateField to store the enrollment date.
* grade: IntegerField to store the student’s grade (between 1 and 12).

**3.2 Code Snippet**

python

Copy code

# students/models.py

from django.db import models

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()

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

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

**3.3 Challenges Encountered**

* Ensuring the email field was unique and adding validation for the grade field to ensure it was between 1 and 12.

A screenshot of a computer

Description automatically generated

**4. Admin Interface**

**4.1 Implementation**

* Registered the Student model in the admin interface.
* Customized the admin list display to show first\_name, last\_name, and enrollment\_date.

**4.2 Code Snippet**

python

Copy code

# students/admin.py

from django.contrib import admin

from .models import Student

class StudentAdmin(admin.ModelAdmin):

list\_display = ('first\_name', 'last\_name', 'enrollment\_date')

admin.site.register(Student, StudentAdmin)

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**5. Views and Templates**

**5.1 Views Created**

* **Student List View:** Display a list of all students.
* **Student Detail View:** Display details of a single student.
* **Add Student View:** Provides a form to add a new student.
* **Edit Student View:** Allows updating existing student information.

**5.2 Templates Created**

* students/templates/students/student\_list.html
* students/templates/students/student\_detail.html
* students/templates/students/add\_student.html
* students/templates/students/edit\_student.html

**5.3 Challenges Encountered**

* Structuring templates for better user experience and ensuring navigation between views was intuitive.

A screenshot of a computer

Description automatically generated

**6. Forms**

**6.1 Implementation**

* Created Django forms in students/forms.py for adding and updating student information.
* Added validation to ensure that the email field is unique and the grade is between 1 and 12.

**6.2 Code Snippet**

python

Copy code

# students/forms.py

from django import forms

from .models import Student

class StudentForm(forms.ModelForm):

class Meta:

model = Student

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

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**7. Authentication**

**7.1 Implementation**

* Added Django’s built-in user authentication system.
* Restricted access to adding, editing, and deleting student records to logged-in users.

**7.2 Code Snippet for Login Required**

python

Copy code

# students/views.py

from django.contrib.auth.decorators import login\_required

@login\_required

def add\_student(request):

# View logic

Pass

A screenshot of a login screen

Description automatically generated

**8. Search Functionality**

**8.1 Implementation**

* Added a search bar to filter students by name using icontains queries.

**8.2 Code Snippet**

python

Copy code

# students/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})

A screenshot of a computer

Description automatically generated

**9. Pagination**

**9.1 Implementation**

* Implemented pagination using Django’s Paginator class to display 10 students per page.

**9.2 Code Snippet**

python

Copy code

from django.core.paginator import Paginator

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

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

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

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**10. Error Handling and Validation**

**10.1 Implementation**

* Handled non-existent student records using Django’s get\_object\_or\_404.
* Added form validation to ensure data integrity.

**10.2 Code Snippet**

python

Copy code

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

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

A screenshot of a computer

Description automatically generated

**11. Project Documentation**

**11.1 README.md**

* Created a README.md file with instructions on setting up and running the project locally.

**11.2 Report Summary**

* Added a detailed report with setup instructions, key features, challenges encountered, and code snippets.

**12. Conclusion**

This project was a successful implementation of a basic student management system using Django. It provided valuable insights into web development, handling user authentication, and integrating additional features like search and pagination.