uOttawa

GNG5300: Full Stack Software Development

Assignment 2 | Building a Student Management System using Django

Professor: Masoud Dorrikhteh

Pouria Bahri - 300352271

Fall 2024

Introduction

This project is a web-based student management system. It allows administrators to manage students' details, including enrollment status, grades, and personal data. Developed using Django, the system features functionalities like adding, editing, searching, and paginating student records. With a secure login system for admins, it provides an interface that simplifies administrative tasks.

Setup Process

Detailed step-by-step instructions for setting up the project has been described on README.md. To set up the project, first clone the repository from GitHub, then navigate to the project directory. Ensure that Python and Django are installed, and create a virtual environment using `python -m venv env`. Activate the environment, and apply the migrations using `python manage.py migrate`. A superuser should be configured, so you can directly run the project with `python manage.py runserver` and access the app at `http://127.0.0.1:8000`.

Features Implemented

1. Student Model

Thank to Django, there is no need to use SQL directly, and managing database is easily handled by the framework. Therefore, we only need to specify the features of the class in a pythonic way.

```
class Student(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
    email = models.EmailField(
        max_length=254,
        validators=[EmailValidator(message="Enter a valid email address.")]
    )
    date_of_birth = models.DateField()
    enrolled = models.DateField()
    grade = models.IntegerField(
        validators=[MinValueValidator(1), MaxValueValidator(12)]
    )
```

After that, a simple mechanism called migration will handle creating the database for us.

2. Admin Interface

for setting up the admin interface, the instructions of the provided source on class with the necessary changes were followed and a superuser (pouria:django4) was created. to access the admin control panel, we can easily go to http://localhost:8000/admin which will ask us the username and password which were set before.

Then we had to handle the proper representation of the Student class.

```
class Student(models.Model):
# ***

    class Meta:
        verbose_name_plural = "Students"

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

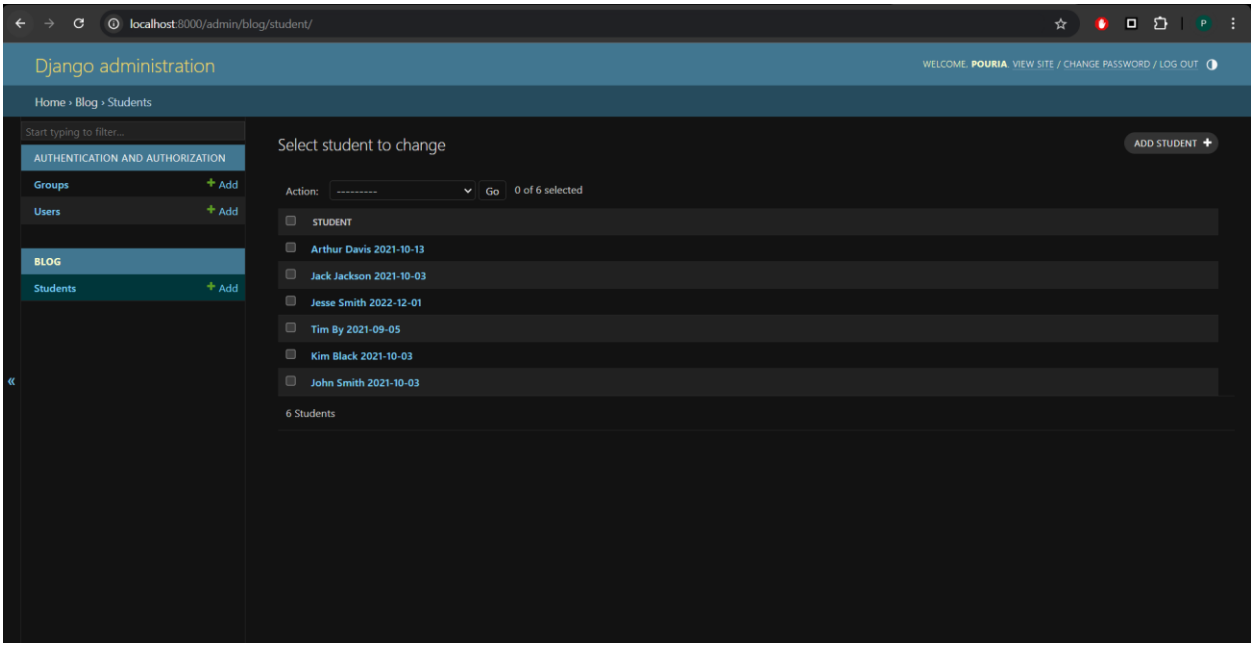

Figure 1- Admin interface

3. Views and Templates:

Handling functionalities of the website was generally done through views and templates. For example, this is student_list view which is the main page of the website.

```
def student_list(request):
    query = request.GET.get('q')  # Get the search query
    if query:
        students = Student.objects.filter(
            Q(first_name__icontains=query) | Q(last_name__icontains=query)
        )
    else:
        students = Student.objects.all()  # Show all students if no search query

    # Pagination logic
    paginator = Paginator(students, 5)  # Display 10 students per page
    page_number = request.GET.get('page')  # Get the current page number from the request
    page_obj = paginator.get_page(page_number)  # Get the appropriate page

    context = {
        "students": page_obj,  # Pass the page object to the template
        "paginator": paginator,  # Optional: pass paginator if you need it for extra logic
        "query": query,  # Pass the query to retain it in the search bar
    }
    return render(request, "blog/student_list.html", context)
```

This page handles the search, add, edit, authentication, and displaying functionality of the website. Then, it needs to also get addressed on urls.py, a file which we built locally to handle custom pages.

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

from . import views

urlpatterns = [
    # path("", views.student_list, name="student_list"),

    path('', views.student_list, name='student_list'),
    path('students/<int:student_id>/', views.student_detail, name='student_detail'),
    path('students/add/', views.add_student, name='add_student'),
    path('students/<int:student_id>/edit/', views.edit_student, name='edit_student'),

    # Authentication URLs
    path('login/',     auth_views.LoginView.as_view(template_name='blog/login.html'), name='login'),
    path('logout/', auth_views.LogoutView.as_view(), name='logout'),
]
```

We have to reference this file on the original urls.py file in the project folder which is done with this code snippet:

```
from django.urls import path, include


urlpatterns = [
    path("admin/", admin.site.urls),
    path("", include("blog.urls")),
]
```

At this point we have to make a nice html page to interact with users.

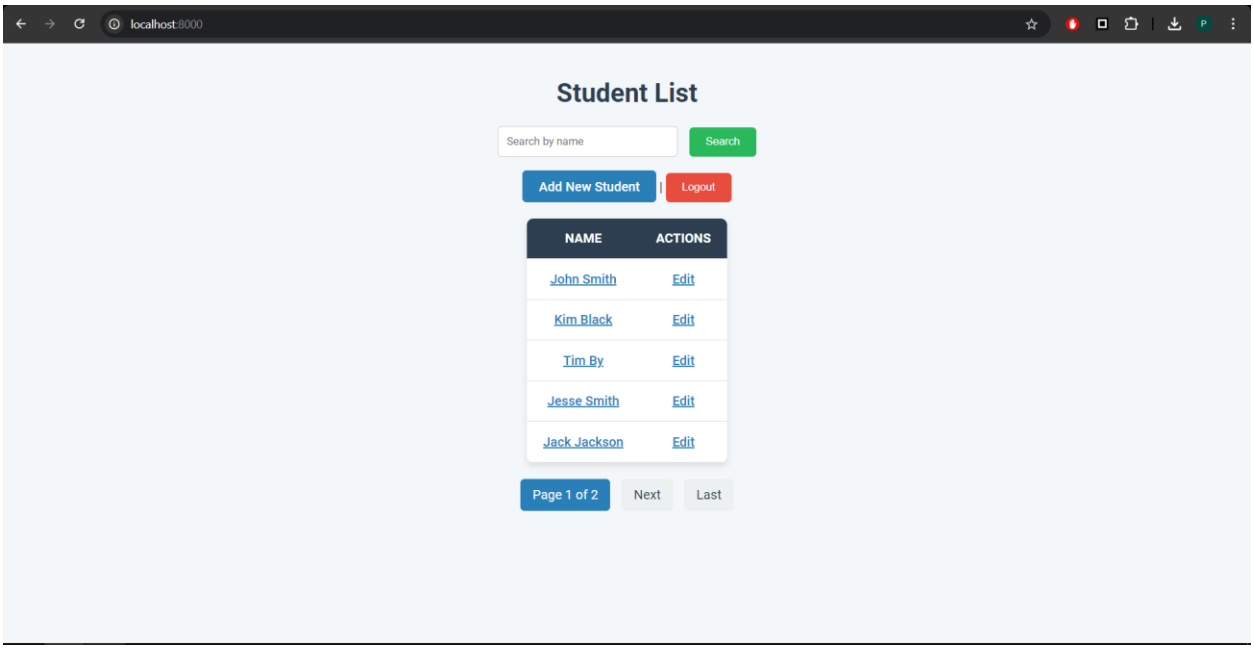

Figure 2- Main page (super user)

Since we logged in initially through admin interface, we have the ability to edit or add students. But if we log out, we see the main page like Figure 3

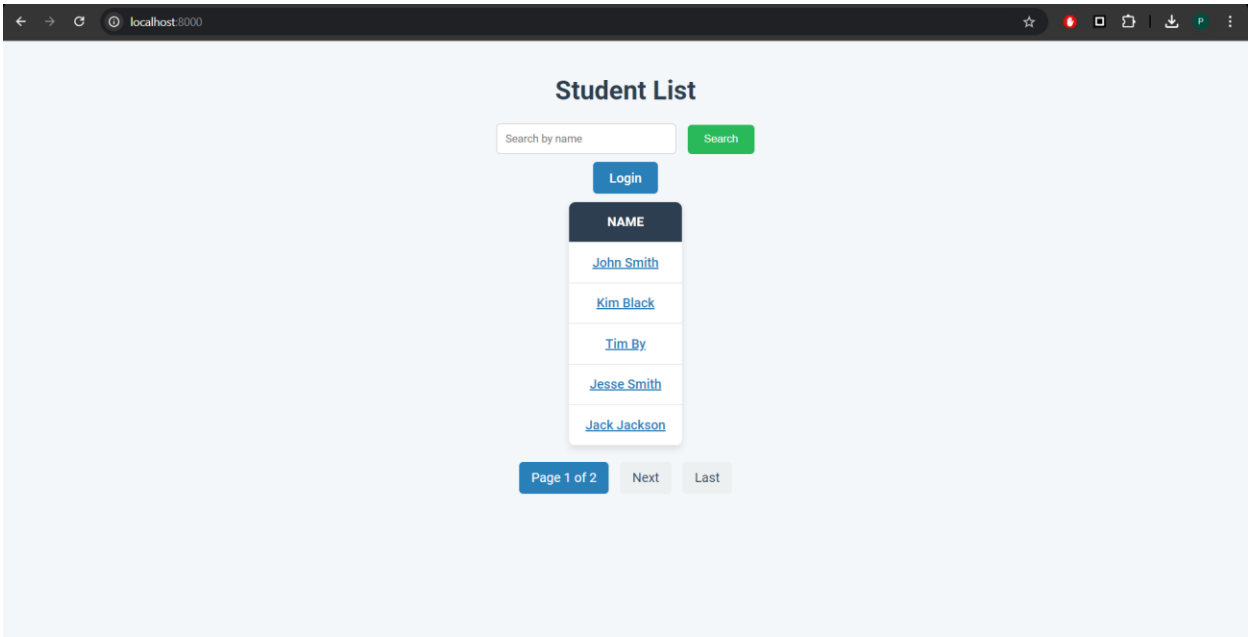

Figure 3- Main page (normal user)

This is handled through a built-in mechanic in Django. To do so, the following snippet was added to settings.py

```
LOGIN_URL = 'login'
LOGIN_REDIRECT_URL = 'student_list'
LOGOUT_REDIRECT_URL = 'student_list'
```

Now, any view can be exclusive to superusers by Django's @login_required decorator. It was added to add_student and edit_student methods:

```
@login_required
def add_student(request):
    if request.method == "POST":
        form = StudentForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('student_list')
    else:
        form = StudentForm()

    return render(request, 'blog/add_student.html', {'form': form})

@login_required
def edit_student(request, student_id):
    student = get_object_or_404(Student, id=student_id)
```

```
    if request.method == "POST":
        if 'save' in request.POST:
            form = StudentForm(request.POST, instance=student)
            if form.is_valid():
                form.save()
                return redirect('student_list')
        elif 'delete' in request.POST:
            student.delete()
            return redirect('student_list')
    else:
        form = StudentForm(instance=student)

    return render(request, 'blog/edit_student.html', {'form': form, 'student': student})
```

it is noteworthy that adding and editing students is handled through built in Django forms. To do so, we just have to change 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', 'enrolled', 'grade']
```

Challenges Encountered

The most major challenge was regarding version control. The goal was to have the assignment submitted on https://github.com/p0ur1a/fullstack_web_development/ repository. But, many challenges which were arose by the lack of expertise had to be overcome. Managing multiple folders and not making a new repository was a relatively new concept which thanks to this assignment, is now done with ease.

Another challenge with git was when something was manually added on GitHub and there was an inconsistency with the local code. While the solution was a simple git pull origin, it took some time to figure this out.