**Student Management CRUD Application**

This project is a basic Django CRUD application to manage student profiles. It allows users to create, view, update, and delete student profiles.

**Models**

The `StudentProfile` model represents students and includes fields such as:

- First Name

- Last Name

- Email (unique)

- Phone Number

- Date of Birth

- Address

- Enrollment Date

- Major

- Status (Active, Graduated, Inactive)

**Features**

- List all students

- View a specific student profile

- Add a new student

- Edit an existing student profile

- Delete a student profile

**Setup Instructions**

Follow these steps to set up and run the Django "student\_management" project:

**1. Create and Activate a Virtual Environment (Optional but Recommended)**

It's a good practice to create a virtual environment for each project to manage dependencies.

1. **Create a virtual environment**:

*python -m venv venv*

1. **Activate the virtual environment:**

On Windows:

*venv\Scripts\activate*

On macOS/Linux:

*source venv/bin/activate*

**2. Install Django and MySQL DB client**

Once the virtual environment is active, install Django:

*pip install Django*

*pip install mysqlclient*

or

*pip install PyMySql*

**3. Check version**

*pip list*

*Package Version*

*----------- -------*

*asgiref 3.8.1*

*Django 5.1.1*

*mysqlclient 2.2.4*

*pip 24.2*

*sqlparse 0.5.1*

*tzdata 2024.2*

**4. Create a Django Project**

To create a new Django project named student\_management:

*django-admin startproject student\_management*

**5. Navigate into the Project Directory**

*cd student\_management*

**6. Create a Django App**

Create an app named student:

*python manage.py startapp student*

**7. Configure the App in the Project**

Open student\_management/settings.py, and add the newly created student to the INSTALLED\_APPS list:

*INSTALLED\_APPS = [*

*Django default apps*

*'django.contrib.admin',*

*'django.contrib.auth',*

*'django.contrib.contenttypes',*

*'django.contrib.sessions',*

*'django.contrib.messages',*

*'django.contrib.staticfiles',*

*Add your app here*

*'student',*

*]*

*Configure Database settings*

*DATABASES = {*

*'default': {*

*'ENGINE': 'django.db.backends.mysql', # Use 'mysql' for mysqlclient or 'pymysql' for PyMySQL*

*'NAME': 'your\_database\_name', # Replace with your database name*

*'USER': 'your\_username', # Replace with your MySQL username*

*'PASSWORD': 'your\_password', # Replace with your MySQL password*

*'HOST': 'localhost', # Or your MySQL server address*

*'PORT': '3306', # Default MySQL port*

*}*

*}*

add the following to configure static files:

*STATIC\_URL = 'static/'*

*STATICFILES\_DIRS = [BASE\_DIR / "static"]*

***Create the static/ directory in the project root***

***Inside the static/ directory, create subdirectories for css, js, and images:***

**8. Test the Database Connection**

Run Migrations: To ensure your Django application can connect to the MySQL database, run the following command to apply migrations:

*python manage.py migrate*

*python manage.py runserver*

**9. Check the DB Structure**

**A screenshot of a computer

Description automatically generated**

You will notice the tables created to handle Django Authentication

**10. Define the StudentProfile Model**

In student/models.py, define Profile model as provided:

*from django.db import models*

*class Profile(models.Model):*

*STATUS\_CHOICES = [*

*('active', 'Active'),*

*('graduated', 'Graduated'),*

*('inactive', 'Inactive'),*

*]*

*# Basic Information*

*first\_name = models.CharField(max\_length=50)*

*last\_name = models.CharField(max\_length=50)*

*email = models.EmailField(unique=True)*

*# Optional fields*

*phone\_number = models.CharField(max\_length=15, blank=True, null=True)*

*date\_of\_birth = models.DateField(blank=True, null=True)*

*address = models.TextField(blank=True, null=True)*

*# Enrollment Information*

*enrollment\_date = models.DateField()*

*major = models.CharField(max\_length=100, blank=True, null=True)*

*# Status with predefined choices*

*status = models.CharField(max\_length=20, choices=STATUS\_CHOICES, default='active')*

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

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

**11. Migrate the model and check DB**

**A screenshot of a computer

Description automatically generated**Run Migrations: To ensure the model created

*python manage.py makemigrations*

*python manage.py migrate*

*python manage.py runserver*

**student\_profile table created**

**Django forms vs templates**

Choosing between using Django forms or templates depends on the specific use case and requirements of your application

**1. Django Forms**

Django forms provide a powerful way to handle user input in your application. They offer a built-in mechanism for form validation, data cleaning, and error handling.

**Advantages of Using Django Forms**

* Validation: Forms automatically validate user input, ensuring that only valid data is processed.
* Rendering: Django forms can automatically render HTML for form fields and error messages, reducing boilerplate code.
* Security: Forms help protect against common web vulnerabilities, such as Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF).
* Reusability: You can create reusable form classes that can be used across different views.
* Integration with Model: Django ModelForms allow you to easily create forms tied to your models, which simplifies saving data to the database.

**When to Use Django Forms**

* When you need to collect user input, such as in registration or contact forms.
* When you require validation and error handling for user data.
* When you want to leverage ModelForms for easy data handling and storage.

**2. Django Templates**

Django templates are used to render HTML views and dynamically display data. Templates can include forms, but they are more general-purpose for generating any HTML content.

**Advantages of Using Django Templates**

* Separation of Concerns: Templates allow you to separate your HTML from your Python code, promoting cleaner, more maintainable code.
* Dynamic Content: You can easily display dynamic data, such as lists of objects, inside your templates.
* Customizability: Templates give you complete control over the HTML structure and styling, allowing for greater flexibility in design.
* Include Tags: You can use {% include %} tags to create reusable HTML components across your application.

**When to Use Django Templates**

* When you need to render complex HTML pages with dynamic content.
* When you want to separate your business logic from presentation logic.
* When you're displaying data, such as lists of items, rather than collecting user input.

**Best Practices**

* Use Forms for User Input: If your application involves forms for user input, always prefer Django forms for better validation, security, and maintainability.
* Use Templates for Rendering: Use Django templates for rendering your application’s views, especially when displaying dynamic content or when you need to customize the layout significantly.

**12. Create the Profile Form with validation**

Student/forms.py

**13. Create the Views (CRUD Operations)**

In student/views.py, define the views for Create, Read, Update, and Delete operations using Django's generic views.

**14. Define the URL Pattern in project**

In student/urls.py, define URL patterns for the inner pages

In student\_management/urls.py, include the student URLs:

*from django.contrib import admin*

*from django.urls import path, include*

*urlpatterns = [*

*path('admin/', admin.site.urls),*

*path('', include('student.urls')), #include the student app urls*

*]*

**10. Create HTML Templates for the Pages**

A screenshot of a computer

Description automatically generated

**11. Run Database Migrations**

Before running the server, apply the initial migrations:

*python manage.py migrate*

**12. Run the Development Server**

Finally, start the Django development server:

*python manage.py runserver*

**13. Launch the Application and see the result**

Open your web browser and visit:

*http://127.0.0.1:8000/*

A screenshot of a computer

Description automatically generated

**14. Deactivating the Virtual Environment**

To deactivate the virtual environment when you're done working:

*Deactivate*