PROJECT REPORT

Student Database Management System

Team: Dev Srijit & Neel Tambe

Registration Numbers:

- Dev Srijit- 235805378(Roll no.:44)
- Neel Tambe- 235805310(Roll no.:36)

Acknowledgement:

I sincerely thank my project guide, **Dr.Manasa CM**, for their invaluable guidance and support throughout the development of the Student Database Management System. I am also grateful to my professors at Manipal Institute of Technology Bengaluru, for their insights and resources, which greatly enhanced my understanding of database design. Additionally, I appreciate the collaboration and suggestions from my peers, which helped address challenges during the project. Finally, heartfelt thanks to my family and friends for their constant encouragement and support. This project's success is a result of the collective efforts of everyone involved, and I am deeply grateful.

Synopsis:

TITLE
STUDENT DATABASE MANAGEMENT SYSTEM

Team Members:

- Dev Srijit
- Neel Tambe

Abstract:

The Student Database Management System is designed to streamline the management of student records in an educational institution. This system will provide functionalities to add new students through a form-based input interface, perform CRUD (Create, Read, Update, Delete) operations on student records, and display a tabular view of all student data , also an Attendance marking system for the students. The project aims to simplify administrative tasks by offering an efficient and user-friendly database management solution.

Problem Statement:

Educational institutions often face challenges in managing large volumes of student data effectively. Manual record-keeping or outdated systems can lead to inefficiencies, errors, and difficulty in retrieving information. The proposed system addresses these issues by providing:
1. A robust database to store and manage student information such as names, roll numbers, courses, and contact details.
2. A user-friendly interface for adding students via a form.
3. CRUD functionalities to edit, delete, or update records seamlessly.
4. A tabular view for quick access and analysis of all stored records.
5.An Attendance tab to mark attendance of students in the database.

The project will focus on both data requirements (e.g., database schema design) and functional requirements (e.g., interface design and query implementation).

Requirements Gathering and Design:

Data Requirements:

Student Table:
- Student ID (Primary Key)
- Name
- Course
- cgpa
- Dob
- Email
- Department(Foreign Key)

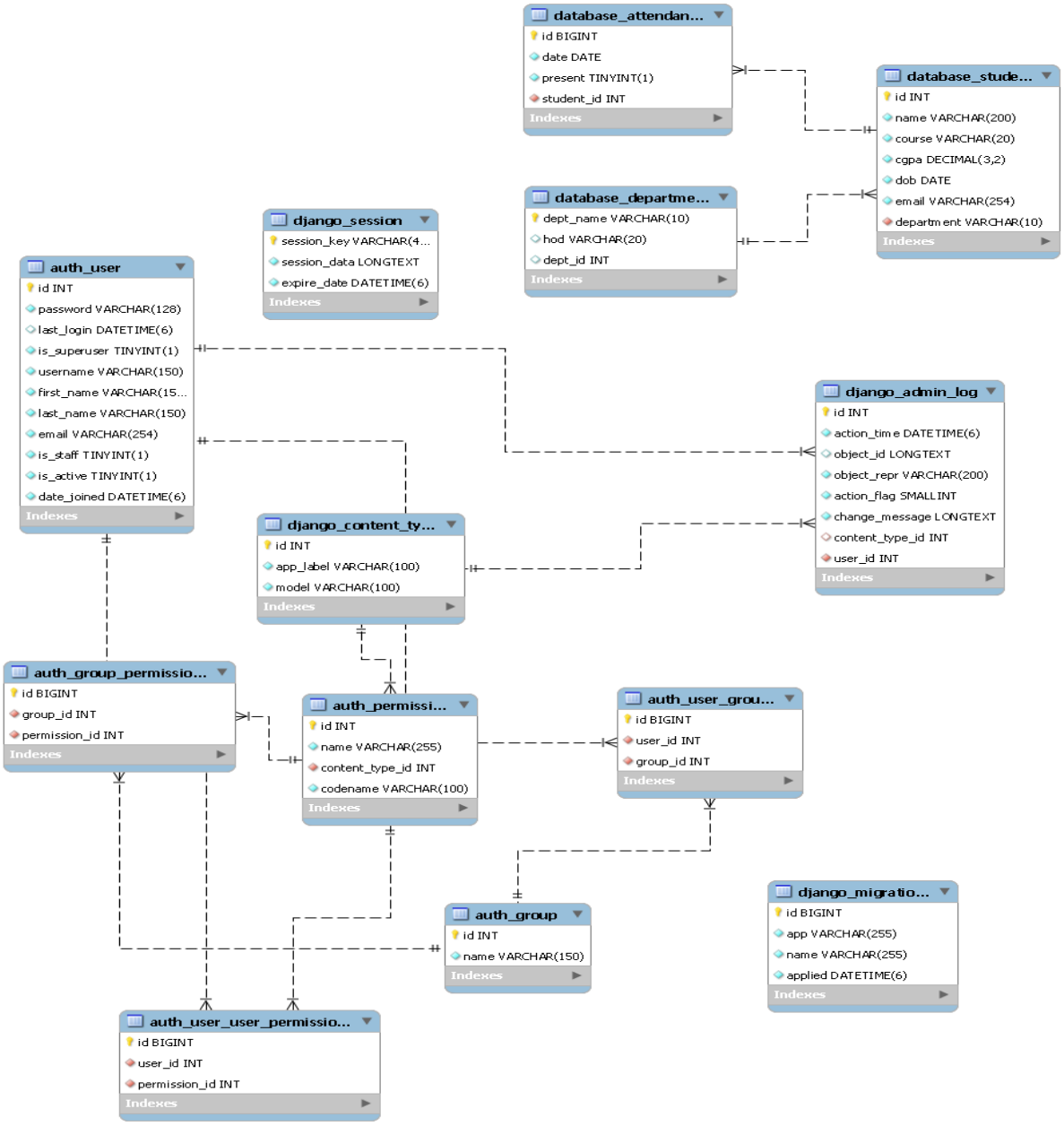
Department Table:
- id(Primary key)
- hod
- dept_name

Attendance Table:
- student(Foreign Key)
- Date
- Present

Functional Requirements:

- Form-based input for adding new students.
- Options to perform CRUD operations on existing records.
- Tabular display of student records with sorting and filtering capabilities, Attendance marking section.
- Django Framework and Mysql for backend and template rendering

E.R Diagram & Relational Tables:



DDL Commands:

Built In Django Admin Panel for User Creation, Adding Roles and Views.

Student Table:

```python
class Student(models.Model):
    DEPARTMENT_CHOICES = [
        ('MIT', 'MIT'),
        ('MLS', 'MLS'),
        ('TAPMI', 'TAPMI'),
        ('DLHS', 'DLHS'),
        ('DOC', 'DOC'),
        ('SMIT', 'SMIT'),
    ]

    COURSE_CHOICES = [
        ('CSE', 'CSE (Core)'),
        ('Cybersecurity', 'Cybersecurity'),
        ('AI', 'Artificial Intelligence'),
        ('DataScience', 'Data Science'),
        ('IT', 'Information Technology'),
        ('ECE', 'Electronics & Communication'),
        ('VLSI', 'VLSI Design'),
        ('Other', 'Other'),
    ]

    id = models.AutoField(primary_key=True)
    name = models.CharField(max_length=200)
    dob = models.DateField()
    email = models.EmailField(max_length=254, default="default@example.com")
    course = models.CharField(max_length=20, choices=COURSE_CHOICES, default='CSE')  # Dropdown
    cgpa = models.DecimalField(max_digits=3, decimal_places=2)  # Allows values like 9.50
    department = models.CharField(max_length=10, choices=DEPARTMENT_CHOICES, default='MIT')  # Dropdown

    def __str__(self):
        return self.name
```

Attendance Table:

```python
class Attendance(models.Model):
    student = models.ForeignKey(Student, on_delete=models.CASCADE)
    date = models.DateField(auto_now_add=True)  # Automatically set on creation
    present = models.BooleanField(default=False)

    class Meta:
        unique_together = ('student', 'date') # One attendance entry per student per day

    def __str__(self):
        return f"{self.student.name} - {self.date} - {'Present' if self.present else 'Absent'}"
```

Department Table:

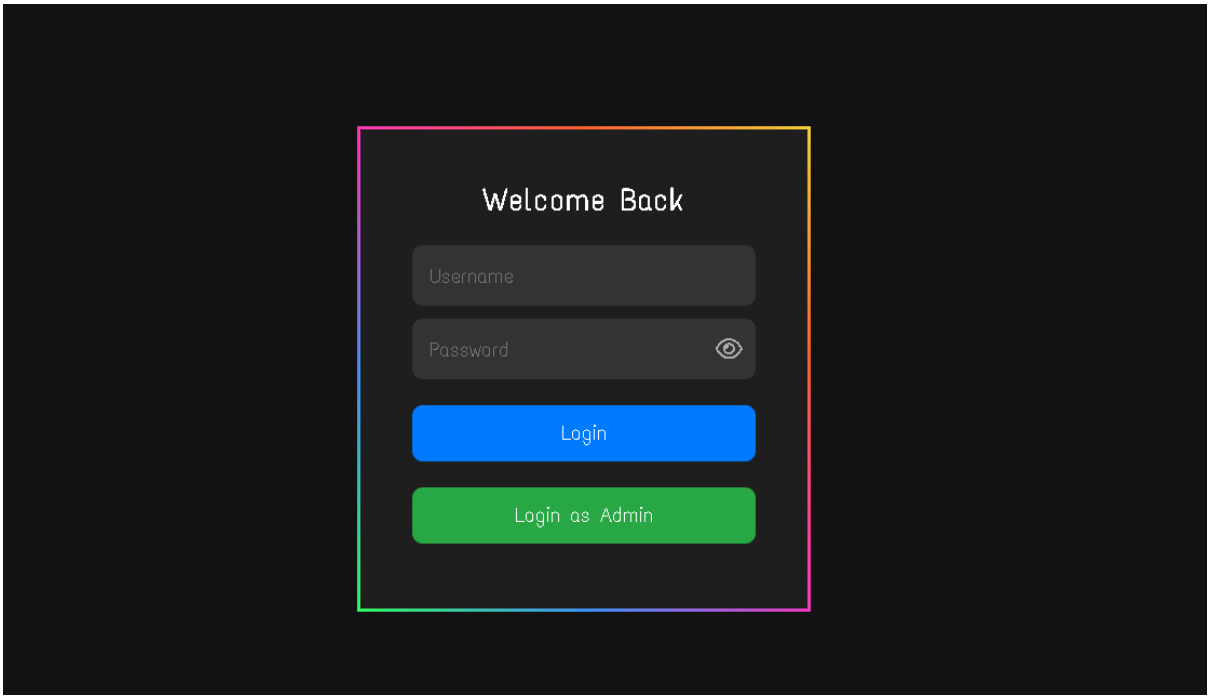
```sql
CREATE TABLE database_department (
    dept_name VARCHAR(10) NOT NULL PRIMARY KEY,
    hod VARCHAR(20),
    dept_id INT
);
INSERT INTO database_department VALUES ('MIT', 'RCB', 123);
INSERT INTO database_department VALUES ('DLHS', 'MI', 456);
INSERT INTO database_department VALUES ('TAPMI', 'RR', 789);
INSERT INTO database_department VALUES ('MLS', 'CSK', 246);
INSERT INTO database_department VALUES ('DOC', 'LSG', 357);
INSERT INTO database_department VALUES ('SMIT', 'SRH', 579);
```

Sql Queries for Student and Attendance Tables:

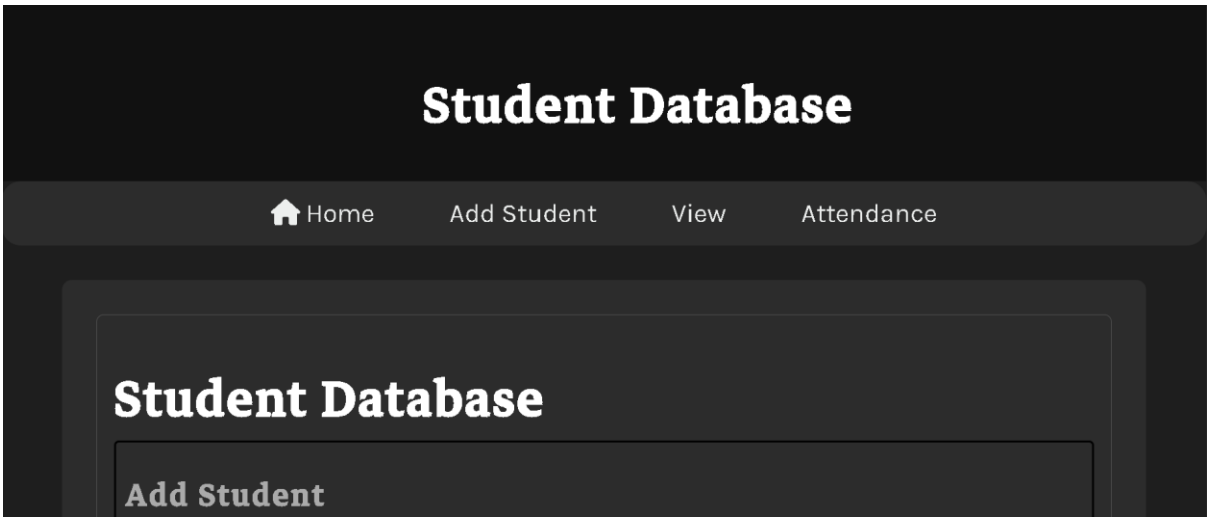
```sql
CREATE TABLE Student (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(200) NOT NULL,
  dob DATE NOT NULL,
  email VARCHAR(254) NOT NULL DEFAULT 'default@example.com',
  course VARCHAR(20) NOT NULL DEFAULT 'CSE',
  cgpa DECIMAL(3, 2) NOT NULL,
  department VARCHAR(10) NOT NULL DEFAULT 'MIT'
);

CREATE TABLE Attendance (
  id INT AUTO_INCREMENT PRIMARY KEY,
  student_id INT NOT NULL,
  date DATE NOT NULL DEFAULT CURRENT_DATE,
  present BOOLEAN NOT NULL DEFAULT FALSE,
  CONSTRAINT fk_student FOREIGN KEY (student_id) REFERENCES Student (id)
    ON DELETE CASCADE,
  UNIQUE (student_id, date)
);
```

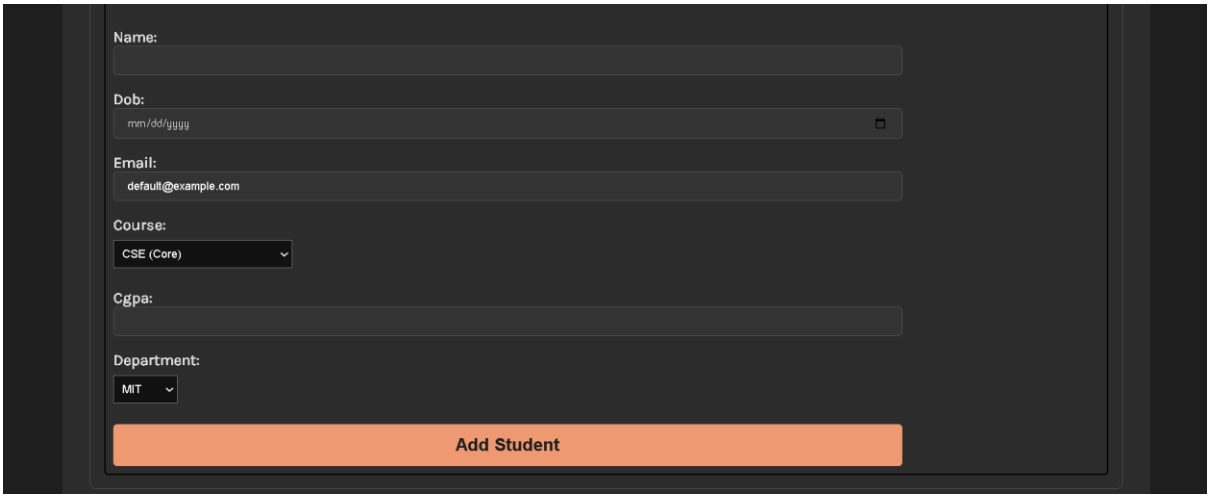
UI Design:

Login Page:

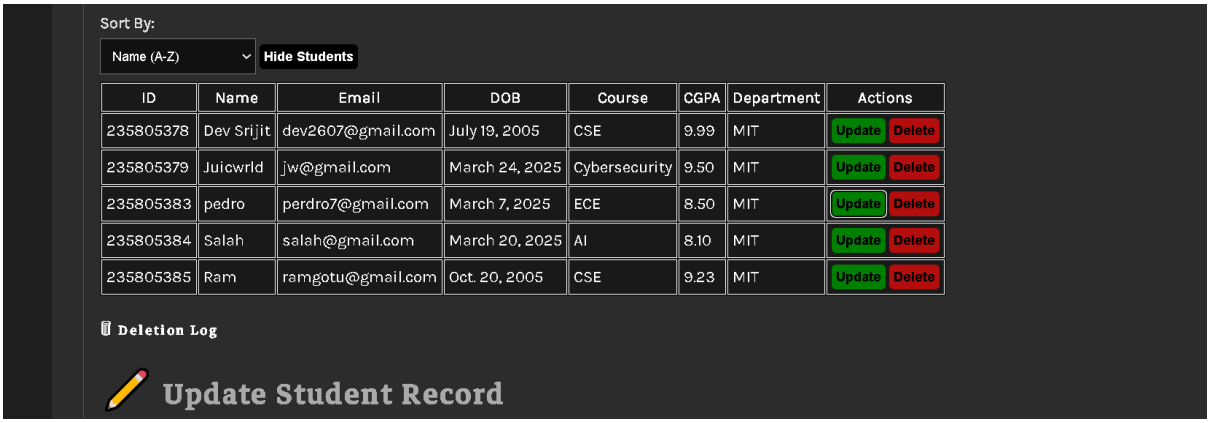


Form to Add New Student Entries:

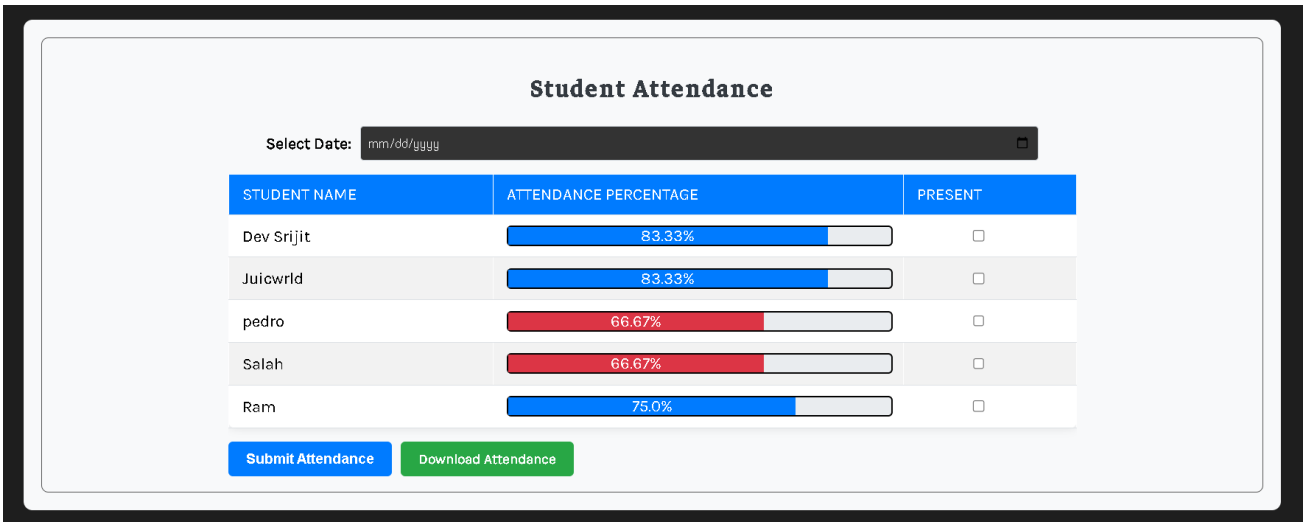




View and Perform basic CRUD operations:



Student Attendance:



Scripts/Functions/Triggers:

Login:

```
<script>
    function togglePassword() {
        var passwordField = document.getElementById("password");
        var eyeIcon = document.getElementById("eyeIcon");

        if (passwordField.type === "password") {
            passwordField.type = "text";
            eyeIcon.classList.remove("fa-eye");
            eyeIcon.classList.add("fa-eye-slash");
        } else {
            passwordField.type = "password";
            eyeIcon.classList.remove("fa-eye-slash");
            eyeIcon.classList.add("fa-eye");
        }
    }
</script>
```

Views.py

```
from django.shortcuts import render, redirect, get_object_or_404
from django.contrib import messages
from django.contrib.auth.decorators import login_required
from django.contrib.auth import authenticate, login, logout
from .models import Student, Attendance
from .forms import StudentForm
from django.http import JsonResponse
from django.views.decorators.csrf import csrf_exempt
import json
from datetime import datetime, date
from django.db import IntegrityError
from django.core.mail import send_mail
from django.conf import settings

def login_view(request):
    if request.method == "POST":
        username = request.POST.get("username")
        password = request.POST.get("password")
        user = authenticate(request, username=username, password=password)

        if user is not None:
            login(request, user)
            return redirect("home/")
        else:
            messages.error(request, "Invalid username or password")
```

```
    return render(request, "login.html")


@login_required
def home(request):
    return render(request, "index.html")

@login_required
def database(request):
    form = StudentForm()
    if request.method == 'POST':
        form = StudentForm(request.POST)
        if form.is_valid():
            form.save()
            messages.success(request, "Student added successfully!")
            return redirect('database')

    return render(request, 'database.html', {'form': form})

@login_required
def view(request):
    students = Student.objects.all()
    return render(request, 'view.html', {'students': students})

@login_required
def attendance(request):
    students = Student.objects.all()

    if request.method == 'POST':
        attendance_date_str = request.POST.get('attendanceDate')
        if attendance_date_str:
            attendance_date = datetime.strptime(attendance_date_str, '%Y-%m-
%d').date()
        else:
            messages.error(request, "Please select a date.")
            return render(request, 'attendance.html', {'students': students})

        for student in students:
            present = request.POST.get(f'attendance_{student.id}') == 'present'
            try:
                # Get or create the attendance record for the student and date
                attendance = Attendance.objects.create(
                    student=student,
                    date=attendance_date,
                    present=present
                )
            except IntegrityError as e:
                messages.warning(request, f"Attendance already recorded for
{student.name} on {attendance_date}.  Skipping.")
```

```
        messages.success(request, "Attendance submitted successfully!")
        return redirect('attendance')  # Redirect to refresh the page
    else:
        # Calculate attendance percentage for each student
        for student in students:
            total_days = Attendance.objects.filter(student=student).count()
            present_days = Attendance.objects.filter(student=student,
present=True).count()

            if total_days > 0:
                percentage = (present_days / total_days) * 100
            else:
                percentage = 0.0  # Or some other default value, like None or -1

            student.attendance_percentage = round(percentage, 2)  # Store percentage
in student object for template

        # Fetch all attendance records for display
        all_attendance = Attendance.objects.all()


        return render(request, "attendance.html", {'students': students,
'all_attendance': all_attendance})

@login_required
def delete_student(request, student_id):
    student = get_object_or_404(Student, id=student_id)
    student.delete()
    return JsonResponse({'success': True})

@csrf_exempt
@login_required
def update_student(request, student_id):
    if request.method == "POST":
        try:
            data = json.loads(request.body)
            student = Student.objects.get(id=student_id)
            student.name = data["name"]
            student.email = data["email"]
            student.dob = data["dob"]
            student.course = data["course"]
            student.cgpa = data["cgpa"]
            student.save()
            return JsonResponse({"success": True})
        except Student.DoesNotExist:
            return JsonResponse({"success": False, "error": "Student not found"})
    return JsonResponse({"success": False, "error": "Invalid request"})
```

Forms:

Form Creation-

```python
from django import forms
from .models import Student

class StudentForm(forms.ModelForm):
    class Meta:
        model = Student
        fields = '__all__'
        widgets = {
            'dob': forms.DateInput(attrs={'type': 'date'}),  # Adds date picker
        } # This includes all fields from Student model
```

```html
<script>
    console.log("Student Database Loaded");

    setTimeout(() => {
        let messageDiv = document.getElementById("messageContainer");
        if (messageDiv) messageDiv.style.display = "none";
    }, 3000);
</script>
```

Login Function:

```html
<script>
    function togglePassword() {
        var passwordField = document.getElementById("password");
        var eyeIcon = document.getElementById("eyeIcon");

        if (passwordField.type === "password") {
            passwordField.type = "text";
            eyeIcon.classList.remove("fa-eye");
            eyeIcon.classList.add("fa-eye-slash");
        } else {
            passwordField.type = "password";
            eyeIcon.classList.remove("fa-eye-slash");
            eyeIcon.classList.add("fa-eye");
        }
    }
</script>
<script src="https://kit.fontawesome.com/e97095fee3.js" crossorigin="anonymous"></script>
```

Urls.py (Rendering):

```python
from django.urls import path
from . import views
from django.conf import settings
from django.conf.urls.static import static
from django.contrib.auth.views import LogoutView

urlpatterns = [
    path("", views.login_view, name="login"),
    path("home/", views.home, name="home"),
    path("database/", views.database, name="database"),
    path("view/", views.view, name="view"),
    path("attendance/", views.attendance, name="attendance"),
    path("delete-student/<int:student_id>/", views.delete_student, name="delete_student"),
    path("update-student/<int:student_id>/", views.update_student, name="update_student"),
    path("logout/", LogoutView.as_view(next_page="login"), name="logout"),
]

if settings.DEBUG:
    urlpatterns += static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
```

Basic CRUD operations:

```html
<script>
console.log("Student Database Loaded");

function searchStudent() {
    let inputId = document.getElementById("searchInput").value.trim();
    let table = document.getElementById("studentTable");
    let rows = table.getElementsByTagName("tr");
    let searchTable = document.getElementById("searchTable");
    let searchTableBody = document.getElementById("searchTableBody");

    searchTableBody.innerHTML = "";
    searchTable.style.display = "none";

    if (inputId === "") {
        alert("Please enter a student ID.");
        return;
    }

    let found = false;
    for (let i = 1; i < rows.length; i++) {
        let idCell = rows[i].getElementsByTagName("td")[0];
        if (idCell && idCell.innerText.trim() === inputId) {
            let clonedRow = rows[i].cloneNode(true);
            searchTableBody.appendChild(clonedRow);
            found = true;
            break;
```

```
        }
    }

    searchTable.style.display = found ? "table" : "table";
    if (!found) searchTableBody.innerHTML = `<tr><td colspan='7'>No student found
with ID ${inputId}.</td></tr>`;
}

function sortTable() {
    let table = document.getElementById("studentTable");
    let rows = Array.from(table.getElementsByTagName("tr")).slice(1);
    let sortType = document.getElementById("sortSelect").value;

    rows.sort((a, b) => {
        let valueA, valueB;
        if (sortType === "name") {
            valueA = a.getElementsByClassName("name")[0].innerText.toLowerCase();
            valueB = b.getElementsByClassName("name")[0].innerText.toLowerCase();
            return valueA.localeCompare(valueB);
        } else {
            valueA = parseFloat(a.getElementsByClassName("cgpa")[0].innerText);
            valueB = parseFloat(b.getElementsByClassName("cgpa")[0].innerText);
            return valueB - valueA;
        }
    });

    let tbody = table.getElementsByTagName("tbody")[0];
    rows.forEach(row => tbody.appendChild(row));
}

function deleteStudent(button) {
    let studentId = button.getAttribute("data-id");
    let row = button.closest("tr");
    let studentName = row.getElementsByClassName("name")[0].innerText;
    if (confirm("Are you sure you want to delete " + studentName + "?")) {
        fetch(`/delete-student/${studentId}/`, {
            method: "POST",
            headers: {
                "X-CSRFToken": "{{ csrf_token }}",
                "Content-Type": "application/json"
            }
        })
        .then(response => response.json())
        .then(data => {
            if (data.success) {
                row.remove();
                updateStudentCount();
                logDeletion(studentName, studentId);
```

```
                alert("Student deleted successfully!");
            } else {
                alert("Error deleting student!");
            }
        })
        .catch(error => console.error("Error:", error));
    }
}

function updateStudentCount() {
    let count = document.querySelectorAll("#studentTable tbody tr").length;
    document.getElementById("studentCount").innerText = count;
}

function logDeletion(name, id) {
    let logList = document.getElementById("deletionLog");
    let logItem = document.createElement("li");
    logItem.innerText = `Deleted: ${name} (ID: ${id})`;
    logList.appendChild(logItem);
}

function saveStudentChanges(event) {
    event.preventDefault();

    let studentId = document.getElementById("updateStudentId").value;
    let updatedData = {
        name: document.getElementById("updateName").value,
        email: document.getElementById("updateEmail").value,
        dob: document.getElementById("updateDOB").value,
        course: document.getElementById("updateCourse").value,
        cgpa: parseFloat(document.getElementById("updateCGPA").value),
        department: document.getElementById("updateDepartment").value
    };

    fetch(`/update-student/${studentId}/`, {
        method: "POST",
        headers: {
            "X-CSRFToken": "{{ csrf_token }}",
            "Content-Type": "application/json"
        },
        body: JSON.stringify(updatedData)
    })
    .then(response => response.json())
    .then(data => {
        if (data.success) {
            updateTable(studentId, updatedData);
            document.getElementById("updateForm").style.display = "none";
            alert("Student updated successfully!");
```

```
        } else {
            alert("Error updating student!");
        }
    })
    .catch(error => console.error("Error:", error));
}

function updateTable(studentId, updatedData) {
    let row = document.getElementById("studentRow" + studentId);
    row.getElementsByClassName("name")[0].innerText = updatedData.name;
    row.cells[2].innerText = updatedData.email;
    row.cells[3].innerText = updatedData.dob;
    row.cells[4].innerText = updatedData.course;
    row.cells[5].innerText = updatedData.cgpa.toFixed(2);
    row.cells[6].innerText = updatedData.department;
}

document.addEventListener("DOMContentLoaded", function () {
    document.getElementById("studentTable").style.display = "none";
    toggleCourseOptions(); // Initialize course options on page load
});

function toggleStudents() {
    let table = document.getElementById("studentTable");
    let button = document.getElementById("toggleStudentsButton");

    if (table.style.display === "none") {
        table.style.display = "table";
        button.innerText = "Hide Students";
    } else {
        table.style.display = "none";
        button.innerText = "Show Students";
    }
}

function editStudent(studentId) {
    let row = document.getElementById("studentRow" + studentId);
    let name = row.getElementsByClassName("name")[0].innerText;
    let email = row.cells[2].innerText;
    let dob = row.cells[3].innerText;
    let course = row.getElementsByClassName("course")[0].innerText.trim();
    let cgpa = row.getElementsByClassName("cgpa")[0].innerText.trim();
    let department = row.getElementsByClassName("department")[0].innerText.trim();

    document.getElementById("updateStudentId").value = studentId;
    document.getElementById("updateName").value = name;
    document.getElementById("updateEmail").value = email;
    document.getElementById("updateDOB").valueAsDate = new Date(dob);
```

```
    document.getElementById("updateCourse").value = course;
    document.getElementById("updateCGPA").value = parseFloat(cgpa);
    document.getElementById("updateDepartment").value = department;

    document.getElementById("updateForm").style.display = "block";
}

function toggleCourseOptions() {
    let departmentSelect = document.getElementById("updateDepartment");
    let courseSelect = document.getElementById("updateCourse");

    // Check if the selected department is MIT
    if (departmentSelect.value === "MIT") {
        // If MIT is selected, enable the course options
        courseSelect.disabled = false;
    } else {
        // If MIT is not selected, disable the course options
        courseSelect.disabled = true;
    }
}
</script>
```

Backend Connectivity and Rendering:

```
"""
Django settings for studentdbms project.

Generated by 'django-admin startproject' using Django 5.1.5.

For more information on this file, see
https://docs.djangoproject.com/en/5.1/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/5.1/ref/settings/
"""

from pathlib import Path
# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent


# Quick-start development settings - unsuitable for production
```

```python
# See https://docs.djangoproject.com/en/5.1/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-(iqv*dry396vmd)981ds&m$a=hxg^#6n1p=korg-dt-uwf+b0o'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'database',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'studentdbms.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': ['database/templates'],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
```

```
]

WSGI_APPLICATION = 'studentdbms.wsgi.application'


# Database
# https://docs.djangoproject.com/en/5.1/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'student',
        'HOST' : 'localhost',
        'PORT' : '3306',
        'USER' : 'root',
        'PASSWORD' : 'root@123',
         'OPTIONS': {
            'init_command': "SET sql_mode='STRICT_TRANS_TABLES'"
         }
    }
}

MEDIA_URL= 'media/'


# Password validation
# https://docs.djangoproject.com/en/5.1/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]


# Internationalization
# https://docs.djangoproject.com/en/5.1/topics/i18n/

LANGUAGE_CODE = 'en-us'
```

```python
TIME_ZONE = 'UTC'

USE_I18N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/5.1/howto/static-files/

STATIC_URL = "/static/"

STATICFILES_DIRS = [
    str(BASE_DIR / "database/static"),
]
STATIC_ROOT = BASE_DIR / "staticfiles"



# Default primary key field type
# https://docs.djangoproject.com/en/5.1/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

Examples of Some Queries/Sample Data:

```
mysql> select * from database_student JOIN database_department;
+-----------+------------+---------------+------+------------+-------------------+------------+-----------+---
---+---------+
| id        | name       | course        | cgpa | dob        | email             | department | dept_name | ho
d  | dept_id |
+-----------+------------+---------------+------+------------+-------------------+------------+-----------+---
---+---------+
| 235805385 | Ram        | CSE           | 9.23 | 2005-10-20 | ramgotu@gmail.com | MIT        | DLHS      | MI
   |     456 |
| 235805384 | Salah      | AI            | 8.10 | 2025-03-20 | salah@gmail.com   | MIT        | DLHS      | MI
   |     456 |
| 235805383 | pedro      | ECE           | 8.50 | 2025-03-07 | perdro7@gmail.com | MIT        | DLHS      | MI
   |     456 |
| 235805379 | Juicwrld   | Cybersecurity | 9.50 | 2025-03-24 | jw@gmail.com      | MIT        | DLHS      | MI
   |     456 |
| 235805378 | Dev Srijit | CSE           | 9.99 | 2005-07-19 | dev2607@gmail.com | MIT        | DLHS      | MI
   |     456 |
| 235805385 | Ram        | CSE           | 9.23 | 2005-10-20 | ramgotu@gmail.com | MIT        | DOC       | LS
G  |     357 |
| 235805384 | Salah      | AI            | 8.10 | 2025-03-20 | salah@gmail.com   | MIT        | DOC       | LS
```

```
mysql> select * from database_attendance;
+----+------------+---------+------------+
| id | date       | present | student_id |
+----+------------+---------+------------+
|  1 | 2025-03-30 |       1 |  235805378 |
|  2 | 2025-03-30 |       1 |  235805379 |
|  5 | 2025-03-30 |       1 |  235805383 |
| 37 | 2025-03-30 |       1 |  235805384 |
| 50 | 2025-04-01 |       1 |  235805378 |
| 51 | 2025-04-01 |       1 |  235805379 |
| 54 | 2025-04-01 |       1 |  235805383 |
| 55 | 2025-04-01 |       1 |  235805384 |
| 68 | 2025-04-02 |       0 |  235805378 |
| 69 | 2025-04-02 |       1 |  235805379 |
```

Conclusion:

The **Student Database Management System** is a robust and efficient solution for managing student-related data, attendance tracking, and administrative tasks. The project leverages Django's built-in authentication system and relational database design to ensure scalability, security, and ease of use.

References:

- Django Documentation- https://docs.djangoproject.com/en/5.2/
- Stack Overflow - https://stackoverflow.com/questions
- Font awesome- https://fontawesome.com/
- W3Schools- https://www.w3schools.com/MySQL/default.asp
- TextBooks- Abraham Silberschatz, Henry F. Korth, S. Sudarshan - Database System Concepts, 6th Edition -McGraw-Hill (2010)

****

Project Guide- Dr.Manasa CM