Project Report on

<u>STUDENT INFORMATION SYSTEM</u>

Submitted by

KRITTIKA DUTTA (12022002016046) AIML-17

SOUMYA PACHAL (12022002016069) AIML-47

Under the supervision and guidance of

Mr (Dr.) Deepsubhra Guha Roy

ASSOCIATE PROFESSOR IEM

&

Mrs Bipasha Mahato

ASSISTANT PROFESSOR IEM

In the partial fulfilment of requirements for the course project in

Database Management System (DBMS)

Batch 2022-2026

Submitted to the

Department of Computer Science and Engineering

(Artificial Intelligence and Machine Learning)

Institute of Engineering & Management

SECTOR – V, KOLKATA - 700091

DECLARATION

We, Krittika Dutta and Soumya Pachal, students of the Department of Artificial Intelligence and Machine Learning (AIML), Batch 2022-2026, at the Institute of Engineering & Management, Kolkata, hereby declare that the work presented in this Database Management System (DBMS) Lab Project report on "Student Information System" is an original work completed under the guidance of Prof. Dr. Deepsubhra Guha Roy and Mrs. Bipasha Mahato. We affirm that this project is a partial fulfilment of the requirements for the Database Management System course.

STUDENT SIGNATURE:

Name: **Krittika Dutta**

Name: **Soumya Pachal**

ABSTRACT

The **Student Information System** (SIS) is a web application designed to streamline and simplify the management of academic data for educational institutions. Built with Flask, SQLite, HTML, and Tailwind CSS, this system provides separate, role-based dashboards for administrators, teachers, and students.

For administrators, the system offers a centralized platform to enroll new students and teachers, manage user records, and maintain academic data with ease. Teachers can use their dashboard to access their personal information and efficiently upload student grades. Students, in turn, are given access to a personal profile page where they can view their information and grades.

By utilizing Tailwind CSS, the interface maintains a clean and responsive design, ensuring a user-friendly experience across devices. The system's database structure, managed by SQLAlchemy and SQLite, securely stores information while facilitating quick data retrieval. This project not only reduces paperwork and administrative workload but also enhances transparency and accessibility for students and teachers alike.

In essence, this Student Information System is a scalable, intuitive solution that serves as a digital backbone for managing educational data efficiently and securely.

ACKNOWLEDGEMENT

Krittika Dutta

Soumya Pachal

B.Tech CSE(AIML)

TABLE OF CONTENT

CHAPTER 1: INTRODUCTION

1.1 Overview

In educational institutions, managing and accessing student information can be a complex and time-consuming task. With traditional systems, administrators, teachers, and students often face challenges related to data organization, grade tracking, and user roles. The **Student Information System** (SIS) is designed to address these issues by providing a streamlined, centralized digital platform for managing academic data.

This system supports multiple user roles—administrators, teachers, and students—each with access to tailored dashboards and functionalities. Administrators can enroll new students and teachers, update records, and maintain an organized academic database. Teachers benefit from a dedicated space to upload and manage student grades, making it easy to handle grading responsibilities. Students, meanwhile, are empowered with quick access to their academic profiles and grades, enhancing transparency and ownership over their educational progress.

Built using Flask for the backend and SQLite for the database, this project ensures secure data handling while maintaining efficient performance. Tailwind CSS is used to create a responsive and visually appealing front end, which ensures that users experience a modern and intuitive interface.

The Student Information System aims to be a practical solution for schools, colleges, and universities, reducing paperwork and administrative burden. It enhances collaboration between administrators and teachers, while giving students a clear, accessible way to view their academic progress. This project represents a significant step toward making educational management more efficient, accessible, and user-centered.

1.2 Objectives

The primary objectives of the **Student Information System (SIS)** are as follows:

- **Centralize Student Data Management**
  Provide a unified platform for securely managing student data, including personal information, course enrollments, and grades, to reduce redundancy and simplify data handling across the institution.

- **Role-Based Access and Permissions**
  Develop a secure, role-based login system to provide administrators, teachers, and students with access to functionalities relevant to their roles. This ensures data confidentiality and minimizes errors due to unauthorized access.

- **Streamline Enrollment and Academic Record-Keeping**
  Enable administrators to enroll new students and teachers easily, manage academic records, and make updates as needed, reducing the time spent on manual record-keeping.

- **Enhance Teacher Workflows for Grading**
  Allow teachers to upload, view, and manage student grades efficiently, thus reducing grading time and ensuring that students receive timely feedback on their academic performance.

- **Improve Student Access to Academic Information**
  Provide students with quick and secure access to their academic profiles, including course details, grades, and personal information, promoting transparency and accountability.

- **Create a User-Friendly Interface**
  Design a responsive and intuitive user interface with Tailwind CSS, ensuring ease of navigation and a positive user experience across devices.

- **Build a Scalable and Maintainable System**
  Develop the application using Flask and SQLite in a way that supports future scalability, whether by integrating with a larger database, expanding features, or accommodating more users.

- **Promote Data Security and Privacy**
  Implement secure data-handling practices, including user authentication and input validation, to safeguard sensitive student information and comply with data protection standards.

These objectives aim to create a functional, user-centered Student Information System that enhances the operational efficiency of educational institutions and improves the overall experience for all users involved.

CHAPTER 2: LITRATURE REVIEW

2.1 Introduction

Student Information Systems (SIS) are software applications designed to manage and store student data in educational institutions. These systems facilitate the management of student

records, including academic performance, personal information, and course enrollment. The objective of SIS is to improve operational efficiency, support decision-making, and enhance communication between students, faculty, and administrative staff.

2.1.1 Key Topics:

- **Purpose and Importance**: SIS systems provide a centralized platform for managing student data, handling course registration, grading, transcripts, attendance, and other educational processes.
- **Evolution**: Traditionally, SIS were standalone systems used for basic record-keeping. With the growth of digital technologies, modern SIS are now cloud-based and integrated with other systems, such as Learning Management Systems (LMS), to enhance functionality.

2.1.2 Technological Advancements

The development of SIS has seen significant advancements with the integration of emerging technologies, including cloud computing, artificial intelligence, and big data analytics.

- **Cloud-Based Systems**: Cloud computing has made SIS more scalable and accessible, providing real-time access to student data. Cloud-based SIS can be accessed remotely, reducing costs associated with maintenance and data storage (Al-Nashash & El-Khatib, 2018).
- **Mobile Integration**: Mobile applications for SIS are becoming increasingly popular, enabling students and faculty to access information from smartphones and tablets. These apps allow for real-time notifications on grades, attendance, and course schedules (Schreiber et al., 2019).
- **Artificial Intelligence**: AI algorithms are used to predict student performance, personalize learning paths, and automate administrative tasks, enhancing the overall student experience (Pérez et al., 2021).

2.1.3 Key Features of a Student Information System

A robust SIS offers multiple functionalities, which can be categorized into:

- **Student Enrollment and Registration**: Handles course selection, scheduling, and enrollment processes, helping students plan their academic year efficiently.
- **Academic Records**: Tracks grades, transcripts, and academic history to provide insights into students' performance.
- **Attendance Tracking**: Allows teachers and administrators to monitor and manage student attendance, with real-time reporting.
- **Communication Tools**: Facilitates communication between students, teachers, and administrators through messaging, notifications, and forums.
- **Data Analytics**: Enables educational institutions to analyze performance metrics, drop-out rates, and other important factors affecting student success.

2.1.4 Challenges in Implementing SIS

Despite the advantages, implementing an SIS in educational institutions comes with challenges:

- **Data kSecurity and Privacy**: Protecting sensitive student information is crucial, especially in the wake of increasing cyberattacks and data breaches. Encryption, access control, and secure cloud storage are necessary for safeguarding data (Feng & Lee, 2020).
- **Integration with Legacy Systems**: Many institutions continue to use older, less flexible systems that make it difficult to integrate new SIS features. This lack of interoperability often leads to inefficiencies and data discrepancies (Johnston & Clark, 2016).
- **User Adoption**: Faculty, staff, and students may resist using a new system, especially if it requires significant changes to existing workflows. Ensuring proper training and support is essential for a successful SIS rollout.

2.1.5 Benefits of SIS

- **Efficiency and Automation**: SIS automates administrative tasks, reducing the workload on staff and freeing up time for more strategic activities.
- **Improved Decision Making**: Data collected by SIS systems can be used to generate reports that assist in decision-making processes at various levels of the institution (Zhang et al., 2020).
- **Personalized Learning**: By analysing student data, institutions can tailor educational offerings to meet individual needs and improve student engagement.

2.1.6 Case Studies and Examples

- **University of Phoenix**: The University of Phoenix implemented a cloud-based SIS that allowed students to register for courses, check grades, and access academic resources online. The system has improved operational efficiency and reduced administrative costs (Johnson, 2017).
- **University of California System**: The University of California has implemented a modern SIS that integrates with various platforms, including learning management systems, providing students and faculty a seamless experience across various services (Williams et al., 2018).

2.1.7 Future Trends and Research Directions

- **Integration with Learning Management Systems (LMS)**: The integration of SIS with LMS will provide a unified platform for managing both administrative and academic aspects of student life.
- **Predictive Analytics and Early Warning Systems**: Research into predictive analytics aims to identify students at risk of underperforming or dropping out, allowing institutions to provide timely interventions (Yadav & Kaur, 2021).
- **Blockchain for Student Records**: Some research suggests using blockchain technology for managing academic records securely. Blockchain can ensure data integrity, enhance security, and reduce fraud in student data (Zhang & Yang, 2022).
- **AI-Powered Chatbots**: The use of AI-based chatbots for administrative support is growing. These chatbots can answer student queries, handle scheduling, and provide information about grades and courses, improving student satisfaction (Sutherland et al., 2020).

2.1.8 Conclusion

The Student Information System plays a crucial role in modern educational institutions by streamlining administrative functions and enhancing student experiences. As technology continues to evolve, SIS systems are expected to become more integrated, intelligent, and user-friendly, helping institutions provide personalized education and improved student outcomes.

CHAPTER 3: SYSTEM DESCRIPTION

3.1 Working Principle

The **Student Information System (SIS)** is a web-based application designed to centralize and simplify the management of academic records within educational institutions. The system is built with Flask for the backend, SQLite as the database, and Tailwind CSS for the frontend, providing a secure, responsive, and scalable solution. The core components of the system include user role-based dashboards for administrators, teachers, and students, each with distinct functionalities and access permissions.

3.1.1 Key Components

1. **User Authentication and Authorization**
- The system offers a secure login mechanism, where users select their role—administrator, teacher, or student—upon login. Flask sessions are used to manage authentication and user access across the platform.
- Role-based access control ensures that each user type has appropriate permissions. For instance, administrators have access to all system features, teachers can manage only their classes and grades, and students can view their profiles and grades.

2. **Admin Dashboard**
- The administrator dashboard is the central management hub of the system, allowing administrators to enroll new students and teachers by providing details like name, ID, course, address, and date of birth.
- Admins can also view, update, and delete records in the database, enabling them to maintain accurate and up-to-date academic information.
- The admin role includes permission to assign courses to teachers and oversee the grading data to ensure accuracy and completeness.

3. **Teacher Dashboard**
- Teachers have a dashboard tailored to their needs, allowing them to view their information, manage course assignments, and upload grades for students.
- The grade upload feature requires teachers to input a student ID and grade, which is then stored in the database and made available to the student.
- The teacher role is designed to simplify the grading process and give teachers an easy, organized way to manage their classes.

4. **Student Dashboard**
- Students have a personal dashboard where they can view their academic profiles, including enrolled courses, grades, and personal information.

- This role promotes transparency by providing students with a self-service portal to track their academic progress and identify areas for improvement.
- The student dashboard is read-only, ensuring that students cannot modify their academic data.

5. **Database Structure and Management**
- The system uses SQLite as the backend database to store user profiles, enrollment details, courses, and grades. SQLAlchemy, Flask's ORM, is employed for data modeling and handling complex database operations.
- The database includes tables for users (separately defining administrators, teachers, and students), courses, enrollments, and grades. Relationships between these tables allow for seamless data retrieval and integration across user roles.

6. **User Interface Design**
- Tailwind CSS is utilized for a clean, modern, and responsive design. The user interface is intuitive, with hover effects, dropdown menus, and forms designed for easy data entry.
- Each dashboard has a visually distinct look, aligned with its functionalities. Administrators see a comprehensive view of system management tools, while students and teachers have focused views tailored to their needs.

7. **Data Security and Privacy**
- The application follows secure data-handling practices, including input validation, sanitized database queries, and session-based access control.
- User passwords are securely stored using hashing techniques, ensuring that sensitive data remains protected from unauthorized access.

8. **Scalability and Future Growth**
- The SIS is built with scalability in mind, allowing for future expansion to support additional features like attendance tracking, report generation, and analytics.
- The system is also capable of integrating with other educational platforms if needed, providing flexibility for institutions to adapt and grow their digital infrastructure.

3.2 System Flow

The system begins with a login page where users select their role and enter their credentials. Upon successful authentication, users are redirected to their respective dashboards based on their roles:

- **Administrators** can access all functionalities, including user enrollment, course management, and oversight of academic records.
- **Teachers** have access to their courses and grading tools, allowing them to view student lists and upload grades.
- **Students** have a simple, read-only view of their profiles and grades.

Each interaction with the system updates the database accordingly, ensuring that administrators, teachers, and students have access to the latest data in real time. This role-based design fosters an organized, accessible, and efficient environment for academic data management.

Systems are designed keeping in mind an issue that is to be solved. Every system is designed in its unique keeping in mind the requirement of the problem or the issue. Our system solves the problem of searching for the good that the customer's needs.

System design involves the design of overall architecture, based on which we design components, modules and interfaces. The beginning of any system architecture is by decomposing it into smaller fragments. Decomposition and binding of components makes the architecture easy to understand and makes it easier to interpret.
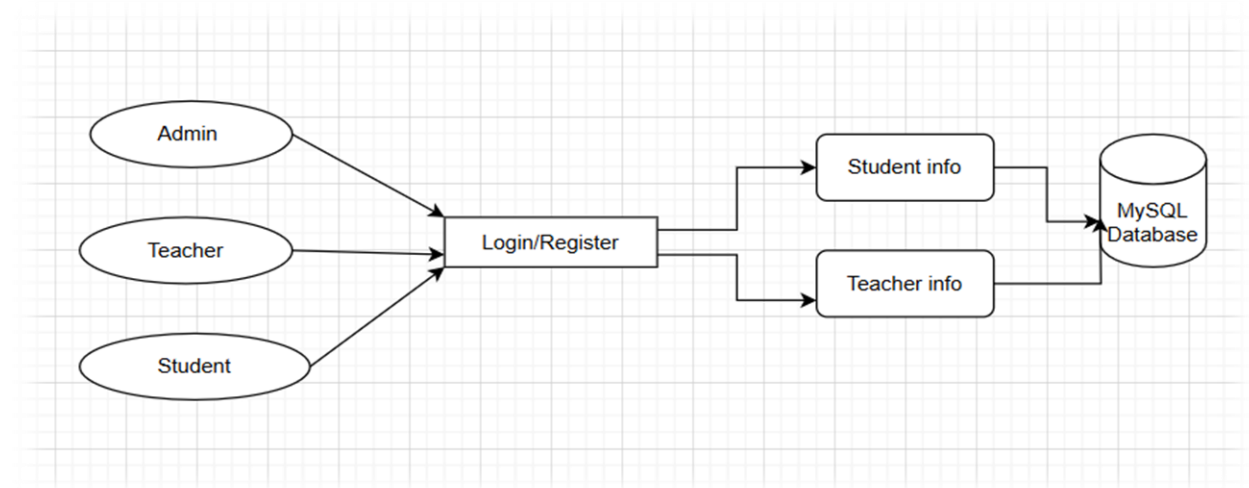
3.3 Architecture Diagram

Fig: Flow Diagram

Flow diagram is a graphic representation of the physical route or flow of people, materials, paper works, vehicles, or communication associated with a process, procedure plan, or investigation. In the second definition the meaning is limited to the representation of the physical route or flow. Here the user ask permission to login and after login the user can search or view.

CHAPTER 4: THEORETICAL ANALYSIS

4.1 Introduction to Tools

4.1.1 Introduction to HTML

Hyper-Text Mark-up Language (HTML) is a simple mark-up system used to create hypertext documents that are portable from one platform to another. HTML documents are SGML documents with generic semantics that are appropriate for representing information from a wide range of applications. HTML mark-up can represent hypertext news, mail, documentation, and hypermedia; menus of options; database query results; simple structured documents with in-lined graphics; and hypertext views of existing bodies of information.

4.1.1.1 Advantages of HTML

- It is widely used.

- Every browser supports HTML language.
- Easy to learn and use.
- It is by default in every window so we don't need to purchase extra software.

4.1.2 Introduction to CSS

Cascading Style Sheets, fondly referred to as CSS, is a simply designed language intended to simplify the process of making web pages presentable. CSS allows you to apply styles to web pages. More importantly, CSS enables you to do this independent of the HTML that makes up each web page.

4.1.2.1 Advantages of CSS

- Greater consistency in design.
- Ease of presenting different styles to different viewers.

4.1.2.2 Friendly Environment

Creating a form, adding controls to form and writing code behind the form are all managed within a friendly Environment.

For Example: This is a sample code of html which is used to make a login form.

```html
<!-- templates/login.html -->

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Login</title>

    <link href="./output.css" rel="stylesheet">

</head>

<body class="bg-blue-500 h-screen flex items-center justify-center">

    <div class="bg-white p--8 rounded-lg shadow-lg w-96">

        <h2 class="text-4xl font-semibold text-center mb-4">Student Information System</h2>

        <form action="./dashboard.html" method="POST">

            <div class="mb-4">
```

```html
        <label for="username" class="block text-gray-700">Username</label>

        <input type="text" id="username" name="username" class="w-full p-2 border
rounded" required>

    </div>

    <div class="mb-4">

        <label for="password" class="block text-gray-700">Password</label>

        <input type="password" id="password" name="password" class="w-full p-2 border
rounded" required>

    </div>

    <div class="mb-4">

        <label for="role" class="block text-gray-700">Role</label>

        <select name="role" id="role" class="w-full p-2 border rounded">

            <option value="admin">Admin</option>

            <option value="teacher">Teacher</option>

            <option value="student">Student</option>

        </select>

    </div>

    <div class="flex justify-center">

        <button type="submit" class="bg-blue-500 text-white py-2 px-6 rounded hover:bg-
blue-700">Login</button>

    </div>

</form>

</div>

</body>

</html>
```

4.1.3 Introduction to Flask

Flask is a lightweight and versatile web framework for Python, widely used to develop web applications due to its simplicity and flexibility. Introduced by Armin Ronacher in 2010, Flask follows the **WSGI** (Web Server Gateway Interface) standard and adopts the **Werkzeug** library for routing and request handling, as well as **Jinja2** for templating. As a **micro-framework**, Flask is minimalistic by design, providing only the core tools needed for web development and allowing developers to add extensions for specific functionality as required. This approach makes Flask particularly popular among developers who appreciate its "build-as-you-grow" philosophy, which avoids imposing excessive dependencies and allows them to maintain control over application architecture.

Flask's simplicity makes it an ideal choice for small to medium-sized applications, prototypes, and even certain production-grade applications, especially where customizability is valued. Despite its lightweight nature, Flask is highly extensible and can accommodate complex features through plugins and integrations, making it a powerful choice for a wide range of projects.

4.1.3.1 Advantages of Flask

- **Lightweight & Flexible** – Provides a minimal core with selective extensions for custom builds.
- **Easy to Learn** – Beginner-friendly with simple syntax and comprehensive documentation.
- **Modular & Extensible** – Supports plugins for ORM, authentication, and forms, making it highly customizable.
- **Built-in Debugging** – Includes a development server and interactive debugger for quick testing.
- **RESTful Routing** – Native support for creating APIs and web services.
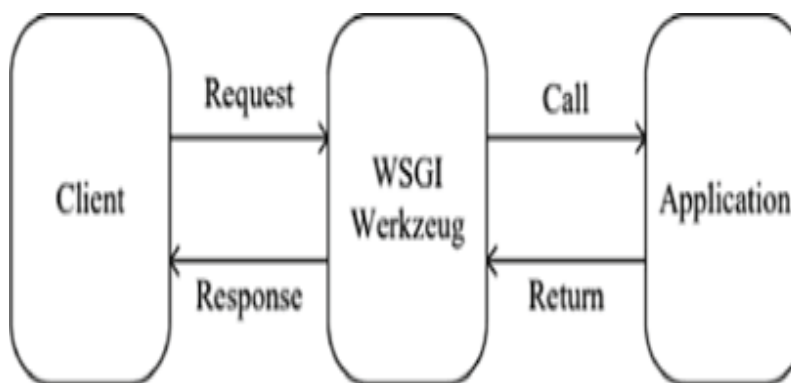

Fig 4.1: Working In Flask

4.1.3.2 Working with Flask

from flask import Flask, render_template, redirect, url_for, request, flash

from flask_sqlalchemy import SQLAlchemy

from models import db, User, Student, Teacher, Grade

```python
from config import Config


# Initialize Flask app

app = Flask(__name__)

app.config.from_object(Config)


# Initialize the db

db.init_app(app)


# Create the tables before the first request

@app.before_first_request

def create_tables():

    with app.app_context():

        db.create_all()


@app.route('/')

def index():

    return render_template('login.html')


# Login route for admin, teacher, and student

@app.route('/login', methods=['POST', 'GET'])

def login():

    if request.method == 'POST':

        username = request.form['username']

        password = request.form['password']

        role = request.form['role']
```

```python
    user = User.query.filter_by(username=username, role=role).first()

    if user:
        return redirect(url_for(f'{role}_dashboard', user_id=user.id))  # Redirect based on role
    else:
        flash('Invalid credentials', 'danger')
        return redirect(url_for('login'))
    return render_template('login.html')


# Admin Dashboard
@app.route('/admin_dashboard/<int:user_id>')
def admin_dashboard(user_id):
    return render_template('admin_dashboard.html', user_id=user_id)


# Teacher Dashboard
@app.route('/teacher_dashboard/<int:user_id>')
def teacher_dashboard(user_id):
    teacher = Teacher.query.get(user_id)
    return render_template('teacher_dashboard.html', teacher=teacher)


# Student Dashboard
@app.route('/student_dashboard/<int:user_id>')
def student_dashboard(user_id):
    student = Student.query.get(user_id)
    grades = Grade.query.filter_by(student_id=user_id).all()
```

```python
    return render_template('student_dashboard.html', student=student, grades=grades)


# Enroll new student or teacher (Admin only)
@app.route('/enroll', methods=['POST', 'GET'])
def enroll():
    if request.method == 'POST':
        role = request.form['role']
        name = request.form['name']
        email = request.form['email']
        username = request.form['username']
        password = request.form['password']

        user = User(username=username, email=email, role=role)
        db.session.add(user)
        db.session.commit()


        if role == 'student':
            student = Student(name=name)
            db.session.add(student)
            db.session.commit()
        elif role == 'teacher':
            teacher = Teacher(name=name)
            db.session.add(teacher)
            db.session.commit()


        flash('New user enrolled!', 'success')
```

```python
        return redirect(url_for('admin_dashboard', user_id=user.id))

    return render_template('enroll.html')


if __name__ == '__main__':

    app.run(debug=True)
```

4.1.3.3 Connecting Flask with MySQL Database

```python
from flask_sqlalchemy import SQLAlchemy

# Create db instance
db = SQLAlchemy()

# User model (Base class for Teacher and Student)
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    role = db.Column(db.String(10), nullable=False)  # 'student', 'teacher', 'admin'

class Student(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100))
    dob = db.Column(db.Date)
    address = db.Column(db.String(200))

class Teacher(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100))
    subject = db.Column(db.String(100))

class Grade(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    student_id = db.Column(db.Integer, db.ForeignKey('student.id'), nullable=False)
    teacher_id = db.Column(db.Integer, db.ForeignKey('teacher.id'), nullable=False)
    marks = db.Column(db.Float)
```

4.1.4 Introduction to MySQL

MySQL is a Relational Database Management System (RDBMS) that runs as a server providing multi-user access to a number of databases. MySQL is pronounced ("My S-Q-L").

MySQL development project has made its source available under the terms of General Public License. MySQL is owned and sponsored by a single for profit firm, the Swedish company MySQL AB, now owned by Sun Microsystem, a subsidiary of Oracle Corporation.

MySQL works on many different system platforms including AIX, BSD i, FreeBSD, HP-UX, i5/OS, Linux, Mac OS X, Net BSD, Novell NetWare, Open BSD, Open Solaris, e com Station, OS/2 Wrap, QNX, IRIX, Solaris, Symbian, SunOS, SCO Open Server, SCO Unix Ware, Sanos, Tru64 and Microsoft Windows. A port of MySQL to Open VMS also exits. All major programming languages with language-specific APIs include Libraries for accessing MySQL database. In addition, an ODBC interface called MYODBC allows additional programming languages that supports the ODBC interface to communicate with a MySQL database, such as ASP or ColdFusion. MySQL server and official libraries are mostly implemented in ANSI C/ ANCI C++.

CHAPTER 5: METHODOLOGY

5.1 Methodology

The development of the Student Information System (SIS) followed a systematic approach to ensure the system meets the needs of different users—administrators, teachers, and students. This methodology outlines the steps taken to design, develop, test, and deploy the system.

5.1.1 Requirement Analysis

- **Goal:** Identify the system requirements and define the functionalities needed by each user role.
- **Process:** Conducted discussions and drafted requirements for each feature. Key functionalities included user authentication, student enrollment, profile viewing, and grade management.
- **Output:** A list of essential features categorized by user role (Admin, Teacher, Student).

5.1.2 System Design

- **Goal:** Outline the architecture, database schema, and user interface (UI) layout.
- **Process:** Created ER diagrams for the SQL database and designed wireframes for the UI components. Defined models for `User`, `Student`, `Teacher`, and `Grade` with Flask SQLAlchemy.
- **Output:** A blueprint for database relationships, routes, and UI structure.

5.1.3 Front-End Development

- **Goal:** Build an interactive, responsive user interface for users to access the system.
- **Process:** Implemented HTML templates for the login, dashboard, and form pages with Tailwind CSS for a modern, clean design. Incorporated interactive elements with Tailwind's hover effects for user engagement.
- **Output:** A fully responsive front end with an easy-to-navigate UI for each user type.

5.1.4 Back-End Development

- **Goal:** Develop the server-side logic to manage authentication, data storage, and role-based access.
- **Process:** Utilized Flask for routing and session management, Flask SQLAlchemy for database interactions, and Flask-Login for secure authentication. Implemented API endpoints for enrolling users, viewing profiles, and updating grades.
- **Output:** A RESTful backend with robust session handling and CRUD functionality.

5.1.5 Database Integration

- **Goal:** Ensure secure, structured storage of user, student, teacher, and grade information.
- **Process:** Designed and created a relational database using SQL, integrating it with Flask SQLAlchemy for ORM support. Established secure connections and implemented relationships between tables.
- **Output:** A normalized database schema with efficient data retrieval and storage.

5.1.6 Testing

- **Goal:** Validate the functionality, performance, and security of the system.
- **Process:** Conducted unit tests on individual routes and functionality, and performed integration testing to verify end-to-end workflows. Tested role-based access control to ensure users can only access their authorized features.
- **Output:** A thoroughly tested system with resolved bugs and validated access control.

5.1.7 Deployment and Documentation
- **Goal:** Deploy the system for real-world use and provide comprehensive documentation for users and developers.
- **Process:** Deployed the Flask app on a local server environment, with potential for cloud deployment. Created a README and user guide detailing setup instructions and usage.
- **Output:** A live, operational Student Information System, with documentation for ease of use and further development.

CHAPTER 6: RESULT AND DISCUSSION

6.1 Code for Admin Login Page

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <link href="./output.css" rel="stylesheet">
</head>
```

```html
<div class="p-8 bg-gray-50 rounded-lg shadow-md">
    <h2 class="text-3xl font-semibold text-center text-gray-800">Admin Dashboard</h2>
    <p class="text-lg text-gray-600 mt-4 text-center">Welcome, Admin!</p>

    <div class="mt-8">
        <h3 class="text-2xl font-medium text-gray-800 mb-4">Enroll New User</h3>
        <form action="{{ url_for('enroll') }}" method="POST" class="space-y-6">
            <div class="flex flex-col space-y-4">
                <div class="flex flex-col">
                    <label for="name" class="text-lg font-semibold text-gray-700">Name</label>
                    <input type="text" name="name" id="name" placeholder="Enter the user's full name"
                        class="p-3 border border-gray-300 rounded-md focus:ring-2 focus:ring-blue-500 focus:outline-none" required>
                </div>

                <div class="flex flex-col">
                    <label for="email" class="text-lg font-semibold text-gray-700">Email</label>
                    <input type="email" name="email" id="email" placeholder="Enter the user's email"
                        class="p-3 border border-gray-300 rounded-md focus:ring-2 focus:ring-blue-500 focus:outline-none" required>
                </div>

                <div class="flex flex-col">
                    <label for="username" class="text-lg font-semibold text-gray-700">Username</label>
                    <input type="text" name="username" id="username" placeholder="Choose a username"
                        class="p-3 border border-gray-300 rounded-md focus:ring-2 focus:ring-blue-500 focus:outline-none" required>
                </div>

                <div class="flex flex-col">
                    <label for="password" class="text-lg font-semibold text-gray-700">Password</label>
                    <input type="password" name="password" id="password" placeholder="Enter the password"
                        class="p-3 border border-gray-300 rounded-md focus:ring-2 focus:ring-blue-500 focus:outline-none" required>
                </div>

                <div class="flex flex-col">
                    <label for="role" class="text-lg font-semibold text-gray-700">Role</label>
                    <select name="role" id="role" class="p-3 border border-gray-300 rounded-md focus:ring-2 focus:ring-blue-500 focus:outline-none">
```

```html
          <option value="student">Student</option>
          <option value="teacher">Teacher</option>
        </select>
      </div>

      <button type="submit"
            class="w-full bg-blue-600 text-white py-3 rounded-md text-lg font-semibold
hover:bg-blue-700 focus:outline-none focus:ring-2 focus:ring-blue-500">
          Enroll User
        </button>
      </div>
    </form>
  </div>

  <div class="mt-12">
    <h3 class="text-2xl font-medium text-gray-800 mb-4">View All Users</h3>
    <div class="overflow-x-auto shadow-lg bg-white rounded-lg">
      <table class="w-full text-left table-auto">
        <thead class="bg-blue-500 text-white">
          <tr>
            <th class="py-3 px-6">Name</th>
            <th class="py-3 px-6">Email</th>
            <th class="py-3 px-6">Role</th>
            <th class="py-3 px-6">Actions</th>
          </tr>
        </thead>
        <tbody>
          <!-- Example row, should be dynamically generated with Flask -->
          <tr>
            <td class="py-3 px-6 border-b">Krittika Dutta</td>
            <td class="py-3 px-6 border-b">kd@gmail.com</td>
            <td class="py-3 px-6 border-b">Student</td>
            <td class="py-3 px-6 border-b">
              <a href="#" class="text-blue-500 hover:text-blue-700">Edit</a>
              <span class="mx-2">|</span>
              <a href="#" class="text-red-500 hover:text-red-700">Delete</a>
            </td>
          </tr>
          <tr>
            <td class="py-3 px-6 border-b">Soumya Pachal</td>
            <td class="py-3 px-6 border-b">sp@gmail.com</td>
            <td class="py-3 px-6 border-b">Student</td>
            <td class="py-3 px-6 border-b">
              <a href="#" class="text-blue-500 hover:text-blue-700">Edit</a>
              <span class="mx-2">|</span>
```

```html
            <a href="#" class="text-red-500 hover:text-red-700">Delete</a>
          </td>
        </tr>
        <!-- Repeat rows for each user -->
      </tbody>
    </table>
  </div>
</div>
</div>
<body>

</body>
</html>
```

6.2 Code for Teacher Dashboard

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Teacher Dashboard</title>
  <link href="./output.css" rel="stylesheet">
</head>
<body class="bg-gray-100">

  <div class="min-h-screen flex flex-col items-center justify-center py--8">
    <div class="bg-white w-full max-w-3xl p-8 rounded-lg shadow-md">
      <h2 class="text-3xl font-semibold text-center text-blue-600 mb-8">Teacher
Dashboard</h2>
      <div class="mb-4">
        <h3 class="text-xl font-bold text-blue-500">Your Profile</h3>
        <p>Name: ABC </p>
        <p>Subject:  DBMS </p>
        <p>Address:  Kolkata </p>
        <p>Date of Birth: 21/9/1988</p>
      </div>
      <a href="{{ url_for('upload_marks') }}" class="block text-center py-2 px-4 bg-green-500
text-white rounded-md hover:bg-green-600 mb-4">Upload Marks</a>
      <a href="{{ url_for('dashboard') }}" class="block text-center py-2 px-4 bg-red-500 text-
white rounded-md hover:bg-red-600">Logout</a>
    </div>
  </div>
```

```
</body>
</html>
```

6.3 Code for Student Dashboard

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Student Dashboard</title>
  <link href="./output.css" rel="stylesheet">
</head>
<body class="bg-gray-100">

  <div class="min-h-screen flex flex-col items-center justify-center py-8">
    <div class="bg-white w-full max-w-3xl p-8 rounded-lg shadow-md">
      <h2 class="text-3xl font-semibold text-center text-blue-600 mb-8">Student
Dashboard</h2>
      <div class="mb-4">
        <h3 class="text-xl font--bold text-blue-500">Your Profile</h3>
        <p>Name: Soumya</p>
        <p>Course: B.tech CSE(AIML)</p>
        <p>Address: Howrah</p>
        <p>Date of Birth: DD/MM/YYYY</p>
      </div>
      <div class="mb-4">
        <h3 class="text-xl font-bold text-blue-500">Your Grades</h3>

          <p>CGPA: 9.0</p>

      </div>
      <a href="{{ url_for('dashboard') }}" class="block text-center py-2 px-4 bg-red-500 text-
white rounded-md hover:bg-red-600">Logout</a>
    </div>
  </div>

</body>
</html>
```

24

6.4 Code for connecting Database to Flask

```
from flask_sqlalchemy import SQLAlchemy

# Create db instance
db = SQLAlchemy()

# User model (Base class for Teacher and Student)
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    role = db.Column(db.String(10), nullable=False)  # 'student', 'teacher', 'admin'

class Student(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100))
    dob = db.Column(db.Date)
    address = db.Column(db.String(200))

class Teacher(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100))
    subject = db.Column(db.String(100))

class Grade(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    student_id = db.Column(db.Integer, db.ForeignKey('student.id'), nullable=False)
    teacher_id = db.Column(db.Integer, db.ForeignKey('teacher.id'), nullable=False)
    marks = db.Column(db.Float)
```

6.5 Code for Back-end with Flask

```
from flask import Flask, render_template, redirect, url_for, request, flash
from flask_sqlalchemy import SQLAlchemy
from models import db, User, Student, Teacher, Grade
from config import Config

# Initialize Flask app
app = Flask(__name__)
app.config.from_object(Config)

# Initialize the db
```

```python
db.init_app(app)

# Create the tables before the first request
@app.before_first_request
def create_tables():
    with app.app_context():
        db.create_all()

@app.route('/')
def index():
    return render_template('login.html')

# Login route for admin, teacher, and student
@app.route('/login', methods=['POST', 'GET'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        role = request.form['role']

        user = User.query.filter_by(username=username, role=role).first()

        if user:
            return redirect(url_for(f'{role}_dashboard', user_id=user.id))  # Redirect based on role
        else:
            flash('Invalid credentials', 'danger')
            return redirect(url_for('login'))
    return render_template('login.html')

# Admin Dashboard
@app.route('/admin_dashboard/<int:user_id>')
def admin_dashboard(user_id):
    return render_template('admin_dashboard.html', user_id=user_id)

# Teacher Dashboard
@app.route('/teacher_dashboard/<int:user_id>')
def teacher_dashboard(user_id):
    teacher = Teacher.query.get(user_id)
    return render_template('teacher_dashboard.html', teacher=teacher)

# Student Dashboard
@app.route('/student_dashboard/<int:user_id>')
def student_dashboard(user_id):
    student = Student.query.get(user_id)
    grades = Grade.query.filter_by(student_id=user_id).all()
```

```python
    return render_template('student_dashboard.html', student=student, grades=grades)

# Enroll new student or teacher (Admin only)
@app.route('/enroll', methods=['POST', 'GET'])
def enroll():
    if request.method == 'POST':
        role = request.form['role']
        name = request.form['name']
        email = request.form['email']
        username = request.form['username']
        password = request.form['password']

        user = User(username=username, email=email, role=role)
        db.session.add(user)
        db.session.commit()

        if role == 'student':
            student = Student(name=name)
            db.session.add(student)
            db.session.commit()
        elif role == 'teacher':
            teacher = Teacher(name=name)
            db.session.add(teacher)
            db.session.commit()

        flash('New user enrolled!', 'success')
        return redirect(url_for('admin_dashboard', user_id=user.id))
    return render_template('enroll.html')

if __name__ == '__main__':
    app.run(debug=True)
```
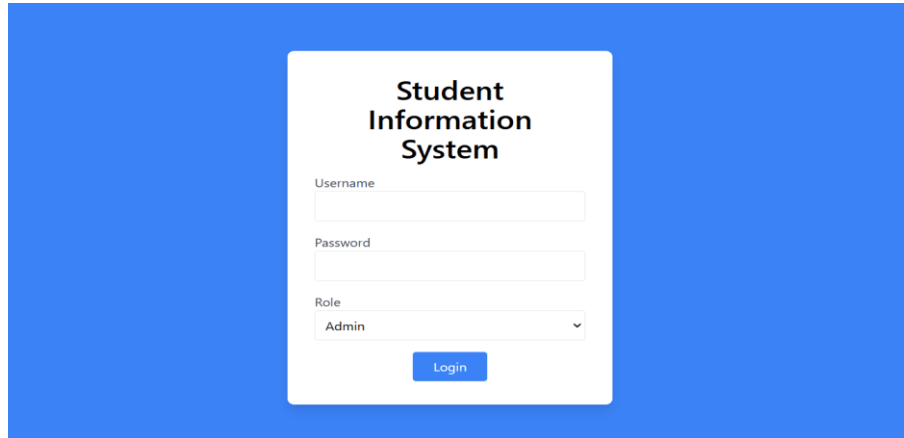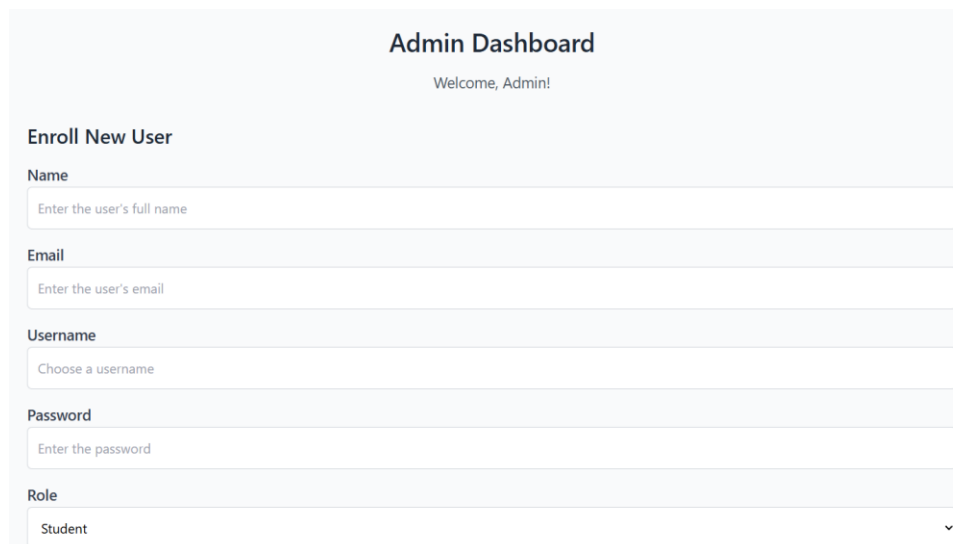
6.6 Results

Fig: Login Page

The **login page** of our web application allows three types of users—**Admin**, **Students**, and **Teachers**—to access their respective dashboards securely. Each user is required to enter their unique **username** and **password**. Based on their role, the system will redirect them to the appropriate interface:

- **Admin**: Access to the administrative panel for managing users, courses, and system settings.
- **Students**: Dashboard to view courses, grades, attendance, and personal details.
- **Teachers**: Interface to manage course content, grades, attendance, and communicate with students.

Fig: Admin Registration Page

The **Admin Registration Page** allows administrators to manage the user base by enrolling new **students** and **teachers**, as well as editing or removing existing users. The key functionalities include:

- **Enroll New Users**: Admins can input personal and academic details to register new students or teachers, including name, email, role, and password.
- **Edit User Details**: Admins can update existing user information such as contact details, course assignments, and roles.
- **Remove Users**: Admins have the option to remove students or teachers from the system entirely.

The page includes fields for user type selection (student or teacher), along with forms for required details. It ensures data validation and confirmation prompts before making any changes. Access to this page is restricted to users with Admin credentials.

Fig: Teacher Dashboard

The **Teacher Dashboard** allows teachers to manage and track student performance in their respective subjects. Key features of the page include:

- **Upload Marks**: Teachers can easily upload student marks for assignments, quizzes, exams, or overall course performance. The system supports bulk uploads via file import (e.g., CSV) or manual entry.
- **Grade Editing**: Teachers can edit individual student grades or comments if needed.

Student Dashboard

Your Profile
Name: Soumya
Course: B.tech CSE(AIML)
Address: Howrah
Date of Birth: DD/MM/YYYY
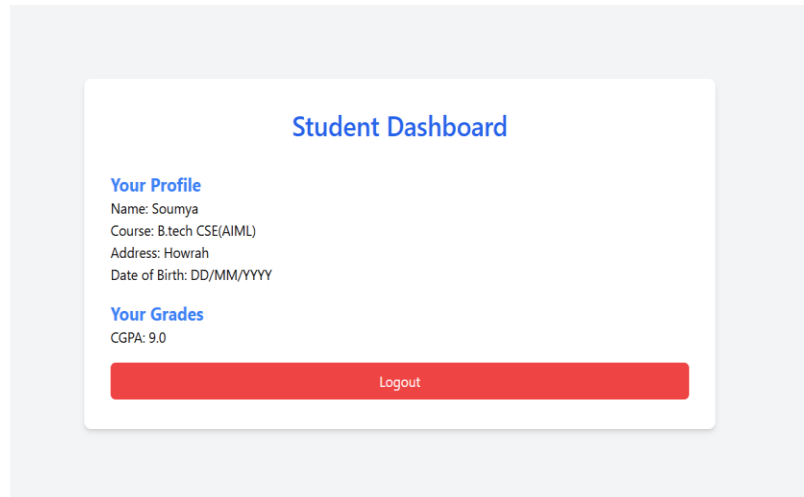
Your Grades
CGPA: 9.0

Logout

Fig: Student Dashboard

The **Student Dashboard** provides students with easy access to their personal and academic information. Key features of the page include:

- **View Personal Details**: Students can view and update their personal information, such as contact details, enrollment status, and course schedule.
- **View Marks and Grades**: Students can see their grades for assignments, exams, and overall course performance across all enrolled subjects.

Chapter 7: CONCLUSION

The Student Information System (SIS) project successfully integrates Flask, SQL, and Tailwind CSS to provide a comprehensive and user-friendly platform for managing student records, courses, grades, and user enrollments. Through systematic development, from requirements gathering to deployment, the system offers efficient role-based access, where administrators can enroll users, teachers can manage student grades, and students can view their profiles and academic progress.

This project demonstrates the power and flexibility of Flask as a backend framework, combining it with a modern front-end design powered by Tailwind CSS. The use of Flask's modularity allowed for the development of a scalable application with secure authentication and smooth database integration using SQLAlchemy.

Overall, the SIS meets the core requirements of managing user data and academic records, offering an intuitive interface and robust functionality for each user type. The system is scalable and can be extended with additional features such as notifications, reports, or advanced analytics in the

future. With further enhancements, the system has the potential to support larger educational institutions or be adapted to other domains requiring user and data management.

Chapter 8: REFERENCE

8.1 Flask Documentation

- Flask's official documentation provides comprehensive details about the framework's features, tools, and usage. It was a key resource for understanding Flask's routing, templating, and database integration.
- URL: https://flask.palletsprojects.com/

8.2 Tailwind CSS Documentation

- Tailwind CSS documentation offered guidance on creating responsive, modern UIs using utility-first CSS classes. It was used for designing the front-end of the system.
- URL: https://tailwindcss.com/docs

8.3 Flask-SQLAlchemy Documentation

- The Flask-SQLAlchemy extension for Flask simplifies database interactions. The documentation was referenced to integrate the SQLAlchemy ORM with Flask for efficient database management.
- URL: https://flask-sqlalchemy.palletsprojects.com/

8.4 MDN Web Docs – HTML & CSS

- Mozilla's web docs were used for understanding fundamental HTML and CSS concepts, which helped in structuring the user interface and styling the application.
- URL: https://developer.mozilla.org/en-US/docs/Web

8.5 YouTube – Flask Tutorials

Various YouTube tutorials were followed to gain practical insights into Flask, database integration, and the implementation of user authentication and authorization. These videos served as additional resources for learning and debugging.

8.6 Stack Overflow

- Stack Overflow was an essential reference for solving common issues and finding solutions for common problems related to Flask, SQLAlchemy, and Tailwind CSS.
- URL: https://stackoverflow.com/