CS2008 Database Systems

Academics Hall and Slot Management System

Department of Computer Science and Engineering

Faculty : Dr. Sivaselvan B

May 2024

Roll Number	
CS22B1054	**K Pranay Sai**
CS22B1093	**Rohan G**
CS22B1095	**R Sai Charish**
CS22B1097	**G Vivek Reddy**

Contents

1 Introduction **6**

2 Scope **6**

3 Functional Requirements **6**

3.1 Authentication and Authorization 6

3.2 Course Management 6

3.3 Hall Management 7

3.4 Slot Management 7

3.5 Instructor Requests 7

3.6 Viewing Schedules 7

4 Non-Functional Requirements **8**

4.1 Performance 8

4.2 Security 8

4.3 User Interface 8

5 System Interfaces **8**

6 Glossary **9**

7 Libraries and Frameworks **12**

8 User Login and Authentication System **12**

8.1 Purpose 12

8.2 Implementation 12

8.3 Authentication Process 12

8.4 Session Management 13

8.5 Access Control 13

9 Database Interaction **13**

9.1 Database Choice 13

9.2 Functionality 14

9.3 Security Measures 14

10 Routes and Templates **14**

10.1 Route Definition 14

10.1.1 Examples 15

10.2 Dynamic Content Generation 15

10.3 Template Rendering 15

10.4 Data Passing and Context 15

11 Technology Stack **16**

11.1 Programming Language 16

11.2 Web Framework 16

11.3 Database 16

11.4 Database Connector 16

11.5 Template Engine 17

11.6 User Authentication 17

12 Data Flow **19**

12.1 User Interaction 19

12.2 Frontend-Backend Communication 19

12.3 Backend Processing 19

12.3.1 Request Routing 19

12.3.2 Authentication 19

12.3.3 Data Manipulation 19

12.3.4 Business Logic Execution 20

12.4 Response Generation 20

12.5 Response Delivery 20
12.6 Frontend Update 20

13 Connecting Frontend and Backend **22**
13.1 Forms 22
13.1.1 HTTP Methods 22
13.1.2 Action Attribute 22
13.1.3 Form Elements 22
13.2 Backend Route Handling 23
13.3 Response Generation and Feedback 23

14 Database and Schema **24**
14.1 Course Table 24
14.2 Enrolled Table 24
14.3 Halls Table 25
14.4 Instructor Table 25
14.5 Instructor Request Table 26
14.6 Slot Table 26
14.7 Student Table 27

15 Source Code **28**

16 Frontend Screenshots **46**
16.1 Home Page 46
16.2 Admin Login 46
16.3 Admin Dashboard 47
16.3.1 Manage Student 47
16.3.2 Manage Instructors 49
16.3.3 Manage Courses 50

16.3.4 Manage Slots 52
16.3.5 Manage Halls 53
16.4 Student Dashboard 54
16.5 Faculty Dashboard 55

17 Conclusion **57**

1 Introduction

The Academic Slot and Hall Management System is a web-based application designed to facilitate the management of academic slots and halls within an educational institution. This system aims to streamline the scheduling of courses, allocation of halls, and management of instructor requests for specific halls and slots.

2 Scope

The system will provide functionalities for administrators, instructors, and students. Administrators will have full control over system configurations and user management. Instructors will be able to schedule courses, request specific halls and slots, and manage course details. Students will have access to view course schedules and hall allocations.

3 Functional Requirements

3.1 Authentication and Authorization

- Users must authenticate themselves with valid credentials (username and password) to access the system.
- The system shall differentiate between administrators, instructors, and students and provide appropriate access privileges.
- Administrators shall have the ability to create, update, and delete user accounts.

3.2 Course Management

- Instructors shall be able to add, update, and delete course details including course name, department, semester, and credit hours.

- Courses shall be assigned to instructors, linking each course to its respective instructor.

3.3 Hall Management

- Administrators shall have the ability to add, update, and delete hall details including hall name, capacity, and location.
- Halls shall be allocated to courses, specifying the hall where each course will be conducted.

3.4 Slot Management

- Administrators shall define slots by specifying days and times.
- Slots shall be allocated to courses, indicating the time slots for each course.

3.5 Instructor Requests

- Instructors shall be able to request specific halls and slots for their courses.
- Each request shall include the instructor ID, course ID, preferred hall, preferred slot, and status (pending/accepted/rejected).
- Administrators shall have the authority to accept or reject instructor requests.

3.6 Viewing Schedules

- Instructors shall be able to view their course schedules, including assigned halls and slots.
- Students shall be able to view course schedules, indicating the halls and slots where each course will be conducted.

4 Non-Functional Requirements

4.1 Performance

- The system shall handle concurrent user requests efficiently, ensuring responsiveness and minimal downtime.
- Response times for common operations (e.g., login, scheduling) shall be within acceptable limits.

4.2 Security

- User authentication shall be implemented securely, utilizing encryption techniques to protect sensitive information.
- Access control mechanisms shall be in place to prevent unauthorized access to system functionalities and data.

4.3 User Interface

- The user interface shall be intuitive and user-friendly, providing clear navigation and feedback.
- The system shall support multiple device types and screen sizes, ensuring compatibility with various browsers.

5 System Interfaces

- The system shall be accessible via a web-based interface, compatible with commonly used web browsers such as Chrome, Firefox, and Safari.

- The backend of the system shall be implemented using a relational database management system (RDBMS) to store and manage data.

6 Glossary

- Instructor: A teaching staff member responsible for delivering courses.
- Student: An individual enrolled in academic programs.
- Administrator: A user with administrative privileges responsible for managing system configurations and user accounts.
- Course: A unit of study offered by the institution.
- Hall: A physical location within the institution where courses can be conducted.
- Slot: A predefined time interval during which courses can be scheduled.

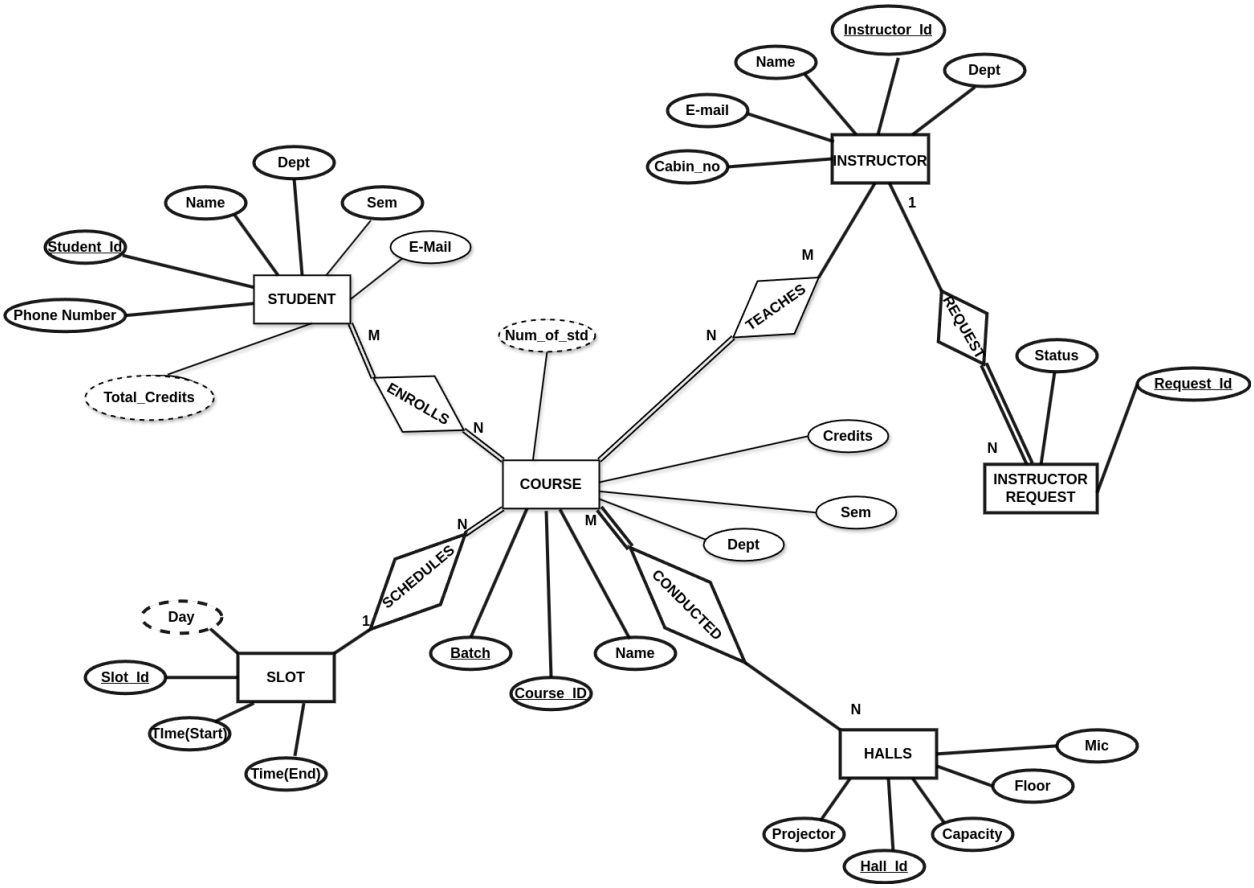

Figure 1: ER Diagram

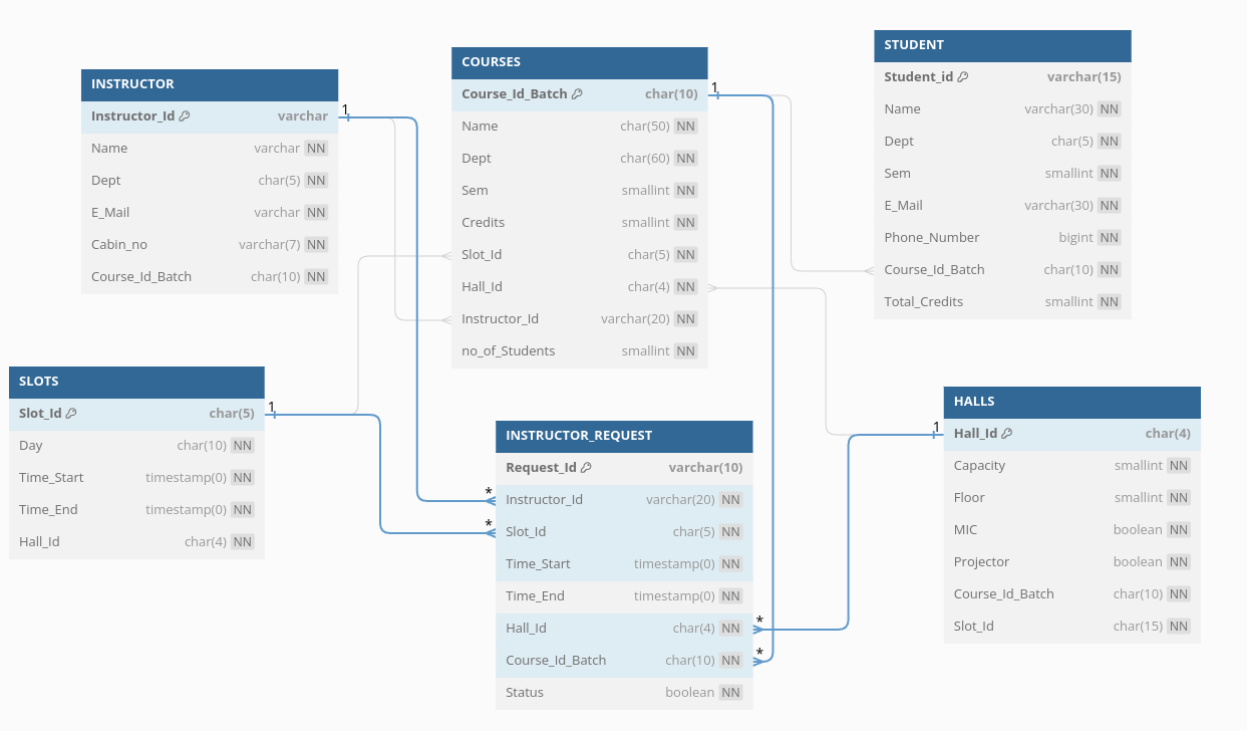

Figure 2: UML Diagram

7 Libraries and Frameworks

- The project uses Flask, a popular Python framework for web application development.Flask is known for its simplicity, flexibility, and ease of use, making it suitable for both beginners and experienced developers. It forms the backbone of the application, managing various backend operations.
- Flask handles routing user requests to the appropriate endpoints, rendering dynamic webpages, and facilitating database interactions.The project utilizes the mysql.connector library to manage database interactions.
- This library acts as a bridge between the Flask application and the MySQL database. It allows for seamless data storage and retrieval, and efficient handling of data operations.

8 User Login and Authentication System

8.1 Purpose

Our website employs a robust login system to ensure that only authorized users have access to specific areas and functionalities.

8.2 Implementation

We have integrated user authentication using the Flask-Login extension, which simplifies the process of managing user sessions in Flask applications.

8.3 Authentication Process

1. When a user attempts to log in, their username and password are verified against a predefined dictionary of users. This dictionary acts as a simple user database for the purpose of this demonstration.

2. If the provided credentials match an entry in the dictionary, a session is created for the user. This session is used to maintain the user's authenticated state across different pages of the website.

8.4 Session Management

- Upon successful login, Flask-Login marks the user as logged in and manages the session for them. This includes handling the creation and termination of sessions.
- The user's session remains active until they log out or the session expires.

8.5 Access Control

- We utilize the login_required decorator provided by Flask-Login to protect certain routes and pages. This decorator ensures that only users who are logged in can access these protected resources.
- If a non-authenticated user tries to access a protected page, they are redirected to the login page.

This implementation not only ensures secure user authentication but also provides a seamless and protected user experience. By leveraging Flask-Login, we can efficiently manage user sessions and safeguard sensitive parts of our website.

9 Database Interaction

9.1 Database Choice

Our application utilizes a MySQL database to store and manage a variety of data, including instructor information, student information, timetable, instructor requests, slots, and halls.

9.2 Functionality

The codebase incorporates several dedicated functions for database interaction:

- **Data Insertion:** Functions such as insert_student_data are designed to add new records into the database. These functions ensure that new entries, whether they pertain to faculty members, projects, or TAs, are properly recorded and stored.
- **Data Retrieval:** Functions like faculty_dashboard are implemented to fetch existing data from the database. These retrieval functions allow us to access detailed information about faculty timetable, faculty's requests, and more.

9.3 Security Measures

To safeguard our database against SQL injection attacks, we employ prepared statements. Prepared statements are a critical security feature that helps protect our database by ensuring that SQL queries are parameterized and that user inputs are appropriately sanitized.

By using prepared statements, we mitigate the risk of malicious SQL code being executed, thereby maintaining the integrity and security of our database.

This approach ensures that our database interactions are both efficient and secure. By adhering to best practices in database management and security, we can maintain a reliable and robust data storage solution that supports the needs of our application.

10 Routes and Templates

10.1 Route Definition

Our website's various functionalities are mapped to specific URLs using the @app.route decorator in Flask. Each route corresponds to a distinct endpoint that users can access.

10.1.1 Examples

- The /login route is designated for handling user authentication. When users navigate to this URL, they are presented with the login interface where they can enter their credentials.
- The /student_dashboard route is responsible for displaying the student's timetable. This route provides a students to look at their timetable and lecture requests.

10.2 Dynamic Content Generation

Flask leverages the power of templates to create dynamic HTML content. Templates enable the separation of HTML structure from the underlying Python code, promoting a cleaner and more maintainable codebase.

10.3 Template Rendering

The render_template function is utilized to render HTML templates and pass data to them. This function takes the name of the template file and any context variables that should be accessible within the template.

For instance, when rendering the login page, render_template('login.html') might be used, potentially passing variables like error messages or user-specific data to enhance the user experience.

10.4 Data Passing and Context

Data passed to templates can include various types of information, such as user details, lists of projects, or status messages. This data is injected into the template context, allowing the HTML to be dynamically updated based on the current state of the application.

This mechanism ensures that web pages are not static but adapt in real-time to reflect changes in data and user interactions.

By effectively utilizing routes and templates, our Flask application delivers a seamless and dynamic user experience. Routes define the structure of our application, directing user requests to the

appropriate handlers, while templates ensure that our web pages are generated dynamically with up-to-date data. This combination allows us to build a responsive and interactive web application.

11 Technology Stack

Our backend infrastructure is built using a carefully selected set of core technologies that ensure reliability, security, and flexibility. Below is a detailed overview of the technologies employed:

11.1 Programming Language

Python: Chosen for its popularity, versatility, and ease of use, Python enables rapid development and clean, readable code. Its extensive library support and active community make it an ideal choice for backend development.

11.2 Web Framework

Flask: A lightweight and highly flexible web framework, Flask allows developers to build web applications quickly and efficiently. Flask's modular design encourages a clean architecture, making it easier to scale and maintain the application.

11.3 Database

MySQL: A widely-used and reliable relational database, MySQL is known for its robustness and performance. It provides comprehensive support for structured data storage and complex queries, making it a cornerstone of our data management system.

11.4 Database Connector

mysql.connector library: This library facilitates secure and efficient communication between our Python application and the MySQL database. It ensures that database operations are performed

reliably and safely, with support for prepared statements to prevent SQL injection attacks.

11.5 Template Engine

Jinja2: As the default templating engine for Flask, Jinja2 allows us to generate dynamic HTML content. It provides powerful features like template inheritance and macros, enabling the creation of reusable and maintainable templates.

11.6 User Authentication

Flask-Login: This extension is essential for managing user authentication and session handling in our application. Flask-Login simplifies the process of logging users in and out, managing user sessions, and protecting routes to ensure only authenticated users have access to certain parts of the application.

Each component of this technology stack has been selected to contribute to a secure, efficient, and scalable backend system. By leveraging these technologies, we ensure that our application can meet the demands of modern web development while providing a seamless and secure user experience.

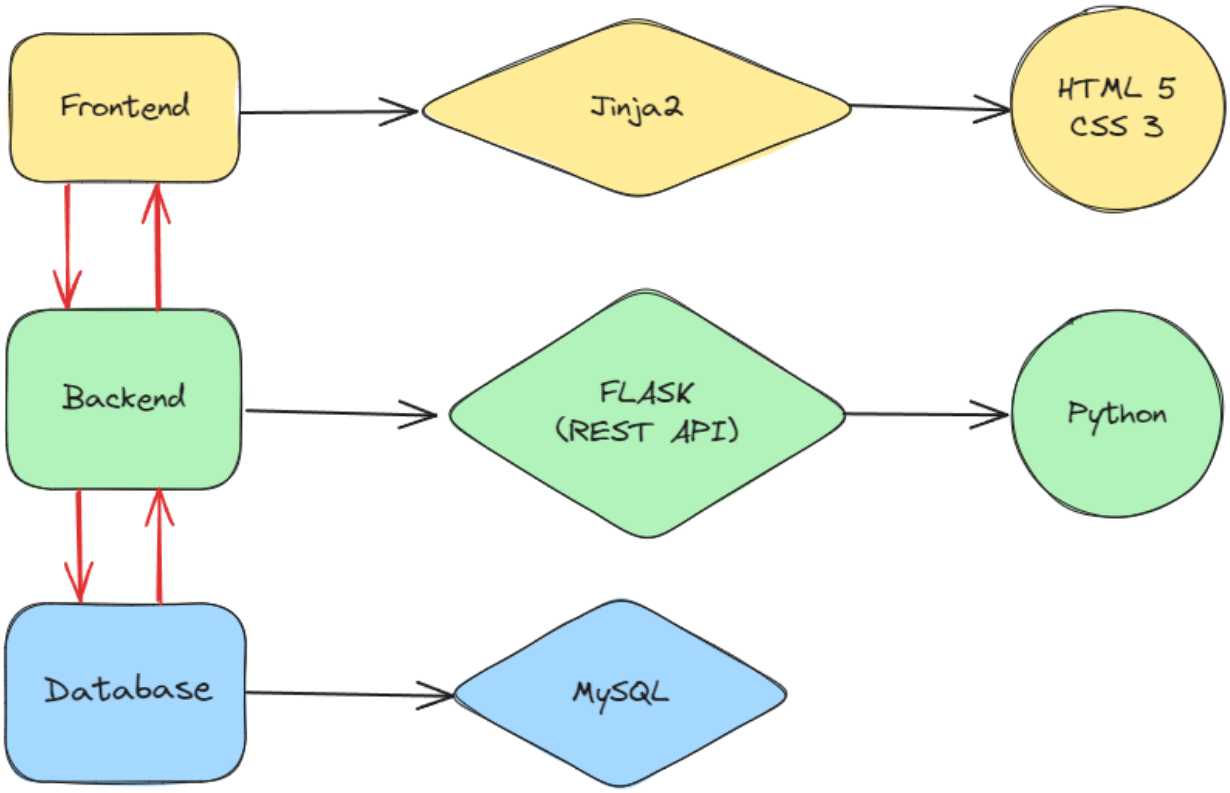

Figure 3: Tech Stack Diagram

12 Data Flow

12.1 User Interaction

Users engage with the website's frontend, which is built using HTML, CSS, and JavaScript. They interact by submitting forms, clicking buttons, and making various requests to the application. This could include actions such as logging in, adding project details, or updating personal information.

12.2 Frontend-Backend Communication

The frontend communicates with the backend by sending HTTP requests. These requests, which might include login credentials or project data, are dispatched through API endpoints or standard form submissions. The communication ensures that user inputs are sent securely and efficiently to the server for processing.

12.3 Backend Processing

12.3.1 Request Routing

Flask handles incoming requests by routing them to the appropriate functions based on the URL and HTTP method (e.g., GET, POST).

12.3.2 Authentication

When users log in, their credentials are validated. The backend compares the provided credentials against hashed passwords stored securely in the database.

12.3.3 Data Manipulation

For operations such as adding new projects or updating faculty information, the backend interacts with the database using prepared SQL statements. This approach ensures secure and efficient data manipulation, protecting against SQL injection attacks.

12.3.4 Business Logic Execution

The backend processes each request according to the application's business logic. This might involve inserting new records, updating existing ones, or performing complex data queries. The logic ensures that the application's functionality is correctly executed.

12.4 Response Generation

Based on the processed request, the backend generates an appropriate response. This response could involve fetching data from the database and formatting it for the frontend. For API endpoints, the data might be formatted as JSON, while for traditional web pages, HTML templates are rendered.

12.5 Response Delivery

The generated response is sent back to the frontend. This response contains the necessary data or status information to inform the frontend of the result of the request.

12.6 Frontend Update

Upon receiving the response, the frontend updates the user interface accordingly. This could involve displaying success or error messages, updating tables with new data, or dynamically altering the content displayed to the user. The update ensures that users receive immediate feedback and that the interface reflects the current state of the application.

By following this data flow, our application ensures smooth and efficient interaction between users and the system. Each step is designed to maintain security, efficiency, and a seamless user experience.

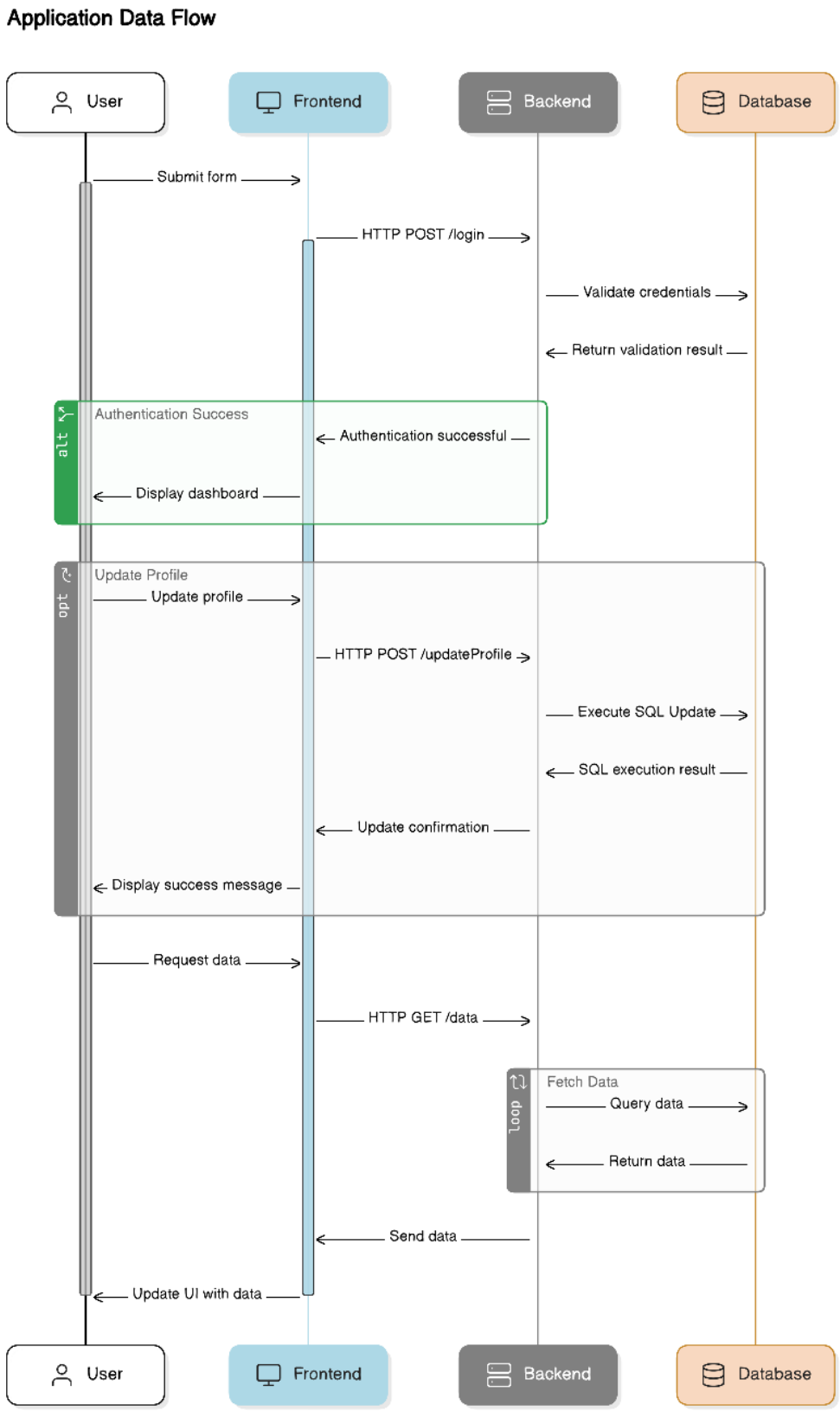

Figure 4: Data Flow Diagram

13 Connecting Frontend and Backend

13.1 Forms

HTML Forms: Our application uses HTML forms as the primary method for users to submit data. These forms are carefully designed with appropriate input fields, ensuring that users can enter the required information easily and accurately.

13.1.1 HTTP Methods

Each form is configured with a specific HTTP method to determine how the data is sent to the backend:

- **POST Method:** For data submission operations, such as user registration, login, or adding new projects, the forms use the POST method. This method securely sends data encapsulated within the request body, ensuring sensitive information like passwords is not exposed in the URL.

13.1.2 Action Attribute

The action attribute of each form specifies the backend route to which the form data will be sent. This route corresponds to a specific endpoint in the Flask application that is designed to handle the particular type of data being submitted.

For example, a login form might have ¡form action=”/login” method=”POST”¿, directing the submitted credentials to the /login route for processing.

13.1.3 Form Elements

- **Input Fields:** The forms include various input elements (¡input¿, ¡textarea¿, ¡select¿) tailored to the type of data being collected. Input validation attributes (e.g., required, maxlength) are used to ensure data integrity and improve user experience.

- **Submit Button:** A submit button (¡button type="submit"¿) triggers the form submission, sending the collected data to the backend for processing.

13.2 Backend Route Handling

Route Definition: In the Flask application, routes are defined using the @app.route decorator, specifying the URL and accepted HTTP methods

13.3 Response Generation and Feedback

After processing the form data, the backend generates an appropriate response. This response could include rendering a new template, redirecting to another page, or sending a JSON response in case of an API call.

The frontend receives this response and updates the user interface accordingly, providing feedback to the user (e.g., success or error messages) and reflecting any changes made by the submitted data.

By utilizing HTML forms and connecting them to specific backend routes with appropriate HTTP methods, our application ensures efficient and secure data transfer from the frontend to the backend. This integration allows for smooth user interactions and robust backend processing, maintaining a cohesive and responsive application.

14 Database and Schema

14.1 Course Table

```
mysql> desc course;
+---------------+--------------+------+-----+---------+-------+
| Field         | Type         | Null | Key | Default | Extra |
+---------------+--------------+------+-----+---------+-------+
| Course_Id     | varchar(10)  | NO   | PRI | NULL    |       |
| Name          | varchar(100) | YES  |     | NULL    |       |
| Credits       | int          | YES  |     | NULL    |       |
| Sem           | int          | YES  |     | NULL    |       |
| Dept          | varchar(100) | YES  |     | NULL    |       |
| Num_of_std    | int          | YES  |     | NULL    |       |
| Instructor_Id | varchar(10)  | YES  |     | NULL    |       |
| Hall_Id       | varchar(10)  | YES  |     | NULL    |       |
| Slot_Id       | varchar(10)  | YES  |     | NULL    |       |
+---------------+--------------+------+-----+---------+-------+
9 rows in set (0.01 sec)

mysql> select * from course;
+-----------+------+---------+------+------+------------+---------------+---------+---------+
| Course_Id | Name | Credits | Sem  | Dept | Num_of_std | Instructor_Id | Hall_Id | Slot_Id |
+-----------+------+---------+------+------+------------+---------------+---------+---------+
| 1         | DBMS |       4 |    4 | cse  |        156 | 1             | 05      | 1       |
| 2         | TOC  |       4 |    4 | cse  |        156 | 2             | 45      | 2       |
+-----------+------+---------+------+------+------------+---------------+---------+---------+
2 rows in set (0.00 sec)
```

Figure 5: Course Table

14.2 Enrolled Table

```
mysql> desc enrolled;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| Student_Id | varchar(10) | NO   | PRI | NULL    |       |
| Course_Id  | varchar(10) | NO   | PRI | NULL    |       |
| Slot_Id    | varchar(10) | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
3 rows in set (0.01 sec)

mysql> select * from enrolled;
+------------+-----------+---------+
| Student_Id | Course_Id | Slot_Id |
+------------+-----------+---------+
| 1          | 2008      | 1       |
| 2          | 2008      | 1       |
+------------+-----------+---------+
2 rows in set (0.00 sec)
```

Figure 6: Enrolled Table

14.3 Halls Table

```
mysql> desc halls;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| Hall_Id   | varchar(10) | NO   | PRI | NULL    |       |
| Capacity  | int         | YES  |     | NULL    |       |
| Projector | tinyint(1)  | YES  |     | NULL    |       |
| Mic       | tinyint(1)  | YES  |     | NULL    |       |
| Floor     | int         | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)

mysql> select * from halls;
+---------+----------+-----------+------+-------+
| Hall_Id | Capacity | Projector | Mic  | Floor |
+---------+----------+-----------+------+-------+
| 05      |      180 |         1 |    1 |     0 |
| 45      |      200 |         1 |    0 |     4 |
+---------+----------+-----------+------+-------+
2 rows in set (0.00 sec)
```

Figure 7: Halls Table

14.4 Instructor Table

```
mysql> desc instructor;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| Instructor_Id | varchar(10) | NO   | PRI | NULL    |       |
| Name          | varchar(100) | YES  |     | NULL    |       |
| Dept          | varchar(100) | YES  |     | NULL    |       |
| E_mail        | varchar(100) | YES  |     | NULL    |       |
| Cabin_no      | varchar(10) | YES  |     | NULL    |       |
| Course_Id     | varchar(10) | YES  |     | NULL    |       |
| Batch         | varchar(10) | YES  |     | NULL    |       |
| Password      | varchar(255) | YES  |     | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
8 rows in set (0.01 sec)

mysql> select * from instructor;
+---------------+------------+------+-----------------------+----------+-----------+-------+----------+
| Instructor_Id | Name       | Dept | E_mail                | Cabin_no | Course_Id | Batch | Password |
+---------------+------------+------+-----------------------+----------+-----------+-------+----------+
| 1             | sivaselvan | cse  | ss@iiitdm.ac.in       | l109     | 2008      | A     | 1234     |
| 2             | sadagopan  | cse  | sadagopan@iiitdm.ac.in | l119     | 2009      | A     | 1234     |
+---------------+------------+------+-----------------------+----------+-----------+-------+----------+
2 rows in set (0.00 sec)
```

Figure 8: Instructor Table

14.5 Instructor Request Table

```
mysql> desc instructor_request;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| Request_Id    | varchar(10) | NO   | PRI | NULL    |       |
| Instructor_Id | varchar(10) | YES  |     | NULL    |       |
| Course_Id     | varchar(10) | YES  |     | NULL    |       |
| Batch         | varchar(10) | YES  |     | NULL    |       |
| Slot_Id       | varchar(10) | YES  |     | NULL    |       |
| Time_Start    | time        | YES  |     | NULL    |       |
| Time_End      | time        | YES  |     | NULL    |       |
| Status        | varchar(50) | YES  |     | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
8 rows in set (0.00 sec)

mysql> select * from instructor_request;
+------------+---------------+-----------+-------+---------+------------+----------+---------------+
| Request_Id | Instructor_Id | Course_Id | Batch | Slot_Id | Time_Start | Time_End | Status        |
+------------+---------------+-----------+-------+---------+------------+----------+---------------+
| 1          | 1             | 2008      | A     | 1       | 12:00:00   | 14:00:00 | Available     |
| 2          | 1             | 2008      | A     | 2       | 08:00:00   | 10:00:00 | Available     |
| 3          | 1             | 2008      | A     | 1       | 08:00:00   | 10:00:00 | Available     |
| 4          | 1             | 2008      | A     | 1       | 08:01:00   | 09:50:00 | Not Available |
+------------+---------------+-----------+-------+---------+------------+----------+---------------+
4 rows in set (0.00 sec)
```

Figure 9: Instructor Request Table

14.6 Slot Table

```
mysql> desc slot;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| Slot_Id    | varchar(10) | NO   | PRI | NULL    |       |
| Day        | varchar(10) | YES  |     | NULL    |       |
| Time_Start | time        | YES  |     | NULL    |       |
| Time_End   | time        | YES  |     | NULL    |       |
| Hall_Id    | varchar(10) | YES  |     | NULL    |       |
| Batch      | varchar(10) | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)

mysql> select * from slot;
+---------+------------+------------+----------+---------+-------+
| Slot_Id | Day        | Time_Start | Time_End | Hall_Id | Batch |
+---------+------------+------------+----------+---------+-------+
| 1       | 12-05-2024 | 12:00:00   | 14:00:00 | 05      | A     |
| 2       | 12-05-2024 | 08:00:00   | 10:00:00 | 45      | A     |
+---------+------------+------------+----------+---------+-------+
2 rows in set (0.00 sec)
```

Figure 10: Slot Table

14.7 Student Table

```
mysql> desc student;
+---------------+--------------+------+-----+---------+-------+
| Field         | Type         | Null | Key | Default | Extra |
+---------------+--------------+------+-----+---------+-------+
| Student_Id    | varchar(10)  | NO   | PRI | NULL    |       |
| Name          | varchar(100) | YES  |     | NULL    |       |
| Dept          | varchar(100) | YES  |     | NULL    |       |
| Sem           | int          | YES  |     | NULL    |       |
| E_Mail        | varchar(100) | YES  |     | NULL    |       |
| Phone_Number  | varchar(15)  | YES  |     | NULL    |       |
| Batch         | varchar(10)  | YES  |     | NULL    |       |
| Total_Credits | int          | YES  |     | NULL    |       |
| Password      | varchar(255) | YES  |     | NULL    |       |
+---------------+--------------+------+-----+---------+-------+
9 rows in set (0.00 sec)

mysql> select * from student;
+------------+------------+------+------+------------------------+--------------+-------+---------------+----------+
| Student_Id | Name       | Dept | Sem  | E_Mail                 | Phone_Number | Batch | Total_Credits | Password |
+------------+------------+------+------+------------------------+--------------+-------+---------------+----------+
| 1093       | rohan      | cse  |    4 | cs22b1093@iiitdm.ac.in | 9490453915   | A     |            23 | 1234     |
| 1097       | vivekreddy | cse  |    4 | cs22b1097@iiitdm.ac.in | 7396887166   | B     |            20 | 1234     |
+------------+------------+------+------+------------------------+--------------+-------+---------------+----------+
2 rows in set (0.00 sec)
```

Figure 11: Student Table

15 Source Code

```
from flask import Flask, render_template, request, redirect, url_for, session, flash,
get_flashed_messages
from collections import defaultdict
import mysql.connector

app = Flask(__name__)
app.secret_key = 'admin'

# Configuration
app.config['MYSQL_DATABASE_HOST'] = 'localhost'
app.config['MYSQL_DATABASE_USER'] = 'root'
app.config['MYSQL_DATABASE_PASSWORD'] = 'password'
app.config['MYSQL_DATABASE_DB'] = 'acad_hall_slot_management'

# Connect to MySQL
db = mysql.connector.connect(
    host=app.config['MYSQL_DATABASE_HOST'],
    user=app.config['MYSQL_DATABASE_USER'],
    password=app.config['MYSQL_DATABASE_PASSWORD'],
    database=app.config['MYSQL_DATABASE_DB']
)

cursor = db.cursor(dictionary=True)

@app.route('/')
def index():
    return redirect(url_for('login'))
```

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        user_type = request.form['user_type']
        user_id = request.form['user_id']
        password = request.form['password']

        if user_type == 'student':
            query = "SELECT * FROM STUDENT WHERE Student_Id = %s AND Password = %s"
        elif user_type == 'faculty':
            query = "SELECT * FROM INSTRUCTOR WHERE Instructor_Id = %s AND Password = %s"
        else:
            flash('Invalid user type')
            return redirect(url_for('login'))

        cursor.execute(query, (user_id, password))
        user = cursor.fetchone()

        if user:
            session['user_id'] = user_id
            session['user_type'] = user_type
            if user_type == 'student':
                return redirect(url_for('student_dashboard'))
            elif user_type == 'faculty':
                return redirect(url_for('faculty_dashboard'))
        else:
            flash('Invalid credentials')
            return redirect(url_for('login'))
```

```
    return render_template('login.html')

@app.route('/logout')
def logout():
    session.pop('user_id', None)
    session.pop('user_type', None)
    return redirect(url_for('login'))

@app.route('/admin_login', methods=['GET', 'POST'])
def admin_login():
    if request.method == 'POST':
        admin_id = request.form['admin_id']
        password = request.form['password']

        if admin_id == 'admin' and password == 'admin':
            session['admin_id'] = admin_id
            return redirect(url_for('admin_dashboard'))  # Redirect to admin dashboard
        else:
            flash('Invalid admin credentials')

    return render_template('admin_login.html')



# Admin Dashboard
@app.route('/admin_dashboard')
def admin_dashboard():
    if 'admin_id' not in session:
        return redirect(url_for('admin_login'))
```

```
    return render_template('admin_dashboard.html')

@app.route('/student_dashboard')
def student_dashboard():
    if 'user_id' not in session or session['user_type'] != 'student':
        return redirect(url_for('login'))

    student_id = session['user_id']

    # Fetch the student's timetable
    query = """
    SELECT SLOT.Slot_Id, SLOT.Time_Start, SLOT.Time_End, HALLS.Hall_Id, COURSE.Course_Id,
    COURSE.Name AS Course_Name, INSTRUCTOR.Name AS Instructor_Name
    FROM ENROLLED
    JOIN SLOT ON ENROLLED.Slot_Id = SLOT.Slot_Id
    JOIN HALLS ON SLOT.Hall_Id = HALLS.Hall_Id
    JOIN COURSE ON ENROLLED.Course_Id = COURSE.Course_Id
    JOIN INSTRUCTOR ON COURSE.Instructor_Id = INSTRUCTOR.Instructor_Id
    WHERE ENROLLED.Student_Id = %s
    """
    cursor.execute(query, (student_id,))
    timetable = cursor.fetchall()

    # Group the timetable by days
    grouped_timetable = defaultdict(list)
    for entry in timetable:
        grouped_timetable[entry['Day']].append(entry)
```

```
    # Fetch the lecture requests by professors
    query = "SELECT * FROM INSTRUCTOR_REQUEST WHERE Batch =
    (SELECT Batch FROM STUDENT WHERE Student_Id = %s)"
    cursor.execute(query, (student_id,))
    requests = cursor.fetchall()

    return render_template('student_dashboard.html', timetable=grouped_timetable,
    requests=requests)

@app.route('/faculty_dashboard', methods=['GET', 'POST'])
def faculty_dashboard():
    if 'user_id' not in session or session['user_type'] != 'faculty':
        return redirect(url_for('login'))

    instructor_id = session['user_id']

    if request.method == 'POST':
        request_id = request.form['request_id']
        course_id = request.form['course_id']
        batch = request.form['batch']
        slot_id = request.form['slot_id']
        time_start = request.form['time_start']
        time_end = request.form['time_end']

        # Check slot availability
        availability_query = "SELECT * FROM SLOT WHERE Slot_Id = %s AND Hall_Id NOT IN
        (SELECT Hall_Id FROM INSTRUCTOR_REQUEST WHERE Slot_Id = %s AND Time_Start < %s
        AND Time_End > %s)"
        cursor.execute(availability_query, (slot_id, slot_id, time_end, time_start))
```

```
    available_slots = cursor.fetchall()

    if available_slots:
        # Slot is available
        status = 'Available'
    else:
        # Slot is not available
        status = 'Not Available'

    # Insert the request with status
    query = "INSERT INTO INSTRUCTOR_REQUEST (Request_Id, Instructor_Id, Course_Id, Batch,
    Slot_Id, Time_Start, Time_End, Status) VALUES (%s, %s, %s, %s, %s, %s, %s, %s)"
    values = (request_id, instructor_id, course_id, batch, slot_id, time_start,
    time_end, status)
    cursor.execute(query, values)
    db.commit()
    return redirect(url_for('faculty_dashboard'))

# Fetch the faculty's schedule including status
query = """
SELECT SLOT.Slot_Id, SLOT.Day, SLOT.Time_Start, SLOT.Time_End, HALLS.Hall_Id,
COURSE.Course_Id, COURSE.Name AS Course_Name, INSTRUCTOR_REQUEST.Status
FROM COURSE
JOIN SLOT ON COURSE.Slot_Id = SLOT.Slot_Id
JOIN HALLS ON SLOT.Hall_Id = HALLS.Hall_Id
LEFT JOIN INSTRUCTOR_REQUEST ON COURSE.Course_Id = INSTRUCTOR_REQUEST.Course_Id
AND INSTRUCTOR_REQUEST.Instructor_Id = %s
WHERE COURSE.Instructor_Id = %s
"""
```

```
    cursor.execute(query, (instructor_id, instructor_id))
    schedule = cursor.fetchall()

    # Fetch the faculty's requests
    query = "SELECT * FROM INSTRUCTOR_REQUEST WHERE Instructor_Id = %s"
    cursor.execute(query, (instructor_id,))
    requests = cursor.fetchall()

    return render_template('faculty_dashboard.html', schedule=schedule, requests=requests)

@app.route('/students', methods=['GET', 'POST'])
def students():
    if request.method == 'POST':
        student_id = request.form['student_id']
        name = request.form['name']
        dept = request.form['dept']
        sem = request.form['sem']
        email = request.form['email']
        phone = request.form['phone']
        batch = request.form['batch']
        credits = request.form['credits']
        password = request.form['password']

        query = "INSERT INTO STUDENT (Student_Id, Name, Dept, Sem, E_Mail, Phone_Number,
        Batch, Total_Credits, Password) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)"
        values = (student_id, name, dept, sem, email, phone, batch, credits, password)
        cursor.execute(query, values)
        db.commit()
        return redirect(url_for('students'))
```

```
    cursor.execute("SELECT * FROM STUDENT")
    students = cursor.fetchall()
    return render_template('students.html', students=students)

@app.route('/students/delete/<int:student_id>', methods=['POST'])
def delete_student(student_id):
    query = "DELETE FROM STUDENT WHERE Student_Id = %s"
    cursor.execute(query, (student_id,))
    db.commit()
    return redirect(url_for('students'))

@app.route('/students/edit/<int:student_id>', methods=['GET', 'POST'])
def edit_student(student_id):
    if request.method == 'POST':
        name = request.form['name']
        dept = request.form['dept']
        sem = request.form['sem']
        email = request.form['email']
        phone = request.form['phone']
        batch = request.form['batch']
        credits = request.form['credits']
        password = request.form['password']

        query = "UPDATE STUDENT SET Name = %s, Dept = %s, Sem = %s, E_Mail = %s,
        Phone_Number = %s, Batch = %s, Total_Credits = %s, Password = %s
        WHERE Student_Id = %s"
        values = (name, dept, sem, email, phone, batch, credits, password, student_id)
        cursor.execute(query, values)
```

```
        db.commit()
        return redirect(url_for('students'))

    cursor.execute("SELECT * FROM STUDENT WHERE Student_Id = %s", (student_id,))
    student = cursor.fetchone()
    return render_template('edit_student.html', student=student)

@app.route('/instructors', methods=['GET', 'POST'])
def instructors():
    if request.method == 'POST':
        instructor_id = request.form['instructor_id']
        name = request.form['name']
        dept = request.form['dept']
        email = request.form['email']
        cabin_no = request.form['cabin_no']
        course_id = request.form['course_id']
        batch = request.form['batch']
        password = request.form['password']

        query = "INSERT INTO INSTRUCTOR (Instructor_Id, Name, Dept, E_mail,
        Cabin_no, Course_Id, Batch, Password) VALUES (%s, %s, %s, %s, %s, %s, %s, %s)"
        values = (instructor_id, name, dept, email, cabin_no, course_id, batch, password)
        cursor.execute(query, values)
        db.commit()
        return redirect(url_for('instructors'))

    cursor.execute("SELECT * FROM INSTRUCTOR")
    instructors = cursor.fetchall()
    return render_template('instructors.html', instructors=instructors)
```

```
@app.route('/instructors/delete/<int:instructor_id>', methods=['POST'])
def delete_instructor(instructor_id):
    query = "DELETE FROM INSTRUCTOR WHERE Instructor_Id = %s"
    cursor.execute(query, (instructor_id,))
    db.commit()
    return redirect(url_for('instructors'))

@app.route('/instructors/edit/<int:instructor_id>', methods=['GET', 'POST'])
def edit_instructor(instructor_id):
    if request.method == 'POST':
        name = request.form['name']
        dept = request.form['dept']
        email = request.form['email']
        cabin_no = request.form['cabin_no']
        course_id = request.form['course_id']
        batch = request.form['batch']
        password = request.form['password']

        query = "UPDATE INSTRUCTOR SET Name = %s, Dept = %s, E_mail = %s, Cabin_no = %s,
        Course_Id = %s, Batch = %s, Password = %s WHERE Instructor_Id = %s"
        values = (name, dept, email, cabin_no, course_id, batch, password, instructor_id)
        cursor.execute(query, values)
        db.commit()
        return redirect(url_for('instructors'))

    cursor.execute("SELECT * FROM INSTRUCTOR WHERE Instructor_Id = %s", (instructor_id,))
    instructor = cursor.fetchone()
    return render_template('edit_instructor.html', instructor=instructor)
```

```
@app.route('/courses', methods=['GET', 'POST'])
def courses():
    if request.method == 'POST':
        course_id = request.form['course_id']
        name = request.form['name']
        credits = request.form['credits']
        sem = request.form['sem']
        dept = request.form['dept']
        num_of_std = request.form['num_of_std']
        instructor_id = request.form['instructor_id']
        hall_id = request.form['hall_id']
        slot_id = request.form['slot_id']

        query = "INSERT INTO COURSE (Course_Id, Name, Credits, Sem, Dept, Num_of_std,
        Instructor_Id, Hall_Id, Slot_Id) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)"
        values = (course_id, name, credits, sem, dept, num_of_std, instructor_id, hall_id,
        slot_id)
        cursor.execute(query, values)
        db.commit()
        return redirect(url_for('courses'))

    cursor.execute("SELECT * FROM COURSE")
    courses = cursor.fetchall()
    return render_template('courses.html', courses=courses)

@app.route('/courses/delete/<int:course_id>', methods=['POST'])
def delete_course(course_id):
    query = "DELETE FROM COURSE WHERE Course_Id = %s"
```

```
    cursor.execute(query, (course_id,))
    db.commit()
    return redirect(url_for('courses'))

@app.route('/courses/edit/<int:course_id>', methods=['GET', 'POST'])
def edit_course(course_id):
    if request.method == 'POST':
        name = request.form['name']
        credits = request.form['credits']
        sem = request.form['sem']
        dept = request.form['dept']
        num_of_std = request.form['num_of_std']
        instructor_id = request.form['instructor_id']
        hall_id = request.form['hall_id']
        slot_id = request.form['slot_id']

        query = "UPDATE COURSE SET Name = %s, Credits = %s, Sem = %s, Dept = %s,
        Num_of_std = %s, Instructor_Id = %s, Hall_Id = %s, Slot_Id = %s WHERE Course_Id = %s"
        values = (name, credits, sem, dept, num_of_std, instructor_id, hall_id,
        slot_id, course_id)
        cursor.execute(query, values)
        db.commit()
        return redirect(url_for('courses'))

    cursor.execute("SELECT * FROM COURSE WHERE Course_Id = %s", (course_id,))
    course = cursor.fetchone()
    return render_template('edit_course.html', course=course)

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

```
def slots():
    if request.method == 'POST':
        slot_id = request.form['slot_id']
        day = request.form['day']
        time_start = request.form['time_start']
        time_end = request.form['time_end']
        hall_id = request.form['hall_id']
        batch = request.form['batch']

        query = "INSERT INTO SLOT (Slot_Id, Day, Time_Start, Time_End, Hall_Id, Batch)
        VALUES (%s, %s, %s, %s, %s, %s)"
        values = (slot_id, day, time_start, time_end, hall_id, batch)
        cursor.execute(query, values)
        db.commit()
        return redirect(url_for('slots'))

    cursor.execute("SELECT * FROM SLOT")
    slots = cursor.fetchall()
    return render_template('slots.html', slots=slots)

@app.route('/slots/delete/<int:slot_id>', methods=['POST'])
def delete_slot(slot_id):
    query = "DELETE FROM SLOT WHERE Slot_Id = %s"
    cursor.execute(query, (slot_id,))
    db.commit()
    return redirect(url_for('slots'))

@app.route('/slots/edit/<int:slot_id>', methods=['GET', 'POST'])
def edit_slot(slot_id):
```

```
    if request.method == 'POST':
        day = request.form['day']
        time_start = request.form['time_start']
        time_end = request.form['time_end']
        hall_id = request.form['hall_id']
        batch = request.form['batch']

        query = "UPDATE SLOT SET Day = %s, Time_Start = %s, Time_End = %s, Hall_Id = %s,
        Batch = %s WHERE Slot_Id = %s"
        values = (day, time_start, time_end, hall_id, batch, slot_id)
        cursor.execute(query, values)
        db.commit()
        return redirect(url_for('slots'))

    cursor.execute("SELECT * FROM SLOT WHERE Slot_Id = %s", (slot_id,))
    slot = cursor.fetchone()
    return render_template('edit_slot.html', slot=slot)

@app.route('/halls', methods=['GET', 'POST'])
def halls():
    if request.method == 'POST':
        hall_id = request.form['hall_id']
        capacity = request.form['capacity']
        projector = request.form['projector']
        mic = request.form['mic']
        floor = request.form['floor']

        query = "INSERT INTO HALLS (Hall_Id, Capacity, Projector, Mic, Floor)
        VALUES (%s, %s, %s, %s, %s)"
```

```
        values = (hall_id, capacity, projector, mic, floor)
        cursor.execute(query, values)
        db.commit()
        return redirect(url_for('halls'))

    cursor.execute("SELECT * FROM HALLS")
    halls = cursor.fetchall()
    return render_template('halls.html', halls=halls)

@app.route('/halls/delete/<int:hall_id>', methods=['POST'])
def delete_hall(hall_id):
    query = "DELETE FROM HALLS WHERE Hall_Id = %s"
    cursor.execute(query, (hall_id,))
    db.commit()
    return redirect(url_for('halls'))

@app.route('/halls/edit/<int:hall_id>', methods=['GET', 'POST'])
def edit_hall(hall_id):
    if request.method == 'POST':
        capacity = request.form['capacity']
        projector = request.form['projector']
        mic = request.form['mic']
        floor = request.form['floor']

        query = "UPDATE HALLS SET Capacity = %s, Projector = %s, Mic = %s,
        Floor = %s WHERE Hall_Id = %s"
        values = (capacity, projector, mic, floor, hall_id)
        cursor.execute(query, values)
        db.commit()
```

```
        return redirect(url_for('halls'))

    cursor.execute("SELECT * FROM HALLS WHERE Hall_Id = %s", (hall_id,))
    hall = cursor.fetchone()
    return render_template('edit_hall.html', hall=hall)

@app.route('/requests', methods=['GET', 'POST'])
def requests():
    if request.method == 'POST':
        request_id = request.form['request_id']
        instructor_id = request.form['instructor_id']
        course_id = request.form['course_id']
        batch = request.form['batch']
        slot_id = request.form['slot_id']
        time_start = request.form['time_start']
        time_end = request.form['time_end']
        status = request.form['status']

        query = "INSERT INTO INSTRUCTOR_REQUEST (Request_Id, Instructor_Id, Course_Id,
        Batch, Slot_Id, Time_Start, Time_End, Status)
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s)"
        values = (request_id, instructor_id, course_id, batch, slot_id, time_start,
        time_end, status)
        cursor.execute(query, values)
        db.commit()
        return redirect(url_for('requests'))

    cursor.execute("SELECT * FROM INSTRUCTOR_REQUEST")
    requests = cursor.fetchall()
```

```
    return render_template('requests.html', requests=requests)

@app.route('/requests/delete/<int:request_id>', methods=['POST'])
def delete_request(request_id):
    query = "DELETE FROM INSTRUCTOR_REQUEST WHERE Request_Id = %s"
    cursor.execute(query, (request_id,))
    db.commit()
    return redirect(url_for('requests'))

@app.route('/requests/edit/<int:request_id>', methods=['GET', 'POST'])
def edit_request(request_id):
    if request.method == 'POST':
        instructor_id = request.form['instructor_id']
        course_id = request.form['course_id']
        batch = request.form['batch']
        slot_id = request.form['slot_id']
        time_start = request.form['time_start']
        time_end = request.form['time_end']
        status = request.form['status']

        query = "UPDATE INSTRUCTOR_REQUEST SET Instructor_Id = %s, Course_Id = %s,
        Batch = %s, Slot_Id = %s, Time_Start = %s, Time_End = %s, Status = %s
        WHERE Request_Id = %s"
        values = (instructor_id, course_id, batch, slot_id, time_start, time_end,
        status, request_id)
        cursor.execute(query, values)
        db.commit()
        return redirect(url_for('requests'))
```

```
    cursor.execute("SELECT * FROM INSTRUCTOR_REQUEST WHERE Request_Id = %s", (request_id,))
    request = cursor.fetchone()
    return render_template('edit_request.html', request=request)

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

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

16 Frontend Screenshots

16.1 Home Page

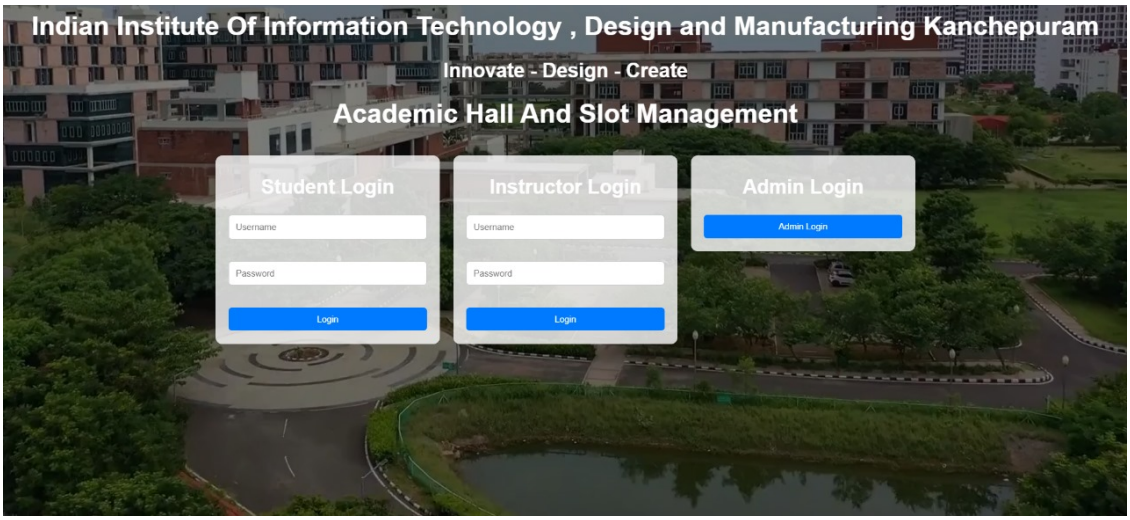

Figure 12: Home Page

16.2 Admin Login

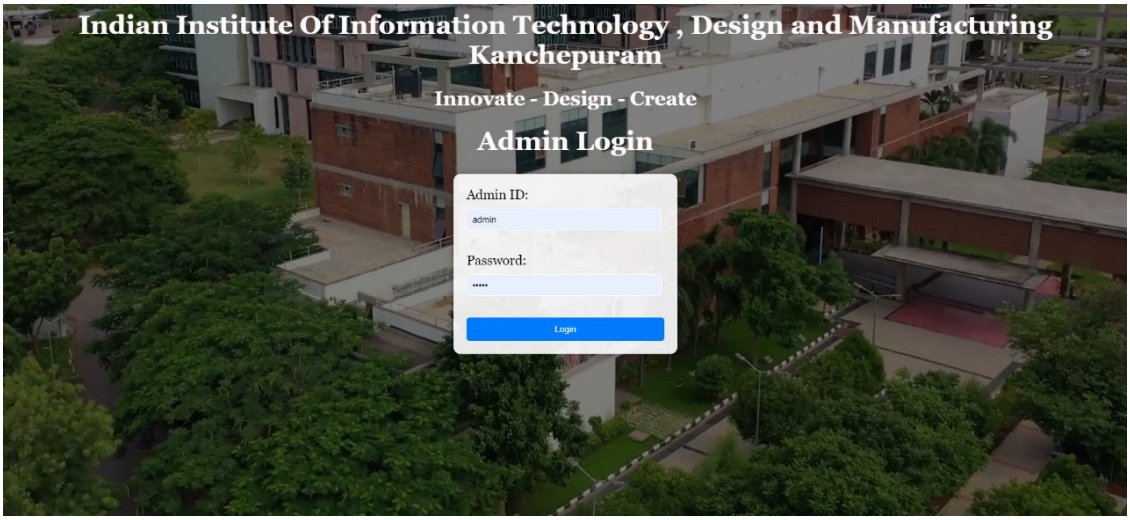

Figure 13: Admin Login Page

16.3 Admin Dashboard

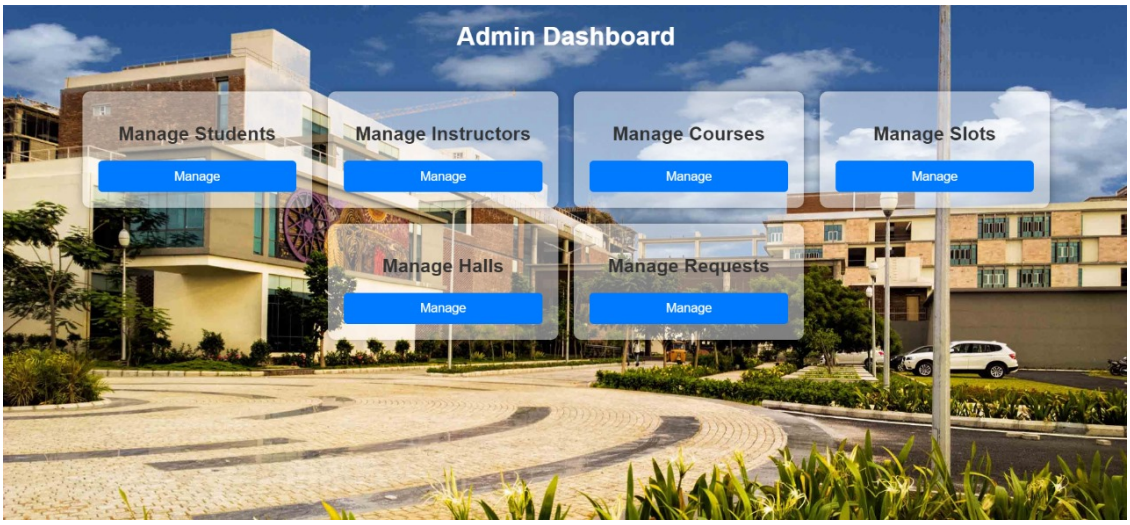

Figure 14: Admin Dashboard with administrative controls

16.3.1 Manage Student

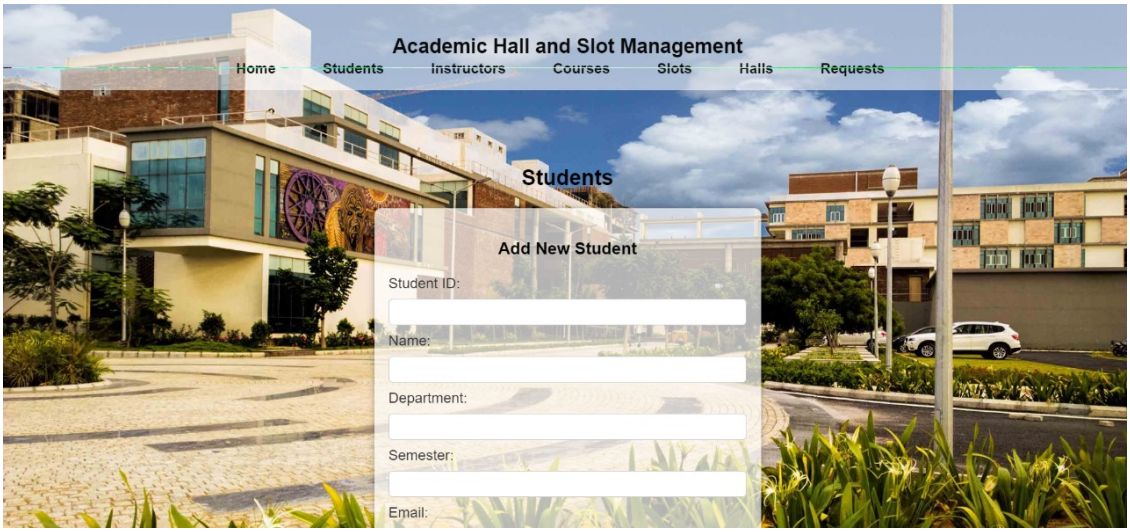

Figure 15: Manage Students Dashboard with CRUD options

```
SQL QUERY : INSERT INTO STUDENT (Student_Id, Name, Dept, Sem, E_Mail,
Phone_Number, Batch, Total_Credits, Password)
VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
```

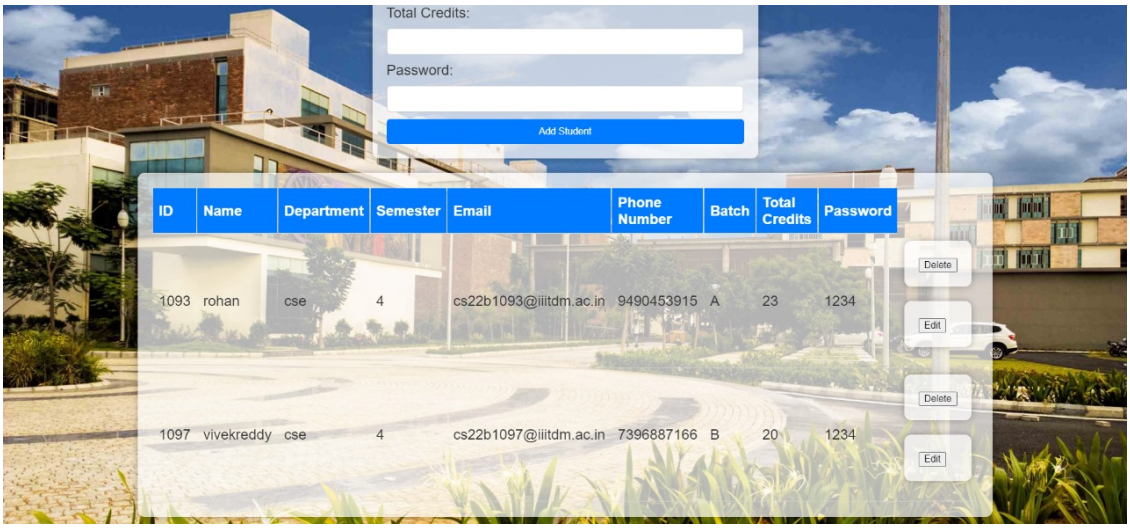

Figure 16: Manage Student Dashboard displaying the current student records

```
SQL QUERY : DELETE FROM STUDENT WHERE Student_Id = %s
SQL QUERY : UPDATE STUDENT SET Name = %s, Dept = %s, Sem = %s, E_Mail = %s,
Phone_Number = %s, Batch = %s, Total_Credits = %s,
Password = %s WHERE Student_Id = %s
```

16.3.2 Manage Instructors

Figure 17: Manage Instructor Dashboard with CRUD options

```
SQL QUERY : INSERT INTO INSTRUCTOR (Instructor_Id, Name, Dept,
E_mail, Cabin_no, Course_Id, Batch, Password)
VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
```

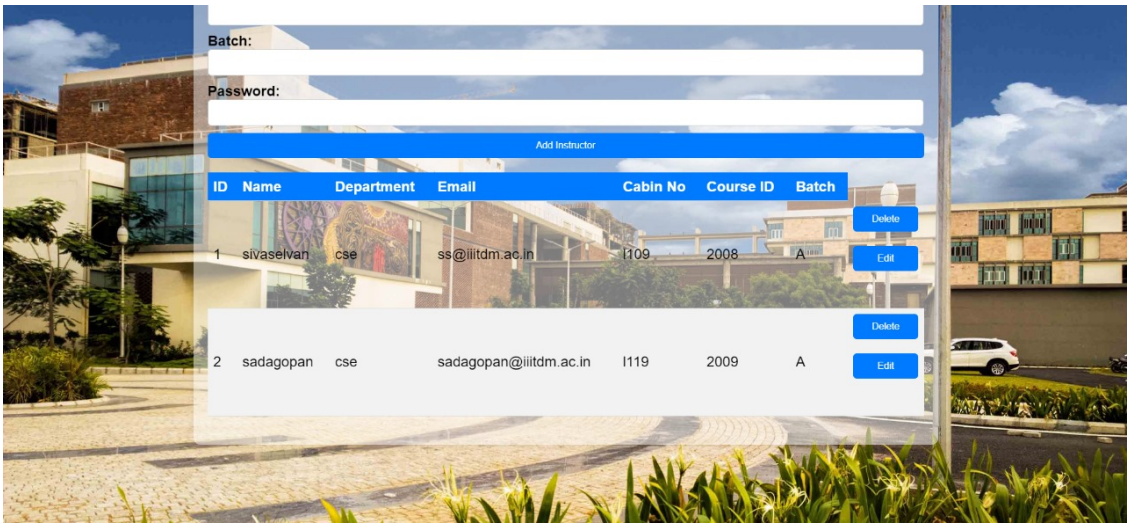

Figure 18: Manage Instructor Dashboard displaying the current instructor/faculty records

```
SQL QUERY : DELETE FROM INSTRUCTOR WHERE Instructor_Id = %s
SQL QUERY : UPDATE INSTRUCTOR SET Name = %s, Dept = %s, E_mail = %s,
Cabin_no = %s, Course_Id = %s, Batch = %s, Password = %s WHERE Instructor_Id = %s
```

16.3.3 Manage Courses

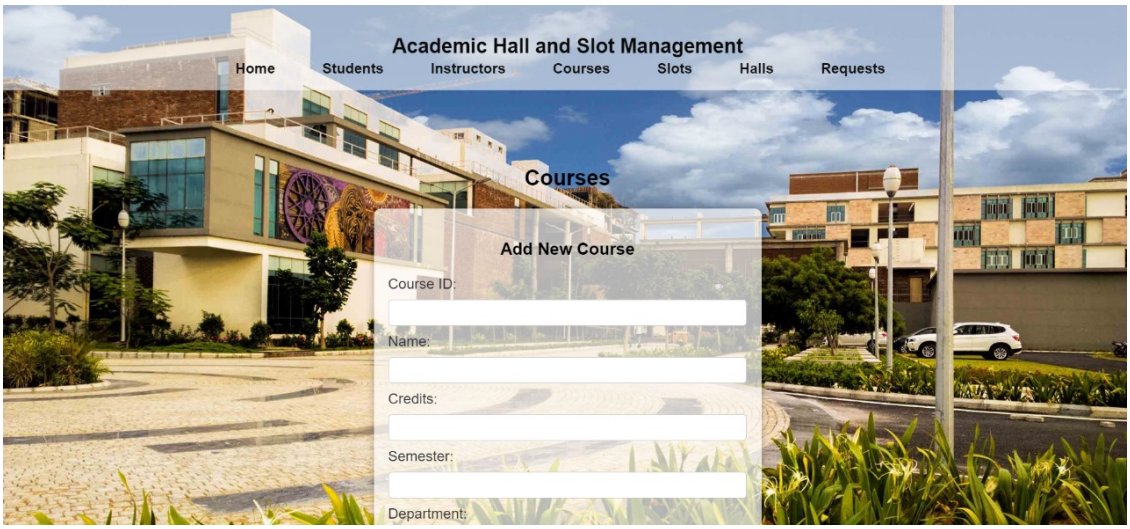

Figure 19: Manage Course Dashboard with CRUD options

```
SQL QUERY : INSERT INTO COURSE (Course_Id, Name, Credits, Sem, Dept, Num_of_std,
Instructor_Id, Hall_Id, Slot_Id)
VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
```

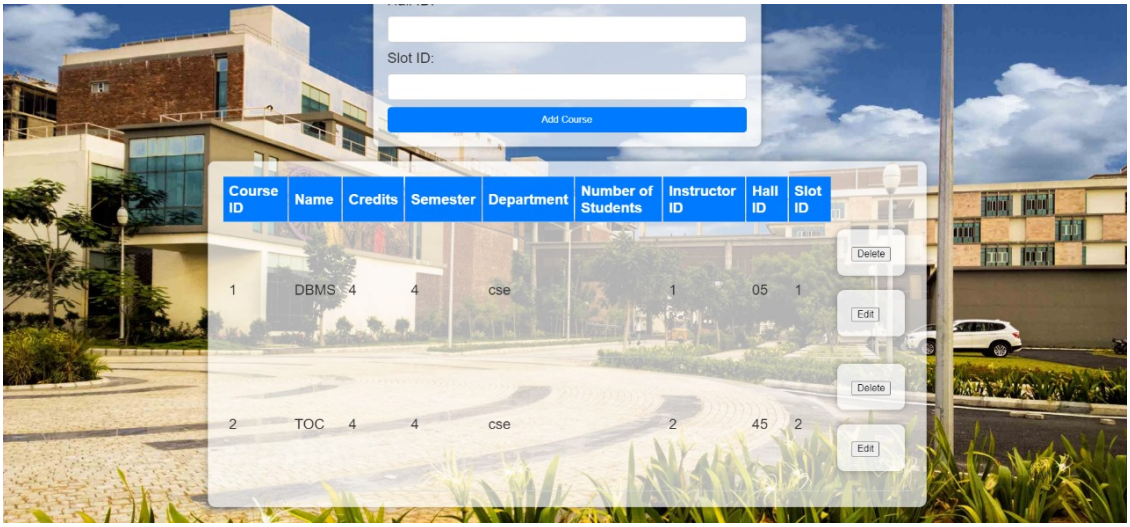

Figure 20: Manage Courses Dashboard displaying the current courses records

```
SQL QUERY : DELETE FROM COURSE WHERE Course_Id = %s
SQL QUERY : UPDATE COURSE SET Name = %s, Credits = %s, Sem = %s, Dept = %s,
Num_of_std = %s, Instructor_Id = %s, Hall_Id = %s,
Slot_Id = %s WHERE Course_Id = %s
```

16.3.4 Manage Slots

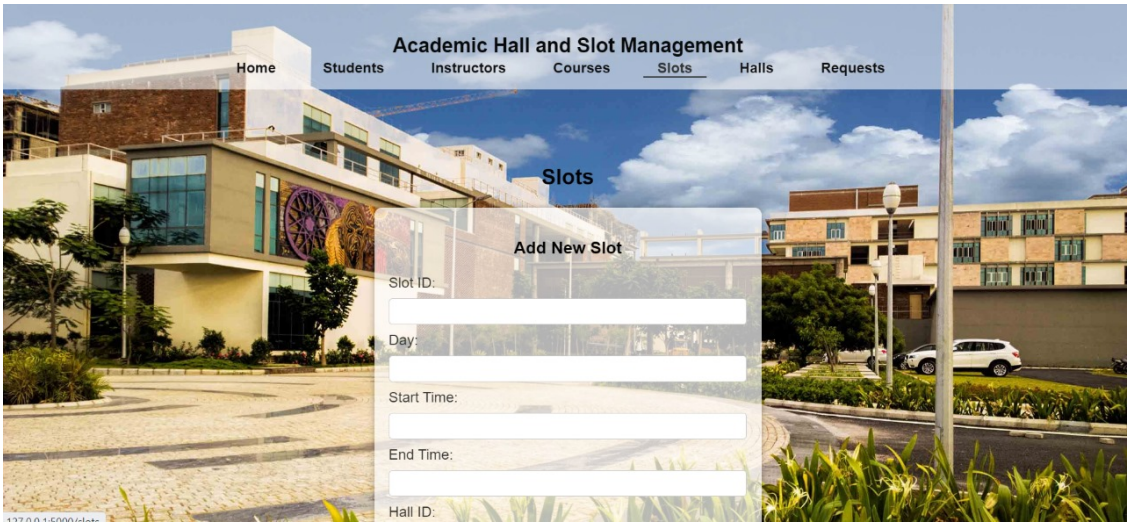

Figure 21: Manage Slots Dashboard with CRUD options

```
SQL QUERY : INSERT INTO SLOT (Slot_Id, Day, Time_Start, Time_End, Hall_Id,
Batch) VALUES (%s, %s, %s, %s, %s, %s)
```

Figure 22: Manage Slots Dashboard displaying the current slot records

```
SQL QUERY : DELETE FROM SLOT WHERE Slot_Id = %s
SQL QUERY : UPDATE SLOT SET Day = %s, Time_Start = %s, Time_End = %s,
Hall_Id = %s, Batch = %s WHERE Slot_Id = %s
```

16.3.5 Manage Halls

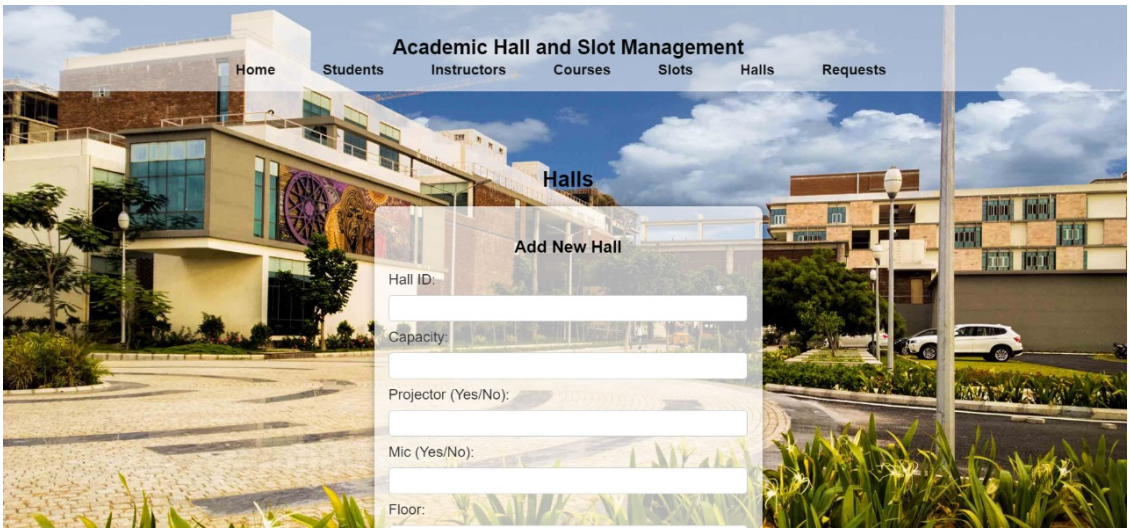

Figure 23: Manage Halls Dashboard with CRUD options

```
SQL QUERY : INSERT INTO HALLS (Hall_Id, Capacity,
Projector, Mic, Floor) VALUES (%s, %s, %s, %s, %s)
```

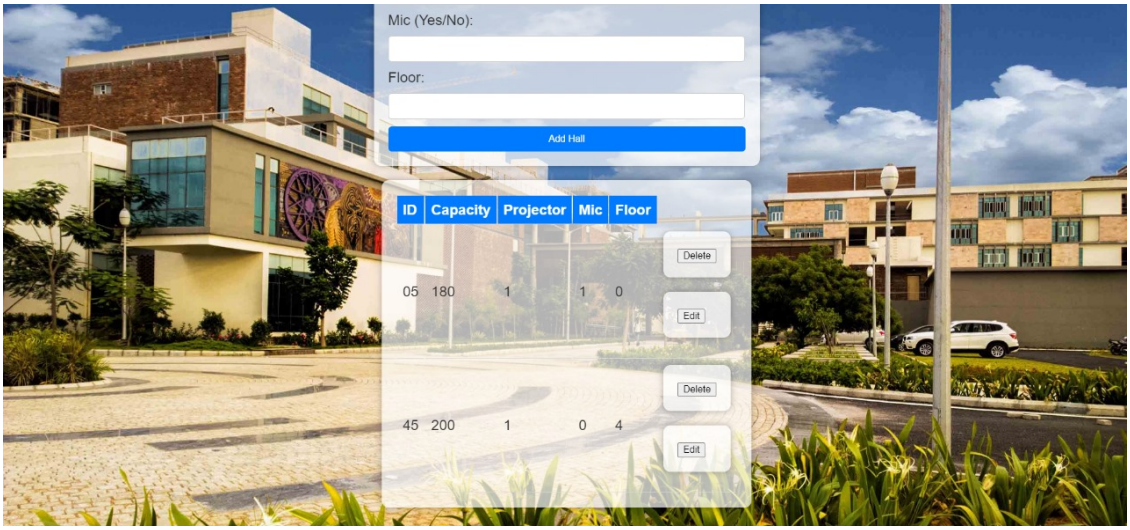

Figure 24: Manage Halls Dashboard displaying the current hall records and equipment conditions

```
SQL QUERY : UPDATE HALLS SET Capacity = %s, Projector = %s,
Mic = %s, Floor = %s WHERE Hall_Id = %s
```

16.4 Student Dashboard

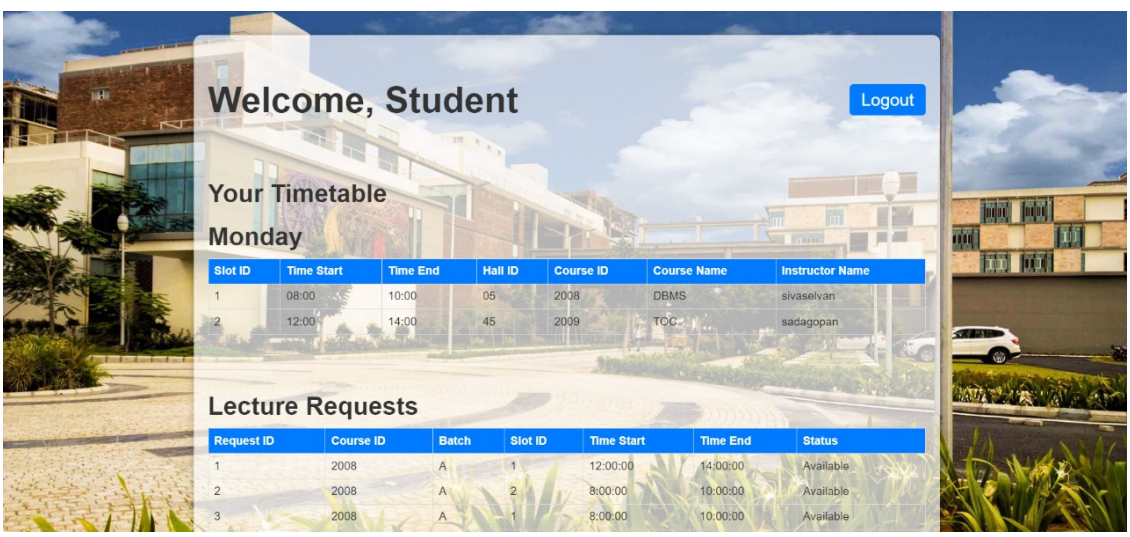

Figure 25: Student Dashboard displaying timetable and lecture requests

```
SQL QUERY : SELECT SLOT.Slot_Id, SLOT.Time_Start, SLOT.Time_End, HALLS.Hall_Id,
COURSE.Course_Id, COURSE.Name AS Course_Name, INSTRUCTOR.Name AS Instructor_Name
FROM ENROLLED
JOIN SLOT ON ENROLLED.Slot_Id = SLOT.Slot_Id
JOIN HALLS ON SLOT.Hall_Id = HALLS.Hall_Id
JOIN COURSE ON ENROLLED.Course_Id = COURSE.Course_Id
JOIN INSTRUCTOR ON COURSE.Instructor_Id = INSTRUCTOR.Instructor_Id
WHERE ENROLLED.Student_Id = %s
```

16.5 Faculty Dashboard

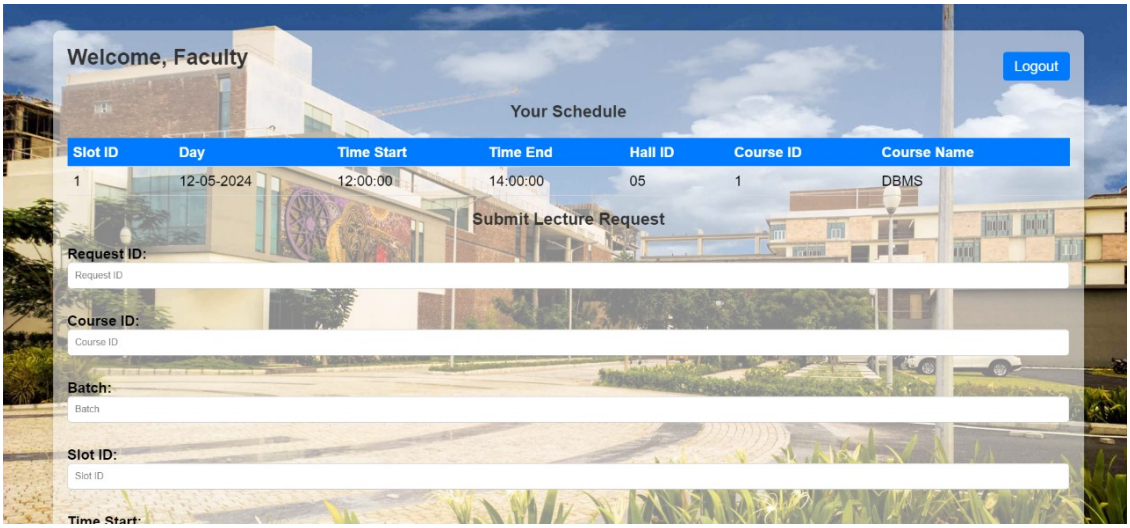

Figure 26: Faculty Dashboard displaying Schedule and Lecture Request Option

```
SQL QUERY : SELECT SLOT.Slot_Id, SLOT.Day, SLOT.Time_Start, SLOT.Time_End,
HALLS.Hall_Id, COURSE.Course_Id, COURSE.Name AS Course_Name,
INSTRUCTOR_REQUEST.Status FROM COURSE JOIN SLOT ON COURSE.Slot_Id = SLOT.Slot_Id
JOIN HALLS ON SLOT.Hall_Id = HALLS.Hall_Id
LEFT JOIN INSTRUCTOR_REQUEST ON COURSE.Course_Id = INSTRUCTOR_REQUEST.Course_Id
AND INSTRUCTOR_REQUEST.Instructor_Id = %s WHERE COURSE.Instructor_Id = %s
```

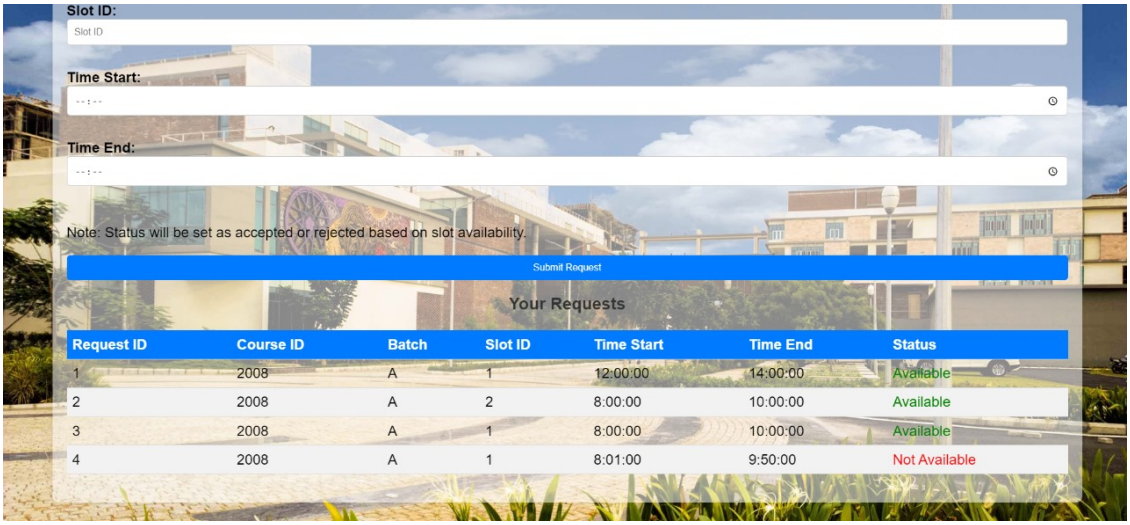

Figure 27: Faculty Dashboard displaying the status of lecture requests

```
SQL QUERY : INSERT INTO INSTRUCTOR_REQUEST (Request_Id, Instructor_Id, Course_Id,
Batch, Slot_Id, Time_Start, Time_End, Status)
VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
```

17 Conclusion

This report has provided a comprehensive explanation of the methodology, implementation details, and technology stack employed in the development of the backend for our Academic Hall and Slot Management System. The backend architecture serves as the cornerstone of the application, ensuring secure user authentication, efficient database interaction, and robust overall application logic. Key features include :

- **Secure User Authentication:** Utilizing Flask-Login, we have implemented a secure and efficient authentication system that ensures only authorized users can access sensitive functionalities. This protects the integrity of the system and user data.
- **Efficient Database Interaction:** With MySQL as our chosen database, we ensure reliable data storage and retrieval. Prepared statements via the mysql.connector library safeguard against SQL injection attacks, maintaining the security and integrity of our data.
- **Dynamic Data Handling:** The backend processes a wide range of requests, from adding and updating faculty information to managing project details and TA associations. This dynamic handling ensures that the system can adapt to various academic scheduling and management needs.
- **Template Rendering with Jinja2:** The use of Jinja2 for template rendering allows us to create dynamic, data-driven web pages. This enhances the user experience by providing real-time updates and intuitive interfaces.
- **Scalability and Flexibility:** The modular design of Flask ensures that our backend can be easily scaled and extended. This makes it adaptable to future requirements, such as adding new features or integrating additional security measures.

Future enhancements for the Academic Hall and Slot Management System could involve integrating advanced security protocols, such as multi-factor authentication, and expanding functionalities to include more detailed reporting, analytics, and automated scheduling algorithms.