STUDENT MANAGEMENT SYSTEM (SMS)

A PROJECT REPORT

Submitted by

Ritik Vishal (24MCA20196)

in partial fulfillment for the award of the degree of

MASTER OF COMPUTER APPLICATION

IN

COMPUTER SCIENCE

Chandigarh University

APRIL, 2025

TABLE OF CONTENT

1.Problem Statement

In the evolving landscape of education, institutions are increasingly tasked with managing large volumes of student information, ranging from personal details to academic performance, enrollment status, and more. Traditionally, many schools and colleges continue to rely on outdated systems such as paper records, handwritten logs, and basic spreadsheet tools for storing and managing this critical information. These traditional methods are often:

- **Error-prone**: Manual data entry leads to increased chances of human error, which can compromise the accuracy of student records.
- **Inefficient**: Searching, updating, or verifying student information manually consumes significant time and administrative effort.
- **Insecure**: Physical files and spreadsheets lack proper access control, increasing the risk of unauthorized access or data breaches.
- **Unscalable**: As the student population grows, these manual methods become increasingly unmanageable and fail to meet institutional demands.
- **Non-collaborative**: Communication between administrative staff and students is hindered when information is scattered or inaccessible in real-time.

These challenges collectively result in delays in academic processes, difficulties in generating timely reports, and reduced productivity for both staff and students. Furthermore, the absence of a centralized platform restricts the ability to maintain uniformity, transparency, and efficiency across departments.

Given this scenario, there is an urgent need for an integrated and automated solution that enables educational institutions to efficiently manage student data. The system must offer secure access, easy management of student records, real-time data retrieval, and a user-friendly interface for administrators and students alike.

This project aims to address these issues by developing a **Student Management System (SMS)** — a web-based application that streamlines the entire process of managing student data. The system will incorporate essential features such as **user authentication, CRUD operations for student records, responsive design, dark mode support, and database integration**, ultimately improving operational efficiency, data accuracy, and administrative communication.

2.Objective

The primary objective of this project is to design and develop a **comprehensive Student Management System (SMS)** that streamlines the management of student information within an educational institution. The system is intended to replace traditional, manual processes with a robust, secure, and scalable digital solution that enhances data accessibility, accuracy, and overall administrative efficiency. The following key objectives outline the functionalities and goals of the project:

1. User Authentication

- To ensure the security and privacy of sensitive student information, the system will include a **user authentication module**.
- This module will facilitate **secure login and registration** for two types of users: **administrators** and **students**.
- Only authenticated users will be able to access or modify the data, protecting the system from unauthorized access.

2. Student Record Management

- The system will enable administrators to **add, edit, delete, and view student records** efficiently.
- Each record will include important personal and academic information such as **student name, ID, email, department, date of birth, and gender**.
- A well-structured user interface will present the data in a clear, organized format, enabling fast navigation and data manipulation.

3. Responsive Design

- The SMS will feature a **responsive web design** that ensures usability across various screen sizes and devices, including desktops, laptops, tablets, and smartphones.
- The goal is to provide a **consistent and intuitive user experience**, regardless of the device being used.

4. Dark Mode Feature

- A **dark mode toggle option** will be implemented to improve user accessibility and visual comfort.
- This feature will allow users to switch between **light and dark themes** based on personal preferences or environmental lighting conditions, enhancing the overall user experience.

5. Database Integration

- The system will use a **MySQL relational database** for efficient storage and retrieval of student data.
- It will support structured queries for data operations, ensuring **data integrity, consistency, and reliability**.
- Proper indexing and relational structures will be maintained to optimize performance, especially as the dataset grows.

6. Error Handling and User Feedback

- Robust **error handling mechanisms** will be incorporated throughout the application to manage potential failures during data entry, authentication, or database operations.
- The system will provide **real-time, user-friendly feedback** messages, guiding users to correct errors and enhancing usability.

7. Future Scalability

- The architecture of the system will be designed with **modularity and scalability** in mind.
- This ensures that the SMS can be easily extended in the future to include **additional modules** (e.g., fee management, academic performance tracking, or messaging).
- The system will also be built to allow **integration with third-party educational tools or platforms**, promoting a broader ecosystem of digital academic services.

3.Introduction

In today's fast-paced digital era, the education sector is undergoing a rapid transformation, with an increasing reliance on technology to enhance institutional efficiency and academic performance. One of the critical aspects of institutional administration is the efficient management of student information, which includes a wide array of data such as personal details, academic performance, enrollment status, and departmental records. Traditionally, many educational institutions have relied on manual or semi-digital methods—such as paper-based records or basic spreadsheets—for handling student data. These conventional systems, however, are often time-consuming, error-prone, difficult to maintain, and lack real-time accessibility, which can hinder the overall productivity of both administrative staff and educators.

The need for a centralized, secure, and user-friendly system to manage student data has become more evident than ever. With increasing student enrollments and growing expectations for administrative transparency and efficiency, educational institutions are seeking modern solutions that streamline day-to-day operations while ensuring data accuracy, security, and accessibility.

To address these challenges, this project introduces a Student Management System (SMS)—a robust, web-based application developed to facilitate the comprehensive management of student information. The system is designed to automate and simplify tasks such as student registration, record updating, and data retrieval, thereby improving the operational workflow of academic institutions.

The SMS platform not only enables administrators to efficiently manage student records but also incorporates modern features such as secure user authentication, a responsive user interface, dark mode support, and error handling mechanisms. With its relational database integration, the system ensures reliable and scalable data storage while maintaining high levels of data integrity and performance.

By implementing this system, educational institutions can reduce administrative burdens, minimize human error, and enhance communication between students and administration. Furthermore, the system's architecture supports future scalability, allowing for the integration of additional modules or third-party tools as institutional needs evolve.

In essence, the Student Management System represents a forward-thinking approach to academic administration—one that embraces digital innovation to meet the growing demands of the educational environment.

4.Input and Output

The **Student Management System (SMS)** is designed to automate and streamline various administrative functions in an educational institution. Each core functionality of the system involves specific inputs and corresponding outputs, ensuring accurate data processing and user feedback. Below is a detailed breakdown of the input and output requirements for each of the major system modules:

1. Add Student

Input:

- **Name**: A non-empty string representing the student’s full name (e.g., "John Doe").
- **Email**: A valid, unique email address used to identify the student (e.g., "john.doe@example.com").
- **Department**: A string denoting the academic department the student belongs to (e.g., "Computer Science").

Output:

- On successful submission:
    - The student record is added to the database.
    - The user is redirected to the **Student List Page (view.jsp)** where the new student is visible.
- On failure (e.g., missing fields, duplicate email, invalid input):
    - An **error message** is displayed on the form page prompting the user to correct the input.

2. Edit Student

Input:

- **ID**: A unique integer value that identifies the student in the database (e.g., 1).
- **Name**: A string with the updated student name (e.g., "John Smith").
- **Email**: A valid and possibly updated email address (e.g., "john.smith@example.com").
- **Department**: A string representing the updated department (e.g., "Information Technology").

Output:

- On successful update:
    - The changes are saved to the database.
    - The user is redirected to the **Student List Page (view.jsp)**, which reflects the updated information.
- On failure (e.g., invalid email format, missing fields):
    - An **error message** is shown on the form, highlighting the issue.

3. Delete Student

Input:

- **ID**: The unique identifier of the student to be deleted (e.g., 1).

Output:

- On successful deletion:
    - The student record is removed from the database.
    - The user is redirected to the **Student List Page (view.jsp)** where the deleted

record is no longer displayed.

- On failure (e.g., invalid ID or database error):
    - An **error message** is displayed to the user.

4. User Registration

Input:

- **Name**: The full name of the registering user (e.g., "Jane Doe").
- **Email**: A valid, unique email address used for user identification (e.g., "jane.doe@example.com").
- **Password**: A string password that meets minimum security requirements (e.g., "securePassword123").

Output:

- On successful registration:
    - The user’s credentials are stored securely in the database.
    - The user is redirected to the **Login Page (login.jsp)**.
- On failure (e.g., email already exists, invalid inputs):
    - An **error message** is shown on the registration form.

5. User Login

Input:

- **Email**: The registered email address of the user (e.g., "jane.doe@example.com").
- **Password**: The password associated with the user account (e.g., "securePassword123").

Output:

- On successful login:
    - A session is created for the user.
    - The user is redirected to the **Student List Page (view.jsp)**.
- On failure (e.g., incorrect credentials):
    - An **error message** is displayed on the login form indicating the reason for failure.

6. User Logout

Input:

- No direct input is required from the user.

Output:

- The current user session is terminated.
- The user is redirected to the **Login Page (login.jsp)**.
- If logout fails due to technical issues, an appropriate **error message** is displayed.

5.Techniques and Technologies Used

The development of the Student Management System (SMS) leverages multiple modern web technologies and software engineering principles to build a robust, scalable, and user-friendly application. Below is an in-depth overview of the key techniques and technologies utilized in the project:

1. JavaServer Pages (JSP)

Description:

JavaServer Pages (JSP) is a server-side technology that enables the creation of dynamically generated web pages based on HTML, XML, or other document types. JSP allows developers to embed Java code directly into HTML pages using special JSP tags, facilitating dynamic content generation and interaction with server-side resources.

Usage in SMS:

JSP is used extensively in this project to create the front-end views for functionalities like adding, editing, and displaying student records. It acts as the presentation layer in the MVC architecture. Form data is collected via JSP forms and passed to Servlets for processing. Additionally, JSP is responsible for rendering database responses back to the user interface.

2. Servlets

Description:

Servlets are Java programs that run on a web server and handle client requests by generating dynamic content, such as HTML. They act as the controllers in a web application, managing the logic between the view and the data model.

Usage in SMS:

Servlets play a central role in controlling the application's behavior. Specific servlets like AddEmployeeServlet, UpdateEmployeeServlet, DeleteEmployeeServlet, LoginServlet, and RegisterServlet handle various business operations. These servlets process form submissions, manage sessions, perform validations, and interact with the database through the DAO layer.

3. Java Database Connectivity (JDBC)

Description:

JDBC is an API in Java that enables applications to connect and execute queries with relational databases. It abstracts the database interaction, providing methods for connecting, executing SQL commands, and processing results.

Usage in SMS:

The project uses JDBC to establish a connection between the application and a **MySQL** database. The DBConnection class handles the setup of the database connection, while the EmployeeDAO class contains methods for Create, Read, Update, and Delete (CRUD) operations on student records. This layer ensures data persistence and retrieval.

4. Model-View-Controller (MVC) Architecture

Description:

The MVC design pattern promotes the separation of concerns by dividing the application into three components:

- **Model:** Manages the data and business logic.
- **View:** Handles the UI and presentation logic.
- **Controller:** Manages input and acts as an intermediary between Model and View.

Usage in SMS:
The SMS project is structured using the MVC architecture:

- **Model:** Represented by the Employee class and DAO (Data Access Object) classes.
- **View:** Implemented using JSP for UI rendering.
- **Controller:** Handled by Servlets, which process requests and direct them to the appropriate view or model layer.

This design improves code modularity, scalability, and maintainability.

5. Responsive Web Design

Description:
Responsive Web Design ensures that web pages look and function well on a variety of devices and screen sizes. It utilizes CSS media queries, flexible grids, and layouts to provide an optimal viewing experience.

Usage in SMS:
The project uses CSS and responsive design principles to make the interface adaptable to different screen sizes. Whether accessed via desktop, tablet, or mobile phone, the layout adjusts fluidly, enhancing usability and accessibility for all users.

6. AJAX (Asynchronous JavaScript and XML)

Description:
AJAX is a client-side technique that enables web pages to communicate with servers asynchronously. It allows content to be updated without reloading the entire page, improving the application's responsiveness.

Usage in SMS:
Although not fully implemented in the current version, AJAX is planned for future enhancements—particularly for features like real-time validation, auto-refreshing student lists, or loading dropdown content dynamically. This can significantly improve the user experience by reducing full page reloads.

7. CSS for Styling

Description:
Cascading Style Sheets (CSS) are used to enhance the appearance of web applications. CSS controls the layout, typography, colors, spacing, and visual transitions.

Usage in SMS:
The system uses custom CSS to style all UI components, including buttons, input forms, navigation, and layout sections. The styling not only improves aesthetics but also improves usability through features like hover effects and visual cues for user interactions.

8. Local Storage for Dark Mode Preference

Description:

Local Storage is part of the Web Storage API that allows websites to store key-value pairs in a user's browser. This data persists even after the browser is closed and reopened.

Usage in SMS:

The SMS incorporates a **Dark Mode toggle** that uses Local Storage to remember the user's theme preference across sessions. Once enabled, the dark theme remains active until the user decides to switch back, improving the overall user experience and accessibility, especially in low-light environments.

9. Error Handling

Description:

Error handling ensures that unexpected conditions (like invalid input or system failures) are managed gracefully without crashing the application. It also provides feedback to users and logs errors for developers.

Usage in SMS:

Throughout the system, robust error handling mechanisms are implemented in Servlets and JSPs. If a form submission fails due to missing or invalid data, the system displays descriptive error messages. Additionally, backend errors are caught and logged, ensuring the application remains stable and user-friendly.

6.System Architecture(MVC)

The Student Management System is designed using the Model-View-Controller (MVC) architectural pattern. This separation of concerns allows for better organization, maintainability, and scalability of the application. Below is a detailed breakdown of each component in the MVC architecture.

1. Model

- Purpose: Represents the data and business logic of the application.
- Components:
    - Employee Class: Represents the employee entity with attributes such as id, name, email, and department.
    - EmployeeDAO Class: Handles database operations related to the Employee entity, including:
        - Retrieving all employees
        - Retrieving a specific employee by ID
        - Adding a new employee
        - Updating an existing employee
        - Deleting an employee
    - DBConnection Class: Manages the database connection using JDBC.

2. View

- Purpose: Represents the user interface of the application, displaying data to the user and capturing user input.
- Components:
    - JSP Files:
        - add.jsp: Form for adding a new student.
        - edit.jsp: Form for editing an existing student.
        - view.jsp: Displays the list of students with options to edit or delete.
        - login.jsp: User login interface.
        - register.jsp: User registration interface.
        - index.jsp: Home page with navigation links.
    - CSS Styles: Stylesheets for enhancing the visual appearance of the application, including responsive design and dark mode support.

3. Controller

- Purpose: Acts as an intermediary between the Model and the View, processing user input and updating the model and view accordingly.
- Components:
    - Servlets:
        - AddEmployeeServlet: Handles the addition of new employees by processing form submissions from add.jsp.
        - UpdateEmployeeServlet: Manages updates to existing employee records from edit.jsp.
        - DeleteEmployeeServlet: Processes deletion requests for employees.
        - LogServlet: Handles user login, validating credentials and managing user sessions.

        - Register: Manages user registration by processing form submissions from register.jsp.

4. Flow of Control

- User Interaction: Users interact with the application through the View (JSP pages).
- Request Handling: When a user submits a form (e.g., adding a student), the request is sent to the appropriate Controller (Servlet).
- Model Update: The Controller processes the request, interacts with the Model (EmployeeDAO) to perform the necessary operations (CRUD).
- Response Generation: After processing, the Controller updates the View (redirects to a JSP page) to reflect the changes (e.g., displaying the updated list of students).

7.Modules Used In The SMS

The Student Management System is structured into several key modules, each responsible for specific functionalities. Below is a breakdown of the modules used in the project:

1. User Interface Module
    - Description: This module handles the presentation layer of the application, providing a user-friendly interface for interaction.
    - Components:
        - JSP Pages:
            - add.jsp: Form for adding new students.
            - edit.jsp: Form for editing existing student details.
            - view.jsp: Displays the list of students with options to edit or delete.
            - login.jsp: User login interface.
            - register.jsp: User registration interface.
            - index.jsp: Home page with navigation links.
        - CSS Styles: Stylesheets for layout, responsiveness, and dark mode support.
        - JavaScript: Scripts for handling user interactions, such as dark mode toggling.
2. Business Logic Module
    - Description: This module contains the core business logic of the application, processing user inputs and managing data.
    - Components:
        - Servlets:
            - AddEmployeeServlet: Handles the addition of new students.
            - UpdateEmployeeServlet: Manages updates to existing student records.
            - DeleteEmployeeServlet: Processes deletion requests for students.
            - LogServlet: Handles user login and session management.
            - Register: Manages user registration.
3. Data Access Module
    - Description: This module is responsible for interacting with the database, performing CRUD operations on student records.
    - Components:
        - DAO Classes:
            - EmployeeDAO: Contains methods for:

        - Retrieving all students.
        - Retrieving a specific student by ID.
        - Adding a new student.
        - Updating an existing student.
        - Deleting a student.
    - DBConnection Class: Manages the database connection using JDBC.

4. Database Module
    - Description: This module defines the database schema and manages data storage.
    - Components:
        - Database Tables:
            - employees: Stores student information (id, name, email, department).
            - users: Stores user information for authentication (name, email, password).

5. Security Module
    - Description: This module ensures secure access to the application, managing user authentication and authorization.
    - Components:
        - Login and Registration: Handles user login and registration processes, including password management.
        - Session Management: Manages user sessions to maintain state across different pages.

8.Constraints

During the development of the Student Management System (SMS), several constraints emerged that influenced design decisions, development processes, and feature implementations. These constraints are categorized into five major types: **Technical**, **Operational**, **User-related**, **Time-based**, and **Budgetary**. Understanding these limitations is essential for evaluating the system's current capabilities and future scalability.

1. Technical Constraints

These constraints are related to the tools, platforms, and technologies used in the development and deployment of the application.

a. Technology Stack Limitation

The project was developed using **Java**, **JavaServer Pages (JSP)**, **Servlets**, and **MySQL**. While this stack offers robustness and platform independence, it restricted the exploration of newer or more flexible frameworks such as Spring Boot, Node.js, or Python-Django that may have provided enhanced development speed, performance optimization, or modularity.

b. Database Performance

The use of **MySQL** posed certain limitations, particularly in terms of query execution speed and indexing when dealing with large datasets. Care had to be taken in designing the database schema and optimizing queries to prevent performance bottlenecks.

c. Browser Compatibility

The application was tested and optimized for modern browsers like **Google Chrome**, **Mozilla Firefox**, and **Microsoft Edge**. However, older browsers (especially Internet Explorer) may not fully support advanced **CSS3** and **JavaScript** features used in the UI, potentially leading to inconsistencies or rendering issues.

d. Server Configuration

The hosting environment for the application required support for **Java EE (Enterprise Edition)**. Limitations in server resources such as **CPU**, **RAM**, or **thread pool size** could impact the application's response time and concurrent user handling, especially under high load conditions or during peak usage.

2. Operational Constraints

Operational constraints are related to the internal policies and processes that affect how the system is implemented, secured, and maintained.

a. User Authentication and Security

Due to the sensitivity of student data, strong **user authentication** and **password security** mechanisms were essential. This included the use of **hashing algorithms** for password storage, secure session management, and input validation—all of which added complexity to the system and extended development time.

b. Data Privacy and Compliance

The system had to comply with data protection policies such as **GDPR (General Data Protection Regulation)**. This imposed constraints on how data could be collected, stored, and accessed, requiring the implementation of features such as **access control**, **encryption**, and **user consent forms**.

c. Scalability Constraints

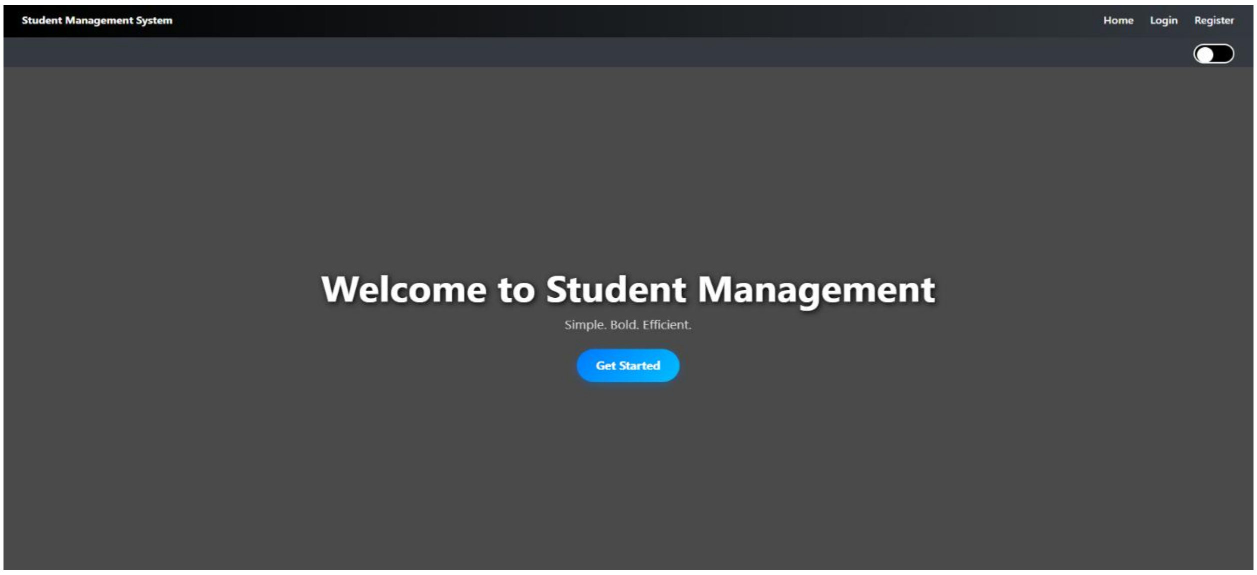
While the architecture is designed with scalability in mind, it is limited by its monolithic design. Scaling horizontally (e.g., distributing the application across multiple servers) would require significant reengineering of the current structure, including session management, load balancing, and database replication.

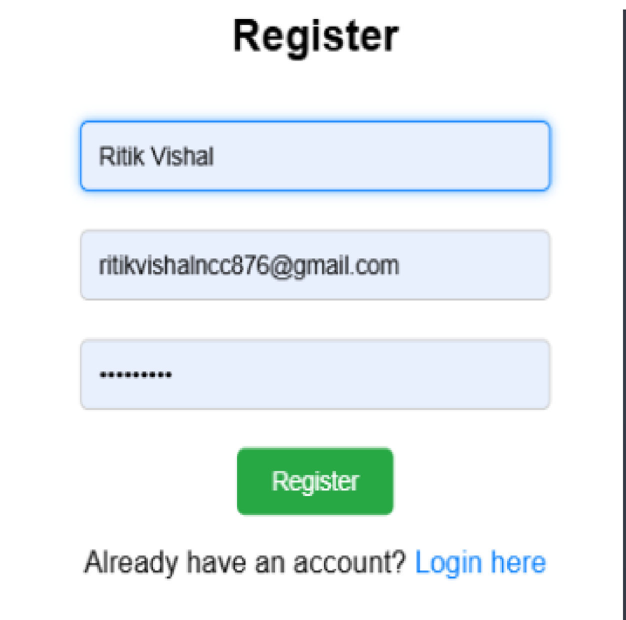
3. User-Related Constraints
These are constraints imposed by the expectations, behavior, and diversity of the users interacting with the system.
a. User Interface Design
The system is intended to serve users with different roles and varying levels of computer literacy, including **students**, **teachers**, and **administrative staff**. Designing an intuitive and clutter-free interface that meets the needs of all users was challenging and required careful UI/UX planning.
b. Accessibility Compliance
The project aimed to comply with **Web Content Accessibility Guidelines (WCAG)** to ensure that users with disabilities (e.g., vision impairments) can use the system. This required thoughtful implementation of **contrast modes**, **keyboard navigation**, **ARIA roles**, and screen reader support, thereby influencing front-end design choices.
c. Training and Documentation
Not all users are tech-savvy; therefore, proper **user onboarding**, **help documentation**, and **training resources** were necessary. Preparing these materials required additional time and effort, which in turn impacted development deadlines.

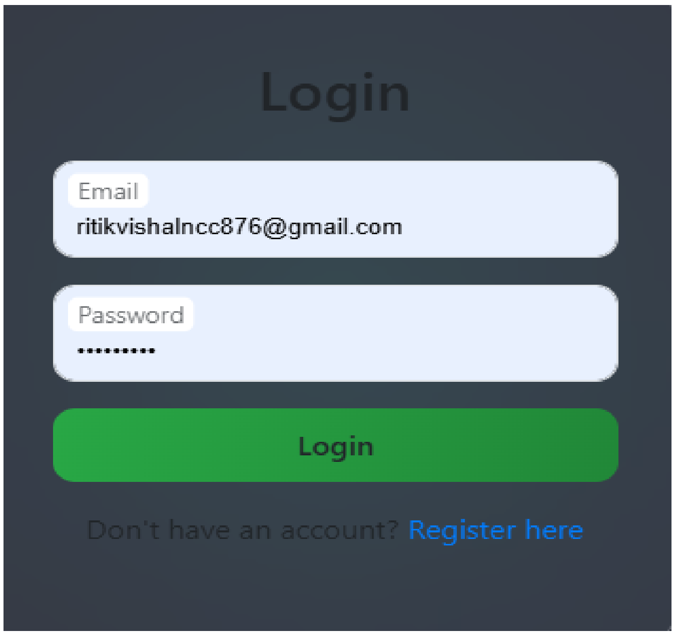
4. Time Constraints
Time was a critical factor that dictated the project’s scope, testing phases, and feature set.
a. Fixed Project Timeline
The project had a pre-defined deadline, which meant that the feature list had to be prioritized. Core functionalities such as **CRUD operations**, **user login**, and **basic UI responsiveness** were completed first, while non-critical or advanced features like **real-time notifications** or **export to Excel/PDF** were deferred for future phases.
b. Limited Testing Window
Due to the compressed timeline, comprehensive testing (unit, integration, and usability testing) had to be scaled down. This increases the risk of minor bugs or performance issues going unnoticed in the production environment.

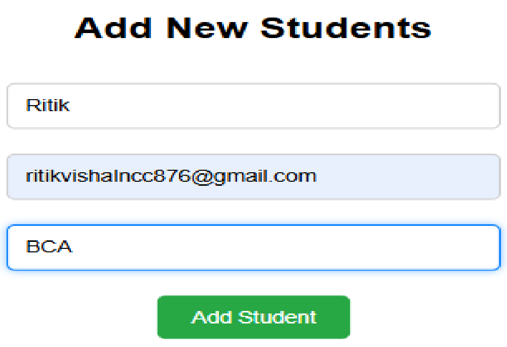
5. Budget Constraints
Financial limitations impacted the scope and scale of resources that could be utilized for the project.
a. Human Resource Limitations
With a limited development budget, the project was executed by a small team (or a single developer, depending on the scenario). This restricted the implementation of advanced features, slowed down development, and reduced the ability to divide work across specialized roles such as UI/UX designers, backend engineers, or testers.
b. Infrastructure and Tools
Due to budget constraints, the project relied on **open-source tools** and **free-tier services**. Advanced tools for testing, monitoring, or continuous integration/deployment (CI/CD) could not be adopted, which might affect long-term maintainability and scalability.

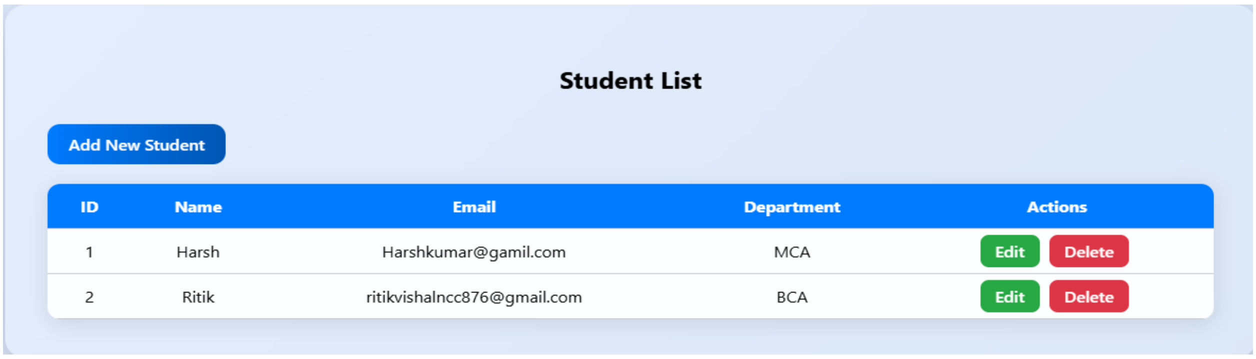
9.ScreenShots




Register
Ritik Vishal
ritikvishalncc876@gmail.com
•••••••••
Register
Already have an account? Login here

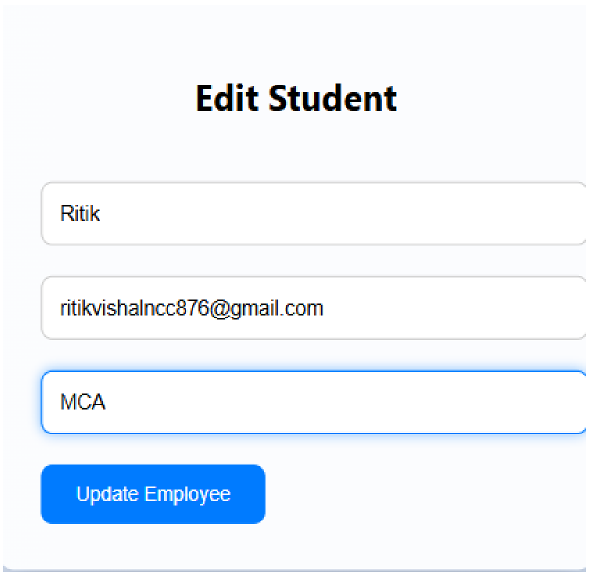

Login
Email
ritikvishalncc876@gmail.com
Password
•••••••••
Login
Don't have an account? Register here





Student List

ID	Name	Email	Department	Actions
1	Harsh	Harshkumar@gamil.com	MCA	Edit Delete
2	Ritik	ritikvishalncc876@gmail.com	BCA	Edit Delete

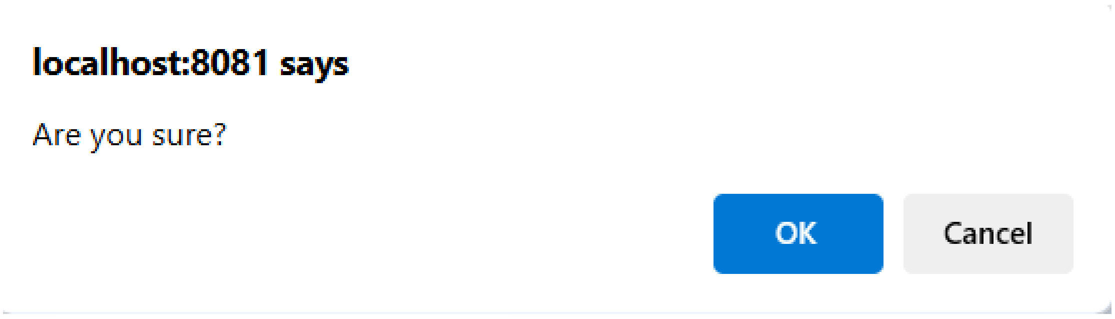


Edit Student

Ritik

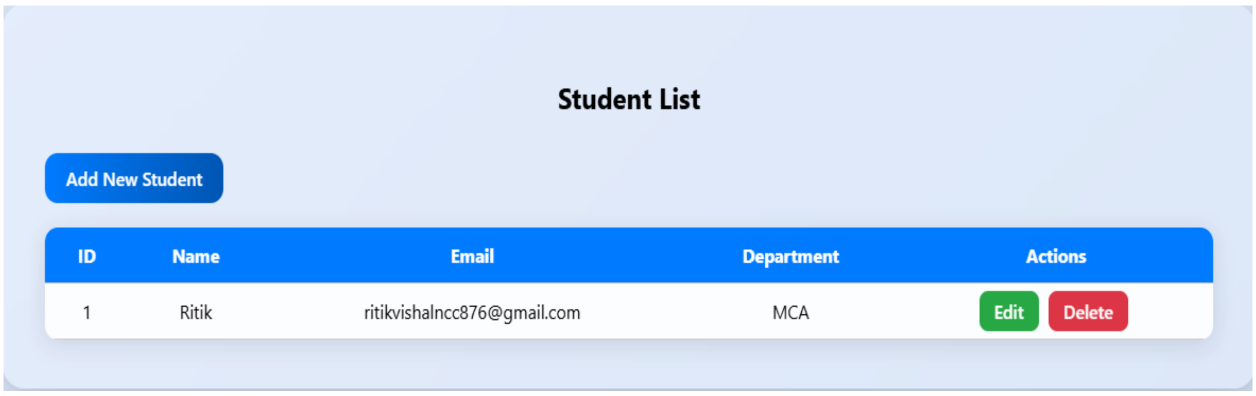
ritikvishalncc876@gmail.com

MCA



localhost:8081 says

Are you sure?



Student List

ID	Name	Email	Department	Actions
1	Ritik	ritikvishalncc876@gmail.com	MCA	Edit Delete

10.References

1. Bootstrap. (n.d.). *Bootstrap documentation*. Retrieved from https://getbootstrap.com/docs/
2. CSS-Tricks. (n.d.). *A complete guide to Flexbox*. Retrieved from https://css-tricks.com/snippets/css/a-guide-to-flexbox/
3. Date, C. J. (2004). *Database design and relational theory: Normal forms and all that jazz*. O'Reilly Media.
4. Eckel, B. (2006). *Thinking in Java* (4th ed.). Prentice Hall.
5. Marcotte, E. (2011). *Responsive web design*. A Book Apart.
6. Mozilla Developer Network (MDN). (n.d.). *JavaScript documentation*. Retrieved from https://developer.mozilla.org/en-US/docs/Web/JavaScript
7. MySQL. (n.d.). *MySQL reference manual*. Retrieved from https://dev.mysql.com/doc/
8. Oracle. (n.d.-a). *Java EE documentation*. Retrieved from https://www.oracle.com/java/technologies/java-ee-glance.html
9. Oracle. (n.d.-b). *Java Servlet specification*. Retrieved from https://jakarta.ee/specifications/servlet/
10. Oracle. (n.d.-c). *JavaServer Pages (JSP) specification*. Retrieved from https://jakarta.ee/specifications/pages/
11. Oracle. (n.d.-d). *Java Database Connectivity (JDBC) API documentation*. Retrieved from https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/
12. OWASP Foundation. (n.d.). *OWASP top ten*. Retrieved from https://owasp.org/www-project-top-ten/
13. Patel, R. (2019). Understanding MVC architecture in web applications. *International Journal of Computer Applications, 182*(12), 1–5. https://doi.org/10.5120/ijca2019918560
14. Smith, J. (2020). Designing an employee management system. *Journal of Software Engineering, 15*(3), 45–60.
15. W3Schools. (n.d.). *HTML tutorial*. Retrieved from https://www.w3schools.com/html/