
Software Requirements Specification

for

E-LEARNING PLATFORM

Version 1.0 approved

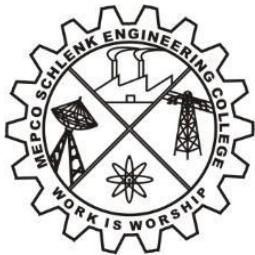
Prepared by SIVA PRASANTH M

KARNASSAGAR S

RAMKUMAR K

MEPCO ADS

28/02/2024

MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI**AUTONOMOUS****DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE****BONAFIDE CERTIFICATE**

Certified that this project report “**E-LEARNING PLATFORM**” is the bonafide work of **KARNASSAGAR S (202109026), SIVA PRASANTH M(202109049), RAMKUMAR K (202109253)** who carried out the project work under my supervision.

SIGNATURE

**Dr.P.Thendral BE, ME, PhD
Associate Professor**

Artificial Intelligence and Data Science
Mepco Schlenk Engineering College,
Sivakasi- 626005, Virudhunagar

SIGNATURE

**Dr.J.Angela Jennifa Sujana, BE, MTech, PhD
Professor & Head**

Artificial Intelligence and Data Science
Mepco Schlenk Engineering College
Sivakasi- 626005, Virudhunagar

The project report submitted for the viva voce held on

Table of Contents

Table of Contents	3
Revision History	3
1. Introduction	4
1.1.Purpose	4
1.2.Document Conventions	4
1.3.Intended Audience and Reading Suggestions	4
1.4.Product Scope.....	5
2. Overall Description	5
2.1.Product Perspective.....	5
2.2.Product Functions.....	6
2.3.User Classes and Characteristics.....	6
2.4.Operating Environment	7
2.5.Design and Implementation Constraints	7
2.6.User Documentation.....	8
2.7.Assumptions and Dependencies.....	9
3. External Interface Requirements.....	9
3.1.User Interfaces.....	9
3.2.Hardware Interfaces	10
3.3.Software Interfaces.....	10
3.4.Communications Interfaces.....	10
4. System Features	10
4.1.System Feature 1	10
5. Other Nonfunctional Requirements.....	11
5.1.Performance Requirements	11
5.2.Safety Requirements.....	11
5.3.Security Requirements.....	11
5.4.Software Quality Attributes.....	12
5.5.Business Rules	12
6. Black box Testing.....	12
Appendix A: Glossary	12
Appendix B: Analysis Models	13
Appendix C: Code.....	19
Appendix D: Complexity Analysis.....	45
Appendix E: Screenshots.....	46
Appendix F:Project Management	82

Revision History

Name	Date	Reason For Changes	Version
Review - I	06/03/24		1.0.0
Review - II	17/04/24		2.0.0

1. Introduction

1.1 Purpose

The purpose of the E-Learning Platform project is to provide an online environment for educational institutions to facilitate teaching and learning activities. In today's digital era, the demand for flexible and accessible learning solutions has grown significantly. This platform aims to bridge the gap between traditional classroom settings and modern online learning by offering a comprehensive suite of tools and features. It is designed to cater to the specific needs of administrators, teachers, and students by offering role-based functionalities.

1.2 Document Conventions

Bold: Headings, subheadings, and emphasis.

Italics: Terms defined in the glossary and references.

Monospace: Code snippets, commands, or system outputs.

1.3 Intended Audience and Reading Suggestions

This document is intended for the development team, project stakeholders, quality assurance team, and end-users including administrators, teachers, and students. The reading suggestions are provided in the document for different audience groups. Developers can refer to the provided source code for implementation details.

1.4 Product Scope

The E-Learning Platform will have three login roles: Admin, Teacher, and Student. Each role will have specific functionalities to perform various tasks related to course management, attendance, assessments, and communication. The platform will utilize MongoDB for data storage and retrieval.

1.5 References

E-Learning Platform Vision and Scope Document, Version 1.0, Author: [Insert Author Name], Date: [Insert Date], Location: [Insert Location]

E-Learning Platform User Interface Style Guide, Version 1.0, Author: [Insert Author Name], Date: [Insert Date], Location: [Insert Location]

E-Learning Platform System Requirements Specification, Version 1.0, Author: [Insert Author Name], Date: [Insert Date], Location: [Insert Location]

E-Learning Platform Use Case Document, Version 1.0, Author: [Insert Author Name], Date: [Insert Date], Location: [Insert Location]

2. Overall Description

2.1 Product Perspective

The E-Learning Platform is a standalone system designed to facilitate online education. It will interact with users through a web-based interface. As an integral part of the educational ecosystem, the platform aims to complement traditional teaching methods by providing a flexible and accessible online learning environment. It will support seamless integration with existing Learning Management Systems (LMS) and educational tools, ensuring a cohesive and enhanced learning experience for all users.

2.2 Product Functions

Admin:

Create Classes: Set up and organize virtual classrooms for different courses.
Create Courses: Define course details, syllabus, and learning objectives.
Manage Teachers and Students: Add, remove, or modify user profiles, assign roles, and manage user permissions.
Take Attendance: Record and track student attendance for each class.
Publish Notices: Share important announcements, updates, and notifications with teachers and students.

Teacher:

Take Attendance: Mark and manage student attendance during classes.
Provide Marks: Evaluate student performance, assign grades, and share feedback.
Upload Materials: Upload lecture notes, presentations, assignments, and additional learning resources.
Upload Quizzes: Create, upload, and manage quizzes and assessments.
Publish Notices: Communicate with students by sharing class schedules, updates, and reminders.

Student:

View Marks and Attendance: Access and review their academic performance, attendance records, and grades.
Download Materials: Download course materials, lecture notes, and additional resources shared by teachers.
Attend Quizzes: Participate in quizzes and assessments as per the course curriculum.
Raise Complaints: Lodge complaints or concerns related to course content, assessments, or any other issues.

2.3 User Classes and Characteristics

Admin:

Role: Administrator

Characteristics:

Manages the overall system and course administration.

Responsible for user management, course creation, and system configuration.

Teacher:

Role: Instructor or Educator

Characteristics:

Conducts classes, assessments, and interactive sessions.

Uploads course materials, evaluates student performance, and provides academic support.

Student:

Role: Learner or Participant

Characteristics:

Engages in learning activities, attends virtual classes, and accesses course materials.

Participates in assessments, quizzes, and collaborative learning activities.

2.4 Operating Environment

The platform will operate in a web-based environment compatible with modern web browsers such as Google Chrome, Mozilla Firefox, Safari, and Microsoft Edge. It will require a stable internet connection to ensure uninterrupted access and optimal performance. The platform will also be accessible on various devices, including desktop computers, laptops, tablets, and smartphones, providing users with the flexibility to learn anytime, anywhere.

2.5 Design and Implementation Constraints

Security Measures:

Implementation of robust security protocols to protect user data and prevent unauthorized access.

Encryption of sensitive information and secure data transmission to safeguard user privacy and confidentiality.

Scalability:

Design and architecture of the platform to accommodate a large number of users, courses, and concurrent sessions.

Optimization of database queries and server resources to ensure efficient performance and scalability.

Compatibility:

Development of a responsive and adaptive design to ensure compatibility with various devices, screen sizes, and resolutions.

Cross-browser compatibility testing and optimization to provide a consistent and seamless user experience across different web browsers. We have used Basic model-Organic for our project.

$$a = 2.4$$

$$b = 1.05$$

$$c = 2.5$$

$$d = 0.3$$

$$E = a * (KLOC) ^ b \text{ person-months.}$$

$$E = 2.4 * (150) ^ 1.05$$

$$E = 462.5 \text{ person-months.}$$

$$T = c * (E) ^ d \text{ months.}$$

$$T = 2.5 * (462.5) ^ 0.38$$

$$T = 25.74 \text{ months.}$$

$$\text{No of Persons required} = E/T \text{ Persons.}$$

$$\text{No of Persons required} = 462.5/25.74$$

$$\text{No of Persons required} = 18 \text{ Persons.}$$

2.6 User Documentation

User documentation will include user manuals, online help, and tutorials. The delivery format will be web-based and accessible within the platform.

2.7 Assumptions and Dependencies

Assumed factors include stable internet connectivity for users.

Dependencies include third party libraries for certain functionalities.

3. External Interface Requirements

3.1 User Interfaces

Intuitive Web-Based Interfaces:

Admin Interface:

Dashboard displaying an overview of system status, user management, course creation, and other administrative functions.

Navigation menu for easy access to various functionalities like class management, course creation, user management, and system settings.

Interactive forms and input fields for data entry, modification, and deletion.

Real-time notifications and alerts for important updates and announcements.

Teacher Interface:

Dashboard providing an overview of classes, student attendance, course materials, and assessments.

Navigation menu for accessing attendance management, grade submission, material upload, and communication tools.

User-friendly interfaces for uploading course materials, managing quizzes, and interacting with students.

Real-time notifications for attendance, assessments, and communication from administrators or students.

Student Interface:

Dashboard displaying course enrollments, attendance records, grades, and available course materials.

Navigation menu for accessing course details, attendance records, material downloads, and communication tools.

User-friendly interfaces for viewing and downloading course materials, attending quizzes, and raising complaints.

Real-time notifications for course updates, grades, and communication from teachers or administrators.

Consistent Layout and Navigation:

Standardized design elements, such as headers, footers, and navigation bars, to maintain consistency across different pages and user roles.

Responsive and adaptive design to ensure compatibility and optimal viewing experience on various devices and screen sizes.

3.2 Hardware Interfaces

No Specific Hardware Interfaces Required:

The platform is designed to operate on standard computing devices, including desktop computers, laptops, tablets, and smartphones, without the need for specialized hardware interfaces.

3.3 Software Interfaces

Integration with Authentication Systems:

Integration with Single Sign-On (SSO) systems or third-party authentication services for secure and seamless user login.

Implementation of OAuth or JWT for secure authentication and authorization processes.

Database Interfaces:

Integration with MongoDB for efficient and scalable data storage and retrieval.

Implementation of CRUD (Create, Read, Update, Delete) operations for managing user data, course details, attendance records, and other relevant information.

3.4 Communications Interfaces

Email Notifications:

Automated email notifications for important events, such as course updates, notices, and announcements.

Customizable email templates and scheduling options for sending notifications to administrators, teachers, and students.

Real-Time Communication Tools:

In-platform chat or discussion forums for facilitating real-time communication and collaboration among users.

Integration with messaging APIs or platforms, such as WebSocket, to enable seamless and interactive communication within the platform.

4. System Features

4.1 Admin Features

Create Classes and Courses:

Add New Class: Admin can create new classes by specifying the class name, subject, time, and other relevant details.

Manage Courses: Admin can create, edit, or delete courses by providing course details such as course name, description, department, and associated class.

Manage Teachers and Students:

Add New Teacher/Student: Admin can add new teachers or students by entering their personal and contact details.

Edit/Delete Teacher/Student: Admin can modify or remove existing teacher or student profiles as needed.

Take Attendance:

Class-wise Attendance: Admin can mark and maintain attendance records for each class and view attendance reports.

Generate Attendance Reports: Admin can generate and download detailed attendance reports for teachers and students.

Publish Notices:

Create Notices: Admin can create and publish notices regarding important updates, announcements, or events.

Notify Users: Admin can send notifications to teachers and students via email or in-platform alerts.

4.2 Teacher Features

Take Attendance:

Mark Attendance: Teachers can mark and update attendance records for their respective classes.

View Attendance Reports: Teachers can view and monitor attendance reports for students in their classes.

Provide Marks:

Enter Marks: Teachers can enter and update assessment marks for students.

Generate Grade Reports: Teachers can generate and share grade reports with students and administrators.

Upload Materials:

Upload Course Materials: Teachers can upload and manage course materials such as lecture notes, assignments, and reference materials.

Categorize Materials: Teachers can categorize and organize uploaded materials for easy access by students.

Upload Quizzes:

Create Quizzes: Teachers can create and upload quizzes with questions, options, and correct answers.

Schedule Quizzes: Teachers can schedule quizzes and notify students about the quiz date, time, and duration.

Publish Notices:

Share Updates: Teachers can publish notices to update students about class schedules, assignment deadlines, or other important information.

Notify Students: Teachers can send notifications to students via email or in-platform alerts.

4.3 Student Features

View Marks and Attendance:

Check Grades: Students can view their assessment marks and grades for each course.

Access Attendance Records: Students can check and monitor their attendance records for each class.

Download Materials:

Access Course Materials: Students can download course materials uploaded by teachers, such as lecture notes, assignments, and reference materials.

Search and Filter Materials: Students can search, filter, and access specific course materials based on course name, subject, or date.

Attend Quizzes:

Take Quizzes: Students can attend quizzes scheduled by teachers, answering questions and submitting their responses within the specified time.

View Quiz Results: Students can view quiz results, including scores and correct answers, upon quiz completion.

Raise Complaints:

Submit Complaints: Students can raise complaints or concerns regarding classes, assessments, or other platform-related issues.

Track Complaint Status: Students can track the status of their complaints and view responses from administrators or teachers.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

Platform Responsiveness:

The platform should load quickly, with a maximum load time of 2-3 seconds for all pages and functionalities.

The system should respond to user actions promptly, with a response time of less than 1 second for basic operations.

Concurrent User Support:

The platform should support a minimum of 1000 concurrent users to ensure smooth operation during peak times.

Load balancing mechanisms should be in place to distribute incoming traffic and maintain optimal performance.

5.2 Safety Requirements

Data Encryption:

All user data transmitted over the platform should be encrypted using secure encryption algorithms such as AES (Advanced Encryption Standard).

Data stored in the database should be encrypted to protect against unauthorized access and data breaches.

Access Control Measures:

Role-based access control (RBAC) should be implemented to restrict access to sensitive information based on user roles (Admin, Teacher, Student).

Two-factor authentication (2FA) should be enabled for enhanced security during login.

5.3 Security Requirements

User Authentication Mechanisms:

Secure user authentication methods such as password hashing, salting, and multi-factor authentication (MFA) should be implemented.

Password policies, including minimum length, complexity, and expiration, should be enforced for user accounts.

Regular Security Audits and Updates:

Regular security audits should be conducted to identify and address potential vulnerabilities. The platform should be updated with the latest security patches and updates to protect against new and emerging threats.

5.4 Software Quality Attributes

Usability:

Intuitive and user-friendly interfaces should be designed for easy navigation and accessibility across different devices and screen sizes.

Clear and concise instructions, tooltips, and help guides should be provided to assist users in using the platform effectively.

Reliability:

The system should be stable and available 24/7, with a minimum uptime of 99.9%.

Automated backup and recovery mechanisms should be in place to ensure data integrity and availability in the event of system failures or data loss.

Maintainability:

The platform should be modular and scalable, allowing for easy updates, modifications, and maintenance of system components.

Documentation and code commenting should be maintained to facilitate understanding and modification of the system by developers and administrators.

5.5 Business Rules

Admin Role:

Admins can only manage courses, classes, and users within their assigned roles and departments.

Admins should have the ability to assign and manage roles for teachers and students.

Teacher Role:

Teachers can only access and manage courses, classes, and students they are assigned to.

Teachers should have the ability to upload and manage course materials, quizzes, and assessments.

Student Role:

Students can only access and view courses, classes, and materials they are enrolled in.

Students should have the ability to view their grades, attendance, and raise complaints or concerns to teachers or administrators.

6. Blackbox Testing

Three major modules:

- Admin
- Teacher
- Student

We are going to test each module by unit-testing and then we will check integration-testing of the module by combining those functionalities with other modules.

Testing of Admin Module:

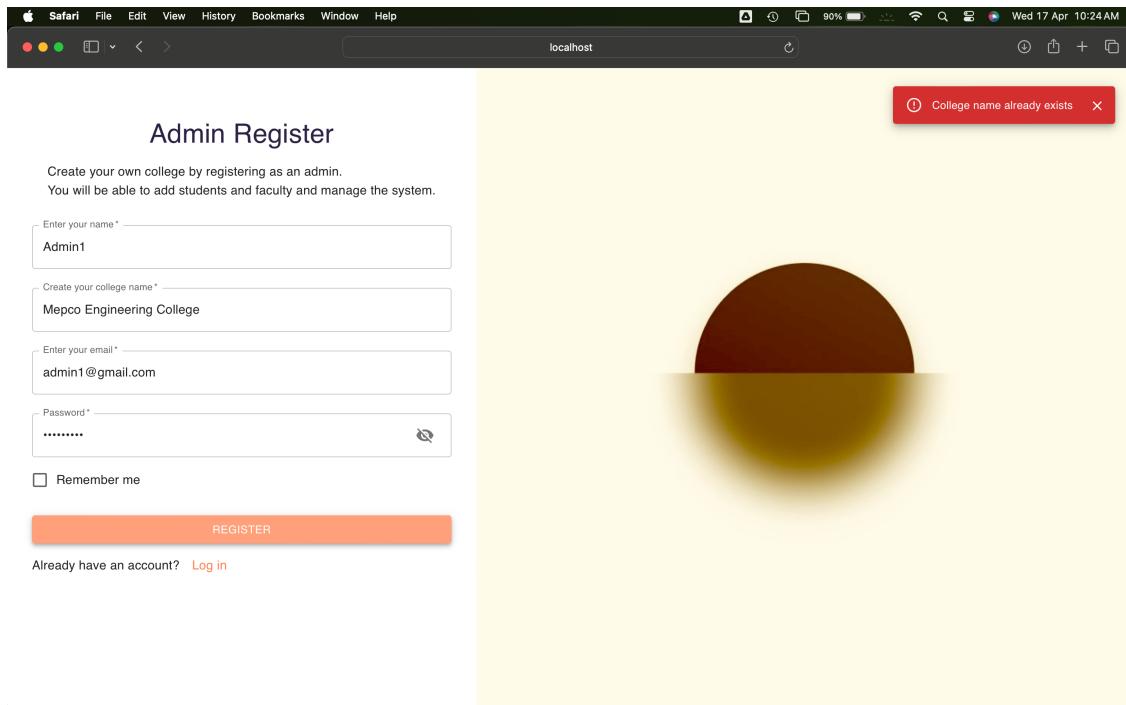
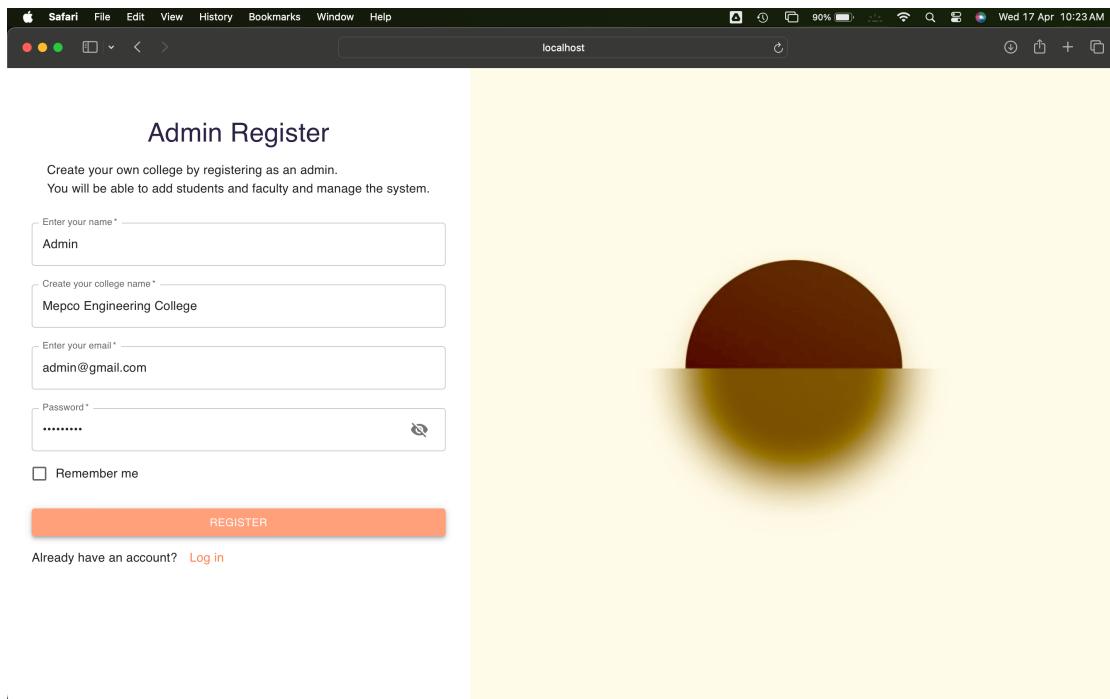
List of Modules:

- Signup Page
- Login Page
- Admin SideBar
- Dashboard
- Class Creation
- Course Creation
- Teacher Login Id creation
- Student Login Id creation
- Notice
- Complain
- Logout

Unit Testing:

Signup Page:

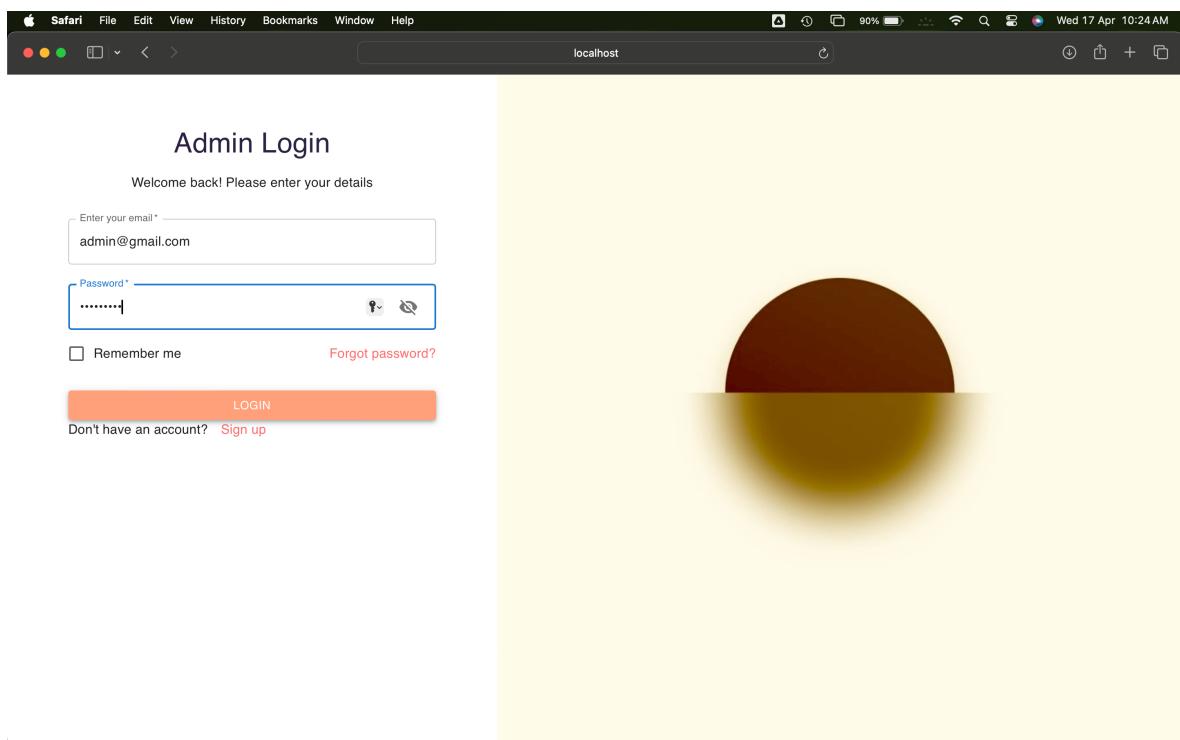
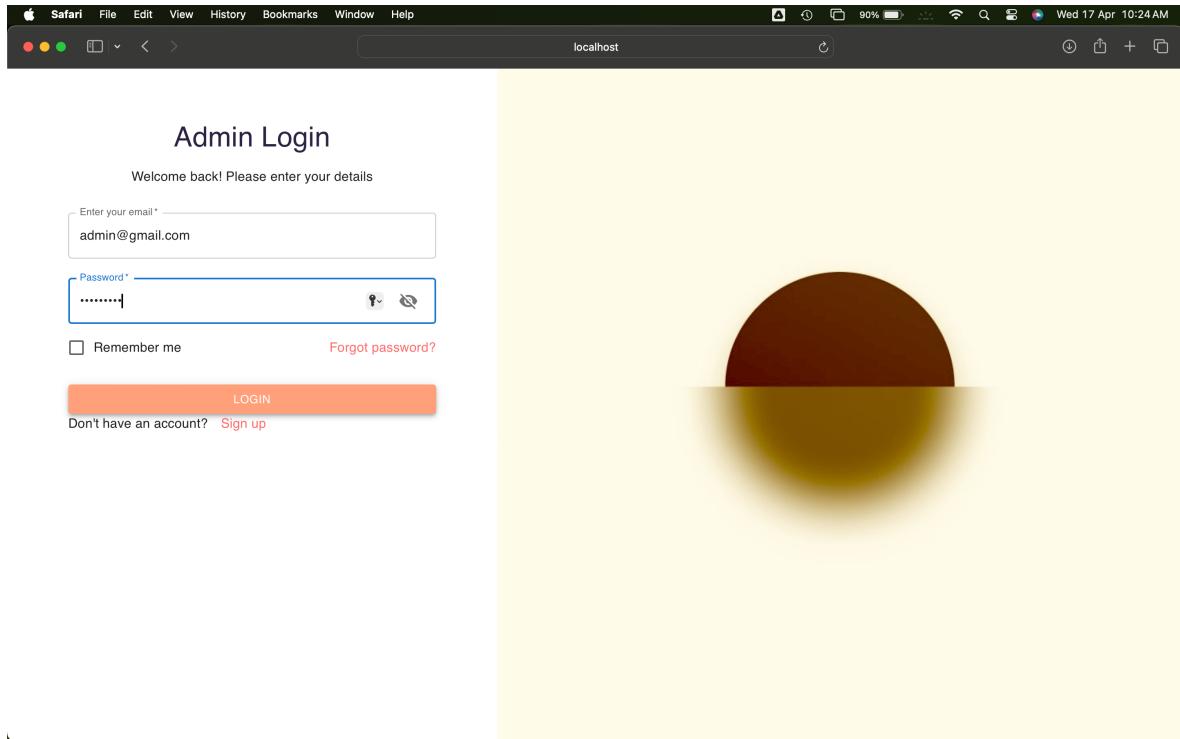
Test case: Check for Signup page loading.



Conclusion: Test case passed as we cannot use a registered college name for a new admin id.

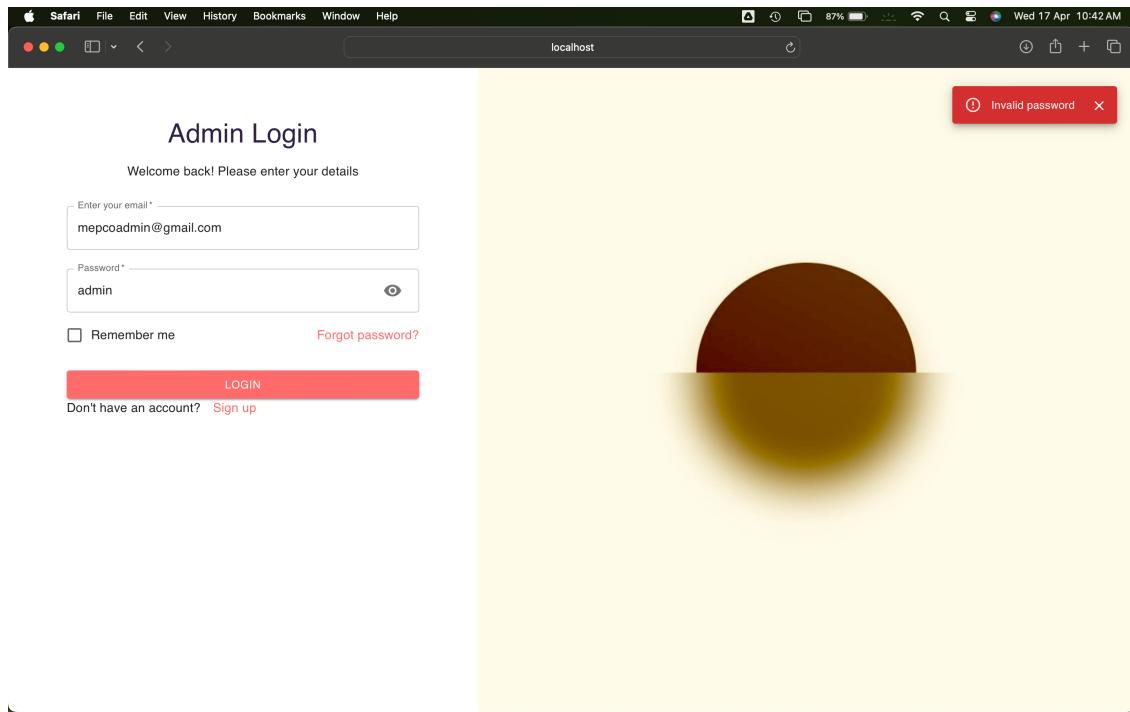
Login Page:

Test case: Login page loading



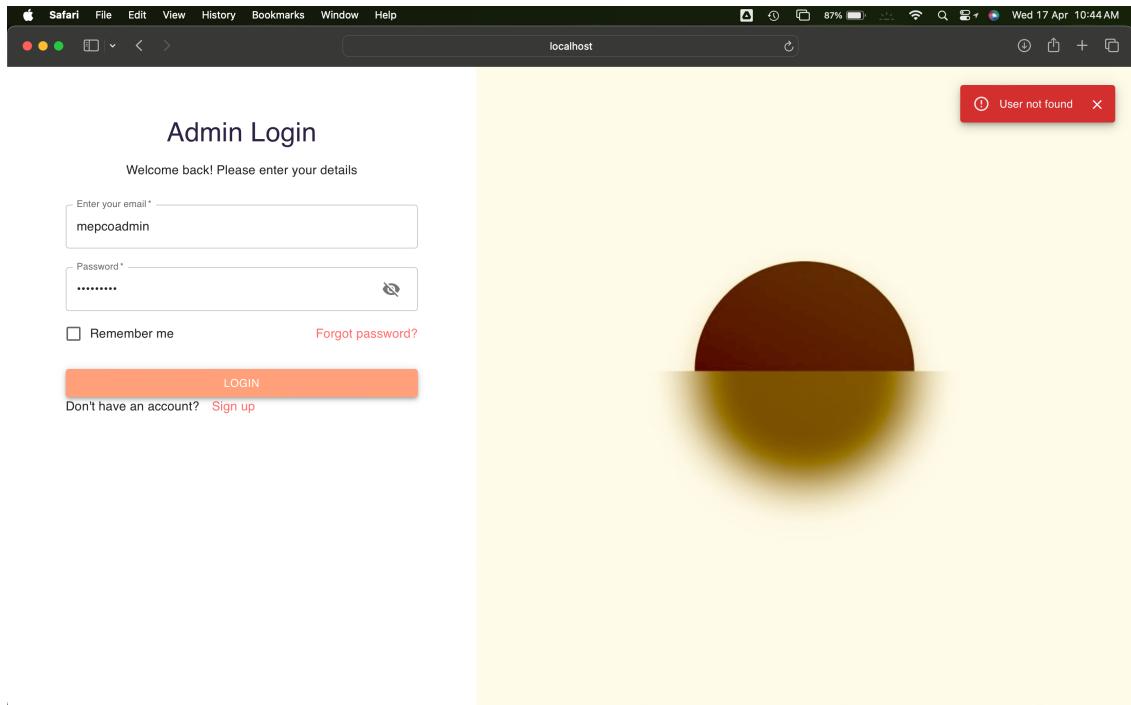
Conclusion: Test case passed. Login page loaded successfully.

Test case: Check for password validation



Conclusion: Test case passed. Validates password successfully.

Test case: Check for mail validation



Conclusion: Test case passed. Validates mail successfully.

Admin Sidebar:

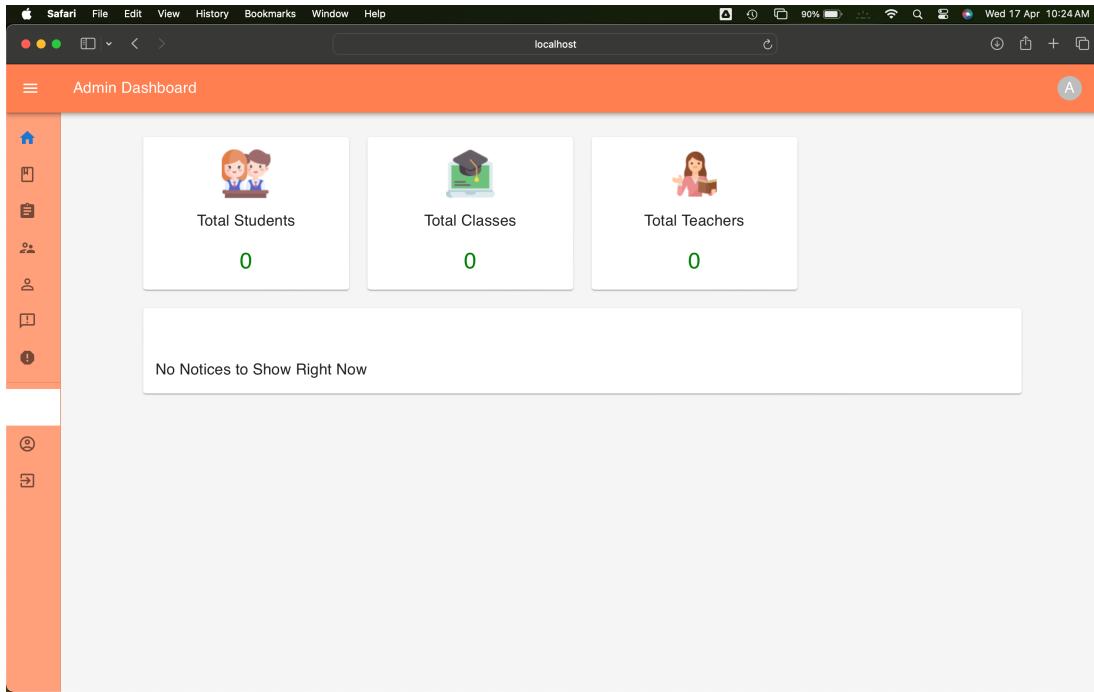
Test case: Check for loading of Admin NavBar



Conclusion: Test case passed. Admin NavBar loaded successfully.

Dashboard:

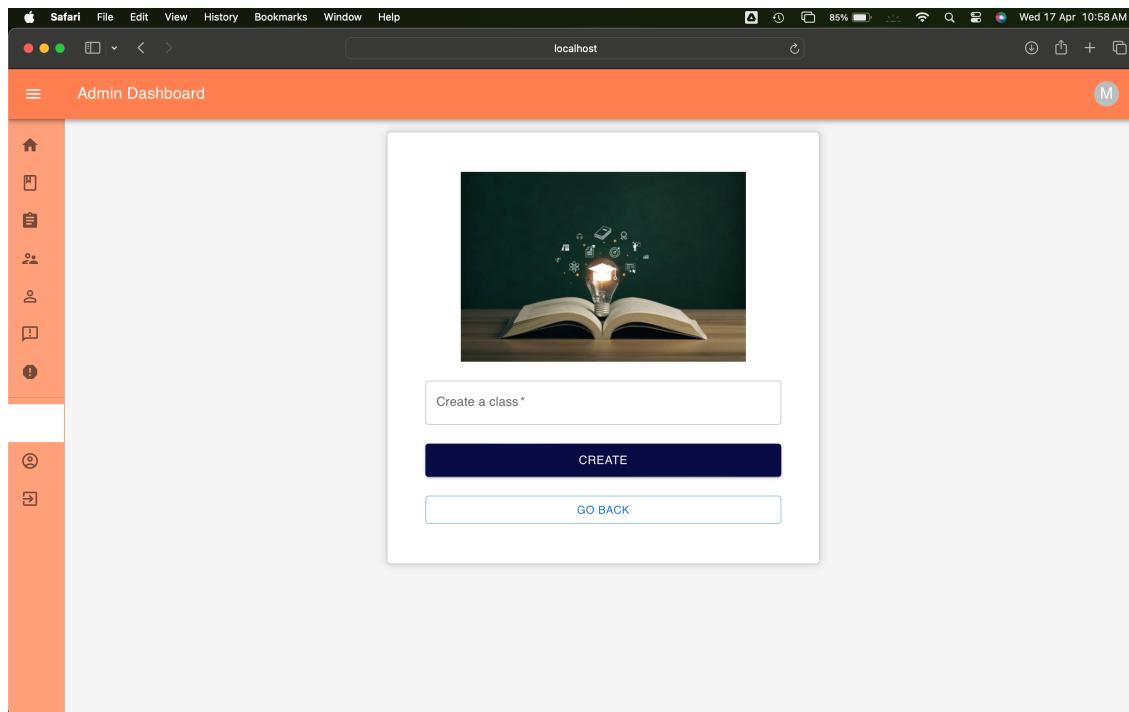
Test case: Check for Admin Dashboard loading.



Conclusion: Test case passed. Admin Dashboard loaded successfully.

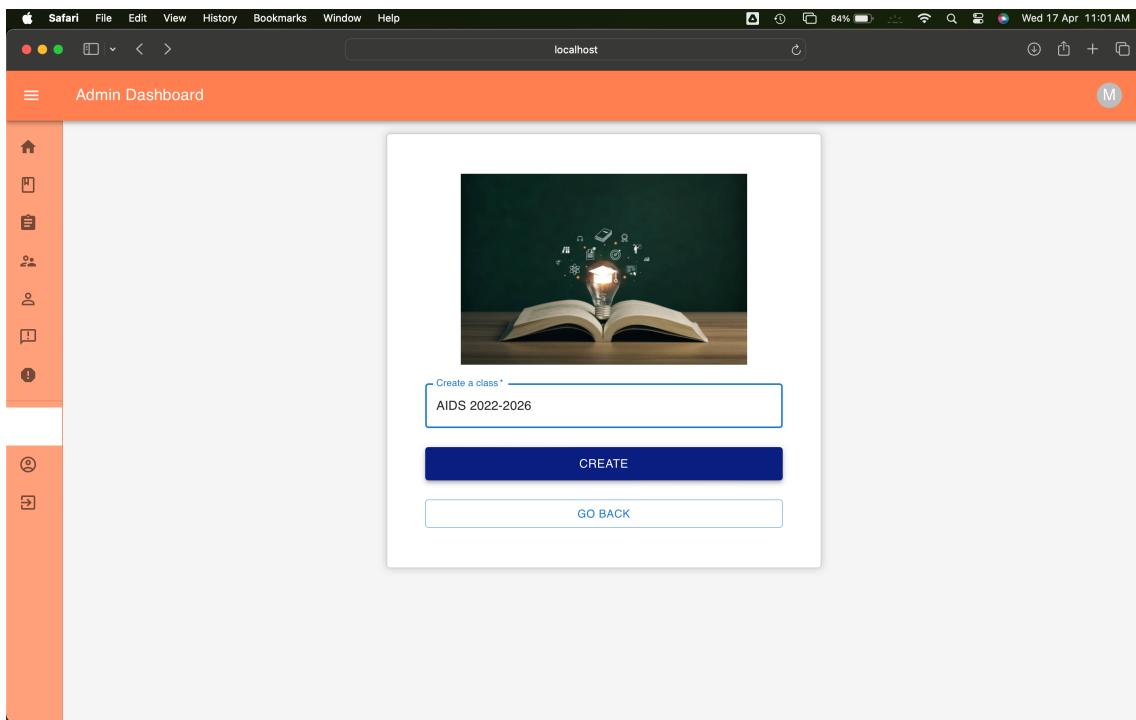
Class Creation:

Test case: Check for class creation page loading.



Conclusion: Test case passed. Class creation page loaded successfully.

Test case: Check for class creation successful.

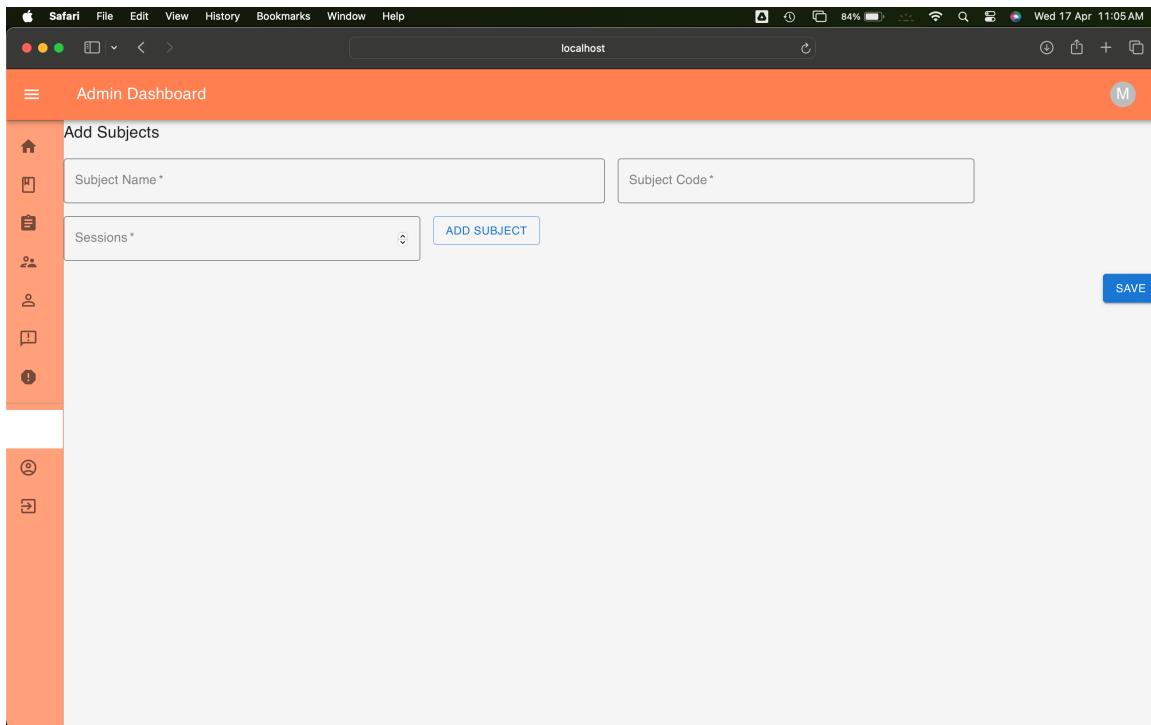


A screenshot of a web browser window titled "Admin Dashboard". The browser's address bar shows "localhost". The main content area displays "Class Details" for the class "AIDS 2022-2026". It shows statistics: "Number of Subjects: 0" and "Number of Students: 0". At the bottom are two buttons: "ADD STUDENTS" and "ADD SUBJECTS". The interface is identical to the previous screenshot, with an orange sidebar on the left.

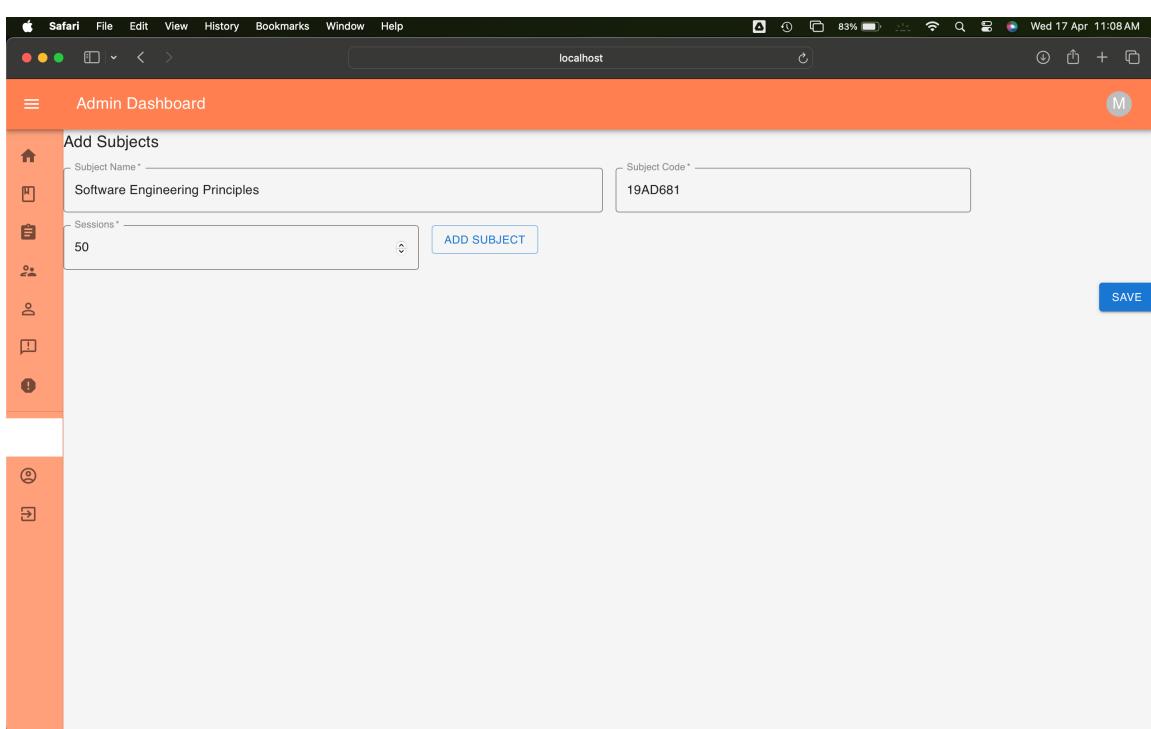
Conclusion: Test case passed. Class creation successful.

Course Creation:

Test case: Check for course creation page loading.



Conclusion: Test case passed. Course creation page loading successful.
Test case: Check for course creation successful.



The screenshot shows the Admin Dashboard interface. On the left is a vertical sidebar with orange icons for Home, Books, Classes, Students, Courses, Notifications, and Help. The main area has a header "Admin Dashboard". Below it is a table with three columns: Sub Name, Sessions, and Class. Two rows are present:

Sub Name	Sessions	Class
Machine Learning Principles	45	ADS semester-5(2021-2025)
Software Engineering Principles	50	AIDS 2022-2026

Actions column contains "VIEW" buttons for each row. At the bottom right of the table are buttons for "Rows per page" (5), "1-2 of 2", and a refresh icon.

Conclusion: Test case passed. Course creation successful.

Test case: Check for multiple course creation.

The screenshot shows the Admin Dashboard with the "Add Subjects" section highlighted. The sidebar on the left is identical to the previous screenshot. The main area has a header "Add Subjects". It contains two sets of input fields for each course:

Subject Name *	Subject Code *
Computer Networking Principles	19AD601
Operating System Principles	19AD601

Each set includes a "Sessions" input field (50) and a "REMOVE" button. A large blue "ADD SUBJECT" button is positioned between the two sets. At the bottom right is a large blue "SAVE" button.

	Sub Name	Sessions	Class	Actions
	Machine Learning Principles	45	ADS semester-5(2021-2025)	
	Software Engineering Principles	50	AIDS 2022-2026	
	Computer Networking Principles	50	AIDS 2022-2026	
	Operating System Principles	50	AIDS 2022-2026	

Conclusion: Test case passed. Multiple course creation successful.

Test case: Check for deletion of a course.

	Sub Name	Sessions	Class	Actions
	Machine Learning Principles	45	ADS semester-5(2021-2025)	
	Software Engineering Principles	50	AIDS 2022-2026	
	Computer Networking Principles	50	AIDS 2022-2026	
	Operating System Principles	50	AIDS 2022-2026	

	Sub Name	Sessions	Class	Actions
	Software Engineering Principles	50	AIDS 2022-2026	
	Computer Networking Principles	50	AIDS 2022-2026	
	Operating System Principles	50	AIDS 2022-2026	

Rows per page: 5 ▾ 1–3 of 3 < >

Conclusion: Test case passed. Course deletion successful.

Teacher Login ID Creation:

Test case: Check for Teacher login id creation successful.

	DETAILS	STUDENTS
Subject Details		
Subject Name : Computer Networking Principles		
Subject Code : 19AD601		
Subject Sessions : 50		
Number of Students: 0		
Class Name : AIDS 2022-2026		
ADD SUBJECT TEACHER		

The screenshot shows the 'Admin Dashboard' interface in a web browser. On the left, there is a vertical orange sidebar with various icons for navigation. The main content area has a light gray background. At the top center, it says 'Add Teacher'. Below that are three input fields: 'Name *' (with a placeholder 'John Doe'), 'Email *' (placeholder 'john.doe@example.com'), and 'Password *' (placeholder 'password123'). Below these fields, there are two small lines of text: 'Subject: Computer Networking Principles' and 'Class: AIDS 2022-2026'. At the bottom center is a blue 'REGISTER' button.

Conclusion: Test case passed. Teacher Id creation successful.

Test case: Check for Teacher Login Id deletion.

The screenshot shows the 'Admin Dashboard' interface in a web browser. The left sidebar is orange with various icons. The main content area displays a table with teacher information. The columns are labeled 'Name', 'Subject', 'Class', and 'Actions'. There are two rows of data:

Name	Subject	Class	Actions
Thendral P	ADD SUBJECT	ADS semester-5(2021-2025)	VIEW
Swathika	Computer Networking Principles	AIDS 2022-2026	VIEW

At the bottom right of the table, there are pagination controls: 'Rows per page: 5', '1–2 of 2', and a refresh/circular arrow icon.

The screenshot shows the 'Admin Dashboard' interface in a web browser. On the left is a vertical orange sidebar with icons for Home, Books, Students, Teachers, and Notifications. The main area has a light gray background with an orange header bar containing the title 'Admin Dashboard' and a user icon.

Add Teacher

Name *: Swathika

Email *: swathika@gmail.com

Password *:|

Subject: Computer Networking Principles
Class: AIDS 2022-2026

REGISTER

The screenshot shows the 'Admin Dashboard' interface in a web browser. The sidebar on the left is identical to the previous screenshot. The main area displays a table of teacher information.

Name	Subject	Class	Actions
Thendral P	ADD SUBJECT	ADS semester-5(2021-2025)	VIEW
Swathika	Computer Networking Principles	AIDS 2022-2026	VIEW

Rows per page: 5 ▾ 1–2 of 2

The screenshot shows a web browser window for the 'Admin Dashboard'. The dashboard has a left sidebar with orange icons for Home, User Management, Class Management, and Notifications. The main area displays a table with one row. The columns are 'Name' (Swathika), 'Subject' (Computer Networking Principles), and 'Class' (AIDS 2022-2026). An 'Actions' column contains a 'VIEW' button. Below the table, it says 'Rows per page: 5' and '1–1 of 1'. The top bar shows the title 'localhost' and the date 'Wed 17 Apr 11:22 AM'.

Conclusion: Test case passed. Teacher Login id deleted successfully.

Student Login Id Creation:

Test case: Check for student login id creation page loading.

The screenshot shows a web browser window for the 'Add Student' form. The sidebar is identical to the previous dashboard. The main form has fields for 'Name *' (with 'I' typed in), 'Roll Number *', 'Password *', and a dropdown 'Class' set to 'AIDS 2022-2026'. A large blue 'ADD' button is at the bottom. The top bar shows the title 'localhost' and the date 'Wed 17 Apr 11:25 AM'.

Conclusion: Test case passed. Student login id creation page loading successful.

Test case: Check for Student login id creation.

The screenshots illustrate the process of adding a student and viewing the resulting list.

Add Student Form (Top Screenshot):

Name *	Karnassagar
Roll Number *	46
Password *
Class	AIDS 2022-2026

Students List (Bottom Screenshot):

Name	Roll Number	Actions
Karnassagar	46	VIEW ATTENDANCE

Conclusion: Test case passed. Student login id created successfully.

Test case: Check for student login id deletion.

Students List:

Name	Roll Number	Actions
Karnassagar	46	VIEW ATTENDANCE
Ramkumar	64	VIEW ATTENDANCE

Rows per page: 5 ▾ 1–2 of 2 < >

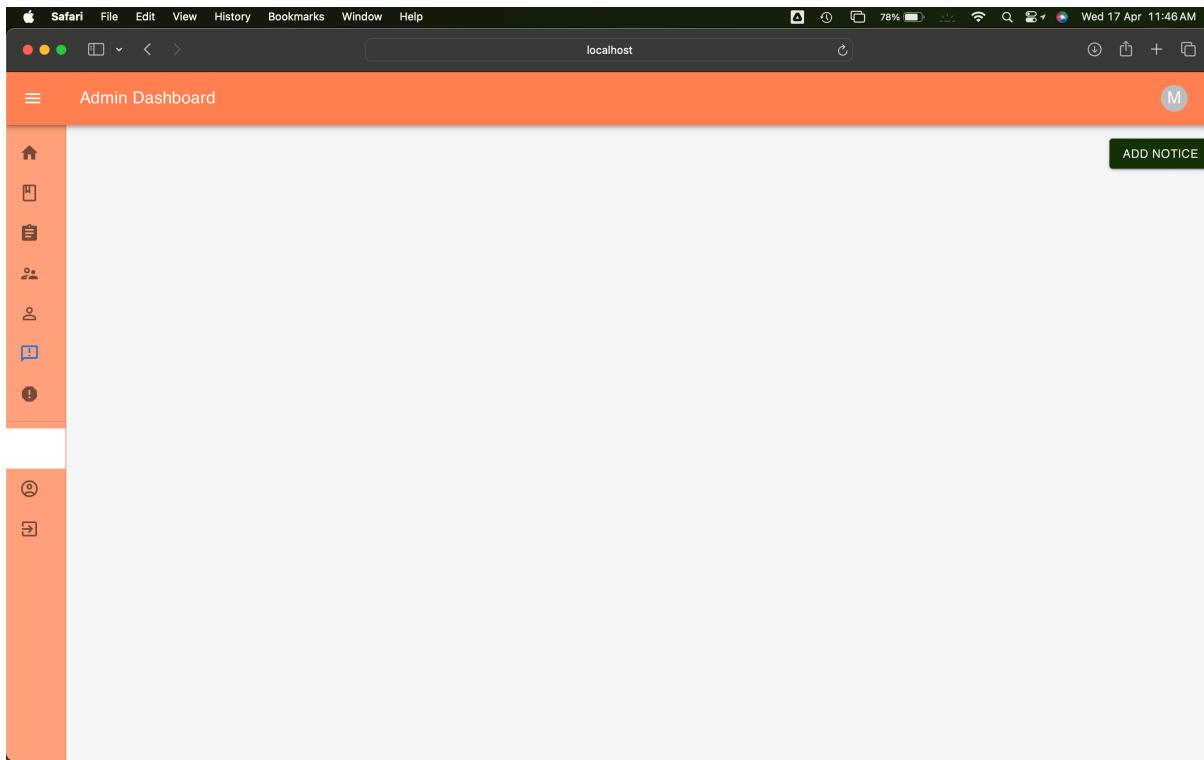
Students List:

Name	Roll Number	Actions
Karnassagar	46	VIEW ATTENDANCE

Rows per page: 5 ▾ 1–1 of 1 < >

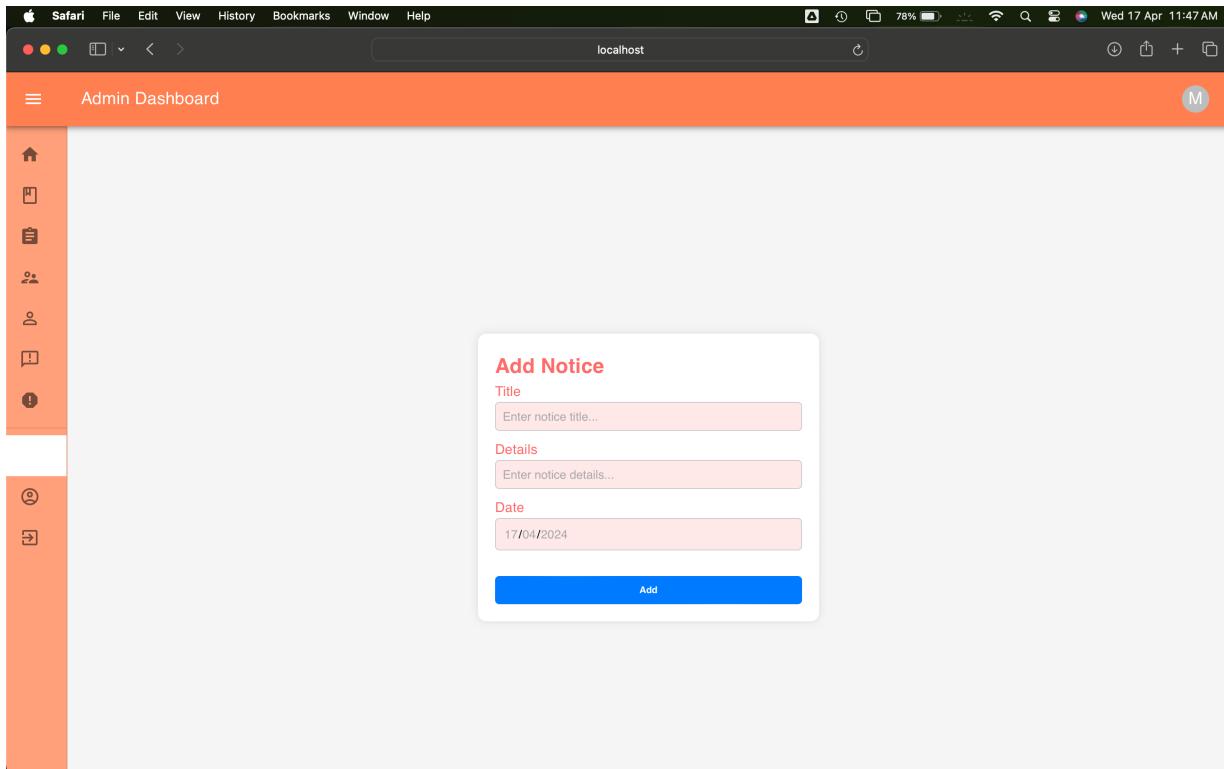
Conclusion: Test case passed. Student login id deletion successful.

Test case: Check for Notice page loading.



Conclusion: Test case passed. Notice page loading successful.

Test case: Check for Notice creation page loading.



Conclusion: Test case passed. Notice creation page loading successful.

Test case: Check for creation of notice.

The screenshot shows two consecutive screenshots of a web-based admin dashboard for an e-learning platform, displayed in a Safari browser on a Mac OS X system. The top screenshot captures the 'Add Notice' modal window, while the bottom screenshot shows the resulting list of notices.

Screenshot 1: Add Notice Form

The 'Add Notice' form is displayed in a modal window. It contains the following fields:

- Title:** Server Maintenance
- Details:** Greetings everyone. Today the website will be under maintenance.
- Date:** 17/04/2024

A blue 'Add' button is located at the bottom right of the form.

Screenshot 2: List of Notices

The list of notices is displayed in a table format. The columns are labeled 'Title', 'Details', 'Date', and 'Actions'. One notice is listed:

Title	Details	Date	Actions
Server Maintenance	Greetings everyone. Today the website will be under maintenance.	2024-04-17	

Below the table, there are pagination controls: 'Rows per page: 5', '1-1 of 1', and navigation arrows. A circular green button with a white icon is visible on the right side of the list area.

The screenshot shows the Admin Dashboard of an e-learning platform. On the left is a vertical sidebar with orange icons for Home, Classes, Subjects, Teachers, Students, Notices, and Complains. The main area has a header "Admin Dashboard" and a sub-header "localhost". It displays three summary boxes: "Total Students" (2), "Total Classes" (2), and "Total Teachers" (1). Below these is a "Notices" section with a table:

Title	Details	Date
Server Maintenance	Greetings everyone. Today the website will be under maintenance.	2024-04-17

At the bottom of the dashboard, there are pagination controls: "Rows per page: 5" and "1-1 of 1".

Conclusion: Test case passed. Notice creation successful.

Test case: Check for Complain page loading.

The screenshot shows the Admin Dashboard with the "Complains" option selected in the sidebar. The main area displays a table of complaints:

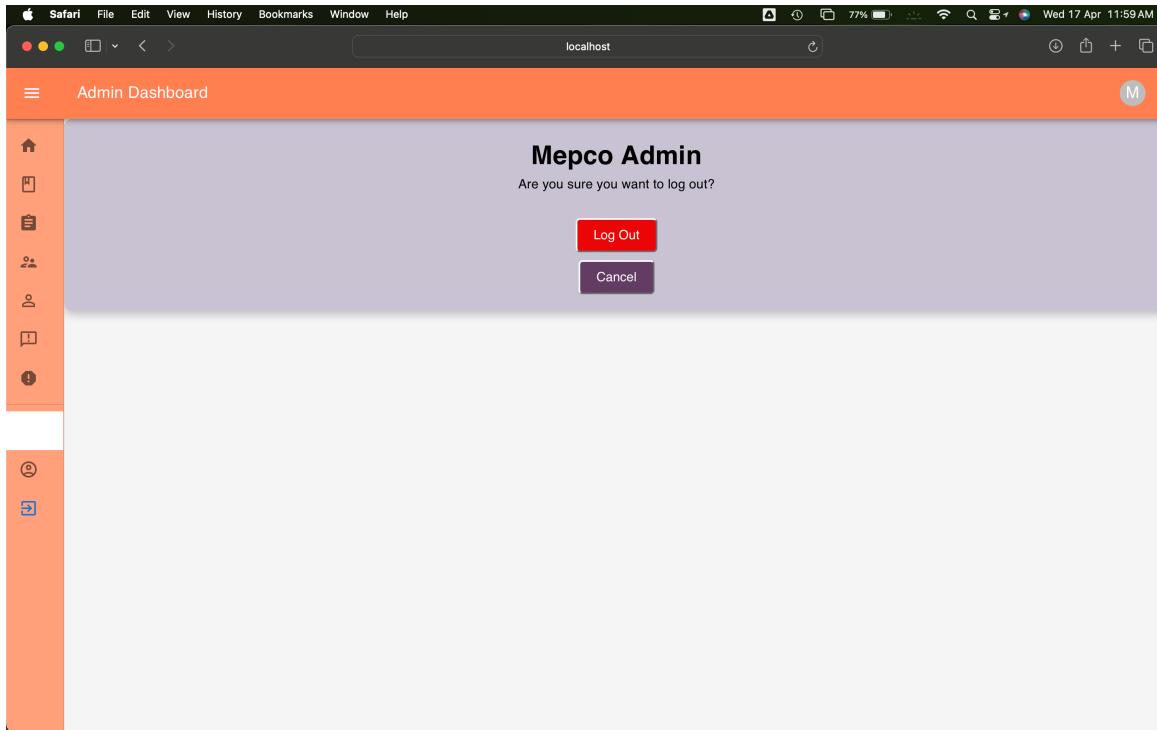
User	Complaint	Date	Actions
RAMKUMAR K	i have forgotten my password on mail	2024-04-17	<input type="checkbox"/>

At the bottom of the dashboard, there are pagination controls: "Rows per page: 5" and "1-1 of 1".

Conclusion: Test case passed. Complain page loading successful.

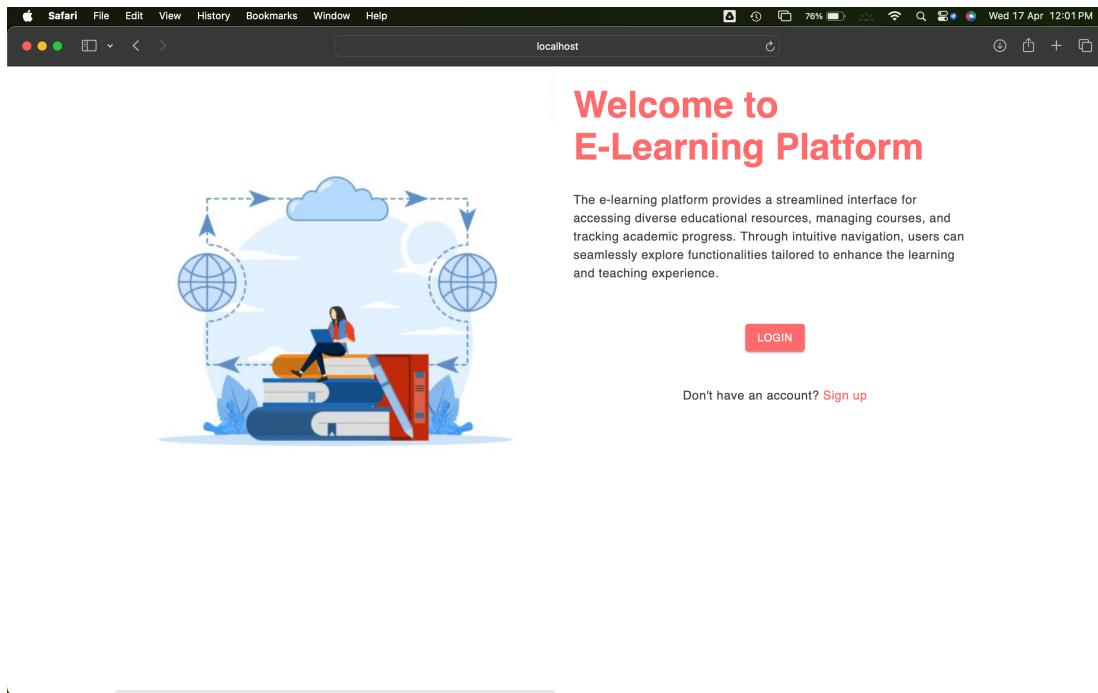
Logout:

Test case: Check for Logout page loading.



Conclusion: Test case passed. Logout page loading successful.

Test case: Check for successful logout.



Conclusion: Test case passed. Logout successful.

Testing of Student Module:

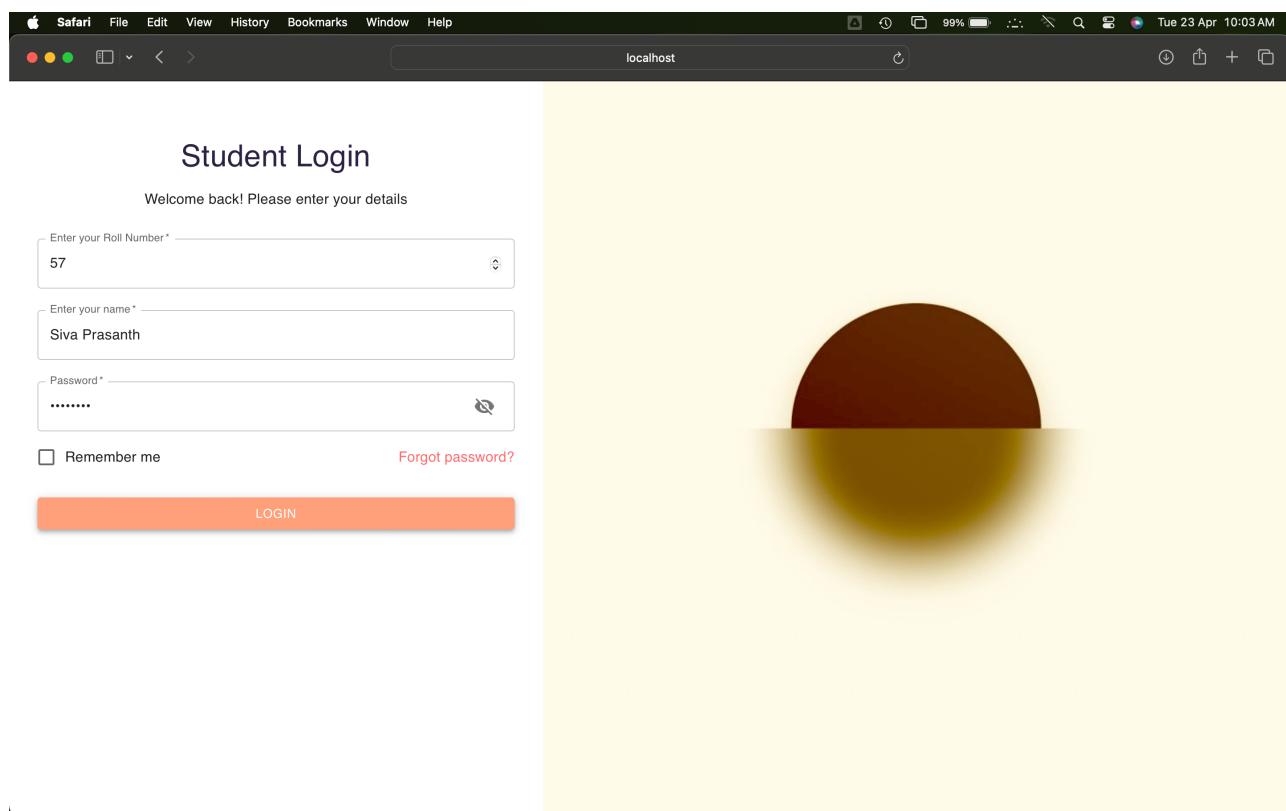
List of Modules:

- Login
- Dashboard
- Subject
- Attendance
- Quiz
- Complain
- Profile
- Logout

Unit Testing:

Login:

Test case: Check for Login screen loading.



Conclusion: Test case passed. Login screen loaded successfully.

Test case: Check for login successful.

The screenshot shows the Student Dashboard. On the left is a vertical sidebar with icons and labels: Home, Subjects, Attendance, Complain, User, Profile, and Logout. The main area is titled "Student Dashboard" and contains two cards: "Total Subjects" (3) and "Total Assignments" (5). To the right is a large green circular progress bar labeled "100%" and "0%". Below these is a section titled "Notices" with a table showing one item: "leave for election day (19-04-2024)" with details "Everyone don't miss out on voting" and date "2024-04-16". At the bottom are pagination controls: "Rows per page: 5", "1–1 of 1", and navigation arrows.

Conclusion: Test case passed. Login successful.

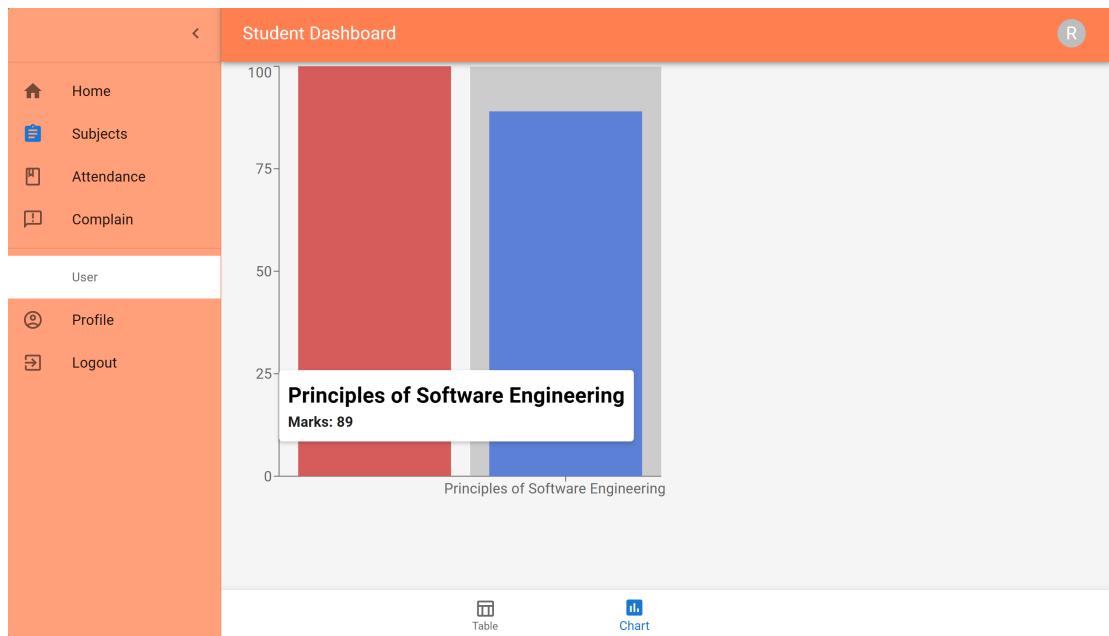
Subject:

Test case: Check for subject page loading.

The screenshot shows the "Subject Marks" page. The sidebar is identical to the previous dashboard. The main area is titled "Subject Marks" and displays a table with two rows: "Full Stack Development" with a mark of "100" and "Principles of Software Engineering" with a mark of "89". At the bottom are links for "Table" and "Chart". The URL "localhost:3001/Student/subjects" is visible at the bottom left.

Conclusion: Test case passed. Subject page loading successful.

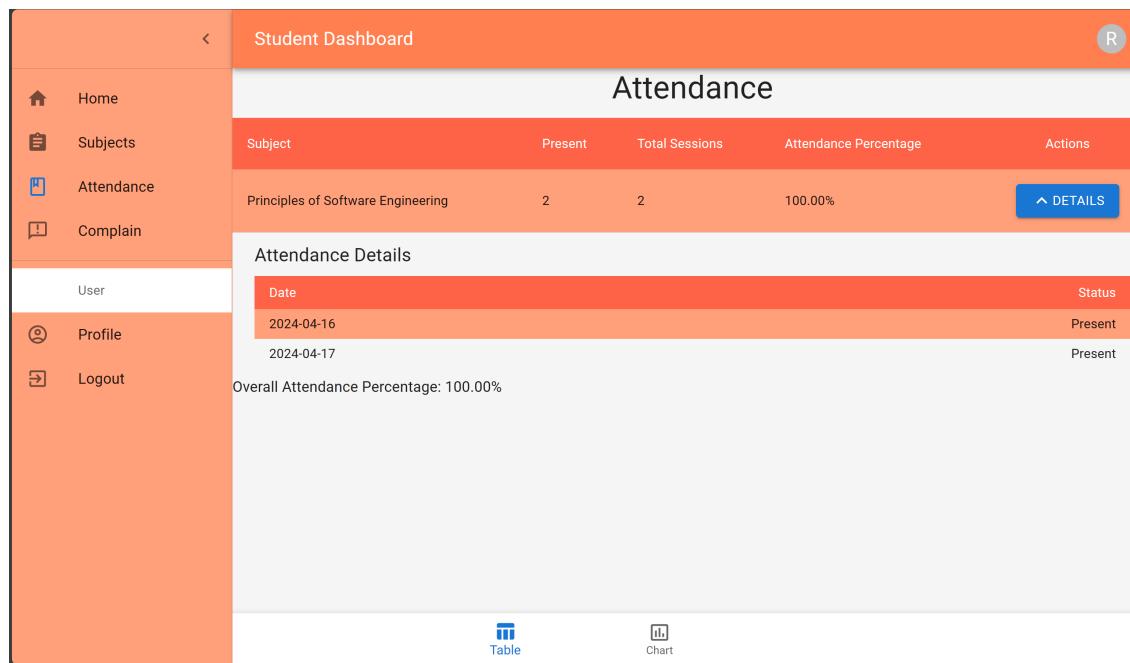
Test case: Check for Mark analysis.



Conclusion: Test case passed. Mark analysis successful.

Attendance:

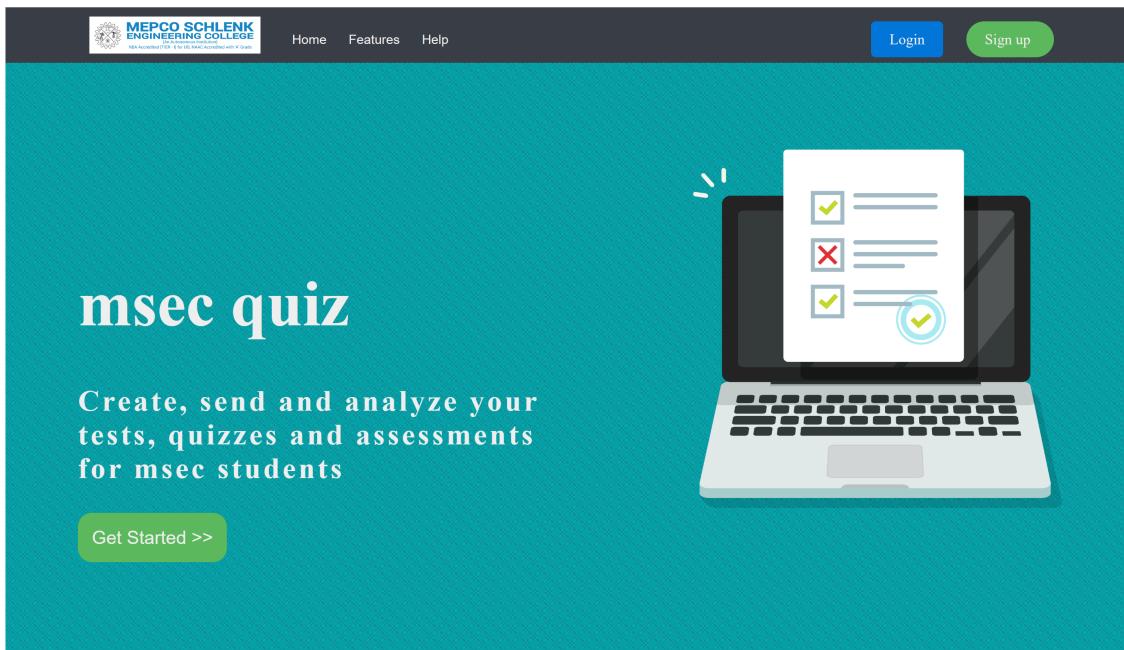
Test case: Check for attendance analysis for a subject.



Conclusion: Test case passed. Attendance analysis for a subject successful.

Quiz:

Test case: Check for Quiz homepage loading.



Conclusion: Test case passed. Quiz homepage loading successful.

Test Case: Check for Quiz dashboard loading.

A screenshot of the Quiz dashboard. The top navigation bar includes the college logo, 'MEPCO SCHLENK ENGINEERING COLLEGE' with accreditation information, 'Dashboard', 'Reports', a help icon, and a 'Logout' button. A prominent teal 'Create Exam' button is centered above a list of quizzes. The 'Quizzes' section shows two entries: 'software engineering => Click for quiz link' and 'full stack => Click for quiz link'. Each entry has three action buttons: 'Analyze' (with a bar chart icon), 'Preview' (with a video camera icon), 'Edit' (with a pencil icon), and 'Delete' (with a trash bin icon). The dashboard has a light gray background and a clean, modern design.

Conclusion: Test case passed. Quiz dashboard loading successful.

Test case: Check for quiz attending.

The screenshot shows a quiz interface for 'MEPCO SCHLENK ENGINEERING COLLEGE'. At the top, there's a navigation bar with the college logo, 'Dashboard', 'Reports', a help icon, and a 'Logout' button. Below the header, it displays '00:04:58' and 'Score : 0'. The main content area is titled 'Question 1 :'. The question is 'Software is defined as _____'. There are four options in boxes: 'set of programs, documentation & configuration of data', 'set of programs', 'documentation and configuration of data', and 'None of the mentioned'. A 'Next Question' button is at the bottom. At the very bottom of the page, there are links for 'Mepco', 'Useful Links', and 'Contact'.

Conclusion: Test case passed. Quiz attending successful.

Test case: Check for quiz result.

The screenshot shows a quiz result page for 'MEPCO SCHLENK ENGINEERING COLLEGE'. The top navigation bar includes the college logo, 'Dashboard', 'Reports', a help icon, and a 'Logout' button. The message 'Final Score : sorry you failed the exam' is displayed. A large red 'FAIL' stamp is centered on the page. A 'Go to dashboard' button is located below the stamp. At the bottom, there are links for 'Mepco', 'Useful Links', and 'Contact'. The 'Useful Links' section includes links for Home, Login, Help, Register, and Features. The 'Contact' section includes links for Sivakasi and an email address: msec@mepcoeng.ac.in. There are also social media icons for LinkedIn, Facebook, and YouTube.

Conclusion: Test case passed. Quiz result successful.

Complain:

Test case: Check for complain creation.

The screenshot shows the Student Dashboard with a sidebar on the left containing links for Home, Subjects, Attendance, Complain, Profile, and Logout. The main content area is titled 'Complain' and includes a date input field set to '04/17/2024' and a text area for the complain message, which contains 'i have forgotten my password on mail'. A blue 'ADD' button is at the bottom right of the form.

Conclusion: Test case passed. Complain creation successful.

Logout:

Test case: Check for logout page loading.

The screenshot shows the Student Dashboard with a sidebar on the left containing links for Home, Subjects, Attendance, Quiz, Complain, Profile, and Logout. A modal dialog box is centered on the screen, displaying the user's name 'Siva Prasanth' and the question 'Are you sure you want to log out?'. It has two buttons: 'Log Out' (in a dark grey box) and 'Cancel' (in a purple box).

Conclusion: Test case passed. Logout page loading successful.

Integration Testing:

Test case: Check for complain raise in admin.

The screenshot shows the Admin Dashboard interface. On the left, there is a sidebar with icons for Home, Classes, Subjects, Teachers, Students, Notices, and Complains. Below these are sections for User, Profile, and Logout. The main area is titled "Admin Dashboard" and contains a table with one row. The table columns are User, Complaint, Date, and Actions. The data in the table is: User - RAMKUMAR K, Complaint - i have forgotten my password on mail, Date - 2024-04-17, and Actions - a checkbox icon. At the bottom right of the main area, there are buttons for "Rows per page:" (set to 5), "1–1 of 1", and navigation arrows.

Conclusion: Test case passed. Complain raise in admin successful.

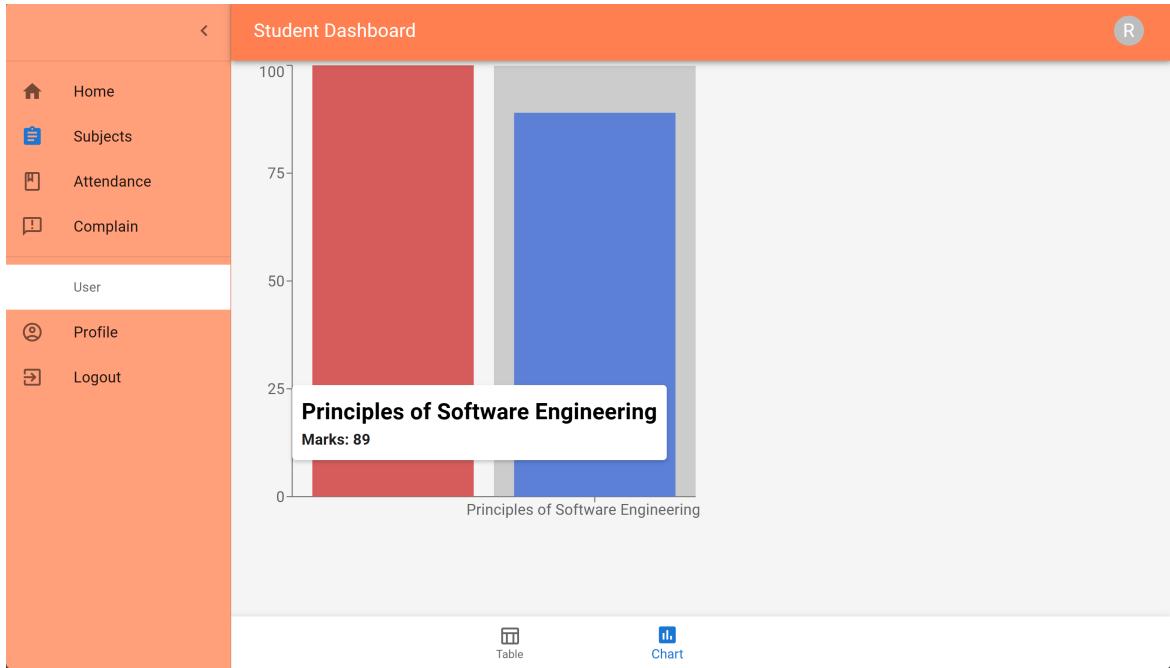
Test case: Check for attendance updation in student.

The screenshot shows the Teacher Dashboard interface. On the left, there is a sidebar with icons for Home, Class TABLEAU, Complain, User, Profile, and Logout. The main area is titled "Teacher Dashboard" and displays a success message "Done Successfully" in a green box. Below the message, it shows "Student Name: chandra prakash" and "Subject Name: Principles of Software Engineering". There are two input fields: "Attendance Status" set to "Present" and "Select Date *" set to "04/16/2024". A large blue "SUBMIT" button is at the bottom.

The screenshot shows the Student Dashboard with the title "Attendance". On the left sidebar, under the "User" section, "Logout" is selected. The main content area displays attendance data for the subject "Principles of Software Engineering". A table shows 2 present sessions out of 2 total, resulting in 100.00% attendance. An "DETAILS" button is visible. Below the table, "Attendance Details" show two entries: "2024-04-16" and "2024-04-17", both marked as "Present". The overall attendance percentage is 100.00%. At the bottom, there are "Table" and "Chart" navigation links.

Conclusion: Test case passed. Attendance updation in student successful.
Test case: Check for mark updation in student.

The screenshot shows the Teacher Dashboard with the title "Teacher Dashboard". On the left sidebar, under the "User" section, "Logout" is selected. The main content area displays student information: "Student Name: RAMKUMAR K" and "Subject Name: Principles of Software Engineering". Below this, a form field shows "Enter marks * 80" with a "SUBMIT" button. A green notification bar at the top right says "Done Successfully".



Conclusion: Test case passed. Mark updation in student successful.

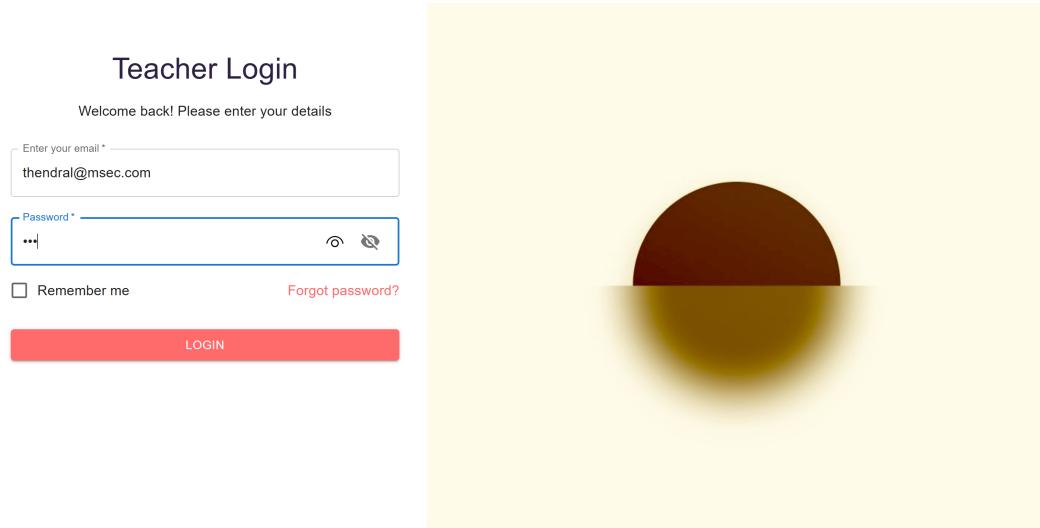
Testing of Teacher Module:

List of Modules:

- Login
- Attendance
- Marks
- Subject
- Quiz
- Logout

Login:

Test case: Check for teacher login.

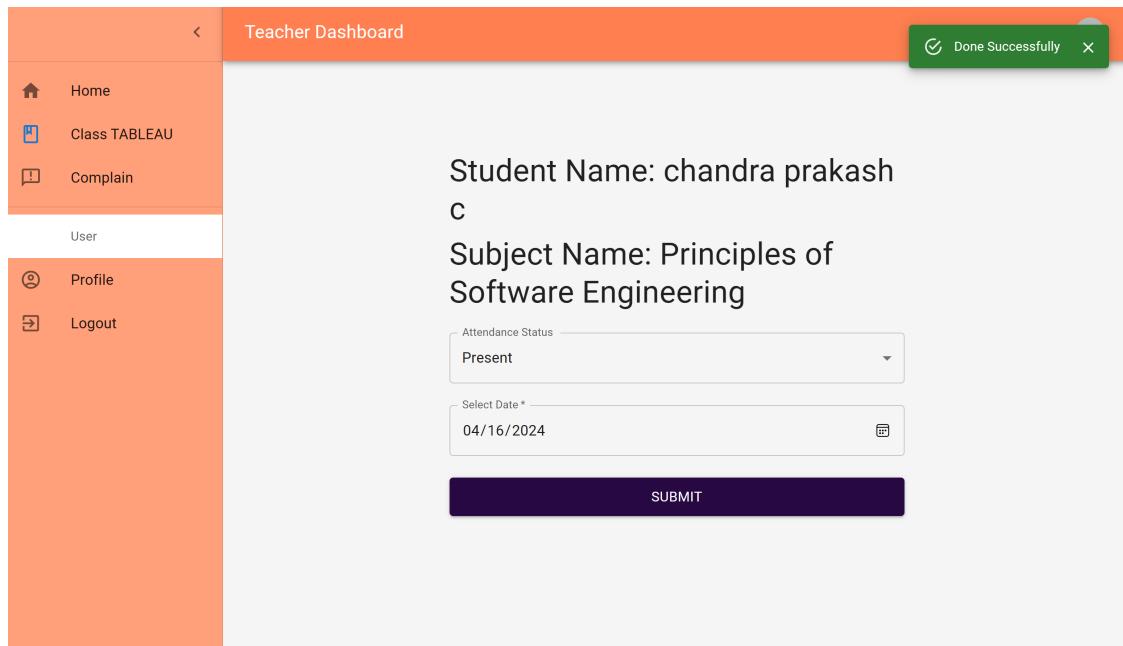


The screenshot shows a 'Teacher Login' form. At the top, a message says 'Welcome back! Please enter your details'. Below it is an email input field containing 'thendral@msec.com'. An password input field is shown with a placeholder 'Password *' and three dots indicating a masked password. To the right of the password field are eye icon and a copy icon. Below the input fields are two buttons: a 'Remember me' checkbox and a 'Forgot password?' link. At the bottom is a large red 'LOGIN' button.

Conclusion: Test case passed. Teacher login successful.

Attendance:

Test case: Check for marking present in the attendance.



The screenshot shows a 'Teacher Dashboard' interface. On the left is a sidebar with icons for Home, Class TABLEAU, Complain, User (selected), Profile, and Logout. The main area displays student and subject information: 'Student Name: chandra prakash' and 'Subject Name: Principles of Software Engineering'. Below this is an 'Attendance Status' dropdown set to 'Present'. A date input field shows 'Select Date *' with '04/16/2024' entered. At the bottom is a dark blue 'SUBMIT' button. A green success toast message 'Done Successfully' is visible at the top right.

Conclusion: Test case passed. Marking present in the attendance.

Test case: Check for marking absent in the attendance.

The screenshot shows the Teacher Dashboard interface. On the left is a sidebar with icons for Home, Class TABLEAU, Complain, User (selected), Profile, and Logout. The main area displays student information: Student Name: chandra prakash and Subject Name: Principles of Software Engineering. Below this, there are two input fields: 'Attendance Status' set to 'Absent' and 'Select Date *' set to '04/15/2024'. A green success message at the top right says 'Done Successfully'.

Conclusion: Test case passed. Marking absent in the attendance.

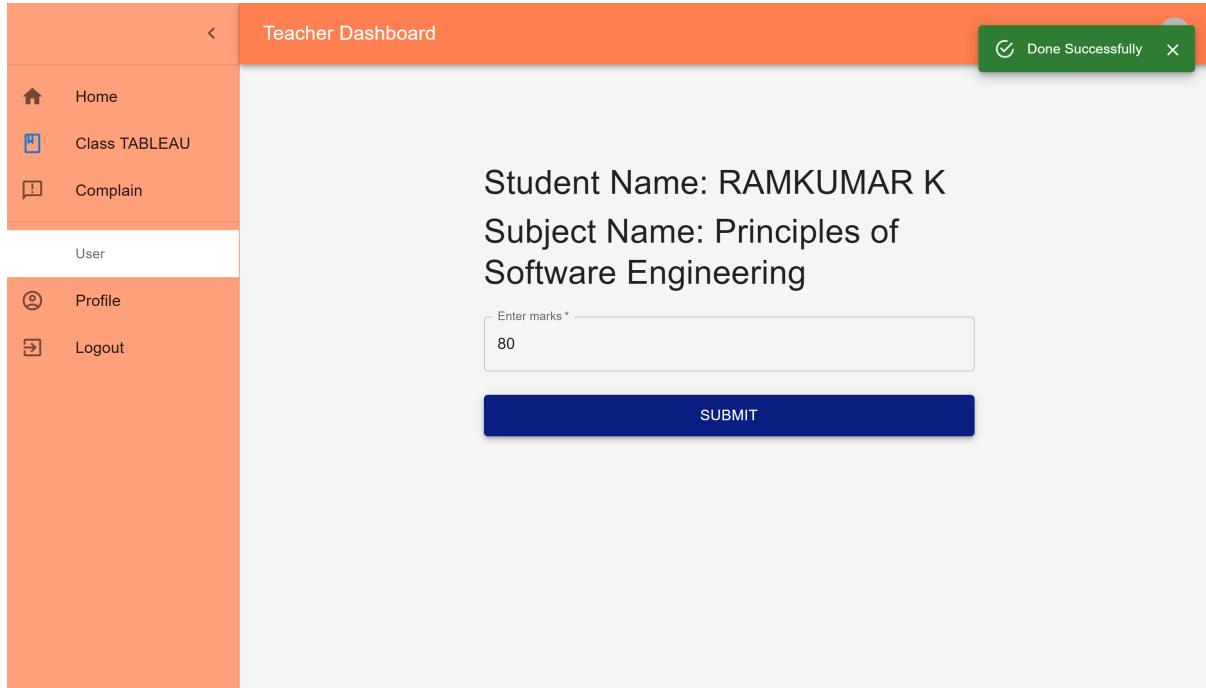
Test case: Check for maximum attendance marking.

The screenshot shows the Teacher Dashboard interface. The sidebar and student information are identical to the previous screenshot. The 'Attendance Status' field is now set to 'Present'. A red warning message at the top right says 'Maximum attendance limit reached'.

Conclusion: Test case passed. Maximum attendance marking successful.

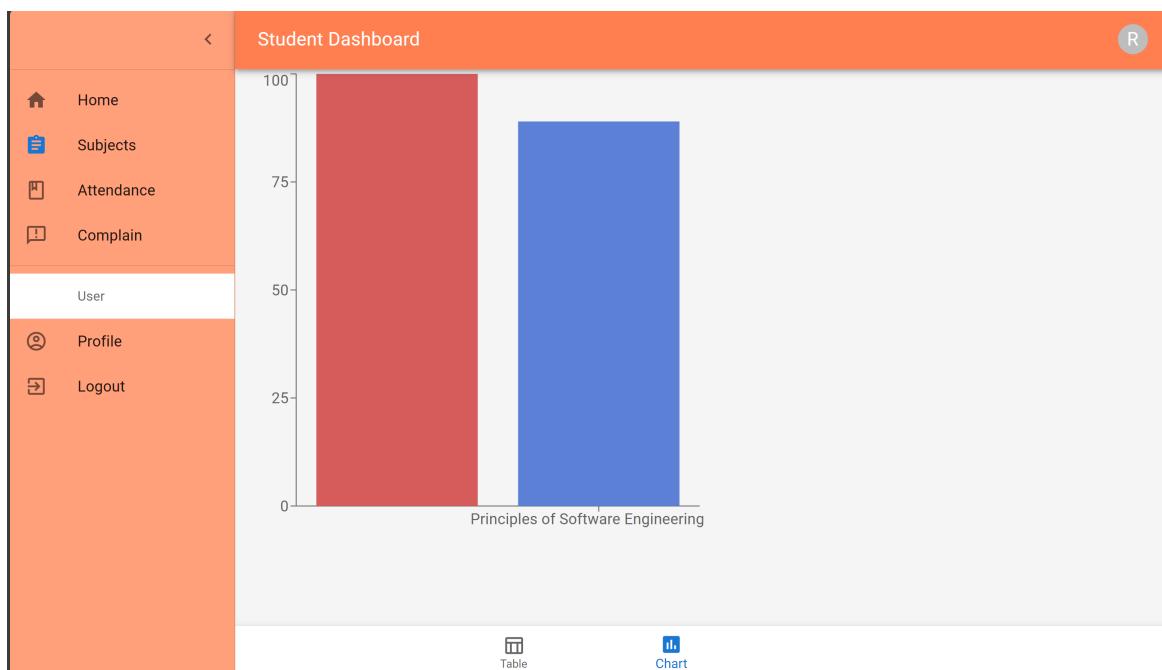
Marks:

Test case: Check for providing mark.



Conclusion: Test case passed. Providing mark successful.

Test case: Check for mark analysis.



Conclusion: Test case passed. Mark analysis successful.

Subject:

Test case: Check for listing students in the subject.

Teacher Dashboard			
Class Details			
	Name	Roll Number	Actions
User	RAMKUMAR K	1	<button>VIEW</button> <button>TAKE ATTENDANCE</button> <button>▼</button>
Profile	chandra prakash c	2	<button>VIEW</button> <button>TAKE ATTENDANCE</button> <button>▼</button>
Logout	vishnu sk	3	<button>VIEW</button> <button>TAKE ATTENDANCE</button> <button>▼</button>
	siva	4	<button>VIEW</button> <button>TAKE ATTENDANCE</button> <button>▼</button>
	arunash kumar	6	<button>VIEW</button> <button>TAKE ATTENDANCE</button> <button>▼</button>

Rows per page: 5 ▾ 1–5 of 6 < >

Conclusion: Test case passed. Listing students in the subject successful.

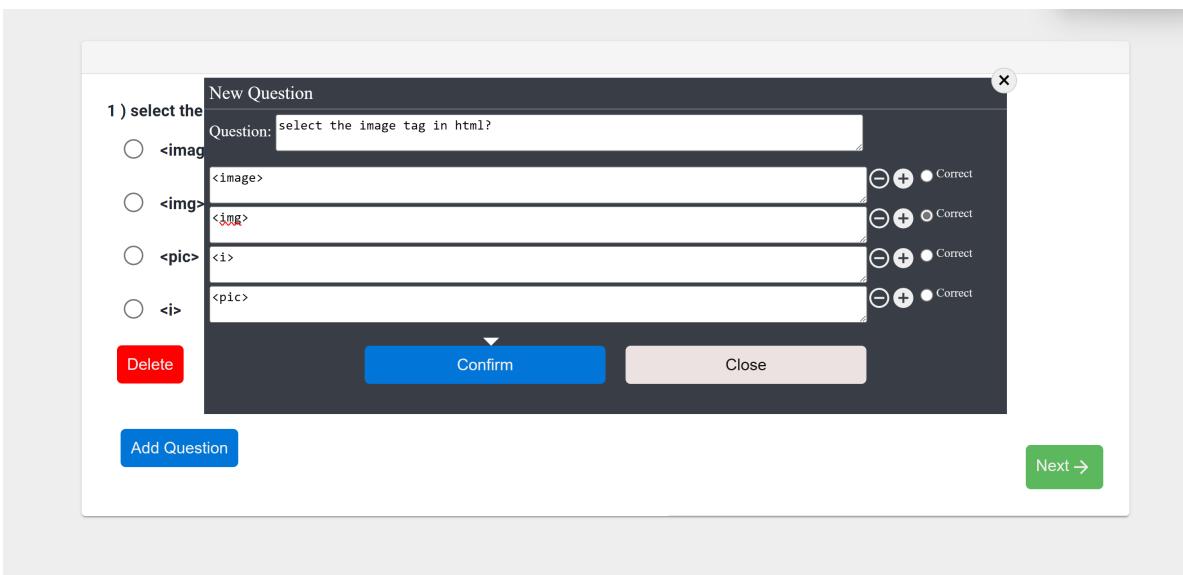
Quiz:

Test case: Check for quiz page loading.

The screenshot shows the MEPCO SCHLENK ENGINEERING COLLEGE dashboard. At the top, there is a header bar with the college logo, 'MEPCO SCHLENK ENGINEERING COLLEGE', 'Dashboard', 'Reports', and a 'Logout' button. Below the header, there is a large teal button labeled 'Create Exam'. Underneath this button is a white rectangular box containing the word 'Quizzes'. At the bottom of the screen, there are three small circular icons (blue, black, red) followed by the text 'Useful Links' and 'Contact'.

Conclusion: Test case passed. Quiz page loading successful.

Test case: Check for quiz creation.



MEPCO SCHLENK
ENGINEERING COLLEGE
NAAC Accredited 'A' & ISO 9001:2015 Certified Academic & Admin

Dashboard Reports

Logout

Quiz Name Software Engineering

Time Limit 5

Pass Grade 2

Save

Useful Links

- Home
- Login
- Help
- Register
- Features

Contact

- Sivakasi
- msec@mepcoeng.ac.in

in linkedin fb

The screenshot shows the MEPCO SCHLENK ENGINEERING COLLEGE dashboard. At the top, there is a navigation bar with the college logo, 'Dashboard', 'Reports', a help icon, and a 'Logout' button. Below the navigation bar is a large teal button labeled 'Create Exam'. Underneath this button is a section titled 'Quizzes' containing two entries:

- 'software engineering => Click for quiz link' with options to 'Analyze', 'Preview', 'Edit', and 'Delete'.
- 'quiz2 => Click for quiz link' with options to 'Analyze', 'Preview', 'Edit', and 'Delete'.

Conclusion: Test case passed. Quiz creation successful.

Logout:

Test case: Check for teacher logout.

The screenshot shows the Teacher Dashboard. On the left is a sidebar with the following menu items:

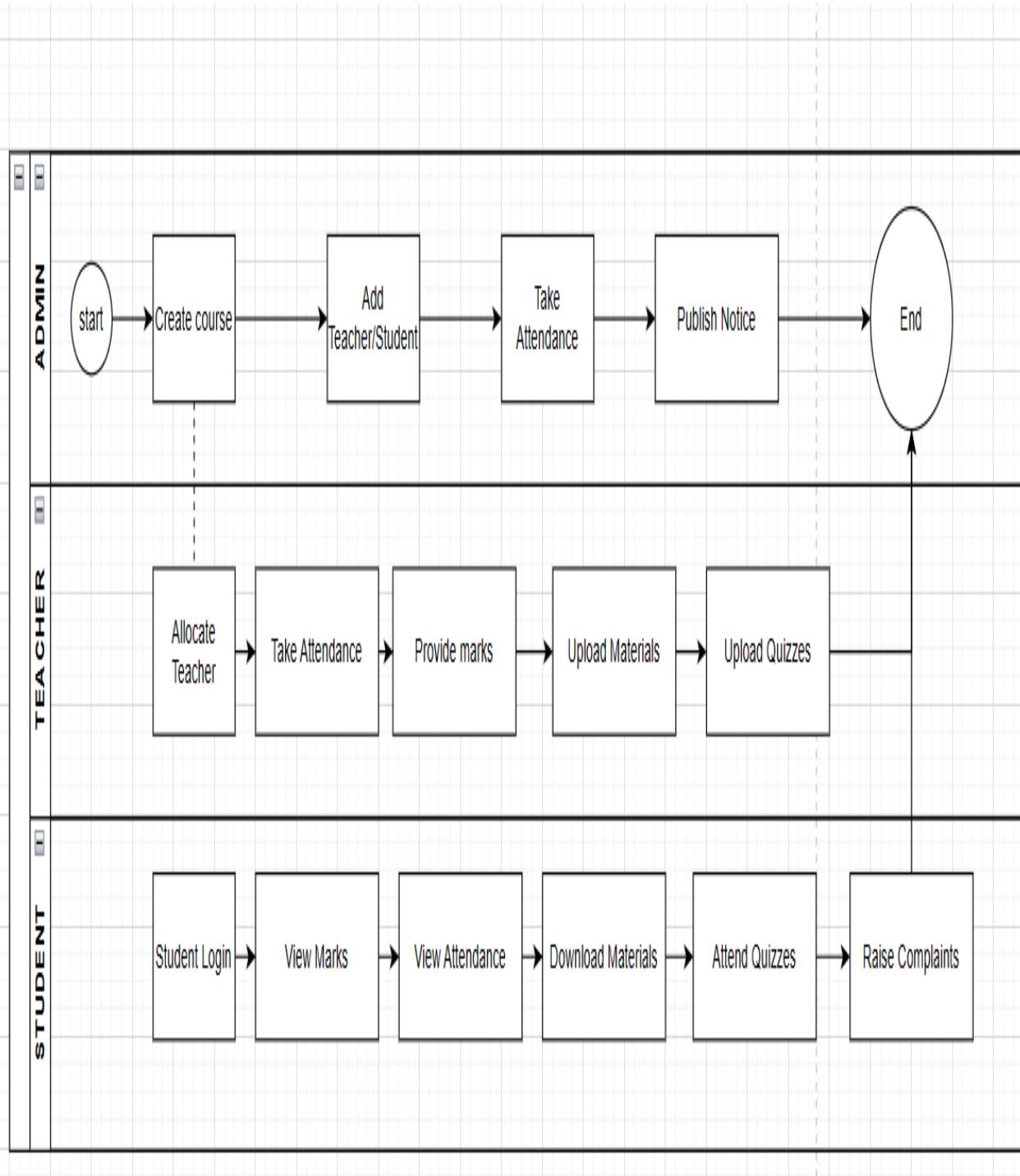
- Home
- Class TABLEAU
- Complain
- User
- Profile
- Logout

The main area is titled 'Thendral' and contains a message: 'Are you sure you want to log out?'. It has two buttons: a red 'Log Out' button and a purple 'Cancel' button.

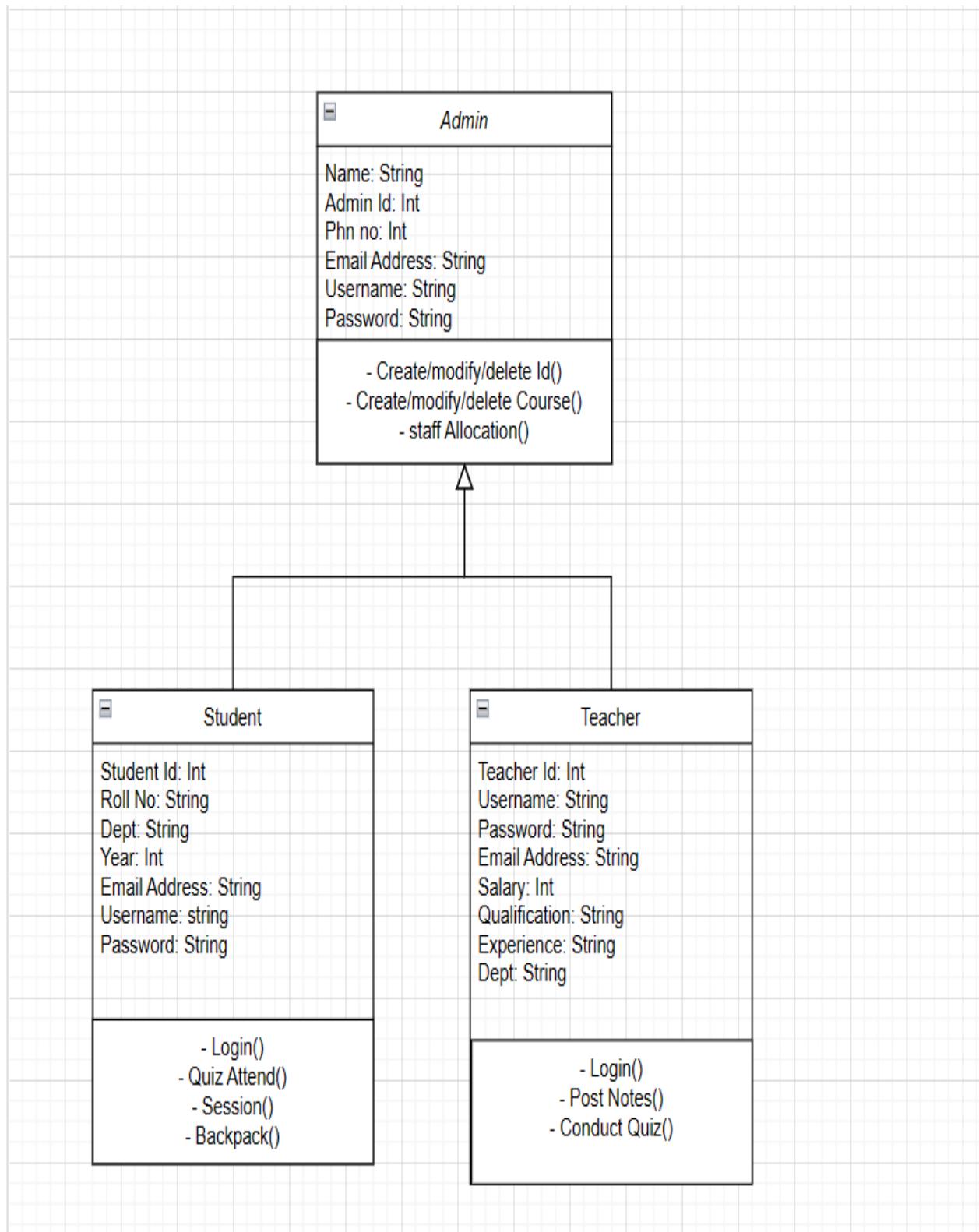
Conclusion: Test case passed. Teacher logout successful.

6. Other Requirements

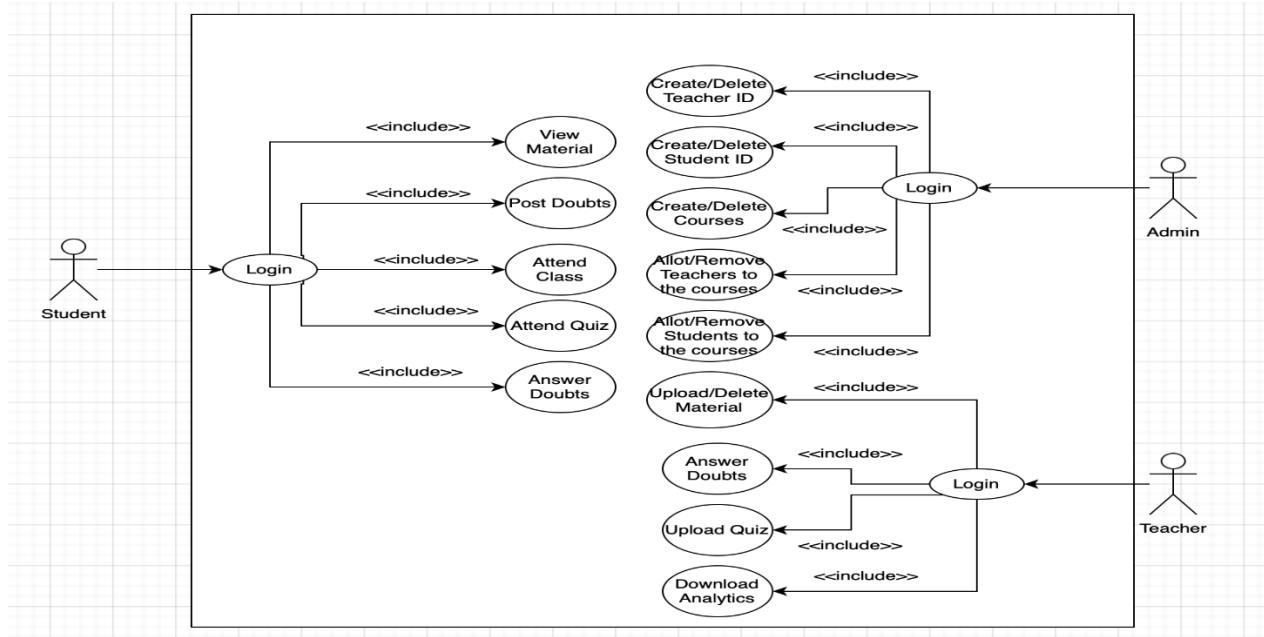
Appendix A: Activity



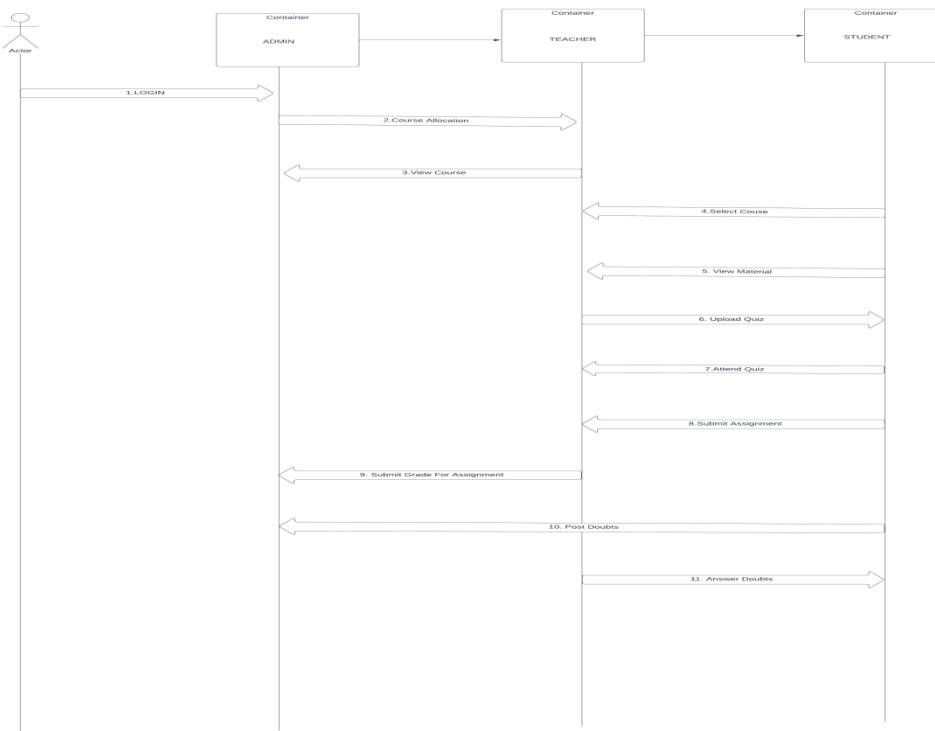
Appendix B: Class diagram



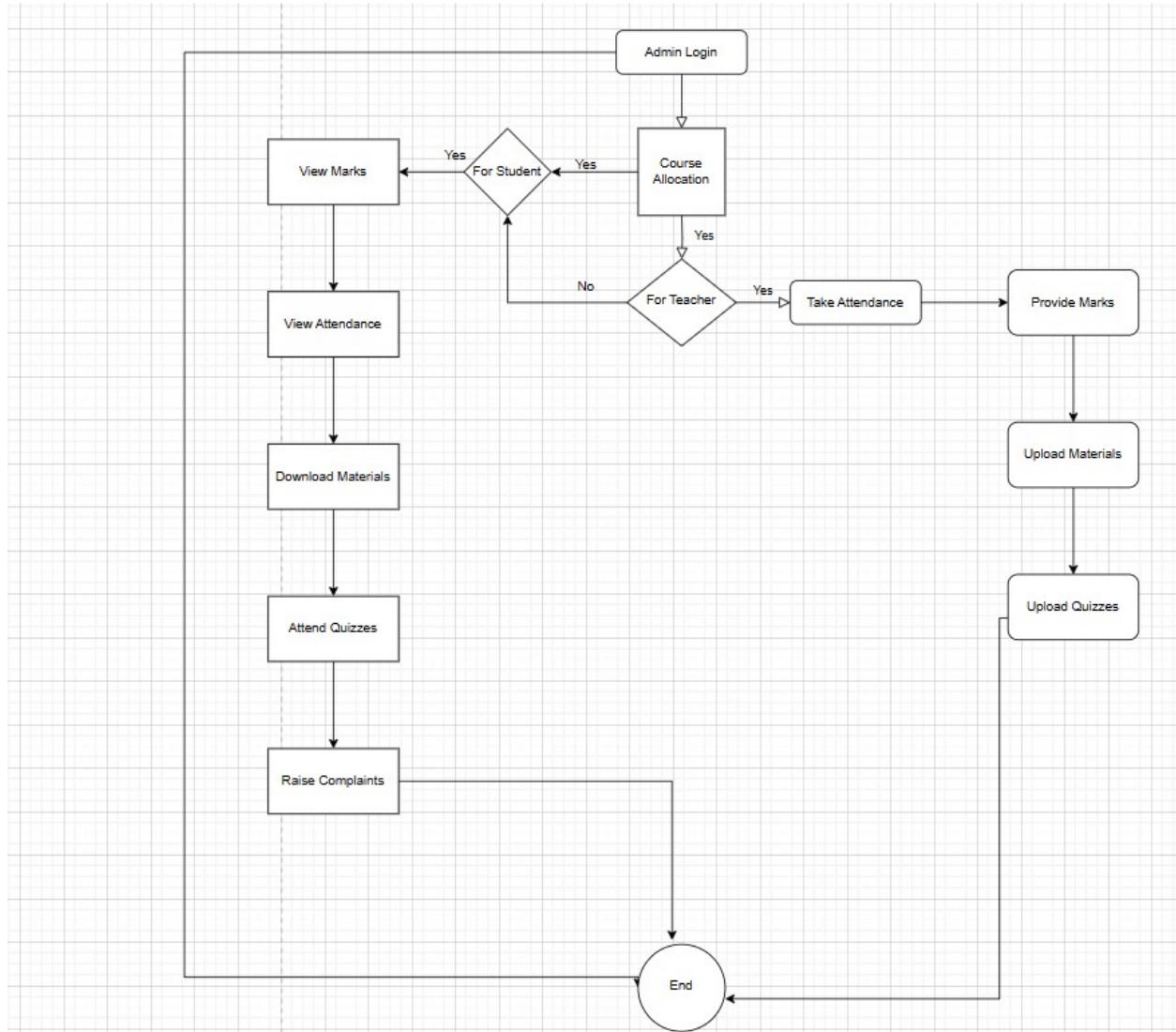
Appendix C:



Appendix D:



Appendix E:



Appendix F: Source Code

/backend/controllers:

```
admin-controller.js
const bcrypt = require('bcrypt');
const Admin = require('../models/adminSchema.js');
const Sclass = require('../models/sclassSchema.js');
const Student = require('../models/studentSchema.js');
const Teacher = require('../models/teacherSchema.js');
const Subject = require('../models/subjectSchema.js');
const Notice = require('../models/noticeSchema.js');
const Complain = require('../models/complainSchema.js');
const Quiz = require('../models/quizSchema.js'); // Import the Quiz model
const Material = require('../models/materialSchema.js');

const adminRegister = async (req, res) => {
  try {
    const salt = await bcrypt.genSalt(10);
    const hashedPass = await bcrypt.hash(req.body.password, salt);

    const admin = new Admin({
      ...req.body,
      password: hashedPass
    });

    const existingAdminByEmail = await Admin.findOne({ email: req.body.email });
    const existingCollege = await Admin.findOne({ collegeName: req.body.collegeName });

    if (existingAdminByEmail) {
      res.send({ message: 'Email already exists' });
    }
    else if (existingCollege) {
      res.send({ message: 'College name already exists' });
    }
    else {
      let result = await admin.save();
      result.password = undefined;
      res.send(result);
    }
  } catch (err) {
    res.status(500).json(err);
  }
}
```

```
        }
    };

const adminLogIn = async (req, res) => {
    if (req.body.email && req.body.password) {
        let admin = await Admin.findOne({ email: req.body.email });
        if (admin) {
            const validated = await bcrypt.compare(req.body.password, admin.password);
            if (validated) {
                admin.password = undefined;
                res.send(admin);
            } else {
                res.send({ message: "Invalid password" });
            }
        } else {
            res.send({ message: "User not found" });
        }
    } else {
        res.send({ message: "Email and password are required" });
    }
};

const getAdminDetail = async (req, res) => {
    try {
        let admin = await Admin.findById(req.params.id);
        if (admin) {
            admin.password = undefined;
            res.send(admin);
        } else {
            res.send({ message: "No admin found" });
        }
    } catch (err) {
        res.status(500).json(err);
    }
};

// New method to handle quiz creation by admin
const createQuiz = async (req, res) => {
    try {
        const quiz = new Quiz(req.body);
        const result = await quiz.save();
        res.send(result);
    } catch (err) {
```

```
        res.status(500).json(err);
    }
};

// New method to handle material upload by admin
const uploadMaterial = async (req, res) => {
    try {
        const material = new Material(req.body);
        const result = await material.save();
        res.send(result);
    } catch (err) {
        res.status(500).json(err);
    }
};

// const deleteAdmin = async (req, res) => {
//     try {
//         const result = await Admin.findByIdAndDelete(req.params.id)

//         await Sclass.deleteMany({ college: req.params.id });
//         await Student.deleteMany({ college: req.params.id });
//         await Teacher.deleteMany({ college: req.params.id });
//         await Subject.deleteMany({ college: req.params.id });
//         await Notice.deleteMany({ college: req.params.id });
//         await Complain.deleteMany({ college: req.params.id });

//         res.send(result)
//     } catch (error) {
//         res.status(500).json(err);
//     }
// }

// const updateAdmin = async (req, res) => {
//     try {
//         if (req.body.password) {
//             const salt = await bcrypt.genSalt(10)
//             res.body.password = await bcrypt.hash(res.body.password, salt)
//         }
//         let result = await Admin.findByIdAndUpdate(req.params.id,
//             { $set: req.body },
//             { new: true })

//         result.password = undefined;
//         res.send(result)
//     }
// }
```

```
// } catch (error) {
//   res.status(500).json(err);
// }
// }

// module.exports = { adminRegister, adminLogIn, getAdminDetail, deleteAdmin,
// updateAdmin };

module.exports = { adminRegister, adminLogIn, getAdminDetail, createQuiz, uploadMaterial };

class-controller.js
const Sclass = require('../models/sclassSchema.js');
const Student = require('../models/studentSchema.js');
const Subject = require('../models/subjectSchema.js');
const Teacher = require('../models/teacherSchema.js');

const sclassCreate = async (req, res) => {
  try {
    const sclass = new Sclass({
      sclassName: req.body.sclassName,
      college: req.body.adminID
    });

    const existingSclassByName = await Sclass.findOne({
      sclassName: req.body.sclassName,
      college: req.body.adminID
    });

    if (existingSclassByName) {
      res.send({ message: 'Sorry this class name already exists' });
    }
    else {
      const result = await sclass.save();
      res.send(result);
    }
  } catch (err) {
    res.status(500).json(err);
  }
};

const sclassList = async (req, res) => {
  try {
    let sclasses = await Sclass.find({ college: req.params.id })
```

```
if (sclasses.length > 0) {
    res.send(sclasses)
} else {
    res.send({ message: "No sclasses found" });
}
} catch (err) {
    res.status(500).json(err);
}
};

const getClassDetail = async (req, res) => {
try {
    let sclass = await Sclass.findById(req.params.id);
    if (sclass) {
        sclass = await sclass.populate("college", "collegeName")
        res.send(sclass);
    }
    else {
        res.send({ message: "No class found" });
    }
} catch (err) {
    res.status(500).json(err);
}
}

const getClassStudents = async (req, res) => {
try {
    let students = await Student.find({ sclassName: req.params.id })
    if (students.length > 0) {
        let modifiedStudents = students.map((student) => {
            return { ...student._doc, password: undefined };
        });
        res.send(modifiedStudents);
    } else {
        res.send({ message: "No students found" });
    }
} catch (err) {
    res.status(500).json(err);
}
}

const deleteSclass = async (req, res) => {
try {
```

```

const deletedClass = await Sclass.findByIdAndDelete(req.params.id);
if (!deletedClass) {
    return res.send({ message: "Class not found" });
}
const deletedStudents = await Student.deleteMany({ sclassName: req.params.id });
const deletedSubjects = await Subject.deleteMany({ sclassName: req.params.id });
const deletedTeachers = await Teacher.deleteMany({ teachSclass: req.params.id });
res.send(deletedClass);
} catch (error) {
    res.status(500).json(error);
}
}

const deleteSclasses = async (req, res) => {
try {
    const deletedClasses = await Sclass.deleteMany({ college: req.params.id });
    if (deletedClasses.deletedCount === 0) {
        return res.send({ message: "No classes found to delete" });
    }
    const deletedStudents = await Student.deleteMany({ college: req.params.id });
    const deletedSubjects = await Subject.deleteMany({ college: req.params.id });
    const deletedTeachers = await Teacher.deleteMany({ college: req.params.id });
    res.send(deletedClasses);
} catch (error) {
    res.status(500).json(error);
}
}

```

```

module.exports = { sclassCreate, sclassList, deleteSclass, deleteSclasses, getSclassDetail,
getSclassStudents };

```

```

complain-controller.js
const Complain = require('../models/complainSchema.js');

const complainCreate = async (req, res) => {
try {
    const complain = new Complain(req.body)
    const result = await complain.save()
    res.send(result)
} catch (err) {
    res.status(500).json(err);
}
}

```

```

};

const complainList = async (req, res) => {
  try {
    let complains = await Complain.find({ college: req.params.id }).populate("user", "name");
    if (complains.length > 0) {
      res.send(complains)
    } else {
      res.send({ message: "No complains found" });
    }
  } catch (err) {
    res.status(500).json(err);
  }
};

module.exports = { complainCreate, complainList };

material-controller.js
const Material = require('../models/materialSchema');

const uploadMaterial = async (req, res) => {
  try {
    const { description, fileUrl } = req.body;
    const newMaterial = new Material({
      description,
      fileUrl,
    });
    await newMaterial.save();
    res.status(201).json({ message: 'Material uploaded successfully', material: newMaterial });
  } catch (error) {
    res.status(500).json({ message: 'Could not upload material', error: error.message });
  }
};

const getMaterials = async (req, res) => {
  try {
    const materials = await Material.find();
    res.status(200).json({ materials });
  } catch (error) {
    res.status(500).json({ message: 'Could not fetch materials', error: error.message });
  }
};

```

```
module.exports = { uploadMaterial, getMaterials };

notice-controller.js
const Notice = require('../models/noticeSchema.js');

const noticeCreate = async (req, res) => {
    try {
        const notice = new Notice({
            ...req.body,
            college: req.body.adminID
        })
        const result = await notice.save()
        res.send(result)
    } catch (err) {
        res.status(500).json(err);
    }
};

const noticeList = async (req, res) => {
    try {
        let notices = await Notice.find({ college: req.params.id })
        if (notices.length > 0) {
            res.send(notices)
        } else {
            res.send({ message: "No notices found" });
        }
    } catch (err) {
        res.status(500).json(err);
    }
};

const updateNotice = async (req, res) => {
    try {
        const result = await Notice.findByIdAndUpdate(req.params.id,
            { $set: req.body },
            { new: true })
        res.send(result)
    } catch (error) {
        res.status(500).json(error);
    }
};

const deleteNotice = async (req, res) => {
```

```

try {
  const result = await Notice.findByIdAndDelete(req.params.id)
  res.send(result)
} catch (error) {
  res.status(500).json(err);
}
}

const deleteNotices = async (req, res) => {
  try {
    const result = await Notice.deleteMany({ college: req.params.id })
    if (result.deletedCount === 0) {
      res.send({ message: "No notices found to delete" })
    } else {
      res.send(result)
    }
  } catch (error) {
    res.status(500).json(err);
  }
}

module.exports = { noticeCreate, noticeList, updateNotice, deleteNotice, deleteNotices };

quiz-controller.js
const Quiz = require('../models/quizSchema');

const createQuiz = async (req, res) => {
  try {
    const { question, options, correctOption } = req.body;
    const newQuiz = new Quiz({
      question,
      options,
      correctOption,
    });
    await newQuiz.save();
    res.status(201).json({ message: 'Quiz created successfully', quiz: newQuiz });
  } catch (error) {
    res.status(500).json({ message: 'Could not create quiz', error: error.message });
  }
};

const getQuizzes = async (req, res) => {
  try {

```

```
const quizzes = await Quiz.find();
res.status(200).json({ quizzes });
} catch (error) {
  res.status(500).json({ message: 'Could not fetch quizzes', error: error.message });
}
};

module.exports = { createQuiz, getQuizzes };

student_controller.js
const bcrypt = require('bcrypt');
const Student = require('../models/studentSchema.js');
const Subject = require('../models/subjectSchema.js');
const QuizAttempt = require('../models/quizSchema.js'); // Import the QuizAttempt model
const MaterialAccess = require('../models/materialSchema.js');

const studentRegister = async (req, res) => {
  try {
    const salt = await bcrypt.genSalt(10);
    const hashedPass = await bcrypt.hash(req.body.password, salt);

    const existingStudent = await Student.findOne({
      rollNum: req.body.rollNum,
      college: req.body.adminID,
      sclassName: req.body.sclassName,
    });

    if (existingStudent) {
      res.send({ message: 'Roll Number already exists' });
    } else {
      const student = new Student({
        ...req.body,
        college: req.body.adminID,
        password: hashedPass
      });

      let result = await student.save();

      result.password = undefined;
      res.send(result);
    }
  } catch (err) {
```

```
        res.status(500).json(err);
    }
};

const studentLogIn = async (req, res) => {
    try {
        let student = await Student.findOne({ rollNum: req.body.rollNum, name: req.body.studentName });
        if (student) {
            const validated = await bcrypt.compare(req.body.password, student.password);
            if (validated) {
                student = await student.populate("college", "collegeName")
                student = await student.populate("sclassName", "sclassName")
                student.password = undefined;
                student.examResult = undefined;
                student.attendance = undefined;
                res.send(student);
            } else {
                res.send({ message: "Invalid password" });
            }
        } else {
            res.send({ message: "Student not found" });
        }
    } catch (err) {
        res.status(500).json(err);
    }
};

const getStudents = async (req, res) => {
    try {
        let students = await Student.find({ college: req.params.id }).populate("sclassName", "sclassName");
        if (students.length > 0) {
            let modifiedStudents = students.map((student) => {
                return { ...student._doc, password: undefined };
            });
            res.send(modifiedStudents);
        } else {
            res.send({ message: "No students found" });
        }
    } catch (err) {
        res.status(500).json(err);
    }
};
```

```
};
```

```
const getStudentDetail = async (req, res) => {
  try {
    let student = await Student.findById(req.params.id)
      .populate("college", "collegeName")
      .populate("sclassName", "sclassName")
      .populate("examResult.subName", "subName")
      .populate("attendance.subName", "subName sessions");
    if (student) {
      student.password = undefined;
      res.send(student);
    }
    else {
      res.send({ message: "No student found" });
    }
  } catch (err) {
    res.status(500).json(err);
  }
}
```

```
const deleteStudent = async (req, res) => {
  try {
    const result = await Student.findByIdAndDelete(req.params.id)
    res.send(result)
  } catch (error) {
    res.status(500).json(error);
  }
}
```

```
const deleteStudents = async (req, res) => {
  try {
    const result = await Student.deleteMany({ college: req.params.id })
    if (result.deletedCount === 0) {
      res.send({ message: "No students found to delete" })
    } else {
      res.send(result)
    }
  } catch (error) {
    res.status(500).json(error);
  }
}
```

```
const deleteStudentsByClass = async (req, res) => {
    try {
        const result = await Student.deleteMany({ sclassName: req.params.id })
        if (result.deletedCount === 0) {
            res.send({ message: "No students found to delete" })
        } else {
            res.send(result)
        }
    } catch (error) {
        res.status(500).json(error);
    }
}

const updateStudent = async (req, res) => {
    try {
        if (req.body.password) {
            const salt = await bcrypt.genSalt(10)
            res.body.password = await bcrypt.hash(res.body.password, salt)
        }
        let result = await Student.findByIdAndUpdate(req.params.id,
            { $set: req.body },
            { new: true })
        result.password = undefined;
        res.send(result)
    } catch (error) {
        res.status(500).json(error);
    }
}

const updateExamResult = async (req, res) => {
    const { subName, marksObtained } = req.body;

    try {
        const student = await Student.findById(req.params.id);

        if (!student) {
            return res.send({ message: 'Student not found' });
        }

        const existingResult = student.examResult.find(
            (result) => result.subName.toString() === subName
        );
    }
}
```

```
if (existingResult) {
    existingResult.marksObtained = marksObtained;
} else {
    student.examResult.push({ subName, marksObtained });
}

const result = await student.save();
return res.send(result);
} catch (error) {
    res.status(500).json(error);
}
};

const studentAttendance = async (req, res) => {
    const { subName, status, date } = req.body;

    try {
        const student = await Student.findById(req.params.id);

        if (!student) {
            return res.send({ message: 'Student not found' });
        }

        const subject = await Subject.findById(subName);

        const existingAttendance = student.attendance.find(
            (a) =>
                a.date.toDateString() === new Date(date).toDateString() &&
                a.subName.toString() === subName
        );

        if (existingAttendance) {
            existingAttendance.status = status;
        } else {
            // Check if the student has already attended the maximum number of sessions
            const attendedSessions = student.attendance.filter(
                (a) => a.subName.toString() === subName
            ).length;

            if (attendedSessions >= subject.sessions) {
                return res.send({ message: 'Maximum attendance limit reached' });
            }
        }
    }
}
```

```
        student.attendance.push({ date, status, subName });
    }

    const result = await student.save();
    return res.send(result);
} catch (error) {
    res.status(500).json(error);
}
};

const clearAllStudentsAttendanceBySubject = async (req, res) => {
    const subName = req.params.id;

    try {
        const result = await Student.updateMany(
            { 'attendance.subName': subName },
            { $pull: { attendance: { subName } } }
        );
        return res.send(result);
    } catch (error) {
        res.status(500).json(error);
    }
};

const clearAllStudentsAttendance = async (req, res) => {
    const collegeId = req.params.id

    try {
        const result = await Student.updateMany(
            { college: collegeId },
            { $set: { attendance: [] } }
        );
        return res.send(result);
    } catch (error) {
        res.status(500).json(error);
    }
};

const removeStudentAttendanceBySubject = async (req, res) => {
    const studentId = req.params.id;
    const subName = req.body.subId
```

```
try {
    const result = await Student.updateOne(
        { _id: studentId },
        { $pull: { attendance: { subName: subName } } }
    );

    return res.send(result);
} catch (error) {
    res.status(500).json(error);
}
};

const removeStudentAttendance = async (req, res) => {
    const studentId = req.params.id;

    try {
        const result = await Student.updateOne(
            { _id: studentId },
            { $set: { attendance: [] } }
        );

        return res.send(result);
    } catch (error) {
        res.status(500).json(error);
    }
};

// New method to handle quiz attempts by students
const attemptQuiz = async (req, res) => {
    try {
        const quizAttempt = new QuizAttempt(req.body);
        const result = await quizAttempt.save();
        res.send(result);
    } catch (err) {
        res.status(500).json(err);
    }
};

// New method to handle material access by students
const accessMaterial = async (req, res) => {
    try {
```

```

    const materialAccess = new MaterialAccess(req.body);
    const result = await materialAccess.save();
    res.send(result);
} catch (err) {
    res.status(500).json(err);
}
};


```

```

module.exports = {
    studentRegister,
    studentLogIn,
    getStudents,
    getStudentDetail,
    deleteStudents,
    deleteStudent,
    updateStudent,
    studentAttendance,
    deleteStudentsByClass,
    updateExamResult,
    clearAllStudentsAttendanceBySubject,
    clearAllStudentsAttendance,
    removeStudentAttendanceBySubject,
    removeStudentAttendance,
    attemptQuiz,
    accessMaterial,
};


```

subject-controller.js

```

const Subject = require('../models/subjectSchema.js');
const Teacher = require('../models/teacherSchema.js');
const Student = require('../models/studentSchema.js');


```

```

const subjectCreate = async (req, res) => {
    try {
        const subjects = req.body.subjects.map((subject) => ({
            subName: subject.subName,
            subCode: subject.subCode,
            sessions: subject.sessions,
        }));
    }


```

```

const existingSubjectBySubCode = await Subject.findOne({
    'subjects.subCode': subjects[0].subCode,
}


```

```
college: req.body.adminID,
});

if (existingSubjectBySubCode) {
    res.send({ message: 'Sorry, this subcode must be unique as it already exists' });
} else {
    const newSubjects = subjects.map((subject) => ({
        ...subject,
        sclassName: req.body.sclassName,
        college: req.body.adminID,
        quiz: [], // Initialize an empty quiz array
    }));
}

const result = await Subject.insertMany(newSubjects);
res.send(result);
}

} catch (err) {
    res.status(500).json(err);
}
};

const allSubjects = async (req, res) => {
try {
    let subjects = await Subject.find({ college: req.params.id })
        .populate("sclassName", "sclassName")
    if (subjects.length > 0) {
        res.send(subjects)
    } else {
        res.send({ message: "No subjects found" });
    }
} catch (err) {
    res.status(500).json(err);
}
};

const classSubjects = async (req, res) => {
try {
    let subjects = await Subject.find({ sclassName: req.params.id })
    if (subjects.length > 0) {
        res.send(subjects)
    } else {
        res.send({ message: "No subjects found" });
    }
}
```

```
    } catch (err) {
      res.status(500).json(err);
    }
  };

const freeSubjectList = async (req, res) => {
  try {
    let subjects = await Subject.find({ sclassName: req.params.id, teacher: { $exists: false } });
    if (subjects.length > 0) {
      res.send(subjects);
    } else {
      res.send({ message: "No subjects found" });
    }
  } catch (err) {
    res.status(500).json(err);
  }
};

const getSubjectDetail = async (req, res) => {
  try {
    let subject = await Subject.findById(req.params.id);
    if (subject) {
      subject = await subject.populate("sclassName", "sclassName")
      subject = await subject.populate("teacher", "name")
      res.send(subject);
    } else {
      res.send({ message: "No subject found" });
    }
  } catch (err) {
    res.status(500).json(err);
  }
};

const deleteSubject = async (req, res) => {
  try {
    const deletedSubject = await Subject.findByIdAndDelete(req.params.id);

    // Set the teachSubject field to null in teachers
    await Teacher.updateOne(
      { teachSubject: deletedSubject._id },
      { $unset: { teachSubject: "" }, $unset: { teachSubject: null } }
    );
  }
};
```

```
// Remove the objects containing the deleted subject from students' examResult array
await Student.updateMany(
    {},
    { $pull: { examResult: { subName: deletedSubject._id } } }
);

// Remove the objects containing the deleted subject from students' attendance array
await Student.updateMany(
    {},
    { $pull: { attendance: { subName: deletedSubject._id } } }
);

res.send(deletedSubject);
} catch (error) {
    res.status(500).json(error);
}
};

const deleteSubjects = async (req, res) => {
try {
    const deletedSubjects = await Subject.deleteMany({ college: req.params.id });

    // Set the teachSubject field to null in teachers
    await Teacher.updateMany(
        { teachSubject: { $in: deletedSubjects.map(subject => subject._id) } },
        { $unset: { teachSubject: "" }, $unset: { teachSubject: null } }
    );

    // Set examResult and attendance to null in all students
    await Student.updateMany(
        {},
        { $set: { examResult: null, attendance: null } }
    );

    res.send(deletedSubjects);
} catch (error) {
    res.status(500).json(error);
}
};

const deleteSubjectsByClass = async (req, res) => {
try {
```

```
const deletedSubjects = await Subject.deleteMany({ sclassName: req.params.id });

// Set the teachSubject field to null in teachers
await Teacher.updateMany(
  { teachSubject: { $in: deletedSubjects.map(subject => subject._id) } },
  { $unset: { teachSubject: "" }, $unset: { teachSubject: null } }
);

// Set examResult and attendance to null in all students
await Student.updateMany(
  {},
  { $set: { examResult: null, attendance: null } }
);

res.send(deletedSubjects);
} catch (error) {
  res.status(500).json(error);
}
};

const createQuiz = async (req, res) => {
  try {
    const subjectId = req.params.id;
    const { question, options, correctAnswer } = req.body;
    const quizItem = { question, options, correctAnswer };

    const subject = await Subject.findById(subjectId);
    if (!subject) {
      return res.status(404).json({ message: 'Subject not found' });
    }

    subject.quiz.push(quizItem);
    await subject.save();

    res.status(201).json({ message: 'Quiz item created successfully' });
  } catch (error) {
    res.status(500).json({ message: 'Error creating quiz item', error });
  }
};

const deleteQuiz = async (req, res) => {
  try {
    const subjectId = req.params.id;
```

```
const quizId = req.params.quizId;

const subject = await Subject.findById(subjectId);
if (!subject) {
    return res.status(404).json({ message: 'Subject not found' });
}

subject.quiz.id(quizId).remove();
await subject.save();

res.status(200).json({ message: 'Quiz item deleted successfully' });
} catch (error) {
    res.status(500).json({ message: 'Error deleting quiz item', error });
}
};

const updateQuiz = async (req, res) => {
try {
    const subjectId = req.params.id;
    const quizId = req.params.quizId;
    const { question, options, correctAnswer } = req.body;

    const subject = await Subject.findById(subjectId);
    if (!subject) {
        return res.status(404).json({ message: 'Subject not found' });
    }

    const quizItem = subject.quiz.id(quizId);
    if (!quizItem) {
        return res.status(404).json({ message: 'Quiz item not found' });
    }

    quizItem.question = question;
    quizItem.options = options;
    quizItem.correctAnswer = correctAnswer;

    await subject.save();

    res.status(200).json({ message: 'Quiz item updated successfully' });
} catch (error) {
    res.status(500).json({ message: 'Error updating quiz item', error });
}
};
```

```
module.exports = { subjectCreate, freeSubjectList, classSubjects, getSubjectDetail,
  deleteSubjectsByClass, deleteSubjects, deleteSubject, allSubjects, createQuiz, deleteQuiz,
  updateQuiz};
```

createQuiz.js

```
import { useContext, useEffect, useState } from 'react'
import styled from 'styled-components'
import Footer from '../components/Footer'
import LoginNavbar from '../components/LoginNavbar'
import Table from '@mui/material/Table';
import TableBody from '@mui/material/TableBody';
import TableCell from '@mui/material/TableCell';
import TableContainer from '@mui/material/TableContainer';
import TableHead from '@mui/material/TableHead';
import TableRow from '@mui/material/TableRow';
import Paper from '@mui/material/Paper';
import { ArrowForward, AddCircle, RemoveCircleOutline } from '@mui/icons-material';
import Popup from 'reactjs-popup';
import { v4 as uuidv4 } from 'uuid';
import { useParams } from 'react-router-dom'
import axios from 'axios'
import { Link } from "react-router-dom";
```

```
const Container = styled.table`
  width: 100%;
  height: 40vh;
  border-collapse: collapse;
  text-align: center;
  border-radius: 8px;
  overflow: hidden;
  background-color: #EEEEEE;
`;
```

```
const Wrapper = styled.div`
  width: 86%;
  margin: 7%;
  max-width: 1300px;
`;
```

```
const Check = styled.input`
  transform: scale(1.5);
  margin: 20px;
  color: ;
`;
```

```
const Label = styled.label`
  color: ;
```

```
maxWidth:1400px;
`  
  
const Button = styled.button`  
font-size:16px;  
margin:10px;  
padding:10px;  
border:none;  
border-radius:5px;  
background-color:#0275d8;  
color:#EEEEEE;  
cursor: pointer;  
&:hover {  
    background-color: #228CE9;  
}  
  
const NextButton = styled.button`  
font-size:16px;  
margin:10px;  
padding:10px;  
border:none;  
border-radius:5px;  
background-color:#5cb85c;  
color:#EEEEEE;  
cursor: pointer;  
&:hover {  
    background-color: #75DB75;  
}  
  
const CreateQuiz = () => {  
  
    const params = useParams();  
    const id = params;  
  
    const [options, setOptions] = useState([]);  
    const [correctOption, setCorrectOption] = useState();  
    const [inputFields, setInputFields] = useState([{ id: uuidv4(), option: " " }]);  
    const [examDatas, setExamDatas] = useState([]);  
    const [questionTitle, setQuestionTitle] = useState("");  
    const [dummy, setDummy] = useState(0);  
    const [dumy, setDumy] = useState(0);  
  
    const addQuestion = async (e) => {  
        e.preventDefault();  
        if (correctOption === undefined) {
```

```
    alert("You have select an correct option")
} else if (inputFields[0].option === "") {
    alert("You have to add a question to use Add Question feature")
}
else {

    const inputOption = await Promise.all(inputFields.map((inputF) => inputF.option))
    /* const index = inputOption.indexOf(correctOption)
    if (index > -1) {
        inputOption.splice(index, 1);
    }*/
    console.log(inputOption);
    setOptions(inputOption);

    const newQuestion = {
        examId: id.id,
        questionTitle: questionTitle,
    };
    console.log(newQuestion)
    axios.post("http://localhost:5100/examquestions/", newQuestion).then((response) => {
        console.log(response.status);
        const data = response.data._id;
        handleOptions({ data, inputOption });
    });
}
}

const handleOptions = ({ data, inputOption }) => {
    var questionOptions;
    var control;
    for (let i = 0; i < inputOption.length; i++) {
        var questionOptions = inputOption[i];
        if (questionOptions === correctOption) {
            control = true
        } else {
            control = false
        }
        const option = {
            options: {
                option: questionOptions,
                isCorrect: control
            }
        }
        console.log(option);
        axios.put("http://localhost:5100/examquestions/" + data, option).then((response) => {
            console.log(response.status);
        })
    }
}
```

```
        console.log(response);
    });
}
setDumy(dumy + 1)
}

const handleChangeInput = async (id, event) => {
    const newInputFields = await Promise.all(inputFields.map(i => {
        if (id === i.id) {
            i[event.target.name] = event.target.value
        }
        return i;
    }))
    setInputFields(newInputFields);
}

const handleAddFields = () => {
    setInputFields([...inputFields, { id: uuidv4(), option: "" }])
}

const handleRemoveFields = id => {
    const values = [...inputFields];
    values.splice(values.findIndex(value => value.id === id), 1);
    setInputFields(values);
}

useEffect(() => {
    getExams();
    console.log("Check")
}, [options, dummy, dumy]);

const getExams = async () => {
    const { data } = await axios.get('http://localhost:5100/examquestions/' + id.id);
    setExamDatas(data);
    console.log(data[0].options)
}

const deleteQuestion = (propId) => {
    axios.delete('http://localhost:5100/examquestions/' + propId).then((response) => {
        console.log(response.status);
        console.log(response.data);
        setDummy(dummy + 1);
    });
}
```

```

return (
  <>
  <LoginNavbar />
  <Container>
    <Wrapper>
      <TableContainer component={Paper}>
        <Table sx={{ minWidth: 650 }} aria-label="simple table">
          <TableHead>
            <TableRow style={{ backgroundColor: "whitesmoke" }}>
              <TableCell align="right"></TableCell>
              <TableCell align="right"></TableCell>
              <TableCell align="right"></TableCell>
              <TableCell align="right"></TableCell>
              <TableCell align="right"></TableCell>
            </TableRow>
          </TableHead>
          <TableBody>
            {examDatas.map((exam, index) => (
              <TableRow
                sx={{ '&:last-child td, &:last-child th': { border: 0 } }}
              >
                <TableCell component="th" scope="exam" style={{ color: "#222831", fontSize: "16px", fontWeight: "600", padding: "25px" }}>
                  <Label>{(index + 1) + " ") {exam.questionTitle}</Label>
                  {exam.options.map((option) => (
                    <>
                      <br /><Check type="radio" name={"\$index + 1"} />
                      <Label>{option.option}</Label>
                    </>
                  )))
                <br />
                <Button style={{ backgroundColor: "red" }} onClick={() => deleteQuestion(exam._id)}>Delete</Button>
              </TableCell>
              <TableCell align="right"></TableCell>
              <TableCell align="right"></TableCell>
              <TableCell align="right"></TableCell>
              <TableCell align="right"></TableCell>
            </TableRow>
          )))
        </TableBody>
        <TableHead>
          <TableRow>
            <PopUp
              trigger={<Button style={{ marginLeft: "-65%" }}>Add Question</Button>}>

```

```

        modal
        nested
      >
      {close => (
        <div style={{ fontSize: "12px", backgroundColor: "#393E46", width: "800px", height: "auto" }}>
          <button style={{ cursor: "pointer", position: "absolute", display: "block", padding: "2px 5px", lineHeight: "20px", right: "-10px", top: "-10px", fontSize: "24px", background: "#EEEEEE", borderRadius: "18px", border: "1px solid #cfcece" }} onClick={close}>
            &times;
          </button>

          <div style={{ width: "100", borderBottom: "1px solid gray", fontSize: "18px", padding: "5px", color: "white" }}>New Question</div>
          <div style={{ padding: "5px", fontSize: "16px", fontWeight: "500", color: "white" }}>Question: <textarea style={{ verticalAlign: "middle", maxWidth: "580px", maxHeight: "200px", width: "580px" }} type="text" onChange={e => setQuestionTitle(e.target.value)} /></div>
          <form onSubmit={addQuestion} style={{ width: "100%", padding: "10px 5px" }}>
            {inputFields.map(inputField => (
              <div key={inputField.id}>
                <textarea
                  name="option"
                  label="First Name"
                  variant="filled"
                  value={inputField.option}
                  onChange={event => handleChangeInput(inputField.id, event)}
                  style={{ maxWidth: "650px", maxHeight: "200px", width: "650px" }}
                />

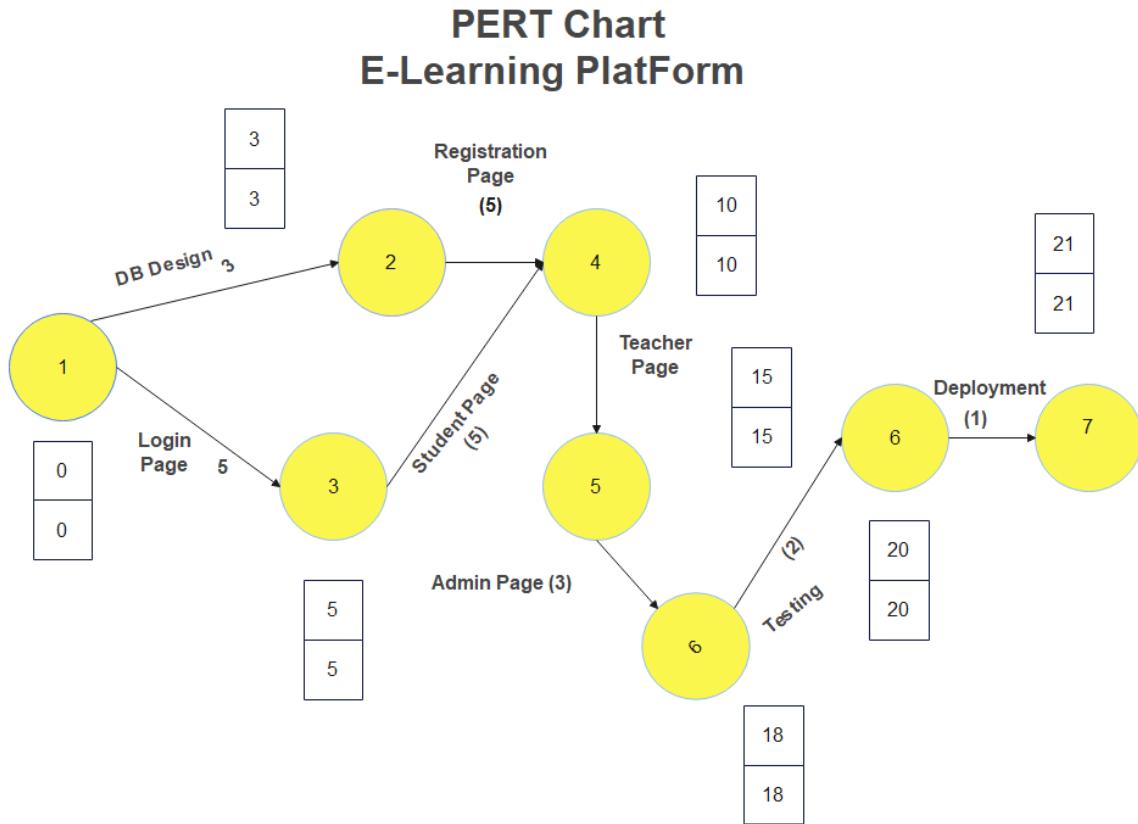
                <RemoveCircleOutline style={{ verticalAlign: "top", color: "#EEEEEE" }} disabled={inputFields.length === 1} onClick={() => handleRemoveFields(inputField.id)} />
                <AddCircle style={{ verticalAlign: "top", color: "#EEEEEE" }} onClick={handleAddFields} />
                <input style={{ verticalAlign: "top", color: "#EEEEEE" }} type="radio" name='correct' value={inputField.option} onClick={(e) => setCorrectOption(e.target.value)} />
                <Label style={{ verticalAlign: "top", color: "#EEEEEE" }} htmlFor="correct">Correct</Label>
              </div>
            )))

```

```
<div style={{ width: "100%", padding: "10px 5px", margin: "auto", textAlign: "center" }}>
    <Popup
        trigger=<Button className="formQButton" type='submit' style={{ width: "30%", marginRight: "10px" }}> Confirm </Button>
        position="top center"
        nested
    >
    </Popup>
    <Button
        className="formQButton" onClick={() => { close(); }}
        style={{ width: "30%", backgroundColor: "#ECE2E1", color: "#100F0F" }}> Close
    </Button>
</div>
</form>
</div>
)}
</Popup>
<TableCell></TableCell>
<TableCell></TableCell>
<TableCell></TableCell>
<TableCell align='right'><Link to={`/configure/${id.id}`}>
    <NextButton>Next<ArrowForward style={{ verticalAlign: "middle", transform: "scale(0.9)" }} />
    </NextButton></Link></TableCell>
</TableRow>
</TableHead>
</Table>
</TableContainer>
</Wrapper>
</Container>
<Footer />
</>
)
}

export default CreateQuiz
```

Appendix G: Project Management



GANTT CHART:

Task ID	Task Description	Start Date	Duration (Days)	End Date	Predecessor(s)
1.	Requirements Analysis	January 2, 2024	8	January 10, 2024	None
2.	Design	January 11, 2024	10	January 20, 2024	1
3.	Development	January 21, 2024	35	February 25, 2024	2
4.	Testing	February 26, 2024	4	February 29, 2024	3
5.	Deployment	March 1, 2024	31	March 31, 2024	4

January 2024 - January 2024 : [Requirements Analysis]

January 2024 - January 2024 : [Design]

January 2024 - February 2024 : [Development]

February 2024 - February 2024 : [Testing]

March 2024 - March 2024 : [Deployment]