EduTrack - IOD Capstone

A Web Application for School Management

Introduction

Purpose

What is the problem or the opportunity that the project is investigating?

The student management system project addresses the inefficiency and inaccuracy of managing student data through traditional methods like spreadsheets or paperwork. These outdated approaches are prone to errors, lack scalability, and make it challenging to access or analyze data effectively. This results in difficulties such as:

- Tracking student information, including attendance, grades, and personal details.
- Facilitating seamless communication between students, teachers, and administrators.
- Managing academic records across multiple departments or classes.
- Scaling the system to support the growth of schools and their student populations.

The project aims to seize the opportunity of creating a centralized, streamlined, and digital system to simplify student data management. By developing a web-based student management system, we can achieve ease of access, automation, data analysis, and scalability:

This solution tackles real-world challenges, improving efficiency, accuracy, and the overall user experience in managing educational data.

Why is this problem valuable to address?

Addressing this problem is valuable because:

- **Enhancing Efficiency and Reducing Errors:** Automating student data management saves time, minimizes manual errors, and streamlines administrative processes, making operations smoother and more reliable.
- **Improving Communication and Transparency:** Real-time access to information like grades, attendance, and schedules improves communication between students, teachers, and parents, fostering trust and accountability within the system.
- **Enabling Scalability and Growth:** A centralised system is designed to scale effortlessly, accommodating an increasing student population and evolving institutional needs, ensuring long-term adaptability and success.

- What is the current state (e.g. unsatisfied users, lost revenue)?

The student management system has made good progress but still needs more work. While it helps with managing student data, time limits have left some areas incomplete:

- **User Needs:** Teachers, administrators, and students may feel the system doesn't fully meet their needs because some important features are missing.
- **Limited Efficiency:** Tasks like advanced reports, better communication tools, and performance tracking are still manual or unavailable, making the system less effective.
- **Scalability Issues:** Without extra features, the system might struggle to handle larger datasets or more complex needs in the future.
- These gaps show the importance of investing more time and resources to improve the system and make it the best it can be.

What is the desired state?

The desired state of the student management system is one that is fully functional, scalable, and easy to use for everyone. The system should:

- **Offer More Features:** Include things like automatic grade calculation, performance tracking, detailed reports, real-time updates, and better communication between teachers, students, and parents.
- **Be Easy to Use:** Have a simple, intuitive interface where users can quickly access grades, attendance, schedules, and academic records.
- **Support Growth:** Handle more students, classes, and departments, and be flexible enough to add new features or integrate with other systems.
- **Protect Data:** Ensure student information is secure with strong data protection and privacy measures.
- **Help with Decisions:** Provide administrators with useful reports and data to help make better decisions about student progress, resources, and improvements.
- Achieving this will make the system more efficient, reliable, and user-friendly, improving productivity, communication, and supporting the growth of the institution.

Has this problem been addressed by other projects? What were the outcomes?

While many schools have adopted student management systems, there are still high schools that do not have one in place. In these schools, most of the work is done manually, such as tracking attendance, grades, and academic records. This manual process can lead to inefficiencies, errors, and difficulties in scaling as the school grows.

For those that have implemented student management systems, the outcomes have generally been positive. These systems have improved efficiency by automating tasks, reduced errors, and provided better access to student information. However, even in schools with a system, many still face challenges with outdated or incomplete features, limited scalability, or lack of integration with other tools.

The need for a more comprehensive and modern solution remains, especially for schools that have not yet transitioned to digital systems.

Industry/ domain

What is the industry/ domain?

The student management system falls within the education technology (EdTech) industry, specifically in the education management sector. This sector focuses on creating digital tools to help manage and improve various aspects of educational institutions, including:

Student Information Systems (SIS): Systems that store and manage student data, such as personal details, grades, and attendance.

Learning Management Systems (LMS): Platforms that deliver educational content, track student progress, and allow communication between students and teachers.

Institutional Administration: Tools that help with scheduling, resource management, and academic planning for schools and universities.

The system I'm building is part of student information management, which aims to make student data more efficient, accurate, and accessible for schools.

What is the current state of this industry? (e.g. challenges from startups)

The three best aspects of the current EdTech industry are:

Rapid Adoption of Digital Tools: More schools are moving to digital systems for managing student data, especially after the COVID-19 pandemic. Cloud-based student management systems are becoming popular because they are scalable and convenient.

Innovation by Startups: Startups are creating new, easy-to-use tools that use technologies like AI and data analytics. These tools focus on personalized learning, real-time progress tracking, and improving user experiences, challenging traditional systems.

Focus on Data-Driven Decision Making: Schools are increasingly using data to make better decisions. There is growing demand for tools that provide insights to track student progress, improve academic results, and better manage resources.

- What is the overall industry value-chain?

The EdTech industry value chain in education management involves several key stages:

1. **Technology Providers:** Develop software (SIS, LMS), hardware, and cloud services for educational needs.
2. **Content and Design:** Create user-friendly platforms with integrated educational content and intuitive interfaces.
3. **Implementation and Training:** Deploy systems, integrate with existing workflows, and train users.
4. **Users (Schools, Teachers, Students):** Use the tools for data management, learning, and communication.
5. **Support and Maintenance:** Offer technical support, updates, and system improvements.
6. **Analytics and Feedback:** Collect data and feedback to refine and enhance tools.
7. **Compliance and Standards:** Ensure alignment with regulations and data security standards.
8. **Market and Innovation:** Promote solutions and develop new technologies to meet evolving educational needs.

This chain ensures the efficient delivery and use of EdTech solutions in educational institutions.

What are the key concepts in the industry?

There are many key concepts in the EdTech industry, but here are five common examples:

1. **Student Information Systems (SIS):** Centralised platforms for managing student data, such as grades, attendance, and personal records.

2. **Learning Management Systems (LMS):** Tools for delivering educational content and tracking student progress.
3. **Automation:** Simplifying tasks like attendance tracking and report generation to save time and reduce errors.
4. **Data Security and Privacy:** Protecting sensitive student information and ensuring compliance with regulations.
5. **Data-Driven Decision Making:** Leveraging analytics to improve academic performance and optimize resources.

These are just a few examples, highlighting the broader range of concepts shaping the EdTech industry.

Is the project relevant to other industries?

Yes, the student management system is relevant to other industries that need to manage data, communication, and processes. For example:

- **Corporate Training:** Tracks employee training, attendance, and performance.
- **Healthcare:** Manages patient records, appointments, and treatments.
- **Human Resources:** Handles employee data, attendance, and evaluations.
- **Event Management:** Organizes registrations, schedules, and updates.
- **Customer Relationship Management (CRM):** Manages client data and communication.

These industries also use similar tools to improve efficiency and organization.

Stakeholders

Administrators:

- **Access:** Full access to all features, including student records, reports, attendance, communication tools, and system settings.
- **Why They Care:** They need the software to streamline data management, improve decision-making, and ensure institutional efficiency.
- **Expectations:** Easy-to-use tools, accurate reporting, and secure data management.

Teachers

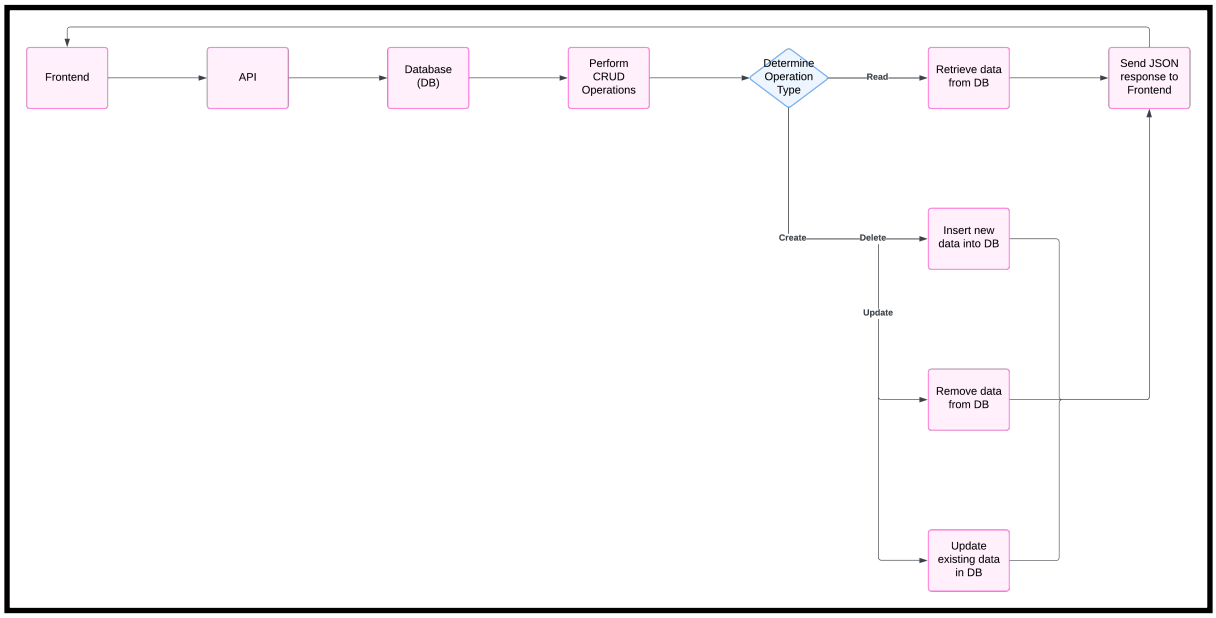
- **Access:** Access to their class rosters, student grades, attendance records, and communication tools for interacting with students and parents.
- **Why They Care:** To simplify classroom management, track student performance, and save time on administrative tasks.
- **Expectations:** User-friendly interface, quick access to data, and automation for grading and reporting.

Students

- **Access:** Limited access to their personal records, grades, attendance, schedules, and notifications.
- **Why They Care:** To stay informed about their academic progress and schedules.
- **Expectations:** Easy navigation, real-time updates, and secure access to their data.

Product Description

Architecture Diagram



The web application is built using a robust architecture that connects the frontend, API (backend), and database, ensuring seamless interaction and functionality. The application leverages modern libraries and frameworks to create an efficient, scalable, and maintainable system.

Frontend:

The frontend, built using React, serves as the user interface where users interact with the system. It utilizes libraries such as Material-UI (MUI) for responsive and modern UI components, react-router-dom for navigation, and axios for making HTTP requests to the backend API. Other libraries, such as dayjs for date management, enhance functionality and user experience.

User interactions, such as form submissions or data retrieval requests, are sent as HTTP requests to the backend.

API (Backend):

The backend is powered by Express.js, a lightweight and fast Node.js framework, which acts as a bridge between the frontend and the database. Key packages include:

Sequelize and mysql2 for database interaction.

jsonwebtoken for secure authentication.

bcryptjs for hashing passwords and ensuring secure user management.

validator to validate incoming data and enforce application rules.

When the frontend sends requests, the backend processes these, performs CRUD operations, and returns JSON responses.

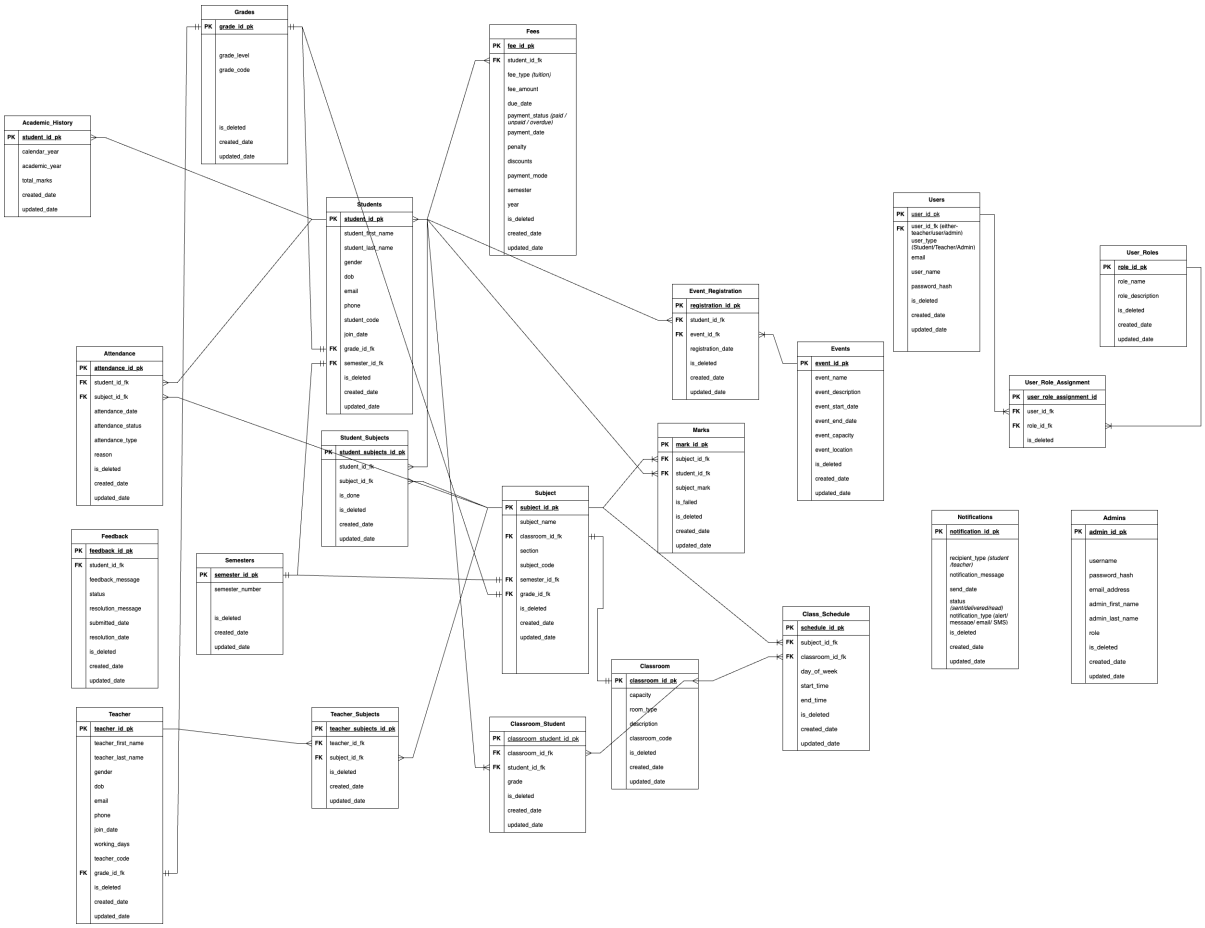
Database:

The application uses a MySQL database, managed with the help of Sequelize, an ORM that simplifies database queries while maintaining flexibility. The database securely stores user information, records, and other critical data.

Flow of Operation:

When a user interacts with the frontend (e.g., submitting a form), the frontend sends a corresponding HTTP request to the backend API using axios.
The backend processes the request, interacts with the MySQL database, performs the necessary operation (create, read, update, or delete), and returns a JSON response to the frontend.
The frontend dynamically updates the UI based on the response.

Entity Relationship Diagram



The **Student Management System (SMS)** database is designed to manage all aspects of student and academic information for an educational institution. It includes key tables like **Students, Teachers, Subjects, Classrooms, and Academic History,** along with tables for tracking **attendance, marks, fees, and notifications.**
The schema is organized to connect different pieces of information, using **primary and foreign keys** to link related data across tables. It also includes a soft delete feature *(is_deleted)* in each table to mark records as deleted without actually removing them. This helps keep track of data for future reference. User roles like **admin, teacher, and student** are also managed to control access to different parts of the system.
Some columns are designed to accept only certain values to maintain consistency. For example, in the **Attendance** table, the **status** column can only have values like *'Present', 'Absent', 'Late', or 'Excused'*. This ensures data is always consistent and accurate.

Overall, this schema supports features like student management, class scheduling, fee tracking, notifications, and report generation, and is designed to work well with both web and mobile platforms.

User Stories

#	User Story Title	User Story Description	Priority	Status
1	Users Sign In	Users sign in and get redirected to a dashboard based on their role (admin/teacher/student)	High	Achieved
2	Admin register a student	Admin can register a new student in a specific class	High	Achieved
3	Admin update a student's details	Admin can update registered student's details	High	Achieved
4	Admin Delete Student	Admin should be able to delete a student from the system	Mediu m	Yet to achieve
5	Admin enrol a student in a specific subject	Admin should be able to enrol a student in a subject - this should already be handled when registering a subject, but this functionality can be there for when a student wants to enrol in an extra subject	Mediu m	Achieved
6	Admin View subjects a student is enrolled in	Admin should be able to view the subjects a student is assigned in	High	Achieved
7	Admin register a teacher	Admin should be able to register a new teacher	High	Achieved
8	Admin update registered teacher	Admin should be able to update the details of already registered teacher	High	Achieved
9	View all subjects assigned to a teacher	Admin should be able to see subjects already assigned to a teacher	High	Achieved
10	Admin Delete Teacher	Admin should be able to delete a teacher from the system	Mediu m	Achieved
11	Admin assign subject to a teacher	Admin should be able to assign a subject to a teacher	High	Achieved
12	Admin Create a Classroom	Admin should be able to create a new classroom in the system	High	Achieved

13	Admin Update a classroom’s details	Admin should be able to update the details for a classroom	High	Achieved
14	Admin View all the classrooms	Admin should be able to view all the classrooms and the subject taught in them and the schedule for each subject and classroom	High	Achieved
15	Admin Create a new subject for a grade	Admin should be able to create a new subject in the system	High	Achieved
16	Admin update subject details	Admin should be able to update the details of a subject	High	Create but more validation needed
17	Admin delete subject	Admin should be able to delete a subject	High	Achieved
18	View all subjects	Admin should be able to view all the subjects	High	Achieved
20	Teacher view all assigned subjects	Teacher should be able view all the subjects that is assigned to	High	Achieved
21	Teacher mark student’s attendance	Teacher should be able to mark student’s attendance	High	Achieved
22	Teacher mark student's marks	Teacher should be able to mark his students’ marks achieved	High	Achieved
23	Teacher promote student top next class	Teacher should be able to promote their student to the next class if they’ve pass all the subjects	High	Achieved
24	Student view all their subjects and marks achieved for the subject	Students should be able to view all their subjects and the marks they’ve got on there	High	Achieved
25	Students view their attendance for each subject	Student should be able to view their attendance for each subject	High	Achieved

User Flow

Admin’s Flow Chart:

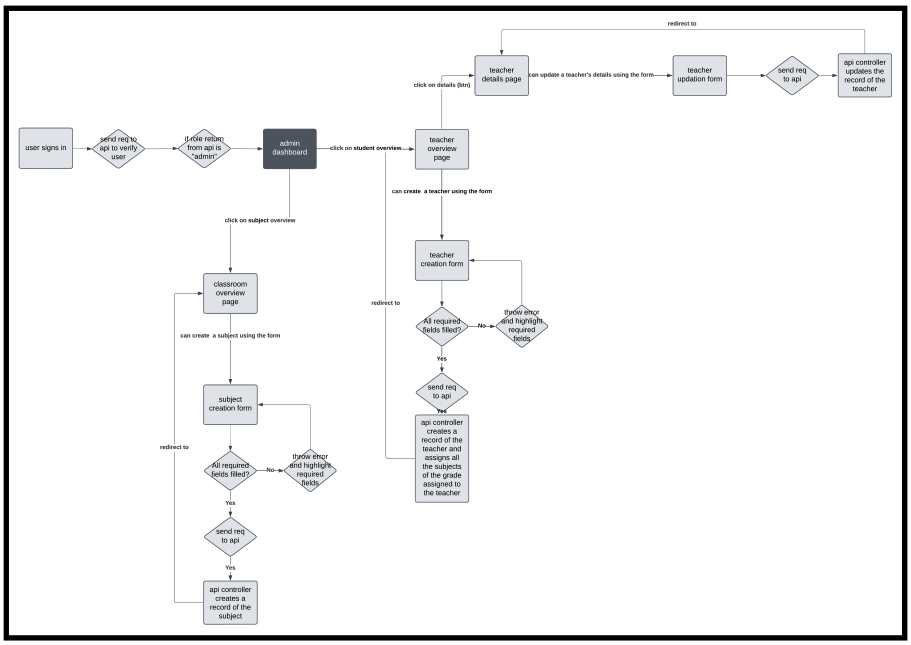


Figure 1: admin role flow chart part 1

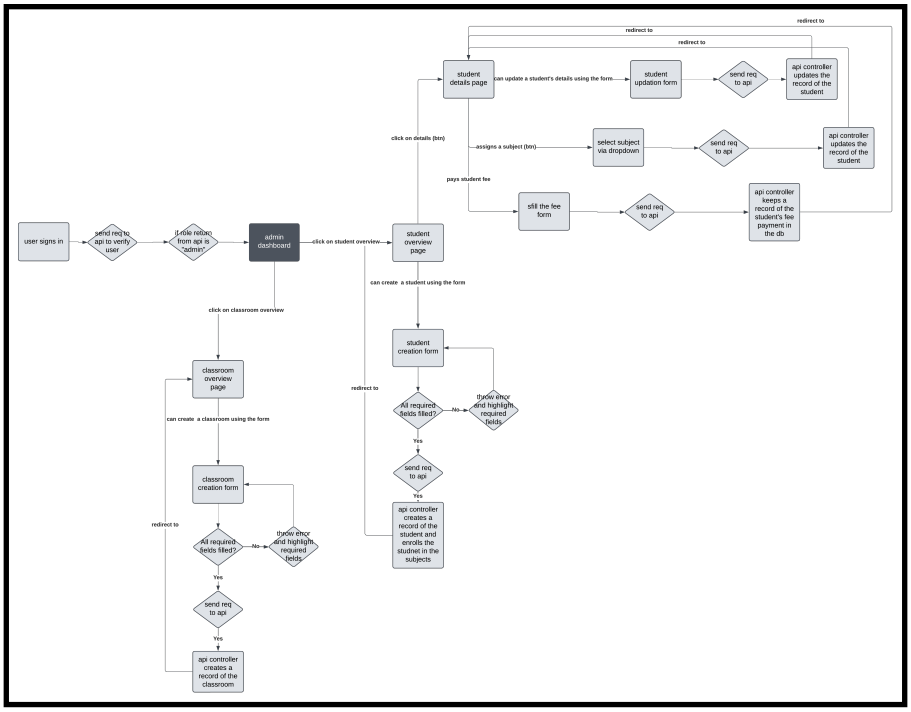


Figure 2: admin role flow chart part 1

Teacher's Flow Chart:

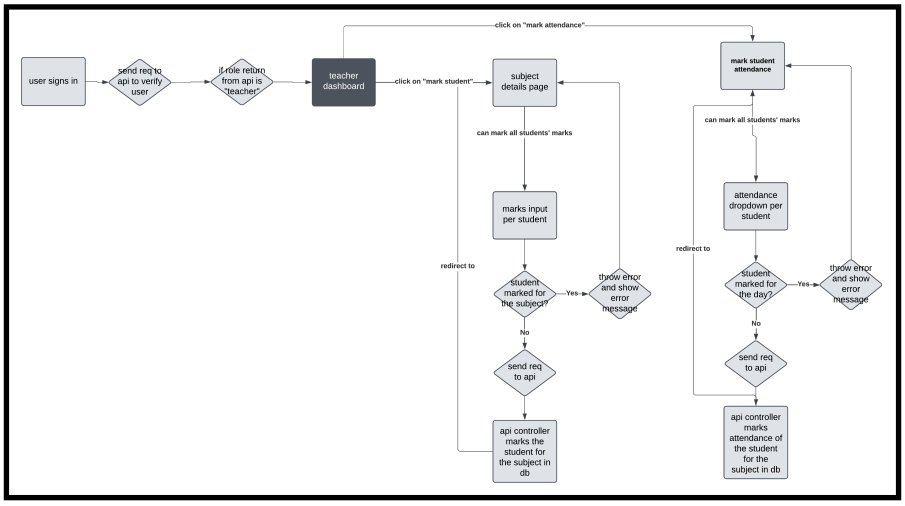


Figure 3: teacher role flow chart part 1

Student Flow Chart:

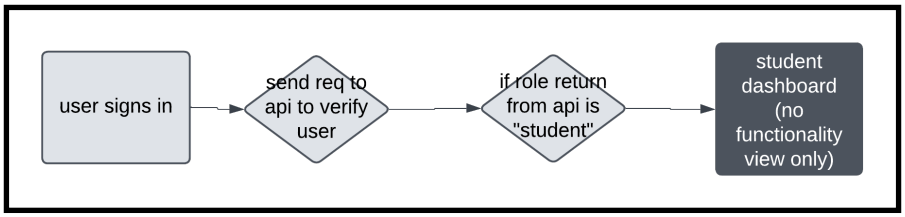
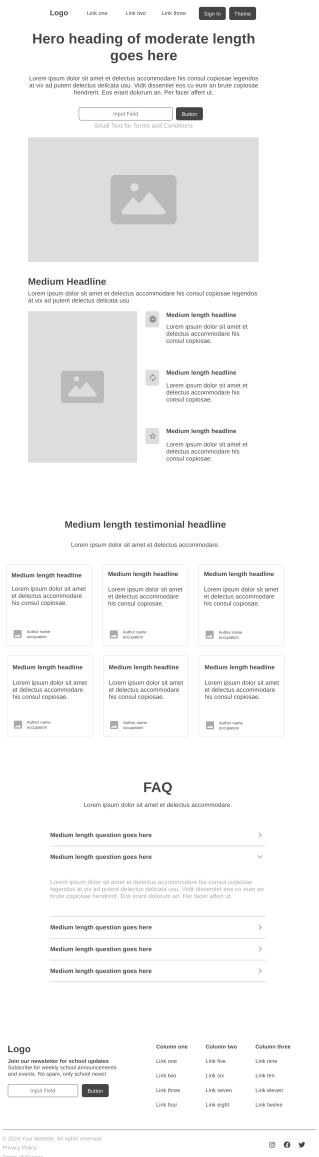


Figure 4: student role flow chart part 1

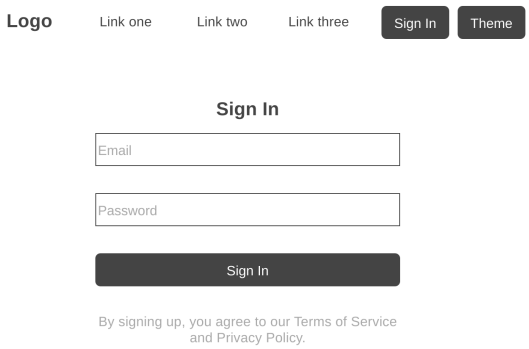
Wireframe Design

The wireframe design section shows the basic layout and structure of the application, acting as a guide for its user interface. These wireframes were created to ensure a clear and user-friendly design that supports the app's functionality. They mapped out the placement of key elements, navigation, and interactions, providing a solid starting point for development and maintaining consistency. While the final website follows the core ideas of the wireframes, it doesn't match them exactly due to changes and improvements made during development.

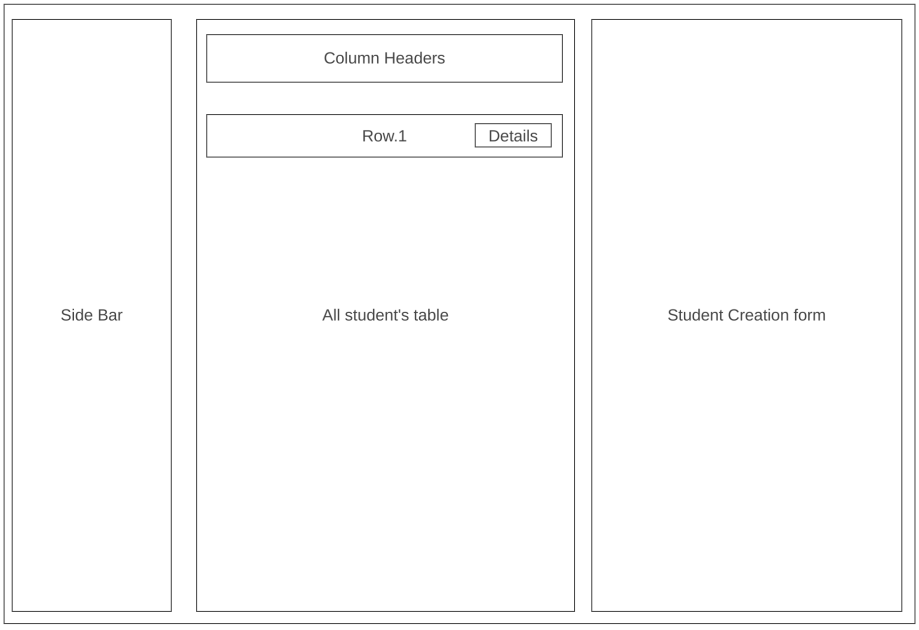
1. Landing Page Wireframe:



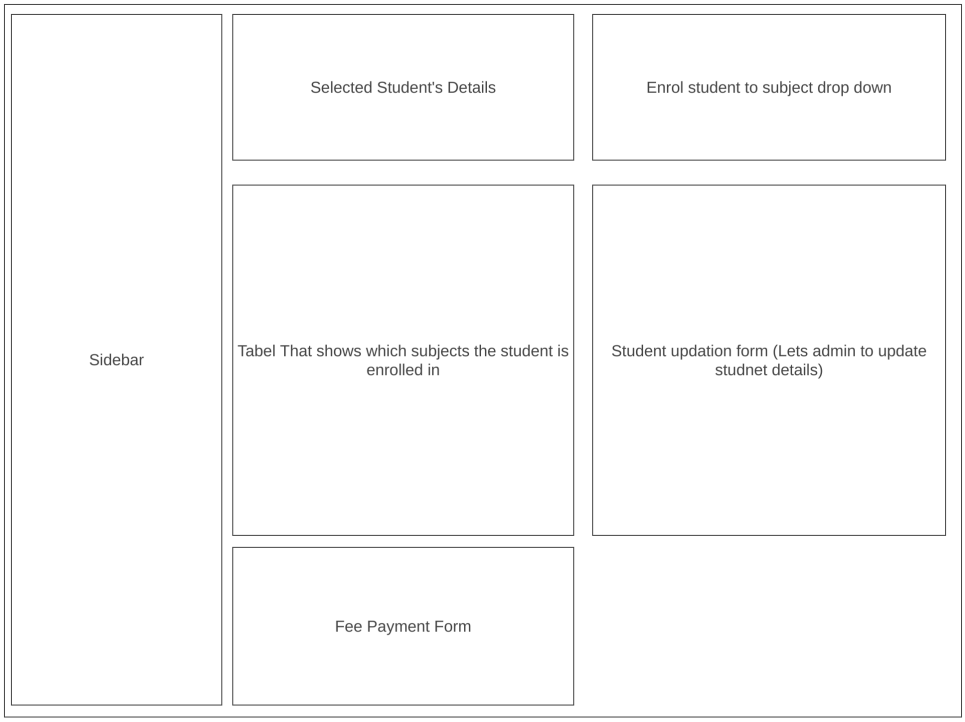
2. Sign In Page:



3. **Admin Student Overview:** The teacher overview page follow the same wireframe design for the UI.**Admin Student Details:** The teacher details page follow the same wireframe design for the UI.

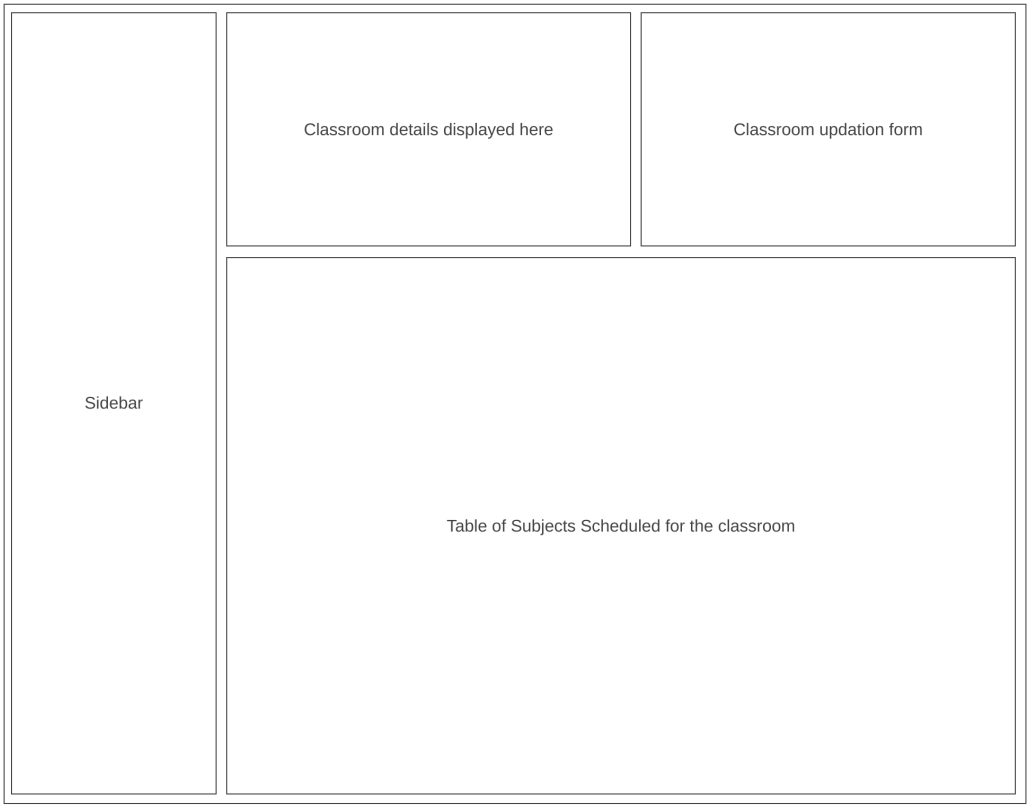


4. Admin Student Details Page:

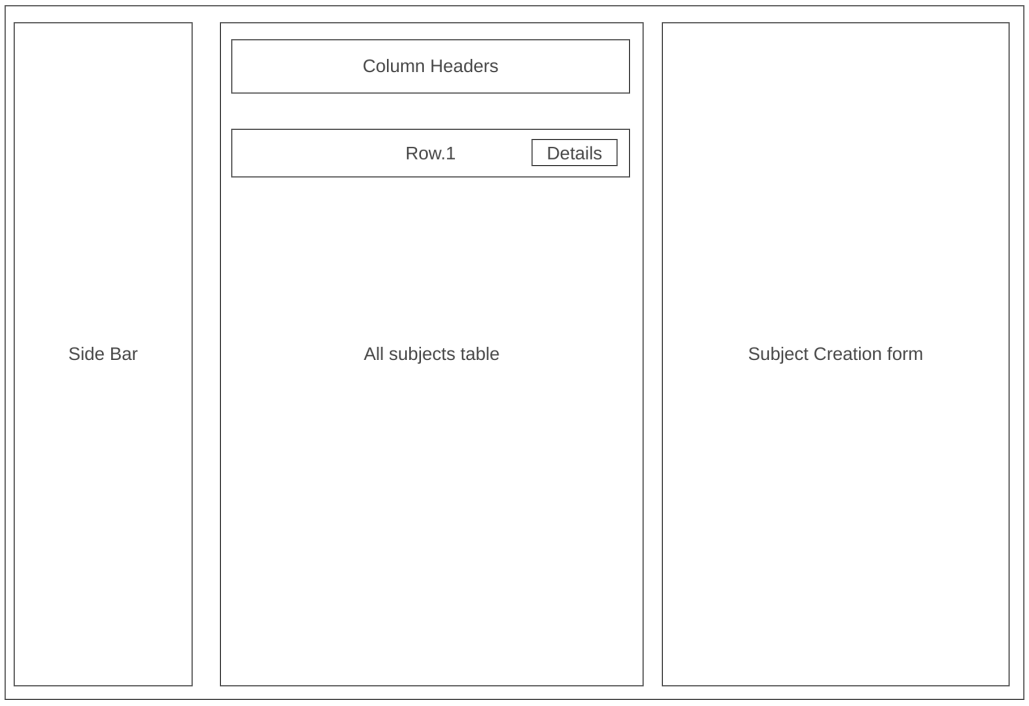


5. Admin Classroom Overview:

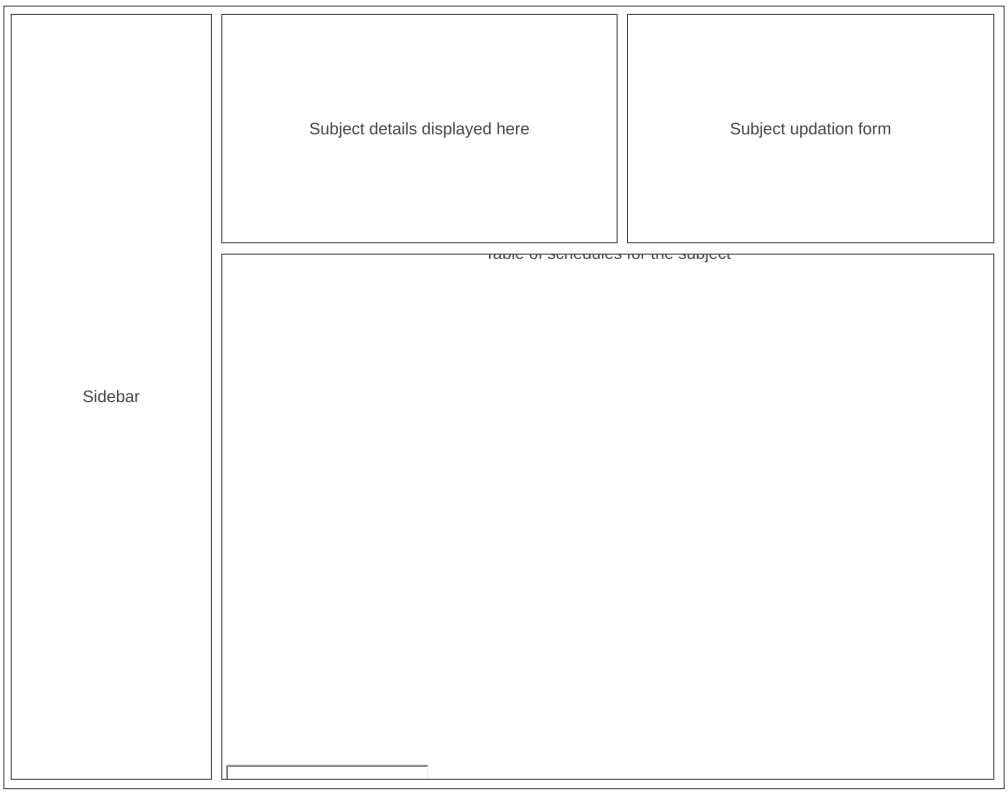
6. Admin Classroom Details



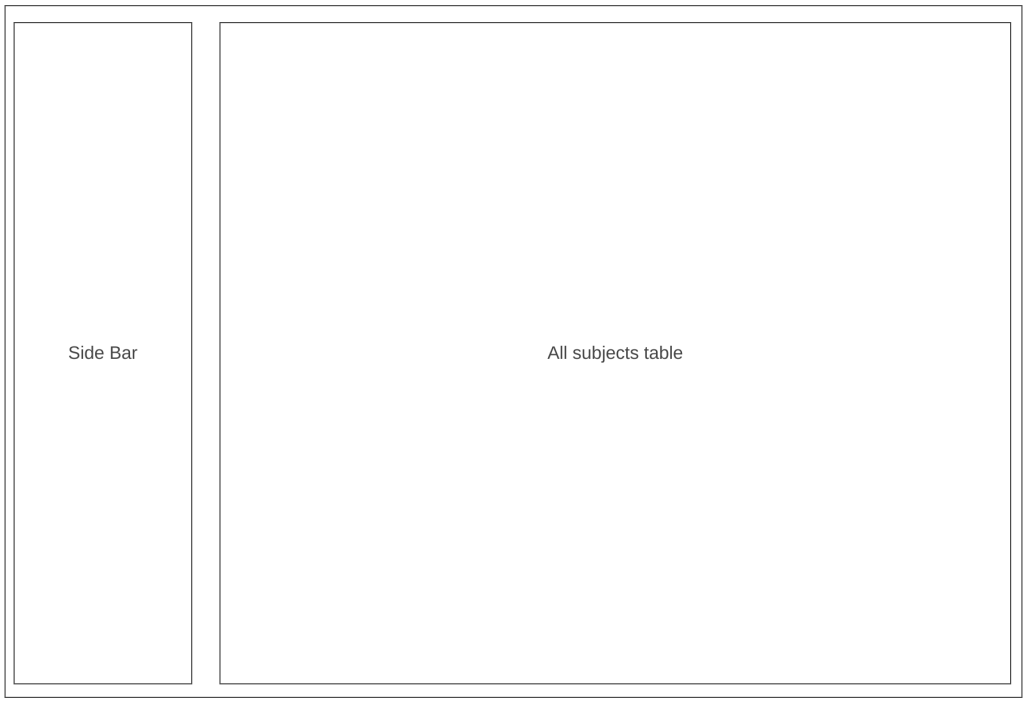
7. Admin Subject Overview:



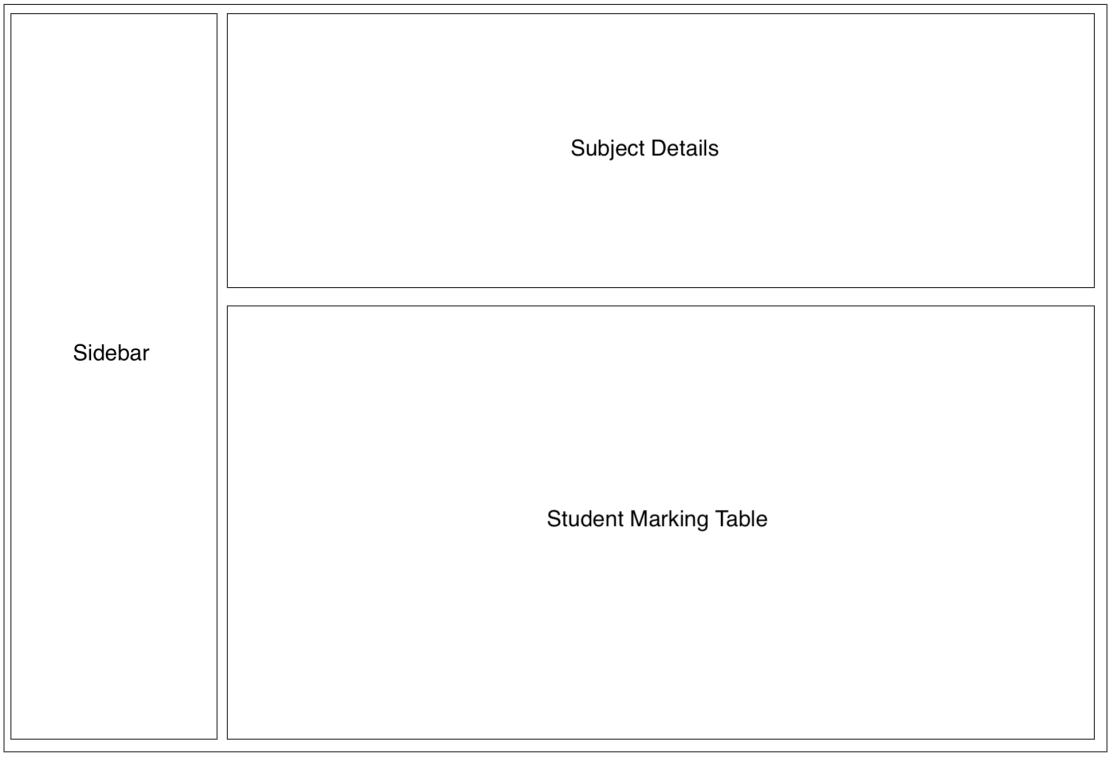
8. Admin Subject Details:



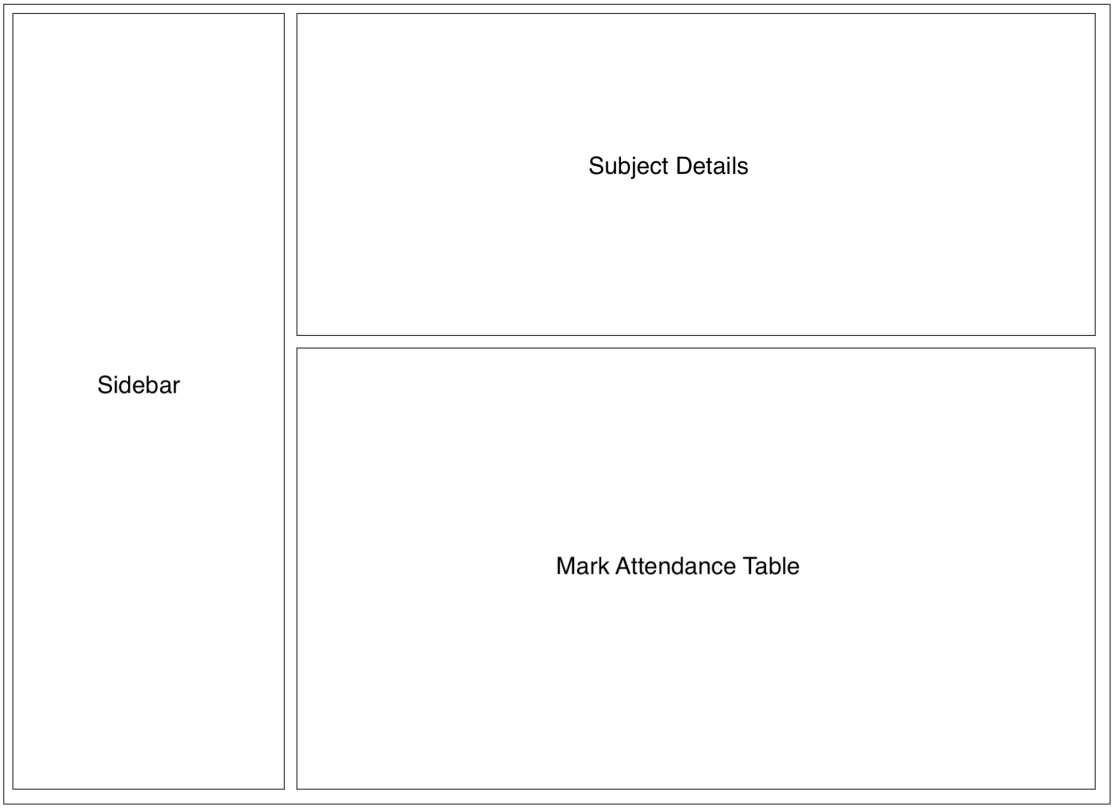
9. Teacher Subject Overview



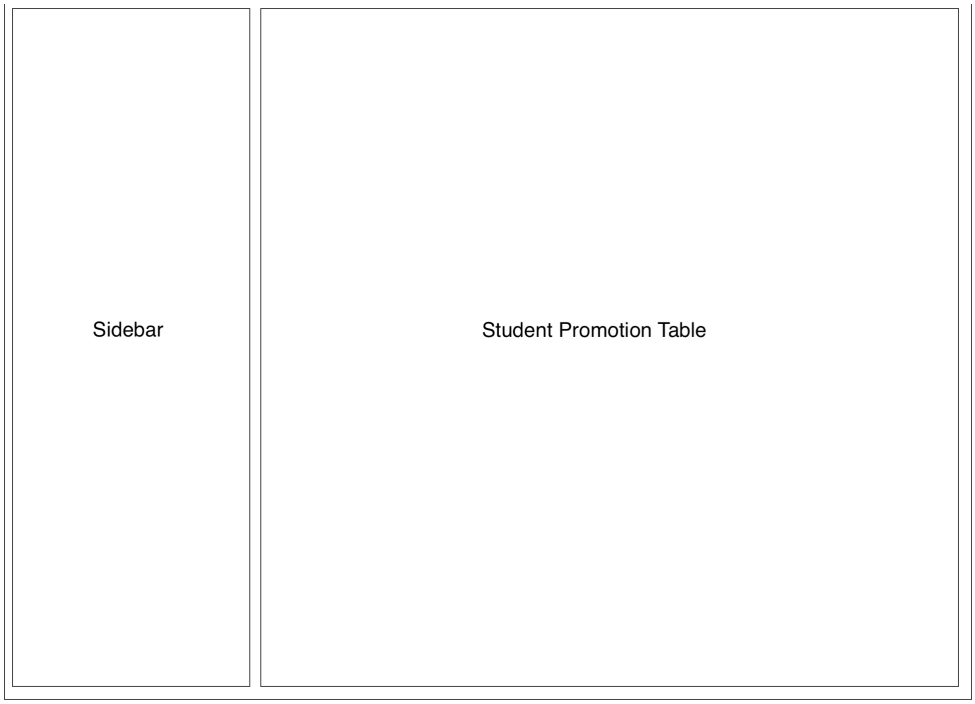
10. Teacher Mark Students:



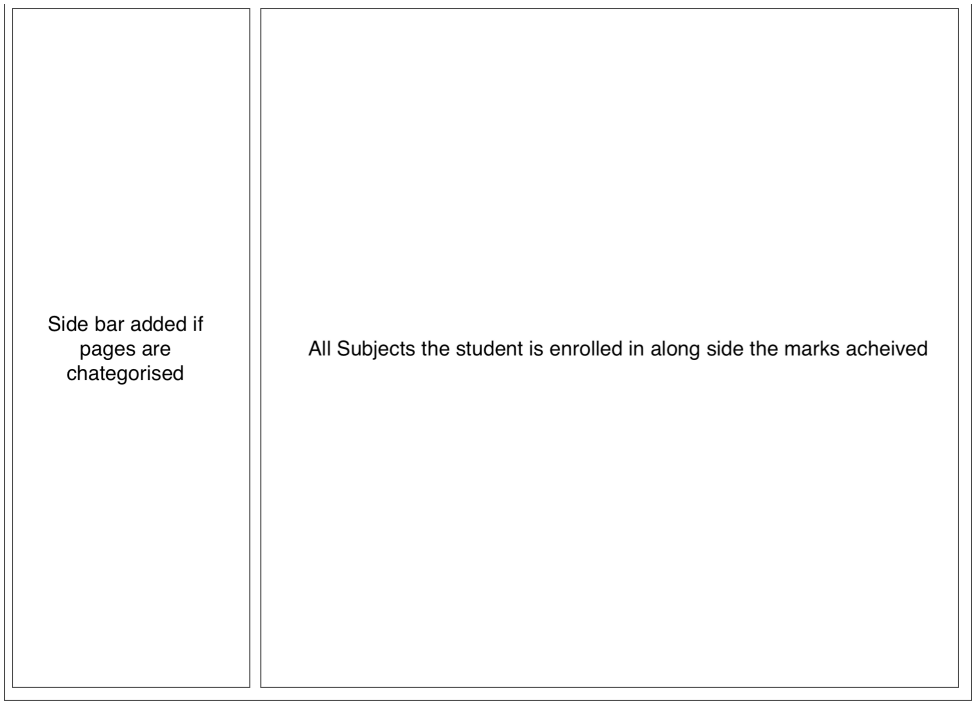
11. Teacher Mark Student Attendance:



12. Teacher Promote Student:

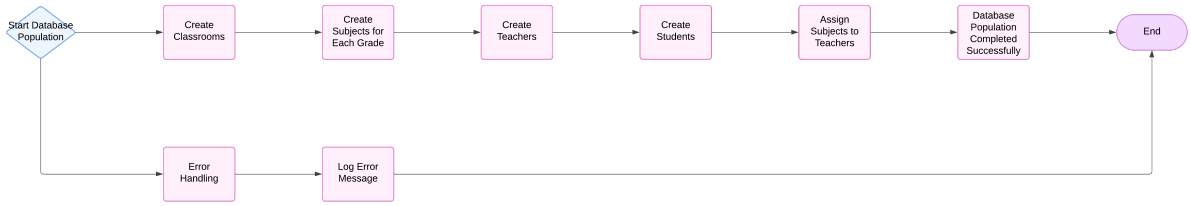


13. Student Subjects Overview:



14. Student View Subject Attendance:

Database Population Flow



This flow represents the step-by-step process for populating the database, intended for implementation in both the backend and frontend. The sequence ensures a structured and error-free setup. The steps described here are essential for initializing the system and should be followed in the order specified. This process ensures consistency across the platform.

Steps for Frontend Implementation

1. Start Database Population
    - Initiates the process to populate the database.
    - A trigger, such as a button click or automated event, initiates the workflow.
2. Create Classrooms
    - Creates classroom records in the database, specifying identifiers such as names, sections, or grades.
    - This step sets up the foundation for organizing subjects, teachers, and students.
3. Create Subjects for Each Grade
    - Assigns subjects to each grade or classroom.
    - The subjects are created dynamically based on grade requirements, ensuring scalability for diverse educational setups.
4. Create Teachers
    - Adds teacher profiles to the system, including their assigned details such as name, department, and unique IDs.
    - Teachers will later be linked to the subjects and classrooms.
5. Create Students
    - Populates the database with student records, including their names, grades, and unique IDs.
    - Students are linked to classrooms for attendance and academic tracking.
6. Assign Subjects to Teachers
    - Maps subjects to the appropriate teachers.
    - Ensures each subject is managed by a specific teacher, establishing clear ownership and responsibilities.
7. Database Population Completed Successfully
    - Confirms the successful completion of the entire process.

- The system ensures that all required entities (classrooms, subjects, teachers, students) are created and interlinked.

8. Error Handling (Optional Path)
    - If an error occurs during any step, the process redirects to error handling mechanisms:
        - Logs error messages for troubleshooting.
        - Prevents partial or corrupt data entry by halting further steps.
9. Log Error Message
    - Captures detailed error information, including the step at which the failure occurred.
    - Logs help identify and resolve issues promptly.
10. End
    - Marks the termination of the database population workflow.
    - Either the process completes successfully or stops due to errors.

Key Points

- **Frontend Role:**
  This flow highlights the steps that need to be implemented on the frontend to interact with the backend APIs. Each action (e.g., classroom creation, subject assignment) should call the respective backend API endpoint in the order specified.
- **Sequential Dependency:**
  Each step depends on the successful completion of the previous step. For instance, subjects cannot be assigned to a classroom until the classroom exists.
- **Error Management:**
  Errors should be caught and displayed to the user in real-time. Suggestions to retry or fix issues can enhance user experience.
- **Scalability:**
  The flow is designed to support future features like assigning additional roles or more granular permissions.

This flow ensures that the database setup is robust and aligned with the platform's operational requirements.

Open Questions/Out of Scope

While the Student Management System successfully implements its core functionalities, certain features were left out of the current scope due to time and resource constraints. These features, though valuable, may be incorporated in future iterations to enhance the system's capabilities.

Features Out of Scope

- **Notification System via Email**
  Although a notification system was planned to alert students and staff about important updates or reminders through email, this feature was not implemented in the current version.
- **Feedback System**
  The addition of a feedback system where users could submit their suggestions or report issues to administrators was planned but not prioritized for this release.
- **Automated Fee Payment System**
  While the application allows tracking of student fees, payment processing through third-party gateways was not included.
- **Advanced Analytics and Visualization**
  Complex data visualizations and analytics, such as performance trends or attendance heatmaps, were excluded to focus on core functionality.
- **Parent/Guardian Access**
  Separate access for parents or guardians to monitor student progress remains a feature for future development.

Open Questions

- What additional channels (e.g., SMS or in-app notifications) should be considered for the notification system?
- How would the feedback system handle anonymous feedback, and what moderation mechanisms are necessary?
- Should the application integrate with external Learning Management Systems (LMS) for seamless data exchange?
- Is there a need for multi-language support to cater to a diverse user base?

By documenting these features and questions, the project lays the groundwork for iterative improvements and aligns future development with user needs and feedback.

Non-functional Requirements

The Student Management System includes several non-functional requirements that enhance user experience, security, and overall usability. These requirements do not directly add to the application's feature set but are essential for ensuring a seamless and secure interaction with the system.

User Authentication

A secure and smooth user authentication process is crucial for protecting student and administrative data. The backend uses JSON Web Tokens (JWT) and HTTP-only cookies to manage user sessions securely. This ensures that sensitive tokens are inaccessible to client-side JavaScript, reducing the risk of security breaches. All requests for restricted actions (e.g., updating student records, viewing

sensitive information) are validated using the stored JWTs, ensuring that only authorized users can access these functionalities.

Ease of Use

The interface is designed to be intuitive and user-friendly, following familiar design patterns that improve navigation and usability. Some key design elements include:
Card-based layouts for displaying data such as student profiles, course details, and attendance records.

- **Clickable elements** like buttons that link to detailed pages for additional information.
- **Interactive tooltips** on hover to describe buttons and actions for better guidance.
- **Consistent iconography** across the platform to make navigation intuitive.
- **Snake bar notifications** for actions like saving or deleting data, providing immediate feedback.
- **Clickable logo** in the header that redirects users to the home dashboard.

Data Handling

The application uses Sequelize ORM for database management, which enforces structured data handling and reduces errors. Additionally, Day.js is used for date and time formatting to maintain consistent timestamp formats throughout the platform.

By implementing these non-functional requirements, the Student Management System ensures security, reliability, and an optimal user experience, allowing administrators and students to interact with the platform confidently.

Project Planning

Each task required to complete the application features was organized with a deadline to ensure timely project completion. Trello was used to track these tasks (see Figure 5), categorizing them into Frontend, Backend, Testing, and a General (Non-Technical) list. This categorization allowed for clear prioritization, ensuring backend tasks were completed before frontend integration, and testing tasks were aligned with the development timeline. Non-technical tasks, such as documentation and project coordination, were tracked separately to streamline the overall workflow. By monitoring progress across these categories, the project stayed on schedule and maintained a structured approach to feature implementation.

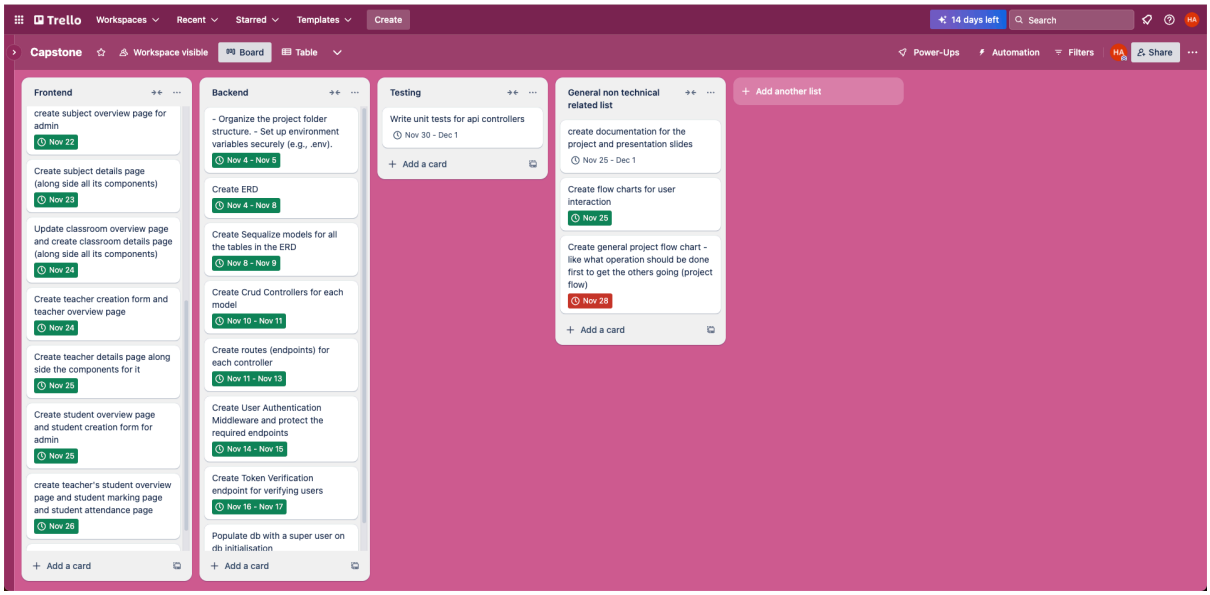


Figure 6: Trello project planning

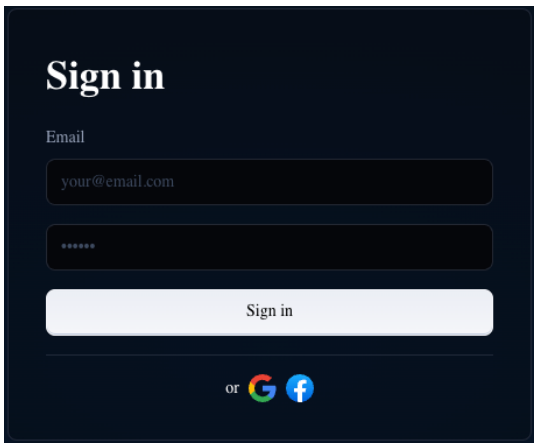
Testing Strategy

The testing strategy for the project ensured that all features were thoroughly evaluated for both backend and frontend functionality. For the backend, each API operation was tested using Postman, with a collection of all endpoints provided, including pre-filled body contents for convenience. Debugging involved extensive use of console logs to identify and resolve issues effectively. Additionally, unit tests were written for nearly 70% of the CRUD operations, ensuring robust backend functionality. For the frontend, I interacted with all implemented features to verify proper integration and usability. Each feature was subjected to two steps of testing: backend validation followed by frontend interaction. Features were only implemented on the frontend after successfully passing backend tests. Below, I have included tables for each endpoint used in the frontend, specifying the type of endpoint, required fields, and optional fields. This systematic approach ensured a reliable and user-friendly application.

NOTE: Below the features are categorised based on user roles (admin/teacher/student) since each has a different dashboard and key of features. If a user that doesn't have enough Insufficient permissions, they will get redirected to the dashboard for the their role.

User Authentication:

API Action used for the feature	**Required Fields**	**API Endpoint**
Create (Sign In User) - POST	**email, password -** email should be unique	http://localhost:3000/user/login-user



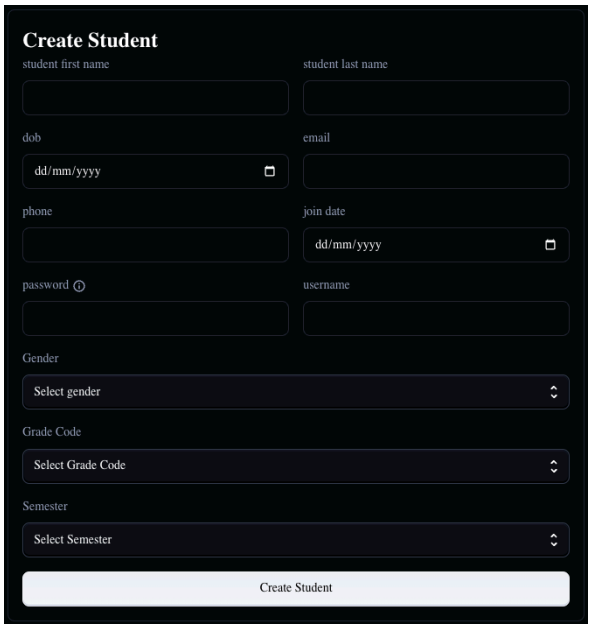
User sign in and gets a role assigned and get a token stored in the local storage for authentication. The role and token gets sent from the API.

ADMIN Related:

Create Student:

This calls one endpoint of the API and the controller populates multiple tables in the backend.

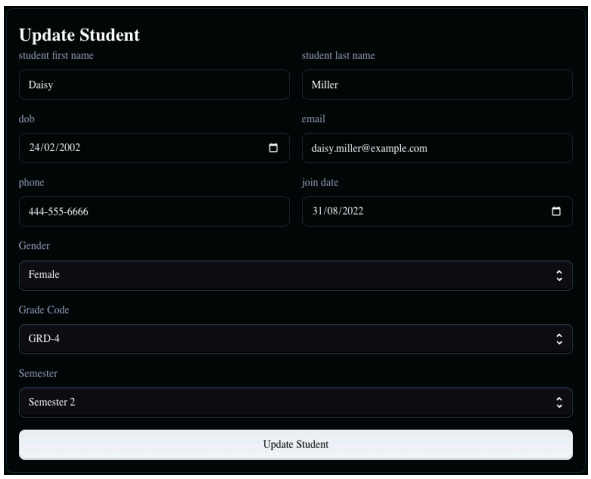
API Action used for the feature	Required Fields	API Endpoint
Create (Register Student) - POST	**multiple*** - all the fields shown in the below figure.	http://localhost:3000/ student/create-student



For selecting the grade a drop down has been used to minimise errors.

Update Student: This calls one endpoint of the API and the controller update the modified fields in tables in the database.

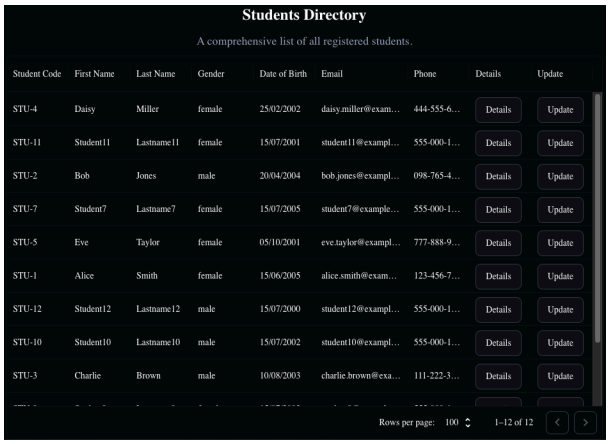
API Action used for the feature	Required Fields	API Endpoint
Update (Update Student Details) - PUT	multiple* - User can update any detail they want the others remain the same	http://localhost:3000/ student/update-student-details-by-id/studnetId



The inputs are already pre-filled with the available data for the student, admin can update any input they want.

View All Students: This calls the API and the controller returns all the students in the system.

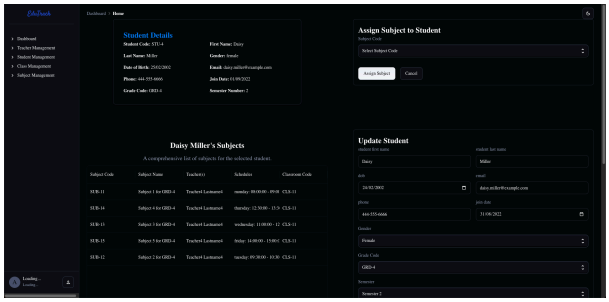
API Action used for the feature	Required Fields	API Endpoint
Reads (Gets all The Students) - GET	It's a get request that returns all the students in the system	http://localhost:3000/ student/get-all-students



This table shows all the students in the system, but you can also filter based on every column.
The below table shows the content yo see when yo click on the details button.

Specific Student's details page: This calls multiple api endpoints which are listed below.

API Action used for the feature	Required Fields	API Endpoint
Reads (Get particular student's data) - GET	It's a get request that returns a student's data by their id.	http://localhost:3000/ student/get-specific-student/ studentId
Update (Update Student Details) - PUT	multiple* - User can update any detail they want the others remain the same	http://localhost:3000/ student/update-student-details-by-id/studnetId
Reads (Gets all the subject for the student)	It get the subject the student is enrolled in.	http://localhost:3000/ subject/get-subjects-for-student/studentId

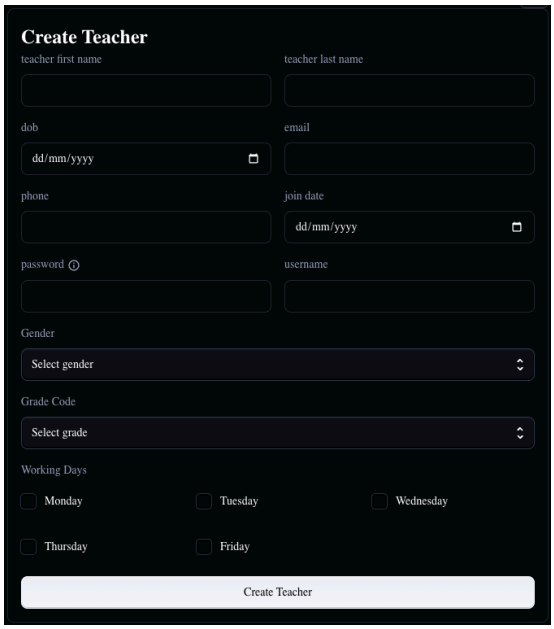


This is the student details page that shows the details related to a specific student.

Register teacher:

This calls one endpoint of the API and the controller populates multiple tables in the backend.

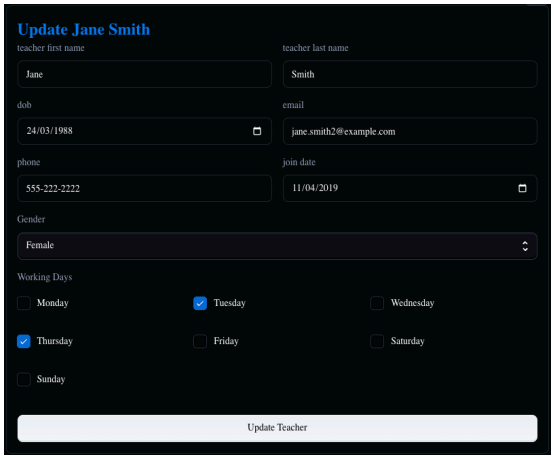
API Action used for the feature	Required Fields	API Endpoint
Create (Register Teacher) - POST	**multiple*** - all the fields shown in the below figure.	http://localhost:3000/ teacher/create-teacher



This form registers a teacher in the school/system.

Update Teacher: This calls one endpoint of the API and the controller updates the modified fields in tables in the database.

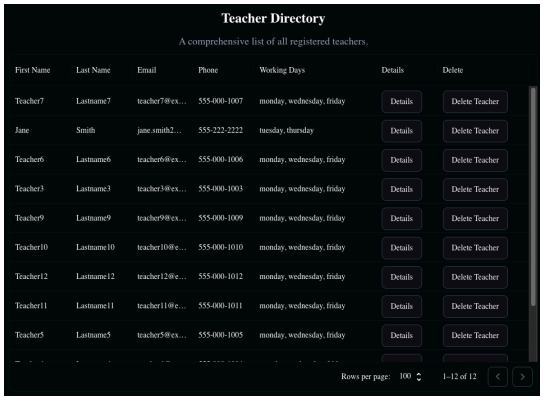
API Action used for the feature	Required Fields	API Endpoint
Update (Update Teacher Details) - PUT	multiple* - User can update any detail they want the others remain the same	http://localhost:3000/ teacher/update-teacher-details-by-id/teacherId



The inputs are already pre-filled with the available data for the teacher, admin can update any input they want. However there are validations, like if a teacher is currently teaching a class and the admin tries to change his grade/class level, he will get an error in a snake bar in the bottom of the screen.

View All Teachers: This calls the API and the controller returns all the teachers in the system.

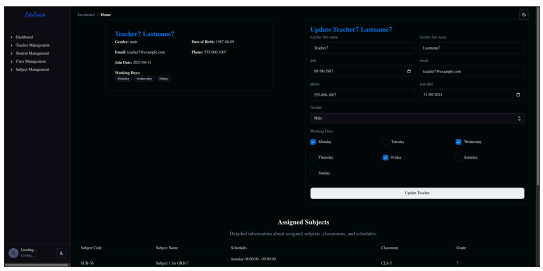
API Action used for the feature	Required Fields	API Endpoint
Reads (Gets all the teachers) - GET	It's a get request that returns all the teachers in the system	http://localhost:3000/ teacher/get-all-teachers



This table shows all the teachers in the system, but you can also filter based on every column.
The below table shows the content yo see when yo click on the details button.

Specific Teacher's details page: This calls multiple api endpoints which are listed below.

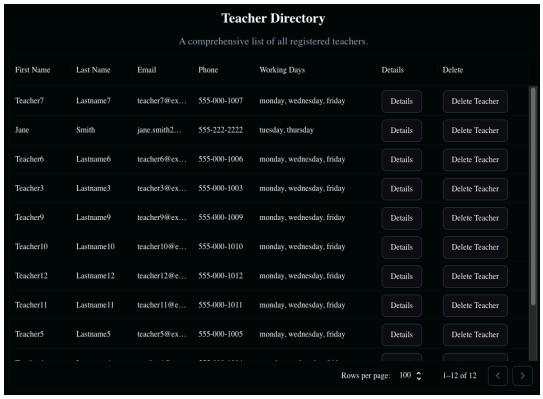
API Action used for the feature	Required Fields	API Endpoint
Reads (Get particular teacher's data) - GET	It's a get request that returns a teacher's data by their id.	http://localhost:3000/ teacher/get-specific-teacher/ teacherId
Update (Update teacher's details) - PUT	multiple* - User can update any detail they want the others remain the same	http://localhost:3000/ teacher/update-teacher-details-by-id/teacherId
Reads (Gets all the subject for the teacher)	It get the subject the teacher is assigned with.	http://localhost:3000/ teacher/get-assigned-subjects/teacherId



This is the teacher's details page that shows the details related to a specific student.

Delete a Specific Teacher: This calls the api endpoint which soft deletes a teacher.

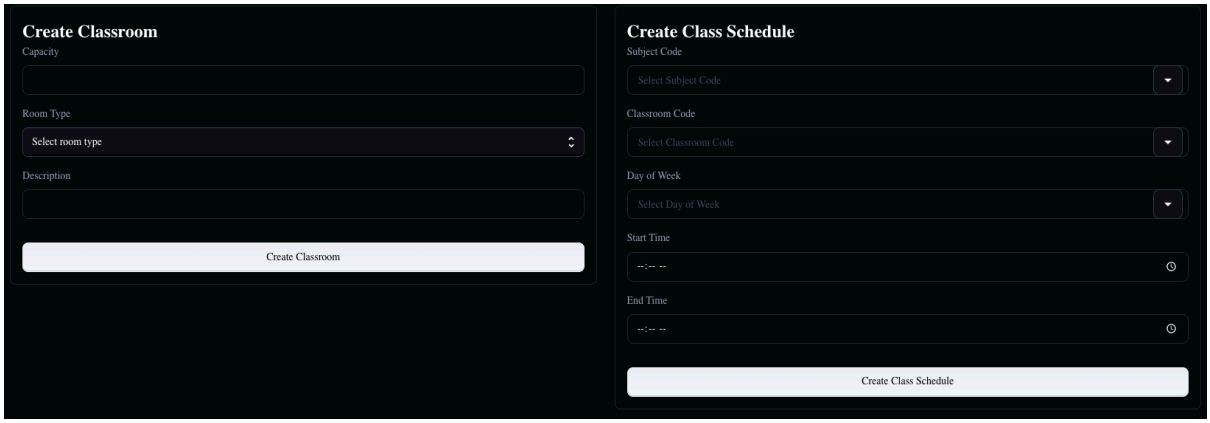
API Action used for the feature	Required Fields	API Endpoint
Deletes (Deletes particular teacher) - DELETE	It's a delete request that deletes a teacher by their id.	http://localhost:3000/ teacher/delete-teacher/ teacherId



The delete Botton will delete a teacher but if a teacher its assigned with a subject, that isn't delete-able.

Create Classroom:
This calls one endpoint of the API and the controller creates a classroom in the system and the right form lets the admin schedule that classroom or any other classroom.

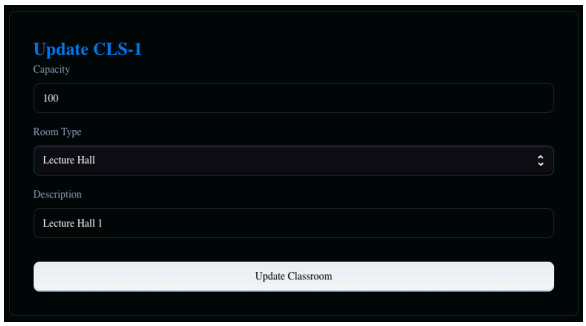
API Action used for the feature	Required Fields	API Endpoint
Create (Register Classroom) - POST	**multiple*** - all the fields shown in the below figure.	http://localhost:3000/ classroom/create-classroom
Create (Schedule a Classroom) - POST		



The first form lets the admin create a classroom and the other one lets him schedule a subject to be taught in that classroom. Admin can also schedule a classroom for a subject when creating a subject.

Update Classroom: This calls one endpoint of the API and the controller updates the modified fields in tables in the database.

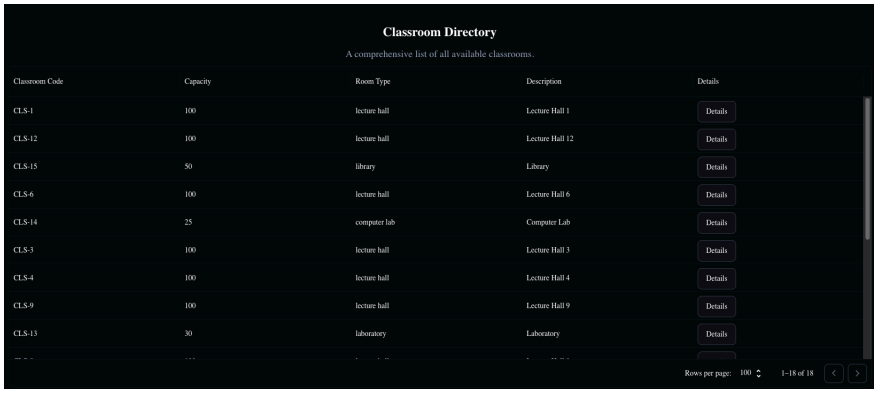
API Action used for the feature	Required Fields	API Endpoint
Update (Update Classroom Capacity) - PUT	Capacity, room type, and description	http://localhost:3000/ classroom/update-classroom-details-by-id/classroomId



This form lets you change the capacity for a classroom.

View All Classrooms: This calls the API and the controller returns all the classrooms in the system along side the subject taught and the schedule for it.

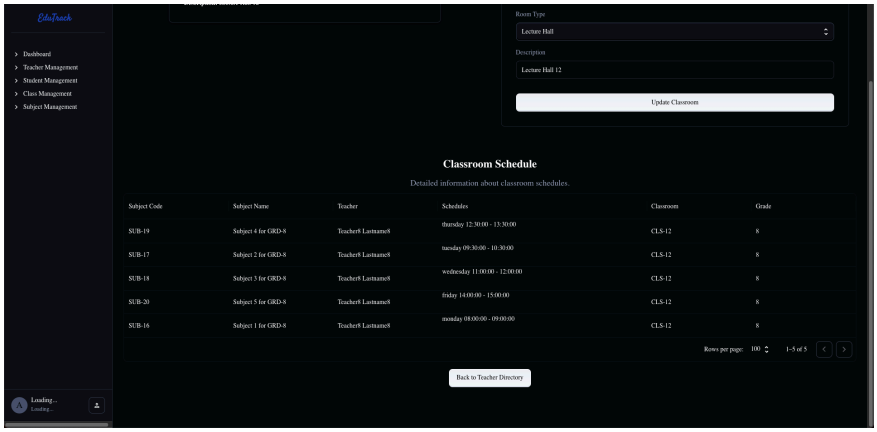
API Action used for the feature	Required Fields	API Endpoint
Reads (Gets all the classrooms) - GET	It's a get request that returns all the classrooms in the system	http://localhost:3000/classroom/get-all-classrooms



This table shows all the classrooms in the system, but you can also filter based on every column. The below table shows the content you see when yo click on the details button.

Specific Classroom's details page: This calls multiple api endpoints which are listed below.

API Action used for the feature	Required Fields	API Endpoint
Reads (Get particular Classroom's data) - GET	It's a get request that returns a Classroom's data by their id.	http://localhost:3000/classroom/get-specific-classroom/classroomId
Update (Update classroom's details) - PUT	multiple* - User can update any detail they want the others remain the same	http://localhost:3000/classroom/update-classroom-details-by-id/classroomId
Reads (Gets all the subject and schedule for the subjects taught in this classroom.	It get the subject the teacher is assigned with.	http://localhost:3000/classroom/get-classroom-schedule/classroomId

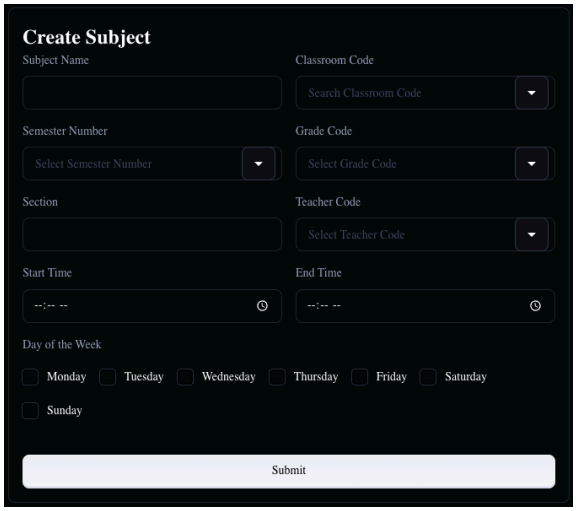


This is the classroom's details page that shows the details related to a specific classroom.

Create subject:

This calls one endpoint of the API and the controller populates multiple tables in the backend.

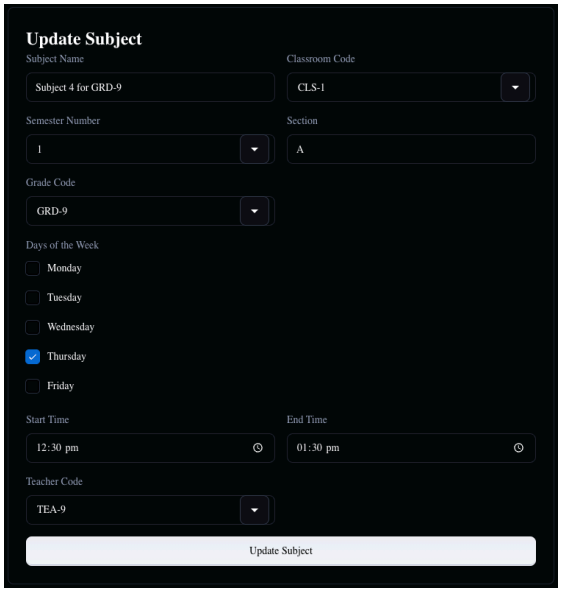
API Action used for the feature	Required Fields	API Endpoint
Create (Register Subject) - POST	**multiple*** - all the fields shown in the below figure.	http://localhost:3000/ subject/create-subject



This form registers a subject in the school/ system.

Update Subject: This calls one endpoint of the API and the controller updates the modified fields in tables in the database.

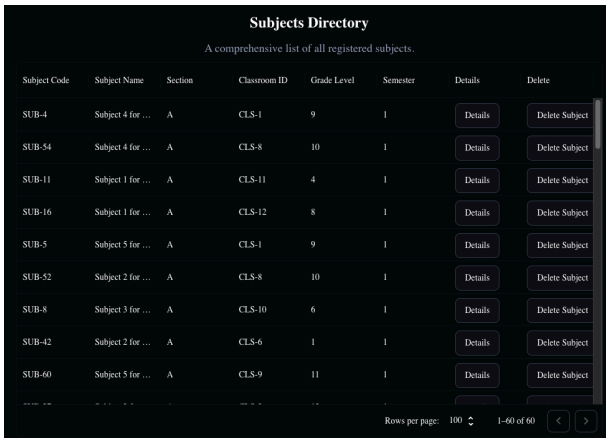
API Action used for the feature	Required Fields	API Endpoint
Update (Update Subject Details) - PUT	multiple* - User can update any detail they want the others remain the same	http://localhost:3000/ subject/update-subject-details-by-id/subjectId



The inputs are already pre-filled with the available data for the subject, admin can update any input they want. However there are validations, like if the teacher doesn't work on the specific days that teacher can't be assigned to take the subject.

View All Subjects: This calls the API and the controller returns all the subjects in the system.

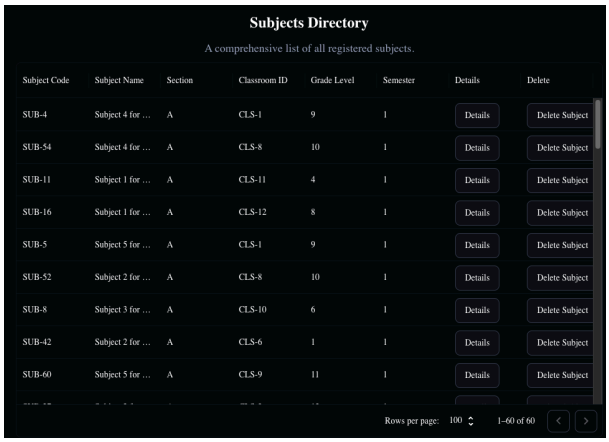
API Action used for the feature	Required Fields	API Endpoint
Reads (Gets all the subjects) - GET	It's a get request that returns all the subjects in the system	http://localhost:3000/ subject/get-all-subjects/ subjectId



This table shows all the subjects in the system, but you can also filter based on every column.

Delete a Specific Subject: This calls the api endpoint which soft deletes a subject.

API Action used for the feature	Required Fields	API Endpoint
Deletes (Deletes particular Subject) - DELETE	It's a delete request that deletes a subject's by their id.	http://localhost:3000/ subject/delete-subject/ subjectId



The delete Botton will delete a subject but if a teacher its assigned with this subject or a student is enrolled in it, it can't be deleted.

Teacher Related:

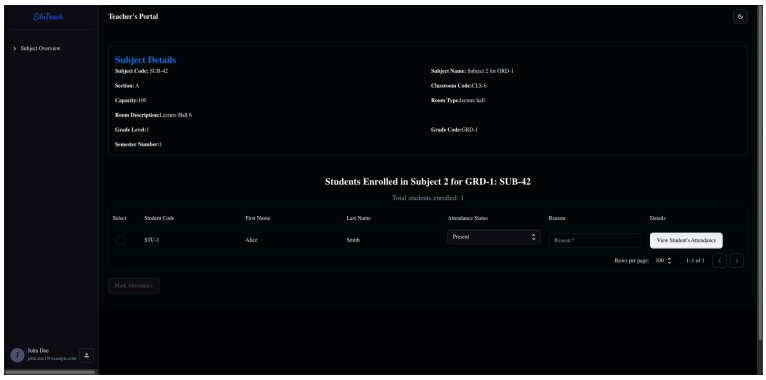
View All Subjects: This calls the API and the controller returns all the subjects the teacher is assigned with.

API Action used for the feature	Required Fields	API Endpoint
Reads (Gets all the subjects) - GET	It's a get request that returns all the subjects the teacher is assigned with.	http://localhost:3000/teacher/get-assigned-subjects/teacherId

This table shows all the subjects in the system, but you can also filter based on every column.

Mark student attendance: This calls the API and the controller lets the teacher mark the attendance for students.

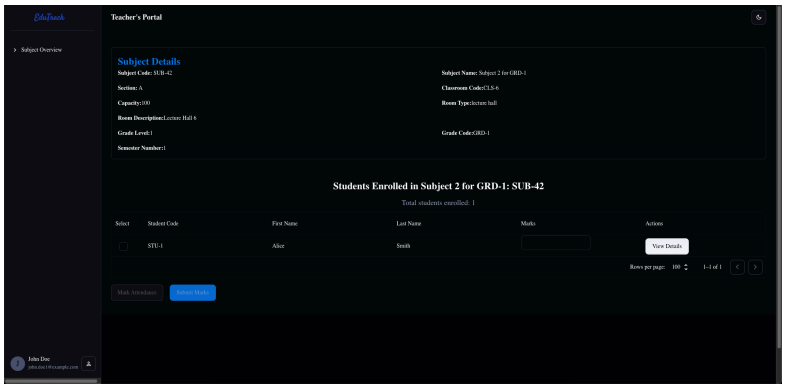
API Action used for the feature	Required Fields	API Endpoint
Creates (Creates attendance record for each student) - POST	It's a post request which creates an attendance record for each student for the day.	http://localhost:3000/attendance/create-attendance



Teacher can mark each student's attendance for the day. The table lets you mark all students' attendance at once using an array.

Mark students marks: This calls the API and the controller lets the teacher mark the marks for students.

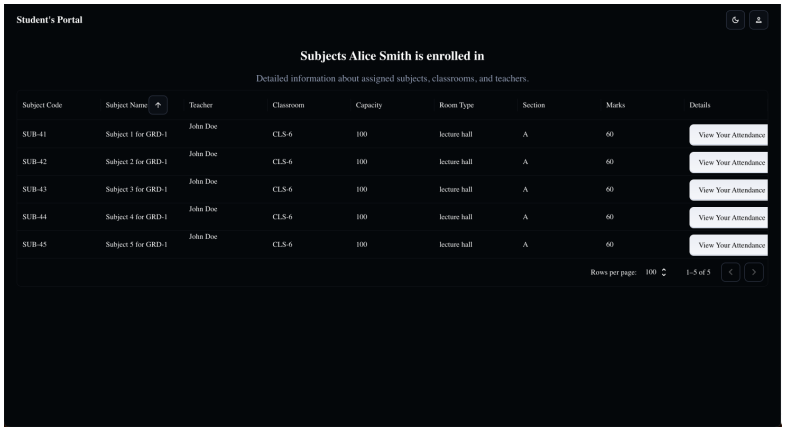
API Action used for the feature	Required Fields	API Endpoint
Creates (Creates marks record for each student) - POST	It's a post request which creates an marks record for each student for the day.	http://localhost:3000/mark/create-mark



Teacher can mark all the students for the subject at once. Keeping in mind that the system is designed to mark each subject only once each year.

Students view their subjects and marks table: This calls the API and the controller lets student view his subjects and marks.

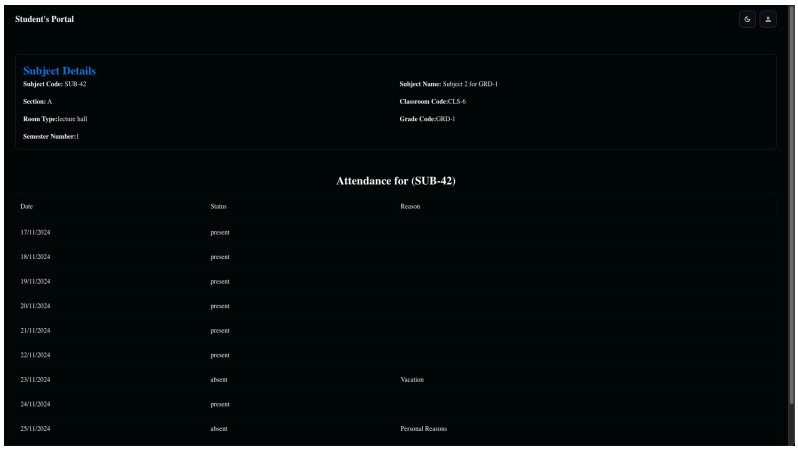
API Action used for the feature	Required Fields	API Endpoint
Reads (Gets all the subject for the student) - POST	It's a get request which gets all the subjects and marks and classes for the subjects	http://localhost:3000/subject/get-subjects-for-student/studentId



Students can view all their subjects and the teacher that teach those subjects and the marks and classes for the subjects.

Students their attendance for a subject: This calls the API and the controller lets student view his attendance for a subject

API Action used for the feature	Required Fields	API Endpoint
Reads (Gets all the attendance for the subject) - POST	It's a get request which gets all the attendance for a subject.	http://localhost:3000/attendance/get-student-attendance-grouped-by-subject/studentId



Student can view all their attendance for a subject.

Implementation

The Student Management System is not yet deployed, but there are plans to use AWS for its reliable and scalable hosting services. The following points outline the key considerations:

1. **Hosting the Application**
   The backend will run on AWS services designed to handle data requests and responses efficiently.
   The database will be hosted securely on AWS to ensure reliable access and backups.
   The frontend will be set up on AWS to make the website fast and easily accessible for users.
2. **Security**
   Secure connections will be enabled to protect user data and communications.
   Access to the application's resources will be carefully controlled to ensure safety.
3. **Performance and Growth**
   The hosting setup will be designed to adjust automatically to handle more users as needed.
   Tools will be in place to monitor how well the application is running and address any issues quickly.
4. **Testing Before Deployment**
   The application will be tested in a trial environment similar to the real one to ensure everything works as expected.
   All features, such as user interactions and data handling, will be verified thoroughly.
5. **Future Enhancements**
   Plans include adding an email notification system for updates and a feedback system to collect user opinions and improve the application.

These steps will help ensure the deployment on AWS is smooth and provides a reliable and user-friendly experience.

End-to-end solution

The Student Management System effectively met its primary objectives by providing a streamlined platform for managing student information, attendance, fee tracking, and academic performance. The software demonstrated robust functionality across both the backend and frontend, ensuring seamless data handling and an intuitive user interface.

Key Achievements

- **Core Functionalities Delivered:** All core features, such as student registration, attendance management, and grade tracking, were implemented successfully and rigorously tested for reliability.
- **Secure User Authentication:** With the integration of JWT-based authentication and secure password hashing, the system ensured data protection and user privacy.
- **Ease of Use:** The application achieved a user-friendly interface by employing a well-organized layout and familiar UI cues, making navigation intuitive for all users.
- **Scalable Design:** The modular architecture ensures scalability, allowing for future enhancements such as notifications or feedback systems to be integrated seamlessly.

Areas for Improvement

While the software met its fundamental goals, some planned features, like email notifications and feedback mechanisms, were not implemented. These are opportunities for future iterations to enhance the application's functionality further.
Overall, the software provided an end-to-end solution that aligned well with its objectives, meeting user needs and delivering a reliable system for managing student data efficiently.

References

Code Repository

The complete code for the Student Management System project is available on GitHub: https://github.com/azimy-hamid/miniProjectThree

Resources Used

The project utilized the following tools, libraries, and technologies:

Frontend:

- @emotion/react, @emotion/styled: For styling components.
- @mui/icons-material, @mui/material: For building responsive UI elements.
- @mui/x-charts, @mui/x-data-grid, @mui/x-date-pickers, @mui/x-tree-view: For data visualization and interactive features.
- axios: For making HTTP requests.
- dayjs: For handling dates and times.
- react, react-dom: Core libraries for building the UI.
- react-router-dom: For navigation and routing.

Backend:

- bcryptjs: For password hashing and security.
- body-parser: To parse incoming request bodies.
- cors: To enable cross-origin requests.
- dotenv: To manage environment variables.
- express: For building the server.
- express-session: For session management.
- jsonwebtoken: For secure user authentication using tokens.
- mysql2, sequelize: For database management and ORM.
- uuid: For generating unique identifiers.
- validator: For validating user input.

Database:

- MySQL: Used to store and manage student, course, and related data.
- Testing Tools:
- Postman: For API testing and debugging.
- Built-in JavaScript console.log and unit testing for backend CRUD operations.

Project Management:

Trello: To track and organize project tasks, divided into frontend, backend, testing, and general categories.

These resources were integral to developing a functional, reliable, and user-friendly application.