## Project-Based Evaluation

Project Report

Semester-V (Batch-2023)

**EDUNOVA - SCHOOL MANAGEMENT SYSTEM**

A red and white sign

Description automatically generated with low confidence

**Supervised By: Submitted By:**

Dr. Salil Adwita Jindal, 2310990268(G4)

Gunn, 2310990277(G4)

Aryan Bansal, 2310990303(G4)

Divya, 2310990328(G4)

**Department of Computer Science and Engineering**

**Chitkara University Institute of Engineering & Technology,**

**Chitkara University, Punjab**

**ABSTRACT**

The rapid digital transformation in the education sector has emphasized the need for efficient, user-friendly, and scalable management solutions that streamline the interaction between administrators, teachers, and students. The **School Management System** developed in this project is a web-based application designed using the **MERN stack with REACT.JS templating for the frontend**. Its objective is to simplify administrative workflows, enhance academic tracking, and promote effective communication across all stakeholders in a school environment.

The system is designed with **role-based access control**, offering distinct functionalities for three primary user groups: Admin, Teacher, and Student. Administrators are responsible for managing users, classes, subjects, and system configurations. Teachers utilize the system to record attendance, assign grades, and provide performance feedback, while students can monitor their academic progress, view attendance history, and communicate with teachers. The incorporation of **data visualization** ensures that students and teachers can interpret academic performance through interactive tables and charts, thereby enhancing learning outcomes.

The architecture of the system follows a **three-tier model** consisting of the Presentation Layer (REACT.JS-based templates for dynamic page rendering), Business Logic Layer (Node.js and Express.js), and Data Layer (MongoDB). This separation of concerns ensures maintainability, scalability, and security. Middleware and authentication mechanisms such as JWT are integrated to safeguard user privacy and enforce role-based restrictions.

From a development perspective, the project adopts an **Agile methodology**, ensuring iterative development and continuous stakeholder feedback. Each module is implemented incrementally, tested rigorously, and refined to meet real-world requirements. This methodology allows the system to evolve dynamically with changing institutional needs.

The significance of the project lies not only in its academic learning value but also in its practical relevance. Schools often struggle with paper-based processes, inefficient communication, and delayed performance tracking. By offering digital solutions such as online attendance, performance management, and real-time communication, this project bridges the gap between administrative efficiency and student engagement. Moreover, its modular architecture provides opportunities for future enhancements, such as mobile application integration, AI-based learning recommendations, and cloud deployment for scalability.

In conclusion, this School Management System represents a **comprehensive digital solution** that combines robust backend services, dynamic templating, and secure data handling to address challenges in modern education. Its implementation highlights the integration of **software engineering principles, database management, web technologies, and system design** into a unified platform. The project not only serves as a practical case study for computer science learners but also demonstrates the potential of web-based applications to transform academic institutions into digitally empowered ecosystems.

### TABLE OF CONTENTS

1. Introduction
   1. Background
   2. Objectives
   3. Significance
   4. Scope of Project
2. Problem Definition and Requirements
   1. Problem Statement
   2. Software Requirements
   3. Hardware Requirements
   4. Data Sets
3. Proposed Design
   1. System Architecture Overview
   2. Methodology
   3. Class and Functional Responsibility
   4. File Structure Description
   5. Data Flow and Processing Logic
4. Results
   1. Screenshots
5. References

**1. INTRODUCTION**

**1.1 Background**

Education is one of the most vital pillars of society, and schools are responsible for nurturing and guiding students toward knowledge, discipline, and skill development. In the modern era, managing a school has become increasingly complex due to the large number of students, teachers, and administrative tasks that must be handled simultaneously. Traditional management methods, which often rely on paperwork and manual processes, are no longer sufficient to meet the growing needs of schools.

The **School Management System (SMS)** has been developed to address these challenges by digitizing and automating key functions such as attendance management, student performance evaluation, communication, and administrative workflows. By providing a centralized platform, the system ensures that stakeholders—administrators, teachers, and students—can collaborate effectively and access real-time information whenever required.

This project uses the **MERN stack (MongoDB, Express.js, REACT.JS, Node.js)** to create a powerful web-based application. The MERN stack was chosen due to its flexibility, scalability, and ability to support large amounts of data while delivering excellent performance. With cloud integration and interactive features, the School Management System is not only a tool for record keeping but also a complete ecosystem that transforms how schools operate on a day-to-day basis.

For decades, schools have relied on registers, paper files, and manual processes to handle everything from attendance tracking to academic performance records. While such methods served their purpose in the past, they come with significant drawbacks:

* **Time Consumption:** Teachers and administrators spend hours on repetitive clerical tasks such as marking attendance, calculating grades, and preparing reports.
* **Error-Prone Processes:** Manual record-keeping is highly susceptible to mistakes, leading to data inconsistencies and loss of information.
* **Limited Accessibility:** Students and parents cannot easily access performance or attendance records unless they physically visit the school.
* **Inefficient Communication:** Messages between students, teachers, and administrators often get delayed, miscommunicated, or lost altogether.

With the rapid growth of technology and increasing expectations from educational institutions, there has been a **shift toward digital transformation in schools**. Many institutions now look for solutions that can **streamline operations, reduce human error, and foster collaboration**.

The **School Management System** aims to fulfill this exact requirement. By leveraging the **MERN stack**, the application benefits from:

* **MongoDB:** A flexible NoSQL database that can handle unstructured and semi-structured data, making it ideal for storing student profiles, attendance logs, and performance records.
* **Express.js and Node.js:** A reliable backend environment for building APIs, managing requests, and ensuring secure user authentication.
* **REACT.JS:** A frontend framework that allows for interactive dashboards, charts, and smooth user experiences.

Thus, the project is built upon modern technologies to solve age-old educational challenges.

**1.2 Objectives**

The objectives of the School Management System go beyond simply replacing manual processes with digital ones. The system has been designed with a **clear vision of improving efficiency, transparency, and collaboration** within the school environment.

The specific objectives include:

1. **Automation of Core Processes**  
   To eliminate repetitive manual work such as attendance marking, report generation, and class scheduling by providing automated workflows.
2. **Centralized Data Management**  
   To store and manage all student, teacher, and administrative records in one secure platform that can be accessed based on user roles.
3. **Improved Communication**  
   To provide an internal messaging system that enables seamless communication between students and teachers, reducing dependency on third-party apps or offline methods.
4. **Performance Monitoring and Feedback**  
   To allow teachers to record student marks, provide comments, and generate progress reports, while enabling students to view their academic history and identify areas for improvement.
5. **User-Friendly Interfaces**  
   To design dashboards tailored for each role (Admin, Teacher, Student), ensuring that users only see the tools and information relevant to their responsibilities.
6. **Data Visualization and Analytics**  
   To present performance metrics and attendance data through charts, tables, and graphs that make information easier to interpret.
7. **Scalability and Extensibility**  
   To develop a system that can grow with the school, supporting more students, teachers, and additional modules like fee management or library systems in the future.

**1.3 Significance**

The **School Management System (SMS)** project holds significance from both an **academic learning perspective** and a **real-world application perspective**. It not only demonstrates the practical benefits of combining MongoDB, Express.js, React.js, and Node.js but also addresses pressing challenges faced by modern educational institutions. Its relevance can be explained in the following dimensions:

**1. Real-World Applicability**

Educational institutions worldwide are increasingly adopting digital solutions to manage their operations. Traditional methods of paper registers, manual grading, and offline communication are inefficient, time-consuming, and error-prone.

This project introduces a **centralized, web-based system** that digitizes:

* **Attendance Management** – Teachers can track attendance seamlessly and generate instant reports.
* **Performance Tracking** – Students can monitor marks and feedback over time.
* **Communication** – Teachers and students can exchange messages without relying on third-party apps.
* **Administration** – Admins can manage user accounts, classes, and subjects efficiently.

By addressing these practical needs, the project demonstrates how technology can improve **efficiency, transparency, and collaboration** in schools.

**2. Academic Value**

From an academic standpoint, this project integrates diverse concepts from **computer science and software engineering**, making it an excellent case study:

* **Web Technologies:** React.js with Material UI for responsive and interactive UIs.
* **Backend Development:** Node.js and Express.js for building RESTful APIs and managing server logic.
* **Database Management Systems:** MongoDB as a NoSQL solution to handle flexible educational data structures.
* **State Management:** Redux for managing complex data flows across different user roles.
* **Software Engineering Principles:** Modularity, layered architecture, reusability, and separation of concerns.
* **Security:** Authentication and role-based authorization to ensure secure access for Admins, Teachers, and Students.

Thus, the project not only solves real-world challenges but also strengthens academic understanding by combining multiple subjects into a **full-stack development experience**.

**3. Skill Development**

The School Management System provides developers with hands-on experience in **industry-relevant technologies** that align closely with the job market:

* **Frontend Development:** Building dynamic, interactive dashboards using React.js + Material UI.
* **Backend APIs:** Developing scalable and secure APIs with Node.js + Express.js.
* **Database Handling:** Designing schemas, handling queries, and managing data with MongoDB.
* **Authentication & Authorization:** Implementing role-based security with JWT tokens.
* **State Management:** Using Redux to handle data consistency across multiple components.
* **Visualization Tools:** Building charts and dashboards to make raw data insightful.

These skills are highly sought after in the industry, making this project a **career-building milestone** for developers and students.

**4. Scalability and Adaptability**

The modular design of the system ensures that it can be easily **extended and adapted** to meet evolving needs. Possible future enhancements include:

* **Fee and Finance Management** for handling student payments.
* **Library Management System** for tracking issued and returned books.
* **Parent Portals** for real-time monitoring of student progress.
* **Mobile Applications** using React Native for cross-platform accessibility.
* **Cloud Integration** for hosting and scaling the application globally.
* **AI-driven Analytics** to identify struggling students and suggest personalized learning paths.

This flexibility makes the system suitable for **schools of all sizes**—from small institutions to large-scale educational organizations.

**5. Enhanced User Experience**

One of the project’s strengths lies in its **focus on user experience (UX)**. Each role (Admin, Teacher, Student) gets a customized dashboard that simplifies their daily tasks:

* **Admins** can manage accounts and settings with minimal effort.
* **Teachers** can track attendance and input grades quickly.
* **Students** can access their performance data in a visually appealing and easy-to-understand format.

The use of **Material UI components** ensures the interface is clean, modern, and responsive across devices. This focus on UI/UX increases adoption rates, builds trust, and enhances user satisfaction.

**6. Foundation for Advanced Features**

By building on a **solid MERN foundation**, this project lays the groundwork for advanced academic and professional research. Future possibilities include:

* **Microservices-based Architecture** for distributed deployment.
* **Integration with Learning Management Systems (LMS)** like Moodle or Google Classroom.
* **Data Analytics Dashboards** for school admins to analyze performance trends and resource utilization.
* **Multi-Language Support** for global accessibility.
* **AI-Powered Chatbots** for answering student queries.

These features ensure that the system is not only **relevant today** but also **future-ready** for the next wave of digital transformation in education.

**1.4. Scope of the Project**

* The **School Management System (SMS)** project is designed to provide a comprehensive digital platform for schools, teachers, and students by leveraging the **technologies (MongoDB, Express.js, REACT.JS, Node.js)**. The scope of the project extends across **functional requirements, technical features, and future enhancements**, ensuring a robust, scalable, and user-friendly system that addresses real-world challenges in education.
* **1. User Registration and Role Management**
* The system supports **role-based access** with distinct privileges for **Admins, Teachers, and Students**.
* **Authentication & Authorization:** Secure login and role management are implemented using JWT tokens to ensure that each user only accesses relevant features.
* **Admin Role:** Admins can create, update, or deactivate accounts for teachers and students.
* **Teacher Role:** Teachers can manage classes, attendance, and student performance.
* **Student Role:** Students can access their profiles, attendance, marks, and communication channels.
* **2. Student Features**
* **Personal Dashboard:** Students have personalized dashboards showing attendance, marks, subjects, and announcements.
* **Performance Tracking:** Students can view grades across subjects, track progress trends, and analyze weaknesses.
* **Attendance Records:** Students can see their attendance percentage in each subject and generate attendance summaries.
* **Communication:** Students can send queries and messages to teachers for academic support.
* **Data Visualization:** Graphs and charts provide an easy-to-understand visual representation of performance and attendance.
* **3. Teacher Features**
* **Class Management:** Teachers can create, assign, and update subjects or classes.
* **Attendance Tracking:** Teachers can mark students present/absent and generate reports for daily, weekly, or monthly attendance.
* **Performance Assessment:** Teachers can enter student marks, provide comments, and evaluate performance over time.
* **Feedback and Communication:** Teachers can send announcements, respond to student queries, and provide additional resources.
* **Report Generation:** Automated reports help teachers analyze class performance at a glance.
* **4. Admin Features**
* **User Management:** Admins can create, update, and manage accounts for teachers and students.
* **Class and Subject Management:** Admins define academic structures such as classes, subjects, and timetables.
* **System Monitoring:** Admins oversee overall system performance, user activity, and reports.
* **Configuration & Settings:** Admins can configure roles, permissions, and institutional policies in the system.
* **5. Backend Features (Node.js & Express.js)**
* **RESTful API Development:** Modular APIs to handle student records, teacher operations, and admin controls.
* **Data Validation & Error Handling:** Ensures reliability and accuracy in input data.
* **Role-Based Authorization:** Middleware ensures secure access for different roles.
* **Scalability:** APIs are structured for easy extension of features like fee management or library systems.
* **6. Database Features (MongoDB)**
* **Flexible Schema Design:** Supports dynamic structures for students, teachers, and classes.
* **Data Persistence:** Stores attendance logs, academic records, communication messages, and system settings.
* **High Performance:** Capable of handling large volumes of student and teacher data efficiently.
* **Backup & Recovery:** Database design supports future integration with cloud services for reliability.
* **7. Frontend Features (REACT.JS + Material UI)**
* **Responsive Dashboards:** Tailored dashboards for Admins, Teachers, and Students.
* **State Management with Redux:** Ensures smooth data flow across different components.
* **Interactive Visuals:** Graphs, charts, and tables provide insights into academic data.
* **Cross-Device Compatibility:** The application works seamlessly across desktops, tablets, and mobile devices.
* **Announcements & Notifications:** Real-time updates for students and teachers on upcoming classes or changes.
* **8. Non-Functional Scope**
* **Security:** JWT-based authentication, role-based access control, and data encryption ensure secure operations.
* **Scalability:** Modular architecture allows the system to support more users and features without redesign.
* **Maintainability:** Clean separation of frontend, backend, and database layers ensures ease of maintenance.
* **Performance:** Optimized queries and caching strategies ensure fast data retrieval.
* **Cross-Platform Availability:** While web-based initially, the system supports future mobile expansion using React Native.
* **9. Future Enhancements**
* **Parent Portal:** Allow parents to monitor student progress and attendance.
* **Fee and Finance Management:** Automating tuition fee tracking, online payments, and financial reports.
* **Library Management System:** Issuing, returning, and tracking books digitally.
* **Mobile App Integration:** Bringing the platform to Android/iOS using React Native.
* **AI-Powered Analytics:** Identifying student performance patterns and predicting areas needing intervention.
* **Cloud Hosting:** Deployment on AWS or Azure for global scalability.
* **Multi-Language Support:** Enabling the platform to be adopted by schools across different regions.

**2. PROBLEM DEFINITION AND REQUIREMENTS**

**2.1 Problem Statement**

Traditional school management processes rely heavily on manual record-keeping, paper-based documentation, and fragmented communication channels between administrators, teachers, and students. This creates inefficiencies, delays, and opportunities for human error. Administrators often struggle to manage student data, teacher information, class organization, and system-wide reporting. Teachers face challenges in maintaining accurate attendance records, assessing student performance consistently, and communicating effectively with students. Similarly, students lack a centralized platform to track their progress, receive feedback, and stay updated with academic activities.

As schools expand and the number of students increases, the volume of data grows exponentially, making manual methods impractical. Without an integrated system, there is no seamless way to manage user roles, generate performance analytics, or ensure transparent communication. Therefore, the problem is the absence of a **centralized, scalable, and digital platform** that streamlines administrative tasks, improves collaboration, and enhances the academic experience for all stakeholders.

The School Management System project aims to solve this by providing a **web-based, role-specific, and data-driven platform** that brings administrators, teachers, and students into a unified ecosystem

**2.2 Software Requirements**

**The software requirements specify the tools, libraries, and frameworks necessary to build and run the School Management System efficiently.**

**Frontend (Client-Side):**

* **REACT.JS–** Useful for server-side rendering**.**
* **Material UI (MUI) –** For a clean, modern, and consistent design.
* **Axios/Fetch API –** For communication between frontend and backend services**.**

**Backend (Server-Side):**

* **Node.js –** For a scalable and event-driven runtime environment**.**
* **Express.js –** For handling RESTful API development and routing**.**
* **JWT (JSON Web Tokens) –** For secure authentication and role-based authorization.
* **Bcrypt.js –** For secure password hashing.

**Database:**

* **MongoDB –** A NoSQL database for flexible data storage, supporting student records, attendance logs, and performance data.
* **Mongoose ORM –** For schema modeling and database interaction.

**Development & Build Tools:**

* **Visual Studio Code (VS Code) –** Primary IDE for development**.**
* **Node Package Manager (NPM) / Yarn –** For dependency management.
* **Git & GitHub –** For version control and collaboration**.**
* **Postman/Insomnia –** For API testing.

**Operating Systems Supported:**

* **Cross-platform compatibility:** Windows, macOS, Linux.
  1. **Hardware Requirements**

To ensure smooth development, deployment, and usage of the School Management System, the following hardware requirements are suggested:

**Minimum Hardware (Development Environment):**

* Processor: Dual-core processor (Intel i3 or equivalent).
* RAM: 4 GB.
* Storage: 500 GB HDD or 128 GB SSD.
* Display: 1366 × 768 resolution.
* Network: Stable internet connection for package installations and database connections.

**Recommended Hardware (Development/Production):**

* Processor: Quad-core processor (Intel i5/i7 or equivalent AMD).
* RAM: 8–16 GB (for smoother multitasking and faster build processes).
* Storage: 256 GB SSD or higher for faster I/O operations.
* Display: Full HD (1920 × 1080) or higher resolution.
* Network: High-speed internet with a minimum of 10 Mbps upload/download.

**Server Deployment Requirements (for production):**

* Cloud Hosting: AWS, Azure, or DigitalOcean.
* CPU: Minimum 2 vCPUs.
* RAM: 4–8 GB (scalable depending on traffic).
* Storage: 50–100 GB SSD.
* Backup: Regular automated backups for database safety.

### Data Sets

The School Management System will handle diverse types of structured and semi-structured datasets, each corresponding to different system modules.

**1. User Data:**

* Student Profiles: Name, roll number, class, section, contact details, guardian information.
* Teacher Profiles: Name, employee ID, subjects taught, qualifications, contact details.
* Admin Accounts: System-level privileges, school-wide management authority.

**2. Academic Data:**

* Class and Subject Details: Class names, subject codes, teacher allocations.
* Timetables: Class schedules, teacher availability, exam schedules.

**3. Attendance Data:**

* Daily attendance logs with timestamps.
* Records of absentees and attendance percentages.
* Attendance reports for class-wise or student-wise summaries.

**4. Performance and Assessment Data:**

* Marks scored in assignments, quizzes, midterms, and finals.
* Teacher feedback on student performance.
* Reports showing overall progress, strengths, and weaknesses.

**5. Communication Data:**

* Messages exchanged between students and teachers.
* Announcements from administrators.
* Notifications for assignments, exams, and system updates.

**6. Visualization Data (Derived):**

* Charts representing student performance trends.
* Attendance graphs for monitoring consistency.
* Tabular reports for administrators to analyze academic health.

By organizing and managing these datasets, the School Management System ensures accuracy, accessibility, and usability, making it easier for stakeholders to make informed decisions.

**3. PROPOSED DESIGN**

**3.1 System Architecture Overview**

The **School Management System** follows a **three-tier architecture**, consisting of the **Presentation Layer (Frontend with REACT.JS templates), Business Logic Layer (Node.js + Express.js), and Data Layer (MongoDB)**. This layered approach ensures modularity, maintainability, and scalability.

* **Presentation Layer (REACT.JS Templates):**  
  The frontend uses Embedded JavaScript (REACT.JS) templating to dynamically generate HTML pages. This enables server-side rendering (SSR), where user-specific data such as attendance records, student marks, or announcements are injected into pre-defined templates before being sent to the browser. This makes the application lightweight, fast, and SEO-friendly compared to client-heavy frameworks.
* **Business Logic Layer (Express.js):**  
  The backend, built with Express.js, serves as the core of the system. It handles HTTP requests, validates input, applies authentication and authorization logic, processes data, and communicates with the database. Role-based access control ensures that Admins, Teachers, and Students only see or modify data relevant to their privileges.
* **Data Layer (MongoDB):**  
  MongoDB acts as the primary storage for structured and semi-structured data. Collections include students, teachers, admins, attendance records, grades, messages, and system configurations. The schema-less design of MongoDB makes it adaptable to evolving requirements, such as adding new fields or extending functionalities.
* **Security and Middleware:**  
  Middleware components (e.g., JWT authentication, input sanitization, logging) are embedded within Express.js to ensure data privacy and security. Sensitive information such as passwords is hashed using Bcrypt before being stored in MongoDB.

In summary, the architecture ensures that the application is **modular, role-specific, secure, and scalable**, suitable for deployment on both on-premise servers and cloud environments.

### 3.2 Methodology

The development of the School Management System follows the **Agile Software Development Methodology**, specifically an **iterative and incremental model**. This methodology was chosen because of its adaptability to changing requirements and focus on delivering working software in short cycles.

**Steps in the methodology:**

1. **Requirement Gathering:**  
   Initial discussions and documentation defined user roles (Admin, Teacher, Student), core functionalities (attendance, performance tracking, communication), and system constraints.
2. **System Design:**  
   UML diagrams, data flow diagrams (DFDs), and entity-relationship diagrams (ERDs) were prepared to visualize data storage and system flow. The REACT.JS-based template structure was outlined here.
3. **Incremental Development:**
   * First Iteration: Implemented Admin features (user management, class creation).
   * Second Iteration: Teacher features (attendance tracking, performance assessment).
   * Third Iteration: Student features (view attendance, marks, communication).
4. **Testing and Debugging:**  
   Each module was tested independently using unit testing (Jest/Mocha for backend, manual testing for REACT.JS views). Integration testing ensured data consistency across modules.
5. **Deployment and Feedback:**  
   The system is deployed on a Node.js runtime. Feedback loops from stakeholders help refine the system iteratively.

This methodology ensures that the project remains aligned with stakeholder expectations and can quickly adapt to new requirements.

**3.3 Classes and Functional Responsibilities**

Since this is a **Node.js + Express.js + REACT.JS project**, “classes” can be interpreted as **modules or models**. Responsibilities are divided into **MVC (Model-View-Controller)**:

**Models (Data Layer):**

* studentSchema.js: Stores student details, class enrollment, attendance logs, grades.
* teacherSchema.js: Contains teacher profiles, assigned classes, subjects.
* adminSchema.js: Manages system-wide settings, user roles, and data integrity.
* sclassSchema.js: Stores class-related information.
* subjectSchema.js: Contains subject details.
* complainSchema.js: Handles student complaints.
* noticeSchema.js: Stores notices for students, teachers, and classes.

**Views (Presentation Layer with REACT.JS):**

* AdminDashboard.React.js, StudentDashboard.React.js, TeacherDashboard.React.js: Separate dashboards for Admin, Teacher, and Student.
* StudentAttendance.React.js (or attendance-related views in student/admin pages): Displays attendance forms and logs.
* StudentExamMarks.React.js (or marks-related views in student/admin pages): Shows marks, progress charts, and teacher feedback.
* SeeNotice.React.js (or messages/notice views in pages/components): Messaging and notice interface for teachers and students.
* AdminRegisterPage.React.js, AdminProfile.React.js, AdminHomePage.React.js, SideBar.React.js: Admin-level system management interface.

**Controllers (Business Logic Layer):**

* admin-controller.js: Manages user accounts, classes, subjects.
* teacher-controller.js: Handles attendance, grading, communication.
* student\_controller.js: Provides student data views (marks, attendance).
* complain-controller.js: Handles complaints from students.
* notice-controller.js: Manages notices and announcements.

This separation of concerns ensures maintainability and clear division of functional responsibilities.

**3.4 File Structure Description**

|-- backend

| |-- controllers

| | |-- admin-controller.js

| | |-- class-controller.js

| | |-- complain-controller.js

| | |-- notice-controller.js

| | |-- student\_controller.js

| | |-- subject-controller.js

| | |-- teacher-controller.js

| |-- models

| | |-- adminSchema.js

| | |-- complainSchema.js

| | |-- noticeSchema.js

| | |-- sclassSchema.js

| | |-- studentSchema.js

| | |-- subjectSchema.js

| | |-- teacherSchema.js

| |-- routes

| | |-- route.js

| |-- .env

| |-- .gitignore

| |-- index.js

| |-- package-lock.json

| |-- package.json

|-- frontend

| |-- public

| | |-- favicon.ico

| | |-- index.html

| |-- src

| | |-- assets

| | | |-- assignment.svg

| | | |-- backg.jpg

| | | |-- classroom.png

| | | |-- designlogin.jpg

| | | |-- img1.png

| | | |-- img2.png

| | | |-- img3.png

| | | |-- img4.png

| | | |-- students.svg

| | | |-- subjects.svg

| | | |-- time.svg

| | |-- components

| | | |-- AccountMenu.js

| | | |-- attendanceCalculator.js

| | | |-- buttonStyles.js

| | | |-- CustomBarChart.js

| | | |-- CustomPieChart.js

| | | |-- ErrorPage.js

| | | |-- mobileChecker.js

| | | |-- Popup.js

| | | |-- SeeNotice.js

| | | |-- SpeedDialTemplate.js

| | | |-- styles.js

| | | |-- TableTemplate.js

| | | |-- TableViewTemplate.js

| | |-- pages

| | | |-- admin

| | | | |-- classRelated

| | | | | |-- AddClass.js

| | | | | |-- ClassDetails.js

| | | | | |-- ShowClasses.js

| | | | |-- noticeRelated

| | | | | |-- AddNotice.js

| | | | | |-- ShowNotices.js

| | | | |-- studentRelated

| | | | | |-- AddStudent.js

| | | | | |-- SeeComplains.js

| | | | | |-- ShowStudents.js

| | | | | |-- StudentAttendance.js

| | | | | |-- StudentExamMarks.js

| | | | | |-- ViewStudent.js

| | | | |-- subjectRelated

| | | | | |-- ShowSubjects.js

| | | | | |-- SubjectForm.js

| | | | | |-- ViewSubject.js

| | | | |-- teacherRelated

| | | | | |-- AddTeacher.js

| | | | | |-- ChooseClass.js

| | | | | |-- ChooseSubject.js

| | | | | |-- ShowTeachers.js

| | | | | |-- TeacherDetails.js

| | | | |-- AdminDashboard.js

| | | | |-- AdminHomePage.js

| | | | |-- AdminProfile.js

| | | | |-- AdminRegisterPage.js

| | | | |-- SideBar.js

| | | |-- student

| | | | |-- StudentDashboard.js

| | | | |-- StudentHomePage.js

| | | | |-- StudentSideBar.js

| | | |-- teacher

| | | | |-- TeacherDashboard.js

| | | | |-- TeacherHomePage.js

| | | | |-- TeacherSideBar.js

| | | |-- ChooseUser.js

| | | |-- Homepage.js

| | | |-- LoginPage.js

| | | |-- Logout.js

| | |-- redux

| | | |-- complainRelated

| | | | |-- complainHandle.js

| | | | |-- complainSlice.js

| | | |-- noticeRelated

| | | | |-- noticeHandle.js

| | | | |-- noticeSlice.js

| | | |-- sclassRelated

| | | | |-- sclassHandle.js

| | | | |-- sclassSlice.js

| | | |-- studentRelated

| | | | |-- studentHandle.js

| | | | |-- studentSlice.js

| | | |-- teacherRelated

| | | | |-- teacherHandle.js

| | | | |-- teacherSlice.js

| | | |-- userRelated

| | | | |-- userHandle.js

| | | | |-- userSlice.js

| | | |-- store.js

| | |-- App.js

| | |-- index.css

| | |-- index.js

| |-- .env

| |-- .gitignore

| |-- netlify.toml

| |-- package-lock.json

| |-- package.json

| |-- README.md

|-- README.md

**3.5 Data Flow and Processing Logic**

The data flow in the School Management System can be explained in stages:

1. **User Interaction:**  
   A user (Admin, Teacher, or Student) interacts with the system through an REACT.JS-rendered interface.
2. **Request Handling:**
   * The user submits a request (e.g., login, mark attendance).
   * Express.js routes map this request to the appropriate controller.
3. **Business Logic Execution:**
   * The controller applies logic, verifies user roles, and interacts with models.
   * Middleware ensures that only authenticated and authorized users proceed.
4. **Database Interaction (MongoDB):**
   * Controllers query or update MongoDB collections using Mongoose.
   * For example, marking attendance updates the Attendance collection with date, student ID, and status.
5. **Response Generation:**
   * The retrieved or updated data is passed into an REACT.JS template.
   * The REACT.JS engine injects the data dynamically into HTML and sends the final page back to the client.
6. **Visualization and Feedback:**
   * Students may see attendance graphs, teachers get performance tables, and admins access reports.
   * Communication messages are immediately rendered and stored for persistence.

This **request–process–response cycle** ensures smooth user experience, data integrity, and role-specific access.

**4. RESULTS**

**4.1 Screenshots**

### 5. REFERENCES

1. Bass, L., Clements, P., & Kazman, R. (2013). *Software Architecture in Practice (3rd Edition)*. Addison-Wesley.
2. Sommerville, I. (2016). *Software Engineering (10th Edition)*. Pearson.
3. Pressman, R. S., & Maxim, B. R. (2020). *Software Engineering: A Practitioner’s Approach (9th Edition)*. McGraw-Hill.
4. MongoDB, Inc. (2024). *MongoDB Documentation*. Retrieved from <https://www.mongodb.com/docs/>
5. Node.js Foundation. (2024). *Node.js Documentation*. Retrieved from [https://nodReact.js.org/en/docs/](https://nodejs.org/en/docs/)
6. Express.js Team. (2024). *Express.js Guide*. Retrieved from <https://expressjs.com/>
7. REACT.JS Templating. (2024). *REACT.JS Official Documentation*. Retrieved from [https://React.js.co/](https://ejs.co/)
8. Redux Toolkit. (2024). *Redux Documentation*. Retrieved from <https://redux.js.org/>
9. Wazid, M., Das, A. K., Shetty, S., & Conti, M. (2019). “A Tutorial and Future Research for Secure Internet of Things: Authentication, Authorization, and Data Protection.” *IEEE Communications Surveys & Tutorials, 21*(4), 3449–3501.
10. Kumar, R., & Singh, P. (2021). “Design and Implementation of a Web-Based Student Management System.” *International Journal of Computer Applications, 183*(24), 12–18.
11. TutorialsPoint. (2024). *MERN Stack Tutorial*. Retrieved from https://www.tutorialspoint.com/mern\_stack/index.htm