School Management System Requirements

1. Introduction

1.1 Purpose

The purpose of this document is to outline the requirements for developing a School Management System (SMS) using the MERN stack, incorporating React or Next.js for the frontend, NestJS for the backend, PostgreSQL as the database, and ensuring the application is containerized. This system aims to streamline administrative tasks by providing essential functionalities for managing student and teacher information, classes, attendance, grades, fees, and basic reporting.

1.2 Scope

The SMS will be a web-based application accessible to administrators, teachers, accountants, and optionally, students and parents. It focuses on core functionalities to deliver immediate value with the possibility of future enhancements.

1.3 Definitions, Acronyms, and Abbreviations

- **SMS**: School Management System
- **MVP**: Minimum Viable Product
- **Admin**: Administrator
- **UI**: User Interface
- **MERN**: MongoDB, Express.js, React.js, Node.js
- **Next.js**: A React framework for server-side rendering
- **NestJS**: A progressive Node.js framework for building efficient and scalable server-side applications
- **DB**: Database
- **Containerization**: Packaging software code with all its dependencies so it can run uniformly and consistently on any infrastructure (e.g., Docker)

2. Overall Description

2.1 Product Perspective

The SMS is an independent system designed to replace manual or fragmented digital systems used in schools. It centralizes data management and provides a user-friendly interface for different user roles, enhancing efficiency and communication within the school.

2.2 Product Functions

- User authentication and role-based access control
- Student and teacher information management
- Class and subject management
- Attendance tracking and reporting
- Grade entry and report generation
- Fee management (including fee structure, collection, and reporting)
- Basic reporting functionalities

2.3 User Classes and Characteristics

- **Administrators**: Manage the entire system, including user roles, classes, subjects, and fee structures.
- **Teachers**: Record attendance, enter grades, and view student information.
- **Accountants**: Handle all aspects of fee management.
- **Students/Parents** (Optional): View attendance records, grades, and fee payment status.

2.4 Operating Environment

- **Client Side**: Modern web browsers (Chrome, Firefox, Edge, Safari)
- **Server Side**: Node.js runtime environment
- **Devices**: Desktops, laptops, tablets, and smartphones
- **Containerization**: Docker or similar containerization platform

2.5 Constraints

- **Security**: Must comply with data protection regulations.
- **Scalability**: Should handle increasing numbers of users and data volumes.
- **Usability**: Interface must be intuitive for users with basic computer skills.
- **Technology Stack**: Must use MERN stack with React or Next.js, NestJS, PostgreSQL, and be containerized.
- **Deployment**: Should be deployable in a containerized environment for consistency across different infrastructures.

2.6 Assumptions and Dependencies

- Users have access to the internet.
- The school provides necessary hardware infrastructure.
- Future modules can be integrated with minimal changes to the core system.
- Development team is proficient in JavaScript, Node.js, and associated frameworks.

3. Specific Requirements

3.1 Functional Requirements

3.1.1 User Authentication and Authorization

- **FR1**: The system shall provide a secure login mechanism for all users using JWT (JSON Web Tokens) for session management.
- **FR2**: The system shall support role-based access control, assigning permissions based on user roles (Admin, Teacher, Accountant, Student, Parent).
- **FR3**: The system shall allow administrators to manage user accounts and roles through an admin dashboard.

3.1.2 Student Information Management

- **FR4**: The system shall allow admins to add, edit, and delete student profiles containing personal and contact information.
- **FR5**: The system shall enable batch import of student data from CSV or Excel files.
- **FR6**: The system shall maintain enrollment status for each student.

3.1.3 Teacher Information Management

- **FR7**: The system shall allow admins to add, edit, and delete teacher profiles.
- **FR8**: The system shall assign teachers to specific classes and subjects.

3.1.4 Class and Subject Management

- **FR9**: The system shall allow admins to create and manage classes/grades.
- **FR10**: The system shall allow admins to define subjects and assign them to classes.
- **FR11**: The system shall enable the assignment of teachers to subjects within classes.

3.1.5 Attendance Management

- **FR12**: The system shall allow teachers to record daily attendance for each class.
- **FR13**: The system shall allow viewing and editing of attendance records by authorized users.
- **FR14**: The system shall generate attendance reports for individual students and classes.

3.1.6 Grade Management

- **FR15**: The system shall allow teachers to enter and update grades for assignments, tests, and exams.
- **FR16**: The system shall calculate overall grades based on inputted scores.
- **FR17**: The system shall generate report cards for students in PDF format.

3.1.7 Fee Management

- **FR18**: The system shall allow admins/accountants to define fee structures, including different fee categories (tuition, transportation, etc.).
- **FR19**: The system shall assign fee structures to classes or individual students as needed.

- **FR20**: The system shall record fee payments made by students, supporting multiple payment methods.
- **FR21**: The system shall generate receipts for each payment transaction.
- **FR22**: The system shall track outstanding fees and notify relevant users.
- **FR23**: The system shall generate reports on fee collection and outstanding balances.

3.1.8 Basic Reporting

- **FR24**: The system shall provide student performance reports combining grades and attendance.
- **FR25**: The system shall generate class performance summaries.
- **FR26**: The system shall produce financial reports related to fee management.

3.1.9 Communication Module (Optional)

- **FR27**: The system shall send notifications to parents/students about important updates (e.g., fee due dates, absenteeism, academic performance) via email or SMS.

3.1.10 Timetable Management (Optional)

- **FR28**: The system shall allow admins to create and manage class timetables.
- **FR29**: The system shall display timetables to teachers and students.

3.2 Non-Functional Requirements

3.2.1 Performance Requirements

- **NFR1**: The system shall load the dashboard within 3 seconds under normal network conditions.
- **NFR2**: The system shall handle up to 500 concurrent users without performance degradation.

3.2.2 Security Requirements

- **NFR3**: The system shall enforce strong password policies (minimum length, complexity).
- **NFR4**: The system shall use HTTPS for all data transmission.
- **NFR5**: Sensitive data, such as passwords and personal information, shall be encrypted in storage.
- **NFR6**: Implement OAuth 2.0 or similar authentication protocols for secure access.

3.2.3 Usability Requirements

- **NFR7**: The system's UI shall be intuitive and require minimal training for users.
- **NFR8**: The system shall provide tooltips and help sections for guidance.
- **NFR9**: The frontend shall be responsive and compatible with various screen sizes.

3.2.4 Reliability Requirements

- **NFR10**: The system shall have an uptime of 99% during operational hours.
- **NFR11**: The system shall automatically back up data daily.

3.2.5 Maintainability Requirements

- **NFR12**: The system shall be built using modular architecture to facilitate easy updates and maintenance.
- **NFR13**: The codebase shall be documented and follow standard coding conventions.
- **NFR14**: Utilize containerization (e.g., Docker) to ensure consistent environments across development, testing, and production.

3.2.6 Scalability Requirements

- **NFR15**: The system shall be scalable horizontally to accommodate increased load.
- **NFR16**: The database design shall support efficient querying and indexing for large datasets.

3.2.7 Deployment Requirements

- **NFR17**: The application shall be containerized using Docker or similar technology.
- **NFR18**: The system shall support deployment using container orchestration tools like Kubernetes (optional for initial deployment).

4. Recommended Tech Stack

4.1 Overview

The chosen tech stack for the SMS includes modern, widely-supported technologies that offer scalability, maintainability, and performance.

4.2 Frontend

4.2.1 React.js or Next.js

- **React.js**: A popular JavaScript library for building user interfaces.
- **Next.js**: A React framework offering server-side rendering and static site generation.

Why React.js/Next.js?

- **Performance**: Efficient rendering and fast user interactions.
- **Developer Experience**: Large ecosystem and community support.
- **SEO Benefits**: Next.js provides server-side rendering, enhancing SEO and performance.

Frontend Tools and Libraries

- **State Management**: Redux or Context API
- **UI Components**: Material-UI, Ant Design, or Bootstrap
- **Form Handling**: Formik or React Hook Form
- **Routing**: React Router or Next.js built-in routing

4.3 Backend

4.3.1 NestJS

- A progressive Node.js framework built with TypeScript, inspired by Angular architecture.

Why NestJS?

- **Modular Architecture**: Facilitates maintainability and scalability.
- **TypeScript Support**: Provides type safety and better code organization.
- **Performance**: Built on top of Express.js or Fastify for efficient request handling.
- **Out-of-the-box Features**: Supports middleware, authentication, and WebSocket integration.

Backend Tools and Libraries

- **ORM**: TypeORM or Prisma for database interactions
- **Authentication**: Passport.js integration with JWT strategy
- **Validation**: Class-validator and class-transformer for input validation

4.4 Database

4.4.1 PostgreSQL

- An open-source relational database known for its robustness and performance.

Why PostgreSQL?

- **Reliability**: ACID-compliant transactions ensure data integrity.
- **Features**: Supports advanced data types and indexing.
- **Scalability**: Handles complex queries and large datasets efficiently.

4.5 Containerization

4.5.1 Docker

- A platform for developing, shipping, and running applications in containers.

Why Docker?

- **Consistency**: Ensures the application runs the same in all environments.
- **Isolation**: Containers are isolated, reducing conflicts between services.
- **Portability**: Easier deployment across different platforms and cloud providers.

Containerization Tools

- **Docker Compose**: For defining and running multi-container Docker applications.
- **Docker Registry**: For storing and distributing Docker images.

4.6 Additional Tools

- **Version Control**: Git with repositories hosted on GitHub or GitLab.
- **CI/CD Pipeline**: Jenkins, GitHub Actions, or GitLab CI for automated testing and deployment.
- Testing Frameworks:
    - **Frontend**: Jest, React Testing Library
    - **Backend**: Jest, SuperTest
- **Logging and Monitoring**: Winston or Bunyan for logging; integrate with monitoring tools like Prometheus or Grafana.
- **Documentation**: Swagger/OpenAPI for API documentation.

5. Additional Considerations

5.1 Security Practices

- **Input Validation**: Use libraries like Joi or class-validator to prevent SQL injection and XSS attacks.
- **Authentication and Authorization**: Implement robust authentication using Passport.js with JWT and role-based access control.
- **Encryption**: Store sensitive data securely using hashing algorithms like bcrypt for passwords.
- **Environment Variables**: Manage secrets and configuration using environment variables, secured in containers.

5.2 API Design

- **RESTful API**: Design clear and consistent API endpoints following REST principles.
- **Error Handling**: Implement standardized error responses with appropriate HTTP status codes.
- **Pagination and Filtering**: For endpoints returning large datasets, support pagination, sorting, and filtering.

5.3 Performance Optimization

- **Caching**: Use Redis or in-memory caching strategies for frequently accessed data.
- **Asset Optimization**: Minify and bundle frontend assets; use code splitting in React.
- **Database Optimization**: Create indexes on frequently queried fields.

5.4 Scalability and Deployment

- **Container Orchestration**: Plan for future scalability using Kubernetes or Docker Swarm.
- **Cloud Deployment**: Consider deploying on cloud platforms like AWS, Azure, or Google Cloud Platform, utilizing their managed database and container services.
- **Load Balancing**: Use Nginx or HAProxy for distributing incoming traffic.

6. Development Plan

6.1 Phase 1: Setup and Core Development

- **Task 1**: Set up version control repositories and initial project scaffolding.
- **Task 2**: Configure Docker files and Docker Compose for development environment.
- **Task 3**: Implement user authentication and authorization modules.
- **Task 4**: Develop APIs for student and teacher management.
- **Task 5**: Create frontend components for user management interfaces.

6.2 Phase 2: Feature Implementation

- **Task 6**: Develop class and subject management modules.
- **Task 7**: Implement attendance and grade management functionalities.
- **Task 8**: Integrate fee management module on both backend and frontend.
- **Task 9**: Set up PostgreSQL database schemas and migrations.

6.3 Phase 3: Testing and Quality Assurance

- **Task 10**: Write unit and integration tests for backend APIs using Jest.
- **Task 11**: Implement frontend testing with Jest and React Testing Library.
- **Task 12**: Perform end-to-end testing using tools like Cypress.
- **Task 13**: Conduct security audits and performance testing.

6.4 Phase 4: Deployment and Release

- **Task 14**: Set up CI/CD pipelines for automated testing and deployment.
- **Task 15**: Deploy application containers to a staging environment.
- **Task 16**: Perform user acceptance testing and gather feedback.
- **Task 17**: Deploy to production environment upon approval.

6.5 Phase 5: Maintenance and Future Enhancements

- **Task 18**: Monitor system performance and user feedback.
- **Task 19**: Plan for the implementation of optional modules like communication tools and timetable management.
- **Task 20**: Schedule regular updates and security patches.

7. Conclusion

This document outlines the requirements for developing a School Management System using the MERN stack with React or Next.js, NestJS, PostgreSQL, and containerization. By focusing on essential features and leveraging modern technologies, the system aims to deliver immediate value while being scalable and maintainable for future enhancements.

Next Steps

- **Requirement Validation**: Review and confirm that all necessary requirements are captured.
- **Resource Allocation**: Assemble a development team proficient in the chosen technologies.
- **Project Planning**: Develop a detailed project plan with timelines and milestones.
- **Development Kickoff**: Begin the development process following the outlined phases.