**Software Requirements Specification (SRS)**

**1. Introduction**

**1.1 Purpose** The purpose of this document is to outline the software requirements for the School Management System (SMS). The system is designed to automate and streamline various administrative and academic tasks within a school, including class scheduling, attendance tracking, and academic performance monitoring. It aims to improve efficiency, reduce manual workload, and enhance collaboration between teachers, students, and parents.

**1.2 Document Conventions** This document follows the IEEE standard for Software Requirements Specifications. Requirements are stated in a clear and structured manner, using numbered lists for easy reference. Key terms and definitions are highlighted in bold. Functional and non-functional requirements are categorized for clarity.

**1.3 Intended Audience and Reading Suggestions** This document is intended for the development team, project managers, and stakeholders, including school administrators and teachers who will interact with the system. Developers should focus on the functional and non-functional requirements, while stakeholders should review the system’s capabilities and constraints. Testers should refer to this document to ensure that the system meets all specified requirements.

**1.4 References**

* IEEE 830-1998 - IEEE Recommended Practice for Software Requirements Specifications
* GDPR (General Data Protection Regulation) Compliance Guidelines
* FERPA (Family Educational Rights and Privacy Act) Compliance Standards
* PostgreSQL Documentation
* Java Spring Boot Official Documentation

**2. Overall Description**

The School Management Application is a comprehensive web-based platform designed to automate and streamline the management of school operations. It provides a centralized system for administrators, teachers, students, and parents to interact, manage academic activities, and access relevant information. The application focuses on 10 core entities: User, Role, Grade, Student, Parent, Employee, Schedule, Lesson, Mark, and Subject. These entities form the foundation of the system, enabling efficient management of users, academic records, schedules, and communication.

**2.1 Product Perspective** The School Management Application is an independent system that integrates various functionalities to support the daily operations of a school. It is designed to replace manual processes with automated solutions, reducing administrative workload and improving efficiency. The system will be accessible via a web interface, ensuring ease of use for all stakeholders.

**2.2 Product Features**

The application provides the following key features

* ***User Management:*** Create, update, and delete users with specific roles (Admin, Teacher, Student, Parent).
* ***Role-Based Access*** Control: Define permissions for each role to ensure secure access to system features.
* ***Grade Management:*** Manage classes or grade levels in the school.
* ***Student Management:*** Maintain student records, including enrollment, grades, and attendance.
* ***Parent Management:*** Link parents to their children and provide access to academic progress.
* ***Employee Management:*** Manage school staff, including teachers and administrators.
* ***Schedule Management:*** Create and manage timetables for classes and lessons.
* ***Lesson Management:*** Define lessons, assign teachers, and link them to subjects.
* ***Mark Management:*** Record and update student grades for lessons and subjects.
* ***Subject Management:*** Define and manage subjects taught in the school.

**2. 3. User Classes and Characteristics**

The system caters to the following user classes:

1. ***Administrator:***

* Manages users, roles, and system settings.
* Oversees the overall functioning of the system.

1. ***Teacher:***

* Manages lessons, schedules, and student grades.
* Communicates with students and parents.

1. ***Student:***

* Views schedules, grades, and lesson materials.
* Interacts with teachers and parents.

1. ***Parent:***

* Monitors their child's academic progress.
* Communicates with teachers and administrators.

**2.4 Operating Environment**

* Web-based application accessible via desktop and mobile browsers.
* Hosted on a cloud server or school intranet.
* **Backend:** Java Spring Boot
* **Main Database:** H2
* **Alternative Database:** PostgreSQL

**2.5 Design and Implementation Constraints**

* The system must support multiple user roles with appropriate access control.
* The database should maintain integrity and security.
* REST API will be used for communication between the backend components.
* The application should be scalable to support up to 2000 concurrent users.
* Compliance with data privacy laws (e.g., GDPR, FERPA).

**2.6 Assumptions and Dependencies**

* The system assumes reliable internet access for cloud-based hosting.
* Users will have access to modern web browsers (Chrome, Firefox, Edge, etc.).
* Email and SMS providers will be used for authentication and notifications.
* Future integration with third-party learning management systems is possible.

**3. Functional Requirements**

***3.1 User Management***

* **3.1.1 The system shall allow administrators to create, update, and delete users.**
* **3.1.2 Each user shall have a unique username and password.**
* **3.1.3 Users shall be assigned roles (e.g., Admin, Teacher, Student, Parent).**

***3.2 Role Management***

* **3.2.1 The system shall define roles with specific permissions (e.g., Admin, Teacher, Student, Parent).**
* **3.2.2 Roles shall determine access to system features.**

***3.3 Subject Management***

* **3.3.1 The system shall allow administrators to create, update, and delete subjects.**
* **3.3.2 Each subject shall have a unique identifier and name (e.g., Math, Science).**

***3.4 Employee Management***

* **3.4.1 The system shall allow administrators to add, update, and delete employee records.**
* **3.4.2 Employees shall be assigned roles (e.g., Teacher, Admin).**

***3.5 Grade Management***

* **3.5.1 The system shall allow administrators to create, update, and delete grades.**
* **3.5.2 Each grade shall have a unique identifier and name (e.g., Grade 10, Grade 11).**

***3.6 Parent Management***

* **3.6.1 The system shall allow administrators to add, update, and delete parent records.**
* **3.6.2 Each parent shall be associated with one or more students.**

***3.7 Student Management***

* **3.7.1 The system shall allow administrators to add, update, and delete student records.**
* **3.7.2 Each student shall be associated with a grade and a parent.**

***3.8 Schedule Management***

* **3.8.1 The system shall allow administrators to create and manage schedules for grades.**
* **3.8.2 Schedules shall include lessons, subjects, and teachers.**

***3.9 Lesson Management***

* **3.9.1 The system shall allow teachers to create and manage lessons.**
* **3.9.2 Each lesson shall be associated with a subject, teacher, and schedule.**

***3.10 Mark Management***

* **3.10.1 The system shall allow teachers to enter and update marks for students.**
* **3.10.2 Marks shall be associated with a student, lesson, and subject.**

**4. Non-Functional Requirements**

**4.1 Performance Requirements**

* The system should handle concurrent access for up to 2000 users.
* Response time for major actions should not exceed 2 seconds.
* Load balancing to distribute traffic effectively.

**4.3 Usability Requirements**

* Simple and intuitive user interface.
* Mobile responsiveness for easy access on different devices.
* Accessibility compliance with WCAG 2.1 standards.

**4.4 Scalability Requirements**

* Cloud-based deployment for easy scalability.
* Microservices architecture to support modular expansion.

**4.5 Validation and Testing**

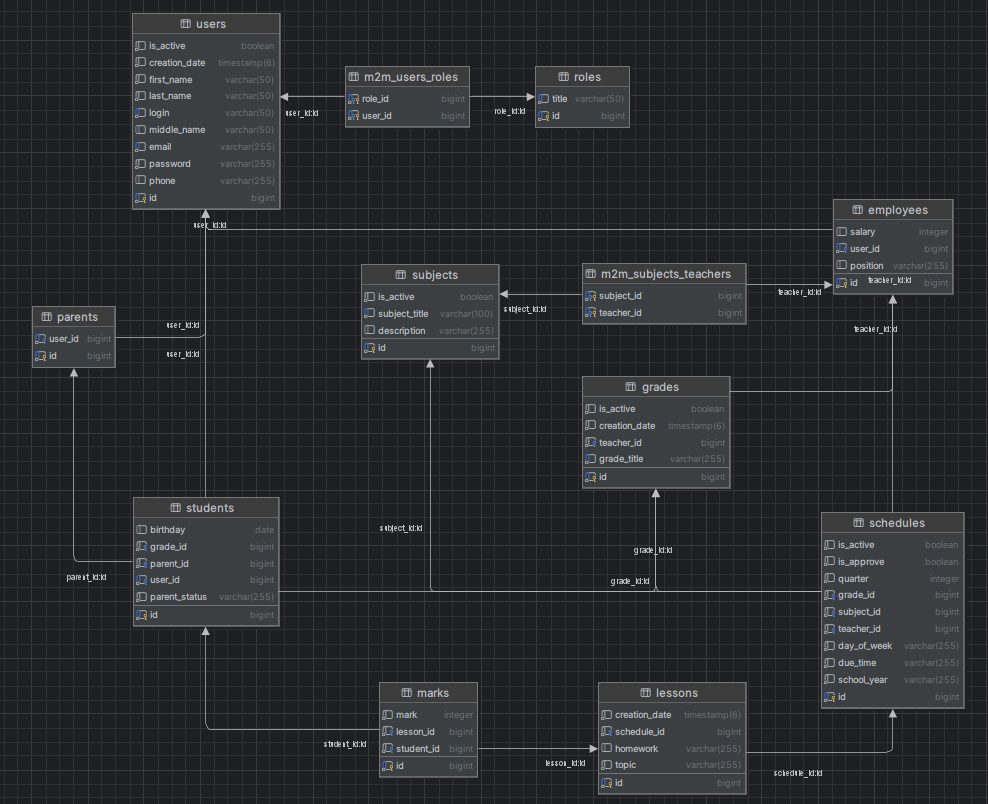
* Validation of user input to ensure data integrity.
* **Unit Tests:**
  + Mappers: Ensure proper data transformation between entities and DTOs.
  + Repositories: Validate correct data storage and retrieval from the database.
  + Controllers: Ensure API endpoints function correctly and handle errors gracefully.

**4.6 API Documentation**

**The system shall provide API documentation using Swagger UI and OpenAPI specifications. The API documentation shall be accessible at:**

* **Swagger UI: *http://localhost:8080/swagger-ui.html***
* **OpenAPI JSON: *http://localhost:8080/v3/api-docs***

1. **Entity-Relationship Diagram (ERD)**

****

**6. Database Schema**

create table if not exists users (

    id bigserial primary key,

    login varchar unique not null,

    first\_name varchar not null,

    last\_name varchar not null,

    middle\_name varchar,

    phone varchar,

    email varchar not null unique,

    password varchar not null,

    creation\_date timestamp with time zone not null default now(),

    is\_active boolean not null default true,

);

create table if not exists roles (

    id bigserial primary key,

    title varchar not null

);

create table if not exists m2m\_users\_roles (

    user\_id bigint references users(id),

    role\_id bigint references roles(id)

);

create table if not exists subjects (

    id bigserial primary key,

    subject\_title varchar not null unique,

    description text,

    is\_active boolean not null default true

);

create table if not exists employees (

    id bigserial primary key,

    position varchar not null,

    salary int not null,

    user\_id bigserial references users(id) not null

);

create table if not exists m2m\_subjects\_teachers (

    subject\_id bigint references subjects(id),

    teacher\_id bigint references employees(id)

);

create table if not exists parents (

    id bigserial primary key,

    user\_id bigserial references users(id) not null

);

create table if not exists grades (

    id bigserial primary key,

    grade\_title varchar not null,

    teacher\_id bigint references employees(id),

    creation\_date timestamp with time zone not null default now(),

    is\_active boolean not null default true

);

create table if not exists students (

    id bigserial primary key,

    birthday date not null,

    grade\_id bigserial references grades(id) not null,

    user\_id bigserial references users(id) not null,

    parent\_id bigint references parents(id) not null,

    parent\_status varchar not null

);

create table if not exists schedules (

    id bigserial primary key,

    day\_of\_week varchar not null,

    quarter int not null,

    due\_time varchar not null,

    year varchar not null,

    subject\_id bigint references subjects(id),

    teacher\_id bigint references employees(id),

    grade\_id bigint references grades(id),

    is\_approve boolean default false,

    is\_active boolean not null default true

);

create table if not exists lessons (

    id bigserial primary key,

    topic varchar,

    homework varchar,

    schedule\_id bigint references schedules(id),

    creation\_date timestamp not null default now()

);

create table if not exists marks (

    id bigserial primary key,

    mark int not null,

    student\_id bigint references students(id),

    lesson\_id bigint references lessons(id)

);

**Database Relationships**

**1. Users**

* **Central table for all users (students, parents, teachers, admins).**
* ***Relationships:***
* **One-to-Many with employees, parents, students.**
* **Many-to-Many with roles via m2m\_users\_roles.**

**2. Employees**

* **Stores school staff (teachers, admins).**
* ***Relationships:***
* **Many-to-One with users.**
* **One-to-Many with grades (as class teacher) and schedules.**

**3. Parents**

* **Stores parent/guardian information.**
* ***Relationships:***
* **Many-to-One with users.**
* **One-to-Many with students.**

**4. Grades**

* **Represents classes/grade levels.**
* ***Relationships:***
* **Many-to-One with employees (class teacher).**
* **One-to-Many with students and schedules.**

**5. Students**

* **Stores student information.**
* ***Relationships:***
* **Many-to-One with users, grades, and parents.**
* **One-to-Many with marks.**

**6. Subjects**

* **Stores subjects taught in the school.**
* ***Relationships:***
* **One-to-Many with schedules.**

**7. Schedules**

* **Represents the timetable for classes.**
* ***Relationships:***
* **Many-to-One with subjects, employees, and grades.**
* **One-to-Many with lessons.**

**8. Lessons**

* **Stores individual lessons.**
* ***Relationships:***
* **Many-to-One with schedules.**
* **One-to-Many with marks.**

**9. Marks**

* **Stores student grades for lessons.**
* ***Relationships:***
* **Many-to-One with students and lessons.**

**10. Roles**

* **Stores user roles (Admin, Teacher, Student, Parent).**
* ***Relationships:***
* **Many-to-Many with users via m2m\_users\_roles.**

**7. Use Cases**

**7.1 Admin Use Cases**

* **Create, update, and delete users.**
* **Assign roles to users.**
* **Manage grades, subjects, and schedules.**

**7.2 Teacher Use Cases**

* **Create and manage lessons.**
* **Enter and update marks for students.**
* **View schedules and student progress.**

**7.3 Student Use Cases**

* **View schedules and marks.**
* **Access lesson materials.**

**7.4 Parent Use Cases**

* **View student progress (marks, attendance).**
* **Communicate with teachers.**

**8. How to Run the Project**

This section provides step-by-step instructions for setting up and running the School Management Application on your local machine or server.

**8.1 Prerequisites**

**Before running the project, ensure you have the following installed:**

* **Java Development Kit (JDK): Version 17 or higher.**
* **Maven: For building and managing dependencies.**
* **H2/PostgreSQL: A relational database for storing application data.**
* **IDE: IntelliJ IDEA, Eclipse, or any Java-supported IDE.**
* **Postman (Optional): For testing REST APIs.**
* **Git: For cloning the repository.**

**8.2 Steps to Run the Project**

**Step 1: Clone the Repository**

1. **Open a terminal or command prompt.**
2. **Run the following command to clone the project repository:**

***https://github.com/NargizaAlieva/school-app***

**Step 2: Set Up the Database**

1. Open your PostgreSQL database client (e.g., pgAdmin).
2. Create a new database named school\_app.
3. Update the database configuration in the application-postgres.properties file:

spring.datasource.url=jdbc:postgresql://localhost:5432/school-app  
spring.datasource.username=postgres  
spring.datasource.password=postgres

**Step 3: Build the Project**

1. Run the application from your IDE:
   * Locate the main class (SchoolAppApplication.java).
   * Right-click and select Run.
2. Alternatively, run the application from the terminal:

mvn spring-boot:run

**8.3 Testing the Application**

1. Use **Postman** or any API testing tool to test the REST APIs.
2. Import the provided Postman collection (if available) to test all endpoints.
3. Alternatively, use the Swagger UI to test APIs:

http://localhost:8080/swagger-ui.html

1. If you want to see database you can navigate to see database in h2

[**http://localhost:8080/h2**](http://localhost:8080/h2)

**JDBC URL- jdbc:h2:mem:school-app**

**9. Future Improvements**

**9.1 Authorization and Authentication Enhancements**

* Implement Single Sign-On (SSO) for better user experience.
* Enhance security with biometric authentication support.
* Introduce passwordless authentication using magic links or device-based authentication.
* Implement fine-grained role management for more flexible access control.
* **OAuth 2.0 Implementation** for secure authentication and authorization.

**10. Development Phases**

* Requirement analysis and planning.
* Database design and implementation.
* Backend development using Spring Boot.
* Integration and system testing.
* Deployment and user training.

**11. Conclusion**

The **School Management Application** is a robust platform designed to automate and streamline school operations. By focusing on the **10 core entities**, the system provides a comprehensive solution for managing users, academic records, schedules, and communication. The application is scalable, secure, and user-friendly, making it an ideal choice for schools looking to modernize their operations.