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, academic performance monitoring. It aims to improve efficiency, reduce manual workload, and enhance collaboration between teachers, students, and parents.

1.2. Objectives

- Automate routine tasks for administrators, teachers, students, and parents.
- Provide a centralized platform for managing academic and administrative activities.
- Improve communication between all stakeholders (teachers, students, parents, and administrators).

1.3 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.

1.4 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.

1.5 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 Scope

The application will focus on the following **10 entities**:

1. **User**: Represents all users of the system (students, parents, teachers, administrators).
2. **Role**: Defines the permissions and access levels for each user.
3. **Grade**: Represents a class or grade level in the school.
4. **Student**: Represents a student enrolled in the school.
5. **Parent**: Represents a parent or guardian of a student.
6. **Employee**: Represents school staff (teachers, administrators, etc.).
7. **Schedule**: Represents the timetable for classes and lessons.
8. **Lesson**: Represents a specific lesson or class session.
9. **Mark**: Represents a grade or score given to a student for a lesson or subject.
10. **Subject**: Represents a subject taught in the school.
11. **Token:** Represents authentication tokens used for managing user sessions and securing API access. This includes access tokens, refresh tokens, and email verification tokens.

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

The system caters to the following classes:

1. ***Administrator:***
   - Manages users, roles, and system settings.
   - Oversees the overall functioning of the system.

2. ***Principal:***
   - Manages subjects, schedules, students, grades.
   - Communicates with students, teachers and parents.

3. ***Teacher:***
   - Manages lessons, schedules, and student grades.
   - Communicates with students and parents.

4. ***Class Teacher:***
   - Manage home grades.
   - Communicates with home grade students and their parents.

5. ***Student:***
   - Views schedules, grades, and lesson materials.
   - Interacts with teachers and parents.

6. ***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, Spring Security, JWT Token
- **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 Managemen

- 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 email.
- 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.

4.3 Security Requirements

- The system implements JWT (JSON Web Token) authentication for secure access to protected endpoints.
- OAuth2 login supported for third-party authentication (e.g., Google, GitHub).
- Role-based access control enforce different permissions for users (e.g., STUDENT, TEACHER, ADMIN).
- Password hashing shall be implemented using a secure algorithm (e.g., BCrypt).
- The system support email verification and two-factor authentication (2FA) to protect user accounts.

4.4 Scalability Requirements

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

4.5 Validation and Testing

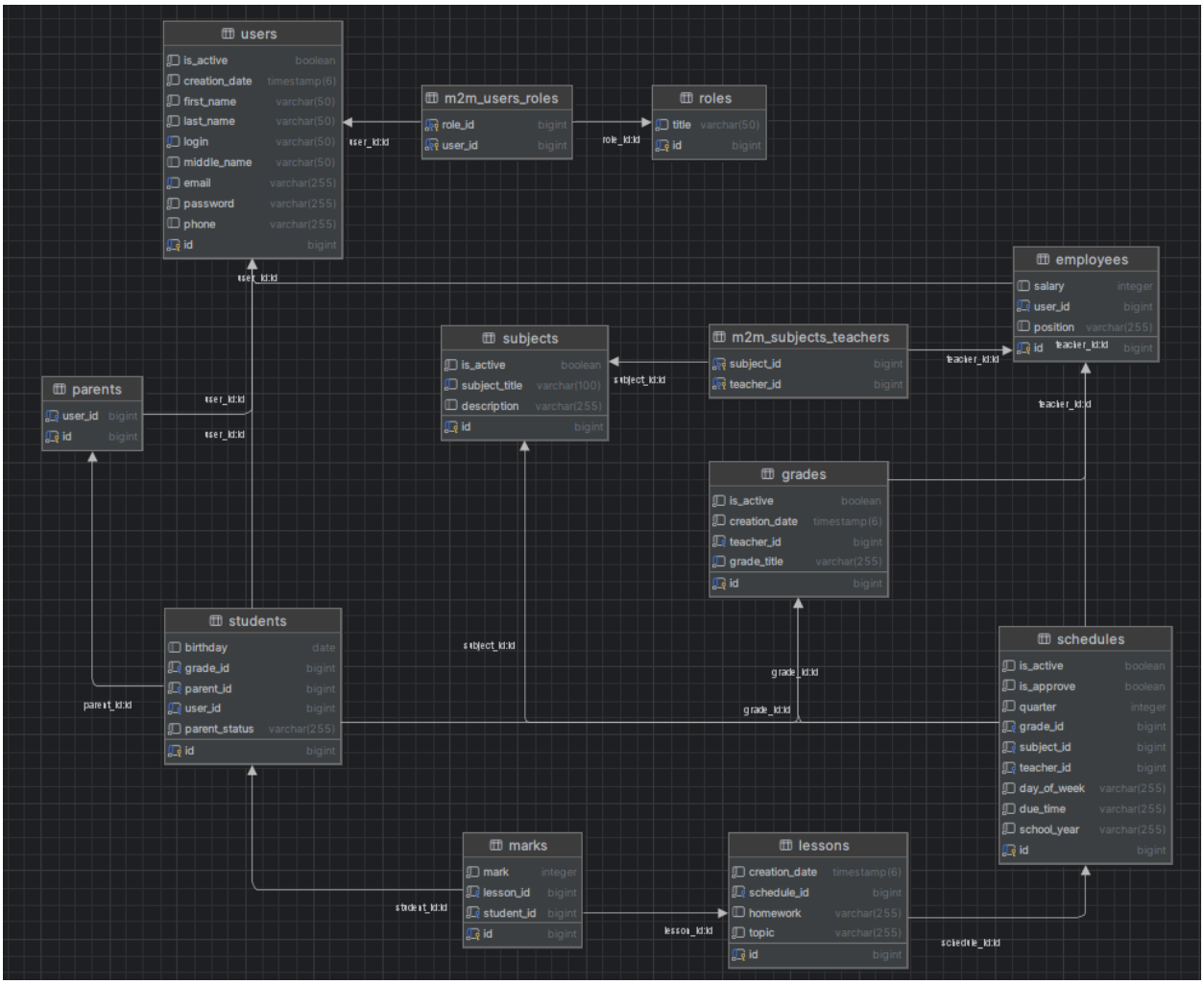
- Validation of user input to ensure data integrity.
- **Unit Tests:**
    - Mappers: Ensure proper data transformation between entities and DTOs.
    - Repositories: Validate correct data storage and retrieval from the database.
    - Controllers: Ensure API endpoints function correctly and handle errors 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:8888/swagger-ui.html***
- **OpenAPI JSON:** ***http://localhost:8888/v3/api-docs***

5. 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)
);

create table if not exists tokens (
    id bigserial primary key,
    token varchar not null,
    token_type varchar not null,
    created_at Date not null,
    expires_at Date not null,
    expired boolean not null,
    revoced boolean not null);
```

Database Relationships

1. Users

- Central table for all users (students, parents, teachers, admins).
- ***Relationships:***
    - One-to-Many with employees, parents, students, tokens.
    - 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.

11. Tokens

- Stores tokens of users.
- ***Relationships:***
    - Many-to-One with users.

7. Use Cases

7.1 Admin Use Cases

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

7.2 Principal Use Cases

- Create, update, and delete students, grades, employees, subjects.
- Manage grades, subjects, and schedules.

7.3 Teacher Use Cases

- Create and manage lessons.
- Enter and update marks for students.

- View schedules and student progress.

7.4 Class Teacher Use Cases
- View schedules and student progress of home grades.

7.5 Student Use Cases
- View schedules and marks.
- Access lesson materials.

7.6 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: Set Up the Jwt

1. Generate secret key by any jwt secret key generator.
2. Create a new database named school_app.
3. Update the secret key configuration in the application.properties file

```
secret.key=your_base64_encoded_secret_key
token.expiration=86400000  # e.g., 1 day in milliseconds
refresh-token.expiration=604800000  # e.g., 7 days
verification-token.expiration=300000  # e.g., 5 minutes
```

Step 4: Set Up the OAuth2

1. Register your application on each provider's developer console (Google, GitHub, Facebook).
2. Retrieve the Client ID and Client Secret from each platform.
3. Update the secret key configuration in the application.properties file

Step 4: Set Up the 2-verification

1. For Gmail: Go to https://myaccount.google.com/apppasswords and generate an app password.
2. For other email providers: Follow their respective documentation.
3. Update the secret key configuration in the application.properties file with your email

```
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=your_email@gmail.com
spring.mail.password=your_app_password
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

Step 5: 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:8888/swagger-ui.html
```

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

   http://localhost:8888/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.
- Implement fine-grained role management for more flexible access control.

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 11 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.