Simplified School Management System Implement Using JWT Token

Type: Projects

Skill: Java

Entity DesignError HandlingJWT AuthenticationRESTful API DevelopmentSpring Boot Framework

Medium

You are tasked with developing a Spring Boot-based web application for a Simplified School Management System. This system will manage students, teachers, and courses with basic authentication and authorization using JSON Web Tokens (JWT). The application will allow teachers to create and assign courses, students to view courses, and both teachers and students to update their personal information securely. Authentication will be handled via JWT tokens to ensure that only authenticated users can perform certain actions.

**Functional Requirements:**

1. User Registration (Student and Teacher)

Endpoint: /users/register

Method: @PostMapping

Description:

This endpoint allows both students and teachers to register by providing the following details:

username (unique identifier for the user)

email

password (encrypted using BCryptPasswordEncoder)

role (either STUDENT or TEACHER)

Access Control:

This API should be accessible to all users without authentication.

Upon successful registration, a confirmation message will be returned along with the created user object.

Input & Output:

The API Should accept User object in request body and return Registered User object in response body.

2. User Authentication (Login)

Endpoint: /users/login

Method: @PostMapping

Description:

This endpoint allows users to log in by providing their username and password.

Upon successful authentication, a JWT token is generated, which can be used for subsequent secure API calls.

The system should return an HTTP status code 401 (Unauthorized) if the login fails.

Access Control:

This API is accessible to all users (without authentication).

Input & Output:

Input: { "username": "user1", "password": "password123" }

Output: { "token": "jwt-token" }

You can use already provided Dto classes `AuthRequest` and `AuthResponse` for this API.

DTO classes for login api are already provided in the project at location `src/main/java/com/wecp/school\_management\_system\_jwt/dto`

If password is incorrect, return a 401 Unauthorized response.

3. Create Course (Teacher Only)

Endpoint: /courses?teacherId={teacherId}

Method: @PostMapping

Description:

This endpoint allows teachers to create new courses by providing the following details:

courseName

courseDescription

Access Control:

Only authenticated teachers with a valid JWT token can create courses.

The course will be saved in the database, and the teacher will be assigned as the course instructor.

Input & Output:

The API should accept Course object in the request body, teacherId as a query parameter and return the created Course object in the response body.

4. Retrieve Course List

Endpoint: /courses

Method: @GetMapping

Description:

This endpoint allows authenticated users (both students and teachers) to retrieve the list of available courses.

The course information (name, description, teacher) will be returned as a JSON array.

Access Control: Only authenticated users with a valid JWT token can access this endpoint.

5. Enroll in a Course (Student Only)

Endpoint: /courses/enroll/{courseId}?studentId={studentId}

Method: @PostMapping

Description:

This endpoint allows students to enroll in a course.

The student will be added to the course as a participant.

Access Control: Only authenticated students with a valid JWT token can enroll in a course.

Input & Output:

The API should accept courseId as a path variable, studentId as a query parameter and return the updated Course object in the response body.

6. Update User Information (Teacher and Student)

Endpoint: /users/{userId}

Method: @PutMapping

Description:

This endpoint allows authenticated users (either students or teachers) to update their personal information (email or password).

Access Control: Only authenticated users with a valid JWT token can update their personal information.

Input & Output:

The API should accept User object in the request body, userId as a path variable and return the updated User object in the response body.

**Entity Design**

* **User Entity**
* Attributes:
* id: Long (unique, auto-generated)
* username: String (unique)
* email: String
* password: String (encrypted)
* role: String (STUDENT or TEACHER)
* enrolledCourses: Set<Course> (courses the student is enrolled in)
* Relationships: Many-to-many with Course entity (a student can enroll in many courses, and each course can have many students).
* **Course Entity**
* Attributes:
* id: Long (unique, auto-generated)
* courseName: String
* courseDescription: String
* teacher: User (many-to-one relationship with User entity, representing the teacher)
* students: Set<User> (students enrolled in the course)

**Notes:**

* Implement Getters and Setters of all fields in the entities as per standard java practices.
* mapped the entities with the table names as users, courses respectively.

**Technical Notes:**

Checkin Permission: Check permission w.r.t authority of the user.

Use hasAuthority() method to check the role of the user. For example hasAuthority("TEACHER") or hasAuthority("STUDENT").

Password Encoding: Use BCryptPasswordEncoder to securely encode user passwords during registration.

JWT Authentication:

After successful login, generate a JWT token that is used to authenticate subsequent API requests.

Ensure all endpoints that require authentication (such as course creation, enrollment, and personal info updates) check for a valid JWT token in the request header.

The token should be passed in the header as Authorization: Bearer <JWT\_token>.

Error Handling:

Return appropriate HTTP status codes and messages for different scenarios:

401 Unauthorized for invalid or missing JWT token.

400 Bad Request for invalid inputs.

404 Not Found if a requested user or course does not exist.

403 Forbidden if a user attempts to access unauthorized resources.