Using JSON Web Token authentication in Library Management System

Type: Projects

Skill: Java

CRUD OperationsRESTful APIRole-Based AuthorizationSpring BootUser Authentication

Medium

Develop a Spring Boot-based web application for a Library Management System that allows users to browse books, check out books, and manage their library account. The system should include user authentication and authorization via Spring Security.

There will be two types of users: Patrons (regular library members) and Librarians. Patrons can search for and check out books, while Librarians can manage the catalog of books and view patron information.

**Functional Requirements:**

Role-Based Authorization:

* PATRON role can search for books, check out books, and view their own checkout history.
* LIBRARIAN role can add, update, or remove books from the catalog, as well as view all patron information and check out history.
* Check permission in SecurityConfig.java file.
* Use hasAuthority() method to check the role of the user.
* For example hasAuthority("LIBRARIAN") or hasAuthority("PATRON")

1. User Authentication and Authorization:

User Registration:

* Implement a RESTful endpoint /users/register with a @PostMapping annotation to allow new users (patrons) to register for a library account.
* The registration request should include username, password, first name, last name, and role (either "PATRON" or "LIBRARIAN").
* Passwords should be encrypted using BCryptPasswordEncoder before storing them in the database.
* The API should accept User objects in RequestBody and return created User objects in ResponseBody.

User Login:

* Implement login functionality using Spring Security for authenticating users by their username and password.
* After successful login, a JWT token should be issued for subsequent requests.
* API = POST /users/login
* 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/book\_library\_management\_system\_jwt/dto`
* If password is incorrect, return a 401 Unauthorized response.

2. Book Management (Librarian Role):

Book Entity:

* A Book entity should have the following fields:
* id (Long) – Unique identifier for the book.
* title (String) – Title of the book.
* author (String) – Author of the book.
* isbn (String) – ISBN of the book.
* availableCopies (int) – Number of copies of the book available in the library.

Book CRUD Operations:

* Create Book: Implement a RESTful endpoint /books with a @PostMapping annotation to allow LIBRARIAN users to add new books to the library catalog.
* The API should accept Book objects in RequestBody and return created Book objects in ResponseBody.
* Update Book: Implement a RESTful endpoint /books/{id} with a @PutMapping annotation to allow LIBRARIAN users to update book details (e.g., availability or price).
* The API should accept Book objects in RequestBody and return updated Book objects in ResponseBody.
* Delete Book: Implement a RESTful endpoint /books/{id} with a @DeleteMapping annotation to allow LIBRARIAN users to delete books from the catalog.
* View All Books: Implement a RESTful endpoint /books with a @GetMapping annotation to allow both PATRON and LIBRARIAN users to view a list of all books in the catalog.

3. Book Checkout (Patron and Librarian Role):

Checkout Entity:

* A Checkout entity should have the following fields:
* id (Long) – Unique identifier for the checkout record.
* book (Book) – Many-to-one relationship with the Book entity.
* patron (User) – Many-to-one relationship with the User entity (who is the patron).
* checkoutDate (LocalDateTime) – Date and time of checkout.
* dueDate (LocalDateTime) – Due date for returning the book.

Checkout Book:

* Implement a RESTful endpoint /checkouts?bookId={bookId}&patronId={patronId}
* with a @PostMapping annotation to allow PATRON users to check out a book.
* Ensure that a patron cannot check out a book if there are no available copies in the library.
* The checkout record should include the book ID and patron ID, and a due date should be set for returning the book. ( set an arbitrary due date, e.g., 30 days from the checkout date)
* Update the available copies of the book after a successful checkout.
* The API Should return Checkout object in ResponseBody.

View Patron Checkout History:

* Implement a RESTful endpoint /users/{userId}/checkouts with a @GetMapping annotation to allow users (both PATRON and LIBRARIAN) to view the list of books that a patron has checked out.
* The API should return a list of Checkout objects in ResponseBody.

4. User Management (Librarian Role):

View User Information:

* Implement a RESTful endpoint /users/{userId} with a @GetMapping annotation to allow LIBRARIAN users to view details of any patron.
* The API should return User object in ResponseBody.

Update User Information:

* Implement a RESTful endpoint /users/{userId} with a @PutMapping annotation to allow LIBRARIAN users to update a patron's personal information (e.g., email, name).
* The API should accept User objects in RequestBody and return updated User objects in ResponseBody.

**Entities:**

User Entity:

* id (Long) – Unique identifier for the user.
* username (String) – Unique username.
* password (String) – Encoded password.
* firstName (String) – Patron's first name.
* lastName (String) – Patron's last name.
* role (String) – Role of the user, either "PATRON" or "LIBRARIAN".

Book Entity:

* id (Long) – Unique identifier for the book.
* title (String) – Title of the book.
* author (String) – Author of the book.
* isbn (String) – ISBN of the book.
* availableCopies (int) – Available copies of the book in the library.

Checkout Entity:

* id (Long) – Unique identifier for the checkout record.
* book (Book) – Book that was checked out.
* patron (User) – User who checked out the book.
* checkoutDate (LocalDateTime) – Date the book was checked out.
* dueDate (LocalDateTime) – Due date for returning the book.

Implement Getters and Setters of all fields in the entities as per standard java practices. Mapped the entities with the table names as users, books, checkouts respectively.

**Notes:**

* Spring Security should be configured for secure endpoints with role-based access control.
* JWT Tokens should be used for authentication, ensuring that users with appropriate roles can access protected endpoints.
* Password Encoding: Ensure passwords are securely encoded with BCryptPasswordEncoder.
* Role-based Access Control: Implement role-based authorization to ensure that only LIBRARIAN users can modify the book catalog and view patron details, while PATRON users can only interact with books (e.g., check out books and view their checkout history).
* Stock Management: When a book is checked out, the number of available copies should be reduced, and no patron should be able to check out more books than are available.