FINAL PROJECT 1

JAVA ACADEMY - XIDERAL

SEPTEMBER 17, 2024

AUTHOR: REGINA RODRIGUEZ CAMPO GARRIDO

1. Introduction

The project focuses on managing a library through an application that uses Spring Data JPA for database interaction and Spring Security for access control based on user roles.

The system allows users to view available books, apply dynamic discounts using lambdas and streams, check if a book is in stock, and retrieve the number of books by genre. There are three types of users: READER, ADMIN, and SUPERADMIN, each with specific permissions. READER users can only perform queries (GET), ADMIN users can add new records (POST), and SUPERADMIN users can delete records (DELETE).

The application manages three main entities and uses a MySQL database with four tables to store and manage this information.

2. Project Structure

The project is organized following a layered architecture:

- spring.jpa.dao: This package contains interfaces that define repositories. In this case, it includes BookRepository, AuthorityRepository, and UserRepository.
- spring.jpa.entity: This package holds the entities that represent the tables in the database.
- spring.jpa.rest: This package contains the REST controllers responsible for handling HTTP requests. The BookController exposes the endpoints to interact with books, while Security manages security configurations.
- spring.jpa.service: The business logic is encapsulated in this package. The BookServiceImpl service manages operations related to books. It also includes SecurityUserDetailsService, which handles user authentication.
- utils: This package contains utility classes to handle security logic.

3. Implementation

3.1 Controller Layer

BookController class, is a REST controller responsible for handling HTTP requests related to the Book entity.

```java
@RestController
@RequestMapping("/rest")
public class BookController {

    private final BookService bookService;

    @Autowired
    public BookController(BookService theBookService) {
        bookService = theBookService;
    }

    @GetMapping("/books")// Get the list of all books
    public List<Book> findAll() {
        return bookService.findAll();
    }

    @GetMapping("/books/{bookId}") //Get the book by id
    public Book getBook(@PathVariable int bookId) {
        Book theBook = bookService.findById(bookId);
        if (theBook == null) {
            throw new RuntimeException("Book id not found - " + bookId);
        }
        return theBook;
    }

    @GetMapping("/books/genre/{genre}") //Number of books by genre
    public long countBooksByGenre(@PathVariable String genre) {
        return bookService.countByGenre(genre);
    }

    @GetMapping("/books/{bookId}/stock")//Books available
    public String IsStock(@PathVariable int bookId) {
        boolean inStock = bookService.IsStock(bookId);
            String message = inStock ? "Book available" : "Book not available";
            return message ;
    }

    @GetMapping("/books/{discountPercentage}/price")//Books available
    public List <String> Discount(@PathVariable double discountPercentage) {
          List<Book> booksWithDiscount = bookService.PriceWithDiscount(discountPercentage);
          return booksWithDiscount.stream()
                  .map(book -> "Title: " + book.getTitle() + ", Price with discount: " + book.getPrice())
                  .collect(Collectors.toList());
    }

    @PostMapping("/books") //add new post
    public Book addBook(@RequestBody Book theBook) {
        theBook.setId(0);
        return bookService.save(theBook);
    }
```

```java
@PutMapping("/books") //update a book
public Book updateBook(@RequestBody Book theBook) {
    return bookService.save(theBook);
}

@DeleteMapping("/books/{bookId}")//delete books
public String deleteBook(@PathVariable int bookId) {
    Book tempBook = bookService.findById(bookId);
    if (tempBook == null) {
        throw new RuntimeException("Book id not found - " + bookId);
    }
    bookService.deleteById(bookId);
    return "Deleted book id - " + bookId;
}
```

3.2 Service Layer

BookServiceImp, implements the business logic for managing books in the system.

```java
@Service
public class BookServiceImpl implements BookService {

    @PersistenceContext
    private EntityManager entityManager;
    private BookRepository bookrepository;

    @Autowired
    public BookServiceImpl(BookRepository thebookrepository) {
        bookrepository = thebookrepository;
    }

    @Override
    public List<Book> findAll() { //find all books
        return bookrepository.findAll();
    }

    @Override
   public Book findById(int theId) { //find by ID
    Optional<Book> result = bookrepository.findById(theId);
    return result.orElse(null);
    }

   @Override
   public long countByGenre(String genre) { //count books by genre
       String jpql = "SELECT COUNT(b) FROM Book b WHERE b.genre = :genre";
       TypedQuery<Long> query = entityManager.createQuery(jpql, Long.class);
       query.setParameter("genre", genre);
       return query.getSingleResult();
    }


    @Override
    public boolean IsStock(int bookId) { //Checks if a book is in stock
        Book book = entityManager.find(Book.class, bookId);
        return book != null && book.getStock() > 0 ;
   }

    @Override
    public List<Book> PriceWithDiscount(double discountPercentage){ //Discount with lambas
         List<Book> books =  findAll();
         return books.stream()
                 .map(book -> {
                     if(book.getPrice() >10){
                         book.setPrice(book.getPrice() - (book.getPrice() * (discountPercentage / 10)));
                     }  return book;
                 })
                 .collect(Collectors.toList());
    }
```

```java
@Transactional
@Override
public Book save(Book theBook) { //Add new book or update of one
    return bookrepository.save(theBook);
}

@Transactional
@Override
public void deleteById(int theId) {//Delete book
    bookrepository.deleteById(theId);
}
```

3.3 Repository Layer

Repositories interacts with the database using Spring Data JPA

```java
import org.springframework.data.jpa.repository.JpaRepository;

public interface BookRepository extends JpaRepository<Book, Integer> {

}
```

```java
public interface AuthorityRepository extends JpaRepository<Authority, Integer> {

    Optional<Authority> findByName(AuthorityName name);

}
```

```java
public interface UserRepository extends JpaRepository<Members, Integer> {

    Optional<Members> findByUsername(String username);
}
```

3.4 Entity Layer

The Book class represents a book in the system. It is mapped to the buk table in the database and includes fields such as id, title, author, genre, price, published, and stock.

```java
package spring.jpa.entity;

import jakarta.persistence.*;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name="buk")
public class Book {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="id")
    private int id;

    @Column(name="title")
    private String title;

    @Column(name="author")
    private String author;

    @Column(name="genre")
    private String genre;

    @Column(name="price")
    private double price;


    @Column(name="published")
    private int published;

    @Column(name="stock")
    private int stock;
}
```

The Members class represents a user in the system. It is mapped to the Person table and includes fields for id, username, password, and a list of authorities. This entity supports many-to-many relationships with the Authority entity, managed through the user_authority join table.

```java
@Entity
@Table(name="Person")
public class Members {


    public Members() {
    }

    public Members(String username, String password, List<Authority> authorities) {
        this.username = username;
        this.password = password;
        this.authorities = authorities;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public List<Authority> getAuthorities() {
        return authorities;
    }

    public void setAuthorities(List<Authority> authorities) {
        this.authorities = authorities;
    }

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="id_user")
    private int id;

    @Column(name="username")
    private String username;
```

```java
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="id_user")
    private int id;

    @Column(name="username")
    private String username;

    @Column(name="_password")
    private String password;

    @ManyToMany(fetch = FetchType.EAGER)
    @JoinTable(
        name = "user_authority",
        joinColumns = @JoinColumn(name = "user_id", referencedColumnName = "id_user"),
        inverseJoinColumns = @JoinColumn(name = "authority_id", referencedColumnName = "id_auto")
    )
    private List<Authority> authorities;
}
```

The Authority class represents a role or permission assigned to a user. It is mapped to the Autorities table and includes fields for id and name. The name field is an enumerated type (AuthorityName) that describes the role's name. This entity is used to manage user roles and permissions in the system.

```java
@Entity
@Table(name="Autorities")
public class Authority {


    public Authority() {
    }

    public Authority(AuthorityName name) {
        this.name = name;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public AuthorityName getName() {
        return name;
    }

    public void setName(AuthorityName name) {
        this.name = name;
    }

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="id_auto")
    private int id;

    @Enumerated(EnumType.STRING)
    @Column(name="name_auto")
    private AuthorityName name;
}
```

4 Spring Security Integration

Spring Security is a powerful and highly customizable authentication and access-control framework.

```java
package spring.jpa.rest;

import org.springframework.context.annotation.Configuration;

@Configuration
public class Security {

//    @Bean
//      public UserDetailsService userDetailsService() {
//        var user = User.withUsername("user")
//          .password("123")
//          .roles("customer")
//          .build();
//
//        return new InMemoryUserDetailsManager(user);
//          }

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .httpBasic()
            .and()
            .authorizeHttpRequests(authorizeRequests ->
                authorizeRequests
                    .requestMatchers(HttpMethod.POST, "/rest/books").hasRole("ADMIN")
                    .requestMatchers(HttpMethod.DELETE, "/rest/books").hasRole("SUPERADMIN")
                    .requestMatchers(HttpMethod.GET, "/rest/books/**").authenticated()
                    .anyRequest().authenticated()
            )
            .csrf(csrf -> csrf.disable());
        return http.build();
    }

    @Bean
      public PasswordEncoder passwordEnconder() {
          return NoOpPasswordEncoder.getInstance();

      }


}
```

5. Database

```sql
CREATE DATABASE bookstoreDB;
USE bookstoreDB;
CREATE TABLE buk (
    id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(100) NOT NULL,
    author VARCHAR(100) NOT NULL,
    genre VARCHAR(50)NOT NULL,
    price double NOT NULL,
    published int(4)  NOT NULL,
    stock int(3) NOT NULL
)ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=latin1;


CREATE TABLE Person(
    id_user INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(100) NOT NULL,
    _password VARCHAR(100) NOT NULL
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=latin1;


CREATE TABLE Autorities(
    id_auto INT AUTO_INCREMENT PRIMARY KEY,
    name_auto VARCHAR(100) NOT NULL
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=latin1;

CREATE TABLE user_authority (
    user_id INT NOT NULL,
    authority_id INT NOT NULL,
    PRIMARY KEY (user_id, authority_id),
    FOREIGN KEY (user_id) REFERENCES Person(id_user),
    FOREIGN KEY (authority_id) REFERENCES Autorities(id_auto)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;


INSERT INTO buk (title, author, genre, price, published, stock)
VALUES
('To Kill a Mockingbird', 'Harper Lee', 'Fiction', 12, 1960,57),
('1984', 'George Orwell', 'Dystopian', 14, 1984, 0),
('The Great Gatsby', 'F. Scott Fitzgerald', 'Fiction', 10, 1925, 15),
('The Catcher in the Rye', 'J.D. Salinger', 'Fiction', 9,1951, 7 ),
('Moby-Dick', 'Herman Melville', 'Adventure', 15, 1851, 13 );
```

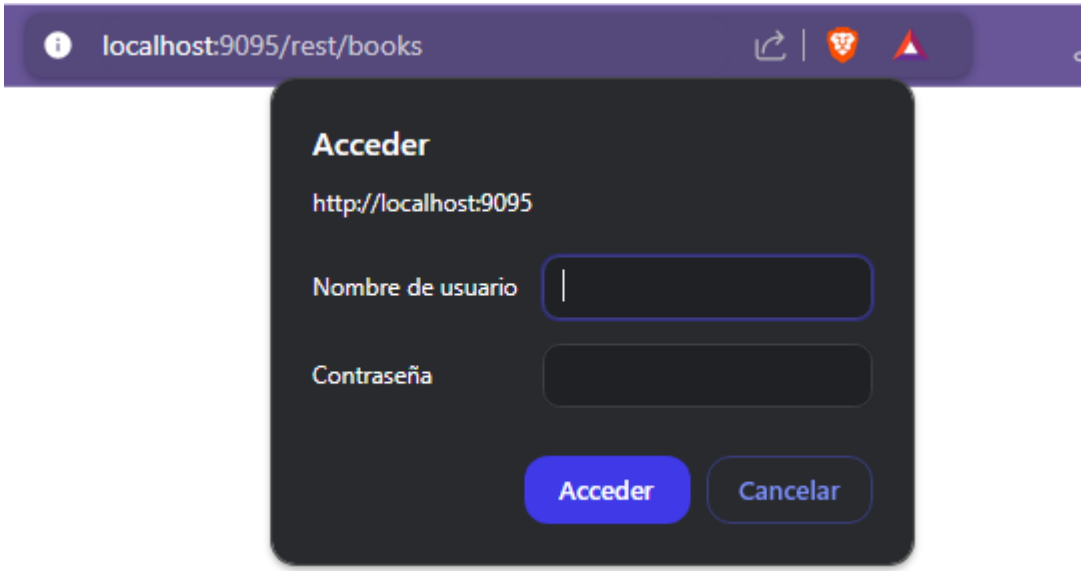
7. Test

```java
@Test
public void testMemberCreation() {
    Members member = new Members("user1", "password1", null);
    assertNotNull(member);
    assertEquals("user1", member.getUsername());
    assertEquals("password1", member.getPassword());
    assertNull(member.getAuthorities());
}

@Test
public void testMemberSettersAndGetters() {
    Members member = new Members();
    member.setUsername("user2");
    member.setPassword("password2");
    assertEquals("user2", member.getUsername());
    assertEquals("password2", member.getPassword());
    assertNull(member.getAuthorities());
}

@Test
public void testMemberWithAuthorities() {
    Authority authority = new Authority(AuthorityName.ROLE_ADMIN);
    Members member = new Members("user3", "password3", List.of(authority));

    assertNotNull(member);
    assertEquals("user3", member.getUsername());
    assertEquals("password3", member.getPassword());
    assertNotNull(member.getAuthorities());
    assertEquals(1, member.getAuthorities().size());
    assertEquals(AuthorityName.ROLE_ADMIN, member.getAuthorities().get(0).getName());
}


@Test
public void testAuthorityCreation() {
    Authority authority = new Authority(AuthorityName.ROLE_ADMIN);
    assertNotNull(authority);
    assertEquals(AuthorityName.ROLE_ADMIN, authority.getName());
}

@Test
public void testAuthoritySettersAndGetters() {
    Authority authority = new Authority();
    authority.setName(AuthorityName.ROLE_READER);
    assertEquals(AuthorityName.ROLE_READER, authority.getName());
}
```

```java
@Test
public void testBookCreation() {
    Book book = new Book(0, "Title", "Author", "Genre", 19.99, 2024, 10);
    assertNotNull(book);
    assertEquals("Title", book.getTitle());
    assertEquals("Author", book.getAuthor());
    assertEquals("Genre", book.getGenre());
    assertEquals(19.99, book.getPrice());
    assertEquals(2024, book.getPublished());
    assertEquals(10, book.getStock());
}

@Test
public void testBookSettersAndGetters() {
    Book book = new Book();
    book.setTitle("New Title");
    book.setAuthor("New Author");
    book.setGenre("New Genre");
    book.setPrice(29.99);
    book.setPublished(2023);
    book.setStock(15);

    assertEquals("New Title", book.getTitle());
    assertEquals("New Author", book.getAuthor());
    assertEquals("New Genre", book.getGenre());
    assertEquals(29.99, book.getPrice());
    assertEquals(2023, book.getPublished());
    assertEquals(15, book.getStock());
}

@Test
public void testSecurityAuthorityGetAuthority() {
    Authority authority = new Authority(AuthorityName.ROLE_ADMIN);
    SecurityAuthority securityAuthority = new SecurityAuthority(authority);

    assertEquals(AuthorityName.ROLE_ADMIN.toString(), securityAuthority.getAuthority());
}

@Test
public void testSecurityUserGetAuthorities() {
    Authority authority = new Authority(AuthorityName.ROLE_ADMIN);
    Members member = new Members("user1", "password1", List.of(authority));
    SecurityUser securityUser = new SecurityUser(member);

    assertNotNull(securityUser.getAuthorities());
    assertEquals(1, securityUser.getAuthorities().size());
    assertEquals(AuthorityName.ROLE_ADMIN.toString(), securityUser.getAuthorities().iterator().next().getAuthority());
}
```

8. OutPut

We enter a user.



FINAL PROJECT 1

Get books

http://localhost:9095/rest/books

GET http://localhost:9095/rest/books

Params Authorization Headers (9) Body Scripts Settings

Body Cookies Headers (14) Test Results

Pretty Raw Preview Visualize JSON


```
[
    {
        "id": 1,
        "title": "To Kill a Mockingbird",
        "author": "Harper Lee",
        "genre": "Fiction",
        "price": 12.0,
        "published": 1960,
        "stock": 57
    },
    {
        "id": 2,
        "title": "1984",
        "author": "George Orwell",
        "genre": "Dystopian",
        "price": 14.0,
        "published": 1984,
        "stock": 0
    },
    {
        "id": 3,
        "title": "The Great Gatsby",
        "author": "F. Scott Fitzgerald",
        "genre": "Fiction",
        "price": 10.0,
        "published": 1925,
        "stock": 15
    },
    {
        "id": 4,
        "title": "The Catcher in the Rye",
        "author": "J.D. Salinger",
        "genre": "Fiction",
        "price": 9.0,
        "published": 1951,
        "stock": 7
    },
    {
        "id": 5,
```



```
{
    "id": 5,
    "title": "Moby-Dick",
    "author": "Herman Melville",
    "genre": "Adventure",
    "price": 15.0,
    "published": 1851,
    "stock": 13
}
```

FINAL PROJECT 1

Get book by Id

http://localhost:9095/rest/books/4


```
{
    "id": 4,
    "title": "The Catcher in the Rye",
    "author": "J.D. Salinger",
    "genre": "Fiction",
    "price": 9.0,
    "published": 1951,
    "stock": 7
}
```


Check if book is available

http://localhost:9095/rest/books/4/stock

```
Book available
```

Give the number of books by genre

http://localhost:9095/rest/books/genre/Fiction

```
4
```

A reader user can't make a post

Daniel

123


```
{
    "timestamp": "2024-09-15T00:50:03.627+00:00",
    "status": 403,
    "error": "Forbidden",
    "message": "Forbidden",
    "path": "/rest/books"
}
```

FINAL PROJECT 1

An admin user can make a post

Username: Regina

Password: 123

```
{
    "id": 11,
    "title": "Example1",
    "author": "Harper Lee",
    "genre": "Fiction",
    "price": 12.99,
    "published": 1960,
    "stock": 25
}
```

POST http://localhost:9095/rest/books

Params | Authorization | Headers (10) | Body | Scripts | Settings

none | form-data | x-www-form-urlencoded | raw | binary | GraphQL | JSON

```
{

    "title": "Example1",
    "author": "Harper Lee",
    "genre": "Fiction",
    "price": 12.99,
    "published": 1960,
    "stock": 25
}

```