

# BÀI 7 PHÂN QUYỀN NGƯỜI DÙNG CHO TÀI KHOẢN ĐĂNG NHẬP

## Bài này giúp người học nắm được các nội dung sau:

Sau khi học xong phân quyền trong Java Spring Boot, người học sẽ nắm được các kiến thức và kỹ năng cơ bản sau đây:

- ✓ Hiểu về cách hoạt động của phân quyền theo phương pháp Role-base trong ứng dụng web.
- ✓ Biết cách triển khai phân quyền người dùng và quản trị viên trong Java Spring Boot.
- ✓ Có khả năng sử dụng các công cụ và thư viện như Spring Security để thực hiện phân quyền trong ứng dụng của mình.
- ✓ Hiểu về cách quản lý quyền truy cập và kiểm soát truy cập của người dùng vào các tính năng cụ thể của ứng dụng.

## Mô tả chức năng:

Trong Java Spring Boot, phân quyền người dùng và quản trị viên là một phần quan trọng để đảm bảo tính bảo mật cho ứng dụng web của bạn. Các chức năng của phân quyền người dùng và quản trị viên trong Java Spring Boot được mô tả như sau:

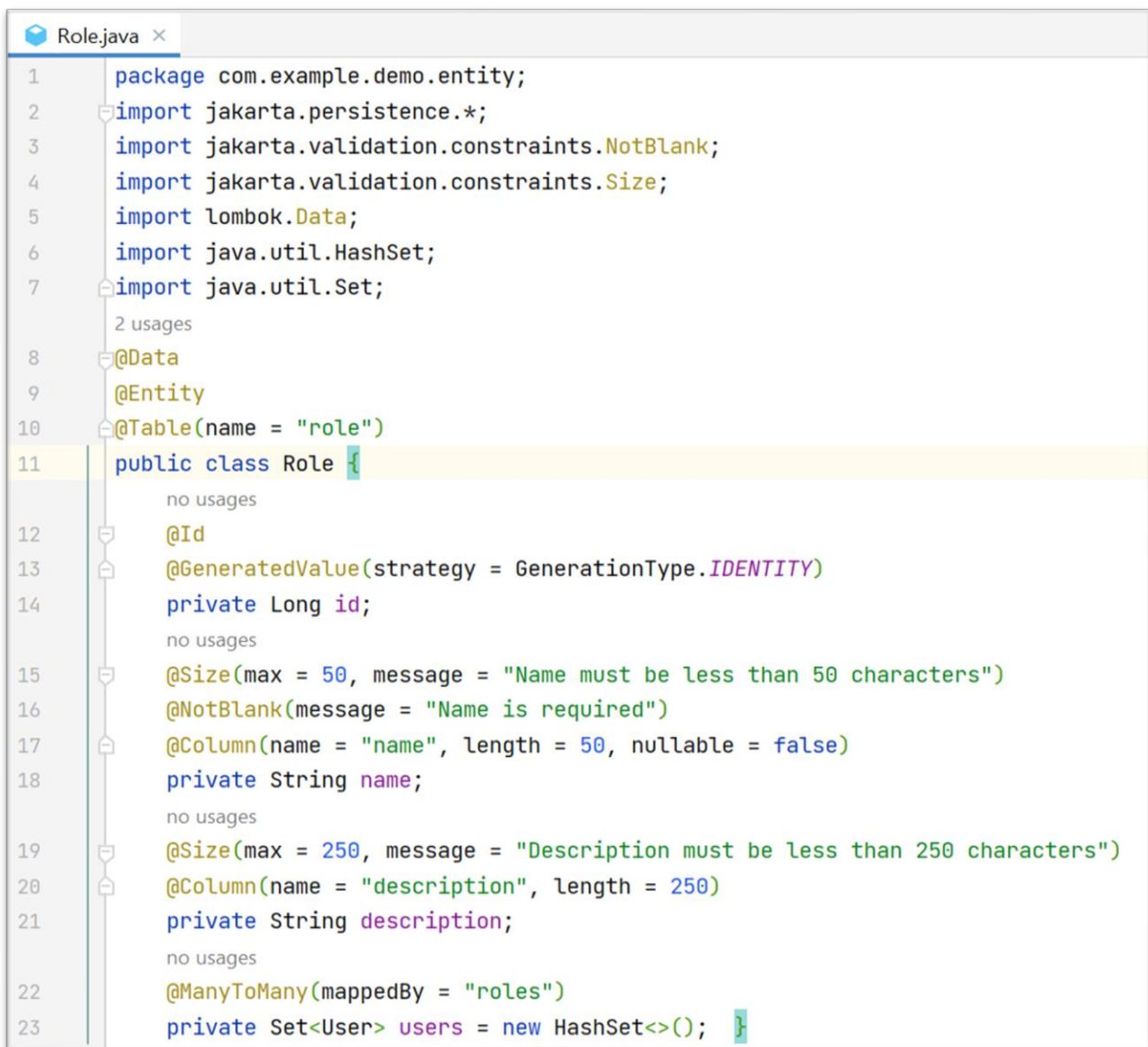
**Phân quyền người dùng (user):** Người dùng được phân quyền sẽ có quyền truy cập vào các tài nguyên cần thiết để sử dụng ứng dụng web. Các tài nguyên này có thể là các trang web, tính năng, thông tin người dùng và các chức năng khác.

**Phân quyền quản trị viên (admin):** Quản trị viên là người sở hữu quyền kiểm soát tất cả các hoạt động của ứng dụng. Quản trị viên có thể xem và chỉnh sửa các thông tin của người dùng, quản lý các tài nguyên và các chức năng khác liên quan đến ứng dụng.

## 7.1 Thêm table và dữ liệu vào table Role trong CSDL

### ➤ Thêm table Role trong CSDL

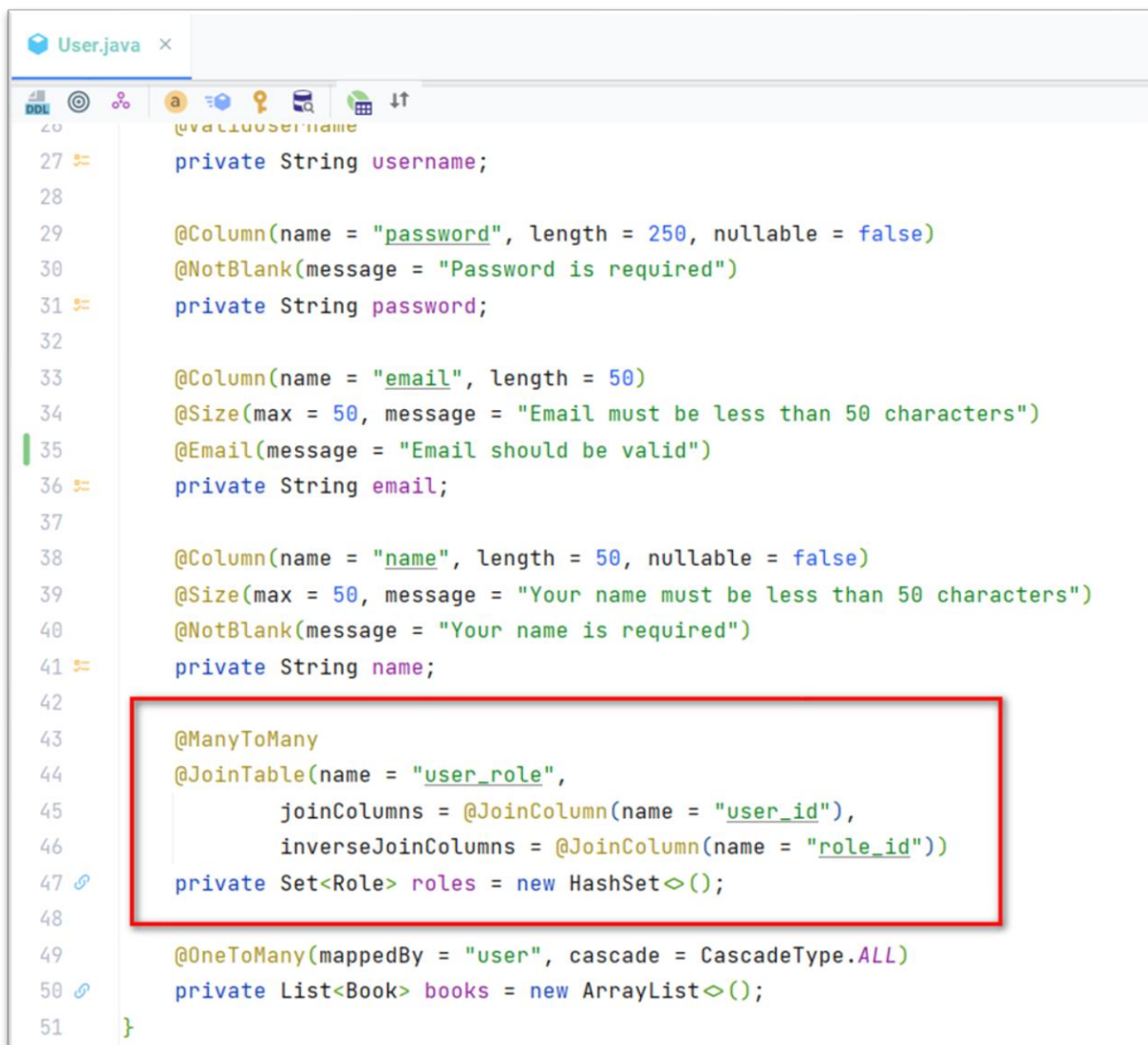
Tạo class **Role.java** trong thư mục **Entity** theo đường dẫn sau đây:  
**src/main/java/com.example.demo/entity**



```
1 package com.example.demo.entity;
2 import jakarta.persistence.*;
3 import jakarta.validation.constraints.NotBlank;
4 import jakarta.validation.constraints.Size;
5 import lombok.Data;
6 import java.util.HashSet;
7 import java.util.Set;
8
9 @Data
10 @Entity
11 @Table(name = "role")
12 public class Role {
13     @Id
14     @GeneratedValue(strategy = GenerationType.IDENTITY)
15     private Long id;
16
17     @Size(max = 50, message = "Name must be less than 50 characters")
18     @NotBlank(message = "Name is required")
19     @Column(name = "name", length = 50, nullable = false)
20     private String name;
21
22     @Size(max = 250, message = "Description must be less than 250 characters")
23     @Column(name = "description", length = 250)
24     private String description;
25
26     @ManyToMany(mappedBy = "roles")
27     private Set<User> users = new HashSet<>();
28 }
```

Hình 7.1. Tạo thêm class role

Tiếp theo, chỉnh sửa thêm nội dung Role.java như ảnh bên dưới, quan hệ n-n (many-to-many) giữa đối tượng User và đối tượng Role trong cơ sở dữ liệu.



```

26  @NotBlank(message = "Username is required")
27  private String username;
28
29  @Column(name = "password", length = 250, nullable = false)
30  @NotBlank(message = "Password is required")
31  private String password;
32
33  @Column(name = "email", length = 50)
34  @Size(max = 50, message = "Email must be less than 50 characters")
35  @Email(message = "Email should be valid")
36  private String email;
37
38  @Column(name = "name", length = 50, nullable = false)
39  @Size(max = 50, message = "Your name must be less than 50 characters")
40  @NotBlank(message = "Your name is required")
41  private String name;
42
43  @ManyToMany
44  @JoinTable(name = "user_role",
45             joinColumns = @JoinColumn(name = "user_id"),
46             inverseJoinColumns = @JoinColumn(name = "role_id"))
47  private Set<Role> roles = new HashSet<>();
48
49  @OneToMany(mappedBy = "user", cascade = CascadeType.ALL)
50  private List<Book> books = new ArrayList<>();
51  }

```

Hình 7.2. Bổ sung thêm đoạn code nối user và role

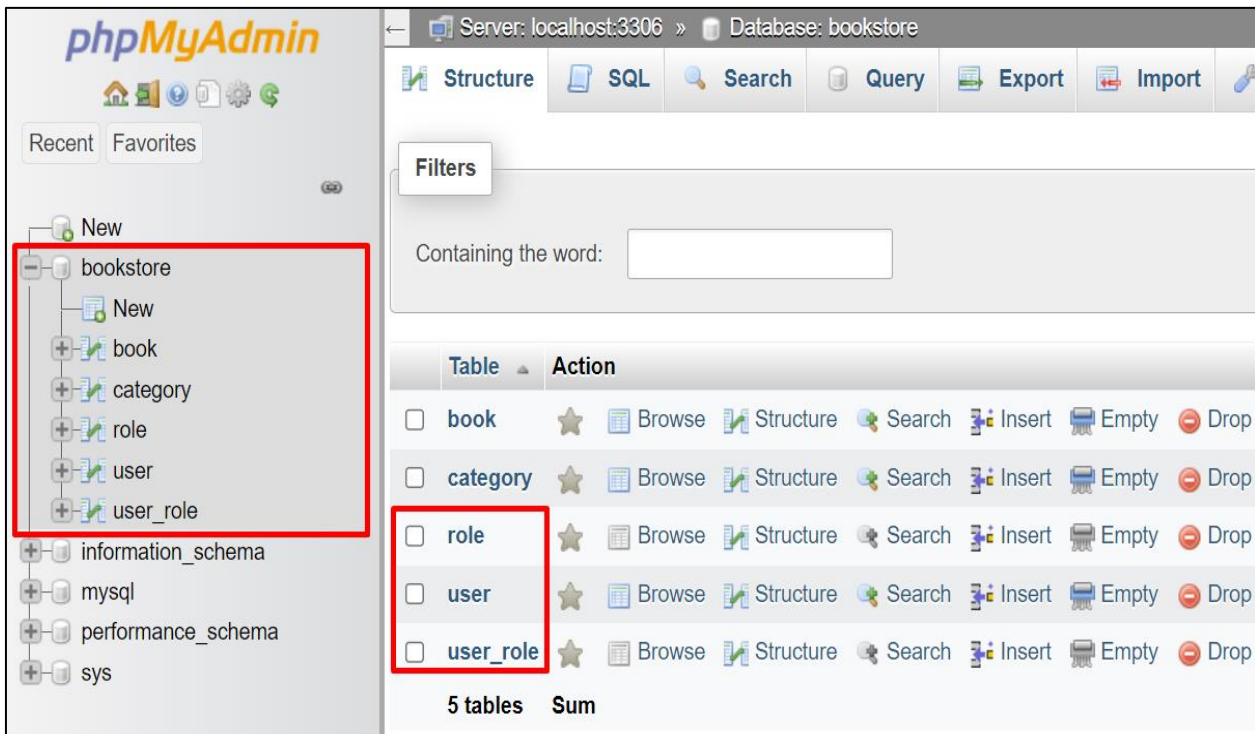
Cụ thể, thuộc tính roles trong đối tượng User được đánh dấu với **@ManyToMany** để biểu thị mối quan hệ n-n với đối tượng Role.

Chú thích **@JoinTable** được sử dụng để chỉ định tên bảng liên kết (join table) để lưu trữ các bản ghi quan hệ giữa các đối tượng User và Role trong cơ sở dữ liệu. Các thuộc tính **joinColumns** và **inverseJoinColumns** được sử dụng để chỉ định tên của các cột khóa ngoại tham chiếu đến bảng User và Role.

Kiểm tra lại các class và interface đã thêm, build lại project (nhấn Ctrl + F9)

Mở **phpMyAdmin** bằng cách truy cập <http://localhost/phpmyadmin/>

Kiểm tra **Database** sẽ xuất hiện thêm table Role đã thêm.



Hình 7.3. Database sẽ xuất table Role và user\_role

User.java lớp Java để xác định thông tin người dùng và quyền truy cập. Thêm role **ADMIN** và **USER** vào bảng **role**

					id	description	name
<input type="checkbox"/>		Edit		Copy		Delete	1 NULL ADMIN
<input type="checkbox"/>		Edit		Copy		Delete	2 NULL USER

Hình 7.4. Thêm dữ liệu vào table role

Tạo 2 tài khoản như bên dưới để thử, ta sẽ gán tài khoản **DEMO1** thành quyền **ADMIN**, tài khoản **DEMO2** là quyền **USER**.

					id	email	name	password	username
<input type="checkbox"/>		Edit		Copy		Delete	1 demo@gmail.com	Nguyễn Văn A	\$2a\$10\$e/7E1/A0oW6GZQDQEO/I6Og87VVK6KFcj/J3Ablo8O... DEMO1
<input type="checkbox"/>		Edit		Copy		Delete	2 demo2@gmail.com	Nguyễn Văn B	\$2a\$10\$JLbb8J.ySbMd6T0mxSM9debRaRtGO1.uyzZH2ftqcbF... DEMO2

☐ Check all    With selected: Edit    Copy    Delete    Export

Hình 7.5. Xem id của user

phpMyAdmin

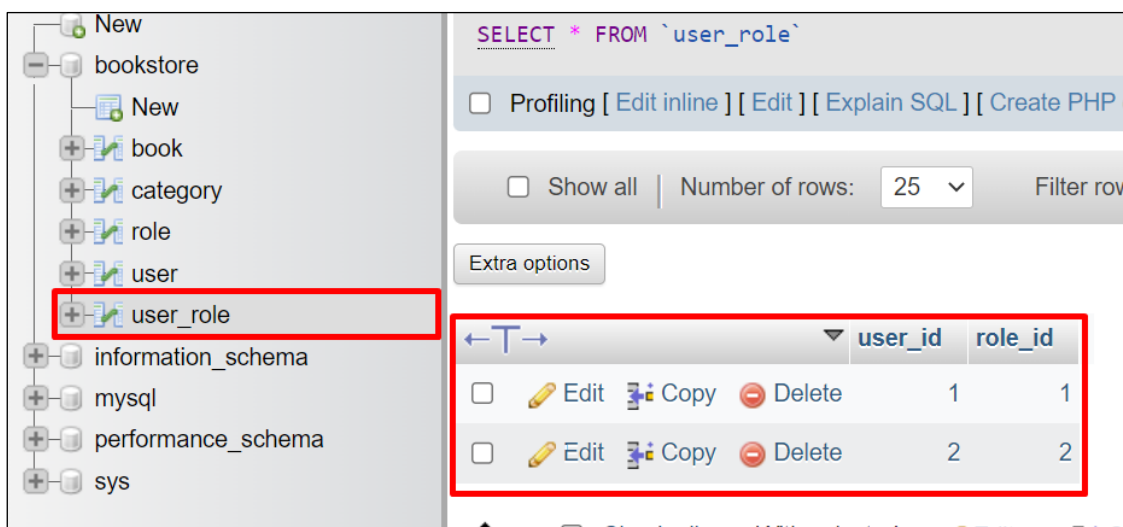
Server: localhost:3306 » Database: bookstore » Table: user\_role

Column	Type	Function	Null	Value
user_id	bigint			<input type="text"/>
role_id	bigint			<input type="text"/>

☐ Ignore

Điền giá trị là id của user và role

Hình 7.6. Điền id của user và role vào bảng user\_role



Hình 7.7. Dữ liệu bảng `user_role` vừa set quyền

Thiết lập mặc định khi tạo tài khoản sẽ có role mặc định là **"USER"**

Tạo file **`IRoleRepository.java`** đặt tại thư mục `repository` theo đường dẫn **`src/main/java/com.example.demo/repository.`**

Khai báo interface **`IRoleRepository`** để truy vấn và quản lý dữ liệu trong cơ sở dữ liệu liên quan đến đối tượng **`Role`**.



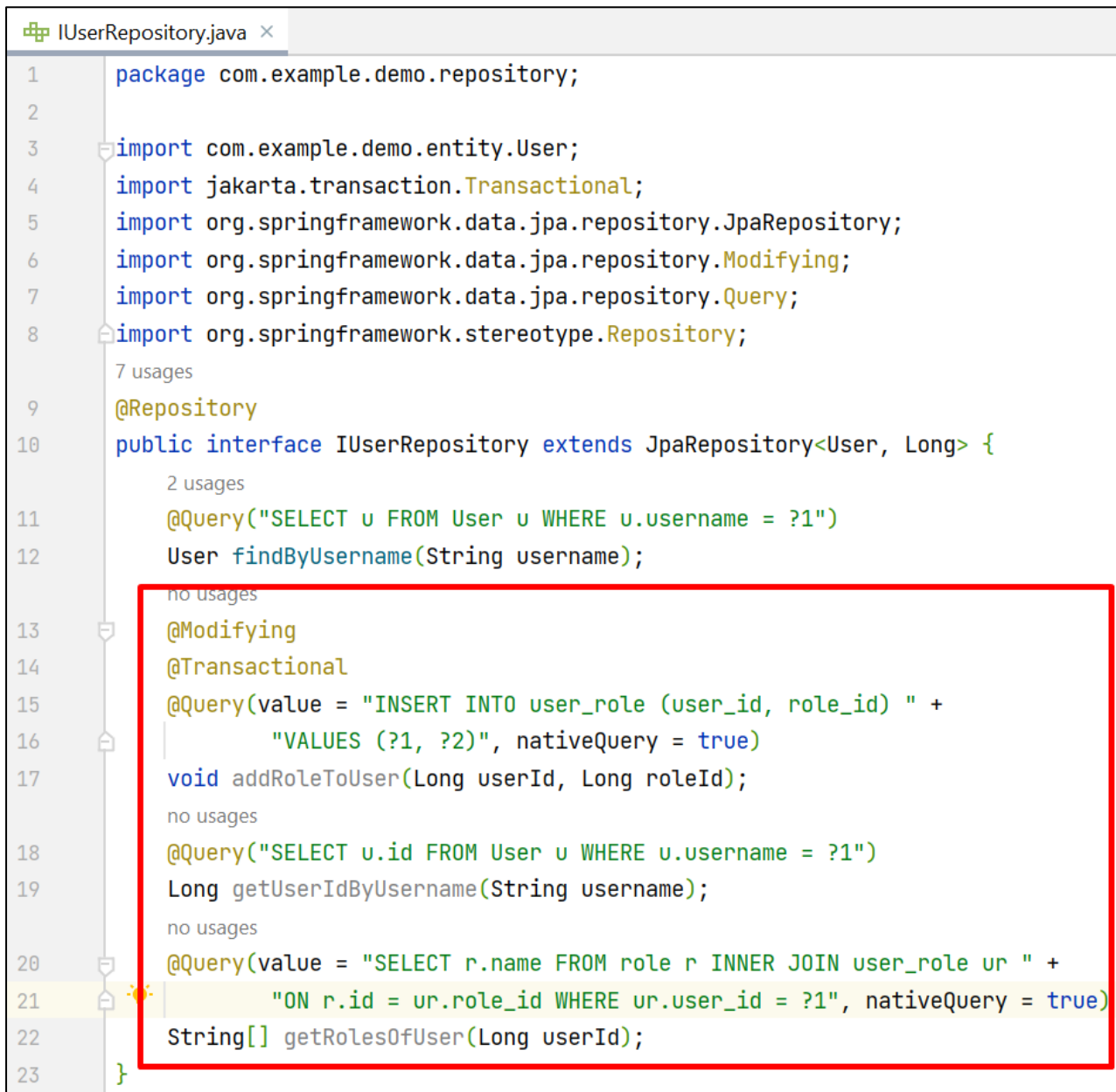
Hình 7.8. Khai báo interface `IRoleRepository`

Tại file **IUserRepository.java** đặt tại thư mục repository theo đường dẫn **src/main/java/com.example.demo/repository** ta bổ sung thêm các phương thức sau:

Phương thức **addRoleToUser** được định nghĩa để thêm một quyền (Role) cho một người dùng (User). Câu truy vấn được sử dụng trong phương thức này là **"INSERT INTO user\_role (user\_id, role\_id) VALUES (?1, ?2)"**. Câu truy vấn này sẽ thêm một bản ghi mới vào bảng user\_role với các giá trị user\_id và role\_id được truyền vào.

Phương thức **getIdByUsername** được định nghĩa để trả về ID của một người dùng dựa trên tên đăng nhập của họ. Câu truy vấn được sử dụng trong phương thức này là **"SELECT u.id FROM User u WHERE u.username = ?1"**. Câu truy vấn này sẽ trả về ID của đối tượng User có tên đăng nhập trùng với tham số đầu vào ?1.

Phương thức **getRolesOfUser** được định nghĩa để trả về tên của các quyền của một người dùng dựa trên ID của họ. Câu truy vấn được sử dụng trong phương thức này là **"SELECT r.name FROM role r INNER JOIN user\_role ur ON r.id = ur.role\_id WHERE ur.user\_id = ?1"**. Câu truy vấn này sẽ trả về một mảng các tên của các đối tượng Role liên quan đến đối tượng User có ID trùng với tham số đầu vào ?1.



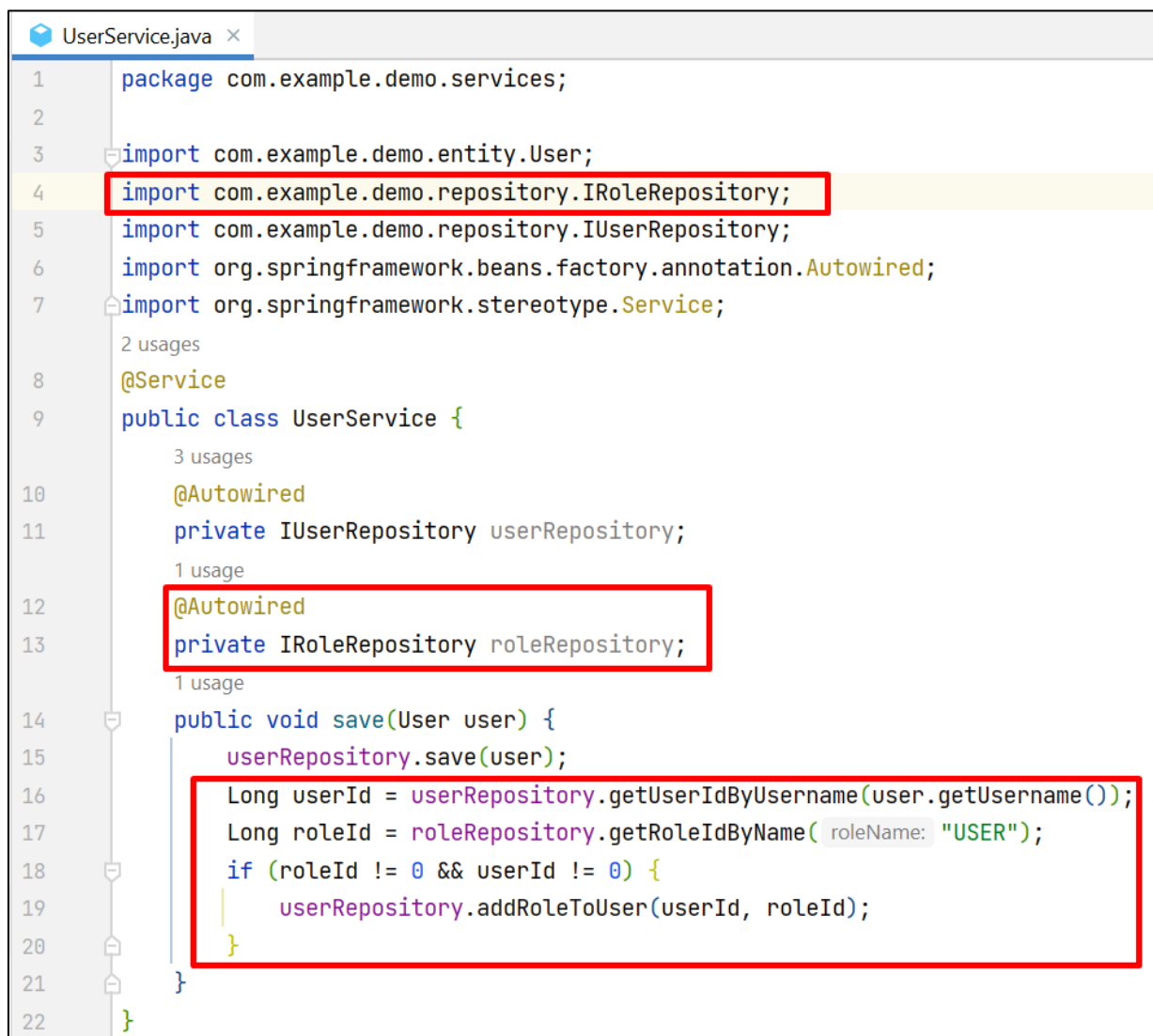
```
1 package com.example.demo.repository;
2
3 import com.example.demo.entity.User;
4 import jakarta.transaction.Transactional;
5 import org.springframework.data.jpa.repository.JpaRepository;
6 import org.springframework.data.jpa.repository.Modifying;
7 import org.springframework.data.jpa.repository.Query;
8 import org.springframework.stereotype.Repository;
9
10 @Repository
11 public interface IUserRepository extends JpaRepository<User, Long> {
12     @Query("SELECT u FROM User u WHERE u.username = ?1")
13     User findByUsername(String username);
14
15     @Modifying
16     @Transactional
17     @Query(value = "INSERT INTO user_role (user_id, role_id) " +
18         "VALUES (?1, ?2)", nativeQuery = true)
19     void addRoleToUser(Long userId, Long roleId);
20
21     @Query("SELECT u.id FROM User u WHERE u.username = ?1")
22     Long getUserIdByUsername(String username);
23
24     @Query(value = "SELECT r.name FROM role r INNER JOIN user_role ur " +
25         "ON r.id = ur.role_id WHERE ur.user_id = ?1", nativeQuery = true)
26     String[] getRolesOfUser(Long userId);
27 }
```

Hình 7.9. Bổ sung thêm các phương thức cho IUserRepository



Tại file **UserService.java** đặt **src/main/java/com/example/demo/services/** ta thay đổi phương thức thêm User như sau:

Bổ sung thêm nội dung như đoạn code bên dưới. Mục đích đoạn code này sử dụng các phương thức được định nghĩa trong các đối tượng **userRepository** và **roleRepository** để thêm một quyền cho một người dùng trong hệ thống.



```
1 package com.example.demo.services;
2
3 import com.example.demo.entity.User;
4 import com.example.demo.repository.IRoleRepository;
5 import com.example.demo.repository.IUserRepository;
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.stereotype.Service;
8
9 @Service
10 public class UserService {
11     @Autowired
12     private IUserRepository userRepository;
13     @Autowired
14     private IRoleRepository roleRepository;
15
16     public void save(User user) {
17         userRepository.save(user);
18         Long userId = userRepository.getUserIdByUsername(user.getUsername());
19         Long roleId = roleRepository.getRoleIdByName("USER");
20         if (roleId != 0 && userId != 0) {
21             userRepository.addRoleToUser(userId, roleId);
22         }
23     }
24 }
```

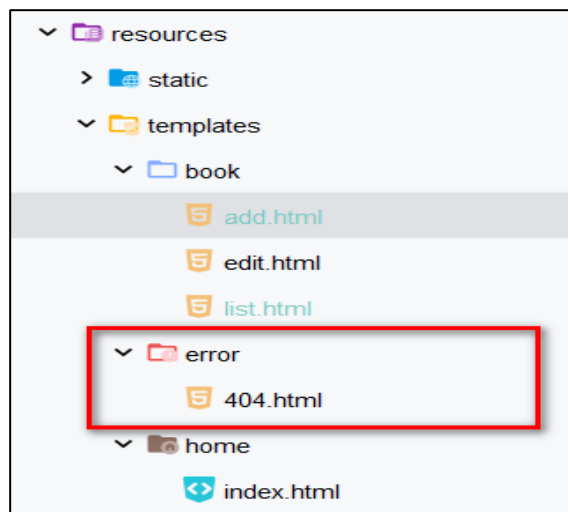
Hình 7.10. Bổ sung đoạn code *UserService.java* để thêm quyền

## 7.2 Tạo thông báo lỗi khi chuyển trang

### ➤ Thiết kế giao diện 404.html

Tạo 1 thư mục mới tên **error** trong thư mục **templates** Tạo file **404.html** tại thư đường dẫn: **src/main/java/com.example.demo/resources/templates/error**

**404.html** là một tệp HTML được sử dụng để hiển thị trang lỗi "404 Not Found". Khi một trình duyệt web yêu cầu một trang không tồn tại trên máy chủ, máy chủ sẽ trả về một mã trạng thái HTTP 404 và trình duyệt web sẽ hiển thị nội dung của tệp 404.html.



Hình 7.11. Thư mục và file 404.html được tạo

```
404.html x
1 <!DOCTYPE html>
2 <html lang="en" xmlns:th="">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>Not Found</title>
8   <th:block th:replace="~{layout :: link-css}"></th:block>
9 </head>
10 <body>
11   <th:block th:replace="~{layout :: header}"></th:block>
12   <div class="d-flex align-items-center justify-content-center vh-25">
13     <div class="text-center">
14       <h1 class="display-1 fw-bold">404</h1>
15       <p class="fs-3"> <span class="text-danger">Oops!</span> Page is not exists.</p>
16       <p class="lead">
17         The page you are looking for might have been removed.
18       </p>
19       <a href="/" class="btn btn-primary">Back to homepage</a>
20     </div>
21   </div>
22   <th:block th:replace="~{layout :: footer}"></th:block>
23 </body>
24 </html>
```

Hình 7.12. File 404.html được tạo ra trong thư mục error

## ➤ Cấu hình lại application.properties

Tìm đến đường dẫn chứa **src/main/resources/application.properties** tiến hành thêm đoạn **server.error.path=/error**.

**server.error.path=/error** là một cấu hình trong tệp application.properties. Khi người dùng truy cập đến một trang không tồn tại trên ứng dụng, hoặc có lỗi xảy ra trong quá trình xử lý yêu cầu của người dùng, Spring Boot sẽ tự động chuyển hướng đến đường dẫn được chỉ định trong cấu hình server.error.path để hiển thị thông báo lỗi cho người dùng.



```
</> application.properties x
1  # Database connection properties
2  spring.datasource.url=jdbc:mysql://localhost:3306/bookstore
3  spring.datasource.username=root
4  spring.datasource.password=
5  spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
6
7  # Hibernate properties
8  spring.jpa.show-sql=true
9  spring.jpa.hibernate.ddl-auto=update
10 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
11 spring.jpa.open-in-view=false
12
13 server.error.path=/error
```

Hình 7.13. Cấu hình lại file application.properties

Tạo file **CustomErrorController.java** đặt tại  
**src/main/java/com.example.demo**

Phương thức **handleError** xác định loại lỗi của yêu cầu bằng cách kiểm tra giá trị của thuộc tính **ERROR\_STATUS\_CODE** trong HttpServletRequest. Nếu lỗi là lỗi 404, nó sẽ trả về tên của tệp HTML hoặc template của trang lỗi 404. Nếu không, nó sẽ trả về null.

```
1 package com.example.demo.controller;
2
3 import jakarta.servlet.RequestDispatcher;
4 import jakarta.servlet.http.HttpServletRequest;
5 import org.springframework.boot.web.servlet.error.ErrorController;
6 import org.springframework.stereotype.Controller;
7 import org.springframework.web.bind.annotation.GetMapping;
8 import org.springframework.web.bind.annotation.RequestMapping;
9
10 import java.util.Optional;
11
12 @Controller
13 @RequestMapping("/error")
14 public class CustomErrorController implements ErrorController {
15     @GetMapping
16     public String handleError(HttpServletRequest request) {
17         return Optional.ofNullable(request.getAttribute(RequestDispatcher.ERROR_STATUS_CODE))
18             .filter(status → Integer.parseInt(status.toString()) == 404)
19             .map(status → "error/404") Optional<String>
20             .orElse( other: null);
21     }
22 }
```

Hình 7.14. Tạo file CustomErrorController.java

Tại **UserRepository.java** ta thêm phương thức lấy ra tất cả các role (quyền truy cập) của user hiện hành

## 7.3 Gán quyền cho User và Role

Tại tệp **CustomUserDetails.java** ta thêm và thay đổi các phương thức sau:

```
8  import java.util.Arrays;
9  import java.util.Collection;
10 import java.util.stream.Collectors;
11
12 2 usages  Nguyễn Xuân Nhân *
12 public class CustomUserDetail implements UserDetails {
13     4 usages
13     private final User user;
14     2 usages
14     private final IUserRepository userRepository;
15
16     1 usage  Nguyễn Xuân Nhân
16 @ public CustomUserDetail(User user, IUserRepository userRepository) {
17     this.user = user;
18     this.userRepository = userRepository;
19 }
20
21 no usages  Nguyễn Xuân Nhân *
21 @Override
22 public Collection<? extends GrantedAuthority> getAuthorities() {
23     return Arrays.stream(userRepository.getRolesOfUser(user.getId())) Stream<String>
24         .map(SimpleGrantedAuthority::new) Stream<SimpleGrantedAuthority>
25         .collect(Collectors.toSet());
26 }
27
```

Hình 7.15. Thay đổi các phương thức trong CustomUserDetail

Tại tệp **CustomUserDetailService.java** ta thay đổi các phương thức sau:



Hình 7.16. Truyền `userRepository` vào `CustomUserDetail`

Lý do phải truyền **userRepository** vào **CustomUserDetail** là để đảm bảo rằng `CustomUserDetail` có thể sử dụng `userRepository` để thực hiện các truy vấn tùy ý khác vào cơ sở dữ liệu.

Tại tệp **SecurityConfig.java** ta tiến hành phân quyền truy cập resource tại **SecurityFilterChain**.

Resource	Quyền
/books/edit", "/books/delete"	ADMIN
"/books", "/books/add"	ADMIN, USER

Hình 7.17. Phân quyền theo admin và user



Hình 7.18. Tiến hành phân quyền truy cập resource tại SecurityFilterChain

## 7.4 Điều chỉnh phân quyền tại giao diện list.html

Tiến hành phân quyền tại file **list.html** tại:

**src/main/java/com.example.demo/resources/templates/book**



```
list.html x
1 <!DOCTYPE html>
2 <html
3     xmlns:th="http://www.thymeleaf.org"
4     xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity4"
5     lang="en">
6 <head>
7     <meta charset="UTF-8">
8     <title>My Book List</title>
9     <th:block th:replace="~{layout :: link-css}"></th:block>
10 </head>
11 <body>
12 <th:block th:replace="~{layout :: header}"></th:block>
13 <div class="container">
14     <table class="table">
15         <tr>
16             <th>ID</th>
17             <th>Title</th>
18             <th>Author</th>
19             <th>Price</th>
20             <th>Category</th>
21             <th sec:authorize="hasAnyAuthority('AD')">Action</th>
22         </tr>
23         <tr th:each="book : ${books}">
24             <td th:text="${book.id}"></td>
25             <td th:text="${book.title}"></td>
26             <td th:text="${book.author}"></td>
27             <td th:text="${book.price}"></td>
28             <td th:text="${book.category != null ? book.category.name : 'N/A'}"></td>
29             <td sec:authorize="hasAnyAuthority('AD')">
30                 <a th:href="@{/books/edit(id=__${book.id}__)}" class="text-info">Edit</a> |
31                 <a th:href="@{/books/delete(id=__${book.id}__)}"
32                     onclick="deleteBook(this); return false;" class="text-danger">Delete</a>
33             </td>
34         </tr>
35     </table>
36 </div>
```

Hình 7.19. Phân quyền tại file list.html

```

37 <script th:src="@{/js/jquery-3.5.1.min.js}"></script>
38 <script th:inline="javascript">
    1 usage  🐞 Nguyễn Xuân Nhân
39     function deleteBook(link) {
40         if (confirm('Are you sure?')) {
41             $.ajax({
42                 url: $(link).attr('href'),
43                 type: 'DELETE',
44                 success: result => {
45                     if (!result.success) {
46                         alert(result.message);
47                     } else {
48                         $(link).parent().parent().remove();
49                     }
50                 }
51             });
52         }
53     }
54 </script>
55 <th:block th:replace="~{layout :: footer}"></th:block>
56 </body>
57 </html>

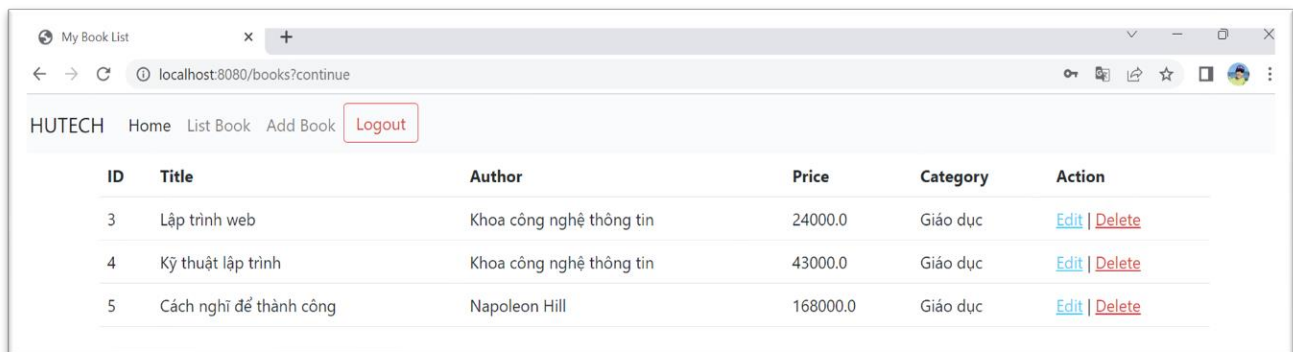
```

} Xóa bằng Ajax

Hình 7.20. Xóa sách bằng Ajax

## 7.5 Kiểm tra kết quả phân quyền

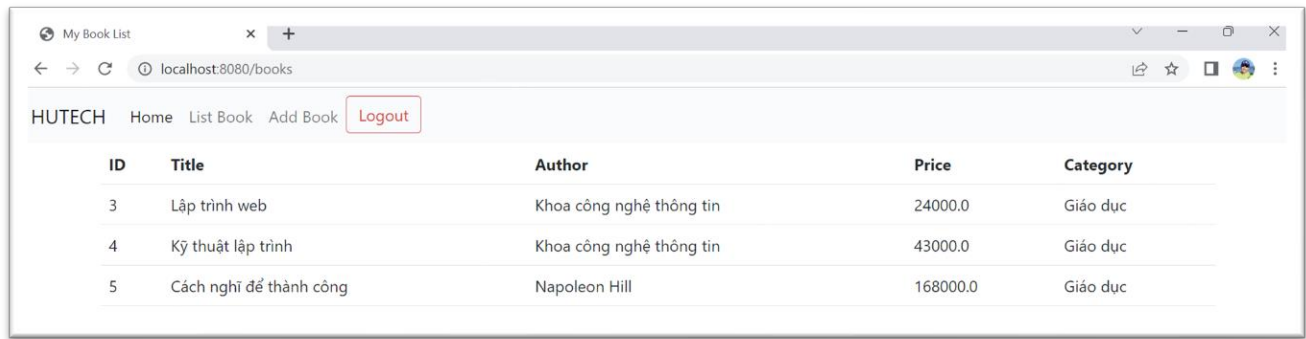
Các bạn kiểm tra bằng cách, đăng nhập bằng tài khoản **admin** sẽ xuất hiện cột **action** có thêm chức năng **Edit và Delete**. Như ảnh bên dưới.



ID	Title	Author	Price	Category	Action
3	Lập trình web	Khoa công nghệ thông tin	24000.0	Giáo dục	<a href="#">Edit</a>   <a href="#">Delete</a>
4	Kỹ thuật lập trình	Khoa công nghệ thông tin	43000.0	Giáo dục	<a href="#">Edit</a>   <a href="#">Delete</a>
5	Cách nghĩ để thành công	Napoleon Hill	168000.0	Giáo dục	<a href="#">Edit</a>   <a href="#">Delete</a>

Hình 7.21. Giao diện xem với quyền ADMIN

Đăng nhập bằng tài khoản user bình thường thì chỉ có thể xem List Book và Add Book.



ID	Title	Author	Price	Category
3	Lập trình web	Khoa công nghệ thông tin	24000.0	Giáo dục
4	Kỹ thuật lập trình	Khoa công nghệ thông tin	43000.0	Giáo dục
5	Cách nghĩ để thành công	Napoleon Hill	168000.0	Giáo dục

Hình 7.22. Giao diện xem với quyền USER

## 7.6 Yêu cầu hoàn thành bài tập bổ sung sau

- ✓ Dựa vào hàm xoá sách bằng Ajax như trên, hãy thực hiện các thao tác như thêm, sửa và lấy dữ liệu bằng Ajax
- ✓ Thử thay đổi phương pháp mã hoá mật khẩu thành Pbkdf2PasswordEncoder và Argon2PasswordEncoder. Nhận xét kết quả?
- ✓ **Nâng cao:** Thực hiện phân quyền theo phương pháp Policy-based

## TÓM TẮT

Sau khi học xong phân quyền 2 quyền user và admin trong Java Spring Boot, bạn sẽ có khả năng triển khai một hệ thống phân quyền bảo mật và an toàn trong ứng dụng web, đồng thời hiểu rõ vai trò và trách nhiệm của người dùng và quản trị viên trong việc sử dụng ứng dụng web.

## CÂU HỎI ÔN TẬP

1. Phân quyền trong lập trình ứng dụng web là gì?
2. Spring Security là gì và nó được sử dụng để làm gì trong phân quyền?
3. Tại sao phân quyền là thành phần quan trọng trong việc bảo mật ứng dụng web?