

BỘ GIÁO DỤC VÀ ĐÀO TẠO
ĐẠI HỌC CÔNG NGHỆ TP.HCM

THỰC HÀNH
LẬP TRÌNH ỨNG DỤNG JAVA

Biên soạn:	Nguyễn Đình Ánh Nguyễn Huy Cường Trần Đăng Khoa Nguyễn Xuân Nhân
-------------------	---

Website: <http://www.hutech.edu.vn>

THỰC HÀNH LẬP TRÌNH ỨNG DỤNG JAVA

Ấn bản 2023

MỤC LỤC

MỤC LỤC	I
MỤC LỤC HÌNH ẢNH	V
HƯỚNG DẪN	XI
BÀI 1 BÀI KHỞI ĐỘNG: LẬP TRÌNH JAVA CORE	1
1.1 Giới thiệu và nội dung mục tiêu bài học.....	1
1.2 Bài tập	3
1.2.1. Khởi tạo Project Console.....	3
1.2.2. Bài tập 1: Viết chương trình menu cho phép quản lý sách	5
1.2.3. Bài tập 2: Xây dựng chương trình kiểm định xe.....	10
1.3 Bài tập làm thêm	13
TÓM TẮT	15
CÂU HỎI ÔN TẬP	15
BÀI 2 LÀM QUEN VỚI SPRING FRAMEWORK.....	16
2.1 Cơ bản về Spring Framework và các cài đặt cần thiết.....	16
2.1.1 Yêu cầu.....	16
2.2 Hướng Dẫn Thiết Lập phpMyAdmin	17
2.3 Download Template Spring từ spring.io	22
2.4 Chạy trang web đầu tiên	32
2.5 Yêu cầu hoàn thành bài tập bổ sung sau	38
TÓM TẮT	39
CÂU HỎI ÔN TẬP	39
BÀI 3 VIẾT CHƯƠNG TRÌNH QUẢN LÝ SÁCH	40

3.1 Xây dựng trang Book List	41
3.1.1 Tạo view layout trong ứng dụng	44
3.2 Xây dựng trang thêm sách	49
3.2.1 Xây dựng phương thức thêm sách.....	49
3.3 Thêm View hiển thị	50
3.4 Xây dựng trang sửa sách	54
3.5 Thực hiện chức năng xóa sách	57
3.6 Hướng dẫn lấy và đưa mã nguồn lên Github.....	58
TÓM TẮT	63
CÂU HỎI ÔN TẬP	63
BÀI 4 THAO TÁC VỚI CƠ SỞ DỮ LIỆU	64
4.1 Khởi động phpMyAdmin – Kết nối CSDL	65
4.2 Khởi tạo các đối tượng	67
4.3 Khởi tạo các thành phần giao diện.....	73
4.4 Build và chạy ứng dụng	78
4.5 Xây dựng giỏ hàng	80
TÓM TẮT	91
CÂU HỎI ÔN TẬP	91
BÀI 5 XÂY DỰNG CÁC CHỨC NĂNG THÊM, XOÁ, SỬA	92
5.1 Viết trang thêm sách – Add Book	93
4.1.1. Thêm các phương thức thêm sách vào Controller	94
5.2 Thêm ràng buộc khi lưu sách.....	100
5.3 Chức năng Edit và delete Book.....	110
5.3.1 Delete Book.....	110

5.3.2 Edit Book	111
5.4 Xây dựng chức năng tìm kiếm theo từ khoá	113
5.5 Xây dựng chức năng Checkout.....	115
TÓM TẮT	124
CÂU HỎI ÔN TẬP	124
BÀI 6 XÂY DỰNG CHỨC NĂNG XÁC THỰC NGƯỜI DÙNG	125
6.1 Xây dựng thêm table mới là User	126
6.2 Thiết kế giao diện đăng nhập và đăng ký	138
6.3 Xây dựng chức năng xác thực bằng OAuth2.....	146
TÓM TẮT	158
CÂU HỎI ÔN TẬP	158
BÀI 7 XÂY DỰNG CHỨC NĂNG PHÂN QUYỀN NGƯỜI DÙNG	159
7.1 Thêm table và dữ liệu vào table Role trong CSDL.....	160
7.2 Tạo thông báo lỗi khi chuyển trang.....	166
7.3 Gán quyền cho User và Role.....	171
7.4 Điều chỉnh phân quyền tại giao diện list.html và layout.html	174
7.5 Kiểm tra kết quả phân quyền.....	179
7.6 Thiết lập bảo mật bằng JWT	Error! Bookmark not defined.
TÓM TẮT	181
CÂU HỎI ÔN TẬP	181
BÀI 8 LÀM QUEN VỚI API VÀ SOCKET	182
8.1 Thêm authorize cho API	183
8.2 Tạo package models chứa các Data transfer object	183
8.3 Tạo ApiController	186

8.4	Chỉnh sửa List.html cho phù hợp	190
8.5	Thực hiện	192
TÓM TẮT	193	
CÂU HỎI ÔN TẬP	193	
TÀI LIỆU THAM KHẢO	194	
PHỤ LỤC CÀI ĐẶT MÔI TRƯỜNG VÀ CÔNG CỤ CẦN THIẾT	195	
Cài đặt Java JDK (Java Development Kit)	195	
Cài đặt Laragon (Quản lý CSDL MySQL)	197	
Cài đặt IntelliJ IDEA	200	

MỤC LỤC HÌNH ẢNH

Hình 1.1. Khởi tạo project console	3
Hình 1.2. Thiết lập cài đặt New project.....	3
Hình 1.3. Download JDK new	4
Hình 1.4. Code file main.java project	4
Hình 1.5. Run chương trình và xem kết quả	5
Hình 1.6. Sử dụng công cụ sinh code	6
Hình 1.7. Sử dụng Tool để Generation: Getter, Setter, Constructor	7
Hình 1.8. Sơ đồ lớp mô tả quan hệ giữa các đối tượng	12
Hình 2.1. Khởi động ứng dụng Laragon	17
Hình 2.2. Cấu hình MySQL, Nginx cho Laragon	18
Hình 2.3. Cài đặt chọn phpMyAdmin 1.....	18
Hình 2.4. Cài đặt chọn phpMyAdmin 2.....	19
Hình 2.5. Quá trình Quick add "phpMyAdmin"	19
Hình 2.6. Khởi động dịch vụ PhpMyAdmin	20
Hình 2.7. Giao diện đăng nhập phpMyAdmin	21
Hình 2.8. Giao diện trang Index PhpMyAdmin.....	21
Hình 2.9. Open Spring Framework tại spring.io.....	22
Hình 2.10. Khởi động Quickstart Your Project	22
Hình 2.11. Lựa chọn cấu hình cho spring initializr	23
Hình 2.12. Tiến hành thêm dependencies vào dự án.....	24
Hình 2.13. Thêm các Dependencies trong spring.io	24
Hình 2.14. Generate dự án Spring Boot	25

Hình 2.15. Tệp tin dự án được generate từ Spring Initializr.....	25
Hình 2.16. Giải nén và mở Spring Template bằng IntelliJ IDEA.	26
Hình 2.17. Thiết lập tin cậy thư mục chứa dự án.....	26
Hình 2.18. Bước cài đặt các Dependencies cần thiết	27
Hình 2.19. Lựa chọn Project Structure.....	27
Hình 2.20. Lựa chọn phiên bản JDK	28
Hình 2.21. Lựa chọn phiên bản JDK - 20.....	28
Hình 2.22. Bổ sung đoạn code cấu hình cho application.properties	29
Hình 2.23. Build project chương trình.....	29
Hình 2.24. Tiến hành chạy ứng dụng.....	30
Hình 2.25. Run chương trình tại file Application	30
Hình 2.26. Giao diện mặc định đăng nhập.....	31
Hình 2.27. Tạo package controller	32
Hình 2.28.Tạo file HomeController.java trong package controllers	33
Hình 2.29. Tạo thư mục home.....	34
Hình 2.30. Hiển thị ngay phía dưới thư mục templates.....	35
Hình 2.31. Tạo file index.html	36
Hình 2.32. Giao diện trang chủ.....	37
Hình 3.1. Thêm 1 Package với tên entities	41
Hình 3.2. Tải thư viện Bootstrap	44
Hình 3.3. Thêm thư viện Bootstrap vào dự án	45
Hình 3.4. Thêm jquery-3.7.0.min.js	45
Hình 3.5. Kết quả, trang Book List.....	48
Hình 3.6. Giao diện trang thêm book	52

Hình 3.7. Thêm 1 cuốn sách	53
Hình 3.8. Trang danh sách Book sau khi thêm Book mới	53
Hình 3.9. Giao diện danh sách các cuốn với nút Edit.....	56
Hình 3.10. Giao diện trang chỉnh sửa Book	56
Hình 3.11. Kết quả sau khi chỉnh sửa thành công, trả về Book List	57
Hình 3.12.Giao diện delete Book	58
Hình 3.13. Chia sẻ mã nguồn lên Github.....	58
Hình 3.14. Nhập các thông tin cơ bản của kho lưu trữ.....	59
Hình 3.15: Kết nối IntelliJ IDEA với Github.....	59
Hình 3.16. Đẩy mã nguồn lên Github	60
Hình 3.17. Kết quả sau khi đưa mã nguồn lên Github	61
Hình 3.18. Lấy mã nguồn từ Github về máy tính.....	61
Hình 3.19. Nhập thông tin kho lưu trữ cần lấy mã nguồn	62
Hình 3.20. Quá trình lấy mã nguồn	62
Hình 4.1. Khởi động phpMyAdmin.....	65
Hình 4.2. Tạo mới CSDL trong phpMyAdmin	66
Hình 4.3. Đặt tên cho CSDL trong phpMyAdmin mới và tạo	66
Hình 4.4. Tạo 2 file Book.java và Category.java trong thư mục entity	67
Hình 4.5. Thư mục css và js.....	75
Hình 4.6. Build và chạy ứng dụng	78
Hình 4.7. Add database bookstore	79
Hình 4.8. Thêm dữ liệu cho bảng Category	79
Hình 4.9. Thêm dữ liệu cho bảng Book	80
Hình 4.10. Truy cập trang danh sách	80

VIII MỤC LỤC

Hình 4.11. Khởi tạo nút thêm vào giỏ hàng	81
Hình 4.12. Khởi tạo Item và Card	82
Hình 4.13. Thêm tab Cart.....	87
Hình 4.14. Trang mặc định khi chưa có sản phẩm nào thêm vào giỏ hàng .	90
Hình 4.15. Thêm một vài sản phẩm vào giỏ hàng	90
Hình 4.16. Kết quả sau khi thêm vào giỏ hàng	90
Hình 5.1. Import service vào controller	97
Hình 5.2. Inject service vào controller.....	97
<i>Hình 5.3. Giao diện add Book.....</i>	99
<i>Hình 5.4. Danh sách Book sau khi Add Book mới</i>	99
<i>Hình 5.5. Thêm ràng buộc vào file pom.xml</i>	100
<i>Hình 5.6. Cập nhật maven.....</i>	101
<i>Hình 5.7. Reload project Maven</i>	102
Hình 5.8. Cấu trúc thư mục Validator	104
Hình 5.9. Kiểm tra ràng buộc.....	110
Hình 5.10. Kết quả tìm kiếm bằng từ khoá.....	115
Hình 5.11. Diagram sau khi tạo ra item và invoice	120
Hình 6.1. Thông báo lỗi chưa định nghĩa lớp ValidUsername	129
Hình 6.2. Kiểm tra database sau khi thêm table User	132
Hình 6.3. Mở Designer và xem kết quả.....	133
Hình 6.4. Sơ đồ Designer của database bookstore.....	133
Hình 6.5. Thêm 2 file login.html và register.html	138
Hình 6.6. Giao diện web chưa đăng nhập.....	143
Hình 6.7. Tạo tài khoản mới.....	144

Hình 6.8. Kiểm tra valid dữ liệu khi không điền dữ liệu	144
Hình 6.9. Đăng ký thử tài khoản mẫu.....	145
Hình 6.10. Đăng nhập bằng tài khoản vừa tạo.....	145
Hình 6.11. Đăng nhập thành công, xuất hiện nút Logout.....	146
Hình 6.12. Kiểm tra thông tin tài khoản đã có trong Database	146
Hình 6.13. Cách hoạt động của OAuth2	146
Hình 6.14. Thiết lập Provider cho Oauth2.....	148
Hình 7.1. Database sẽ xuất table Role và user_role.....	162
Hình 7.2. Thêm dữ liệu vào table role.....	162
Hình 7.3. Xem id của user	163
Hình 7.4. Điền id của user và role vào bảng user_role	163
Hình 7.5. Dữ liệu bảng user_role vừa set quyền	163
Hình 7.6. Thư mục và file 404.html được tạo.....	167
Hình 7.7. Phân quyền theo admin và user.....	171
Hình 7.8. Tiến hành phân quyền truy cập resource tại SecurityFilterChain	174
Hình 7.9. Phân quyền tại file list.html.....	Error! Bookmark not defined.
Hình 7.10. Giao diện xem với quyền ADMIN	179
Hình 7.11. Giao diện xem với quyền USER	180
Hình 8.1. Thêm quyền authorize cho API	183
Hình 8.2. Tạo lớp BookDto.....	184
Hình 8.3. Chỉ định Restful Api và thiết lập CORS	Error! Bookmark not defined.
Hình 8.4. ApiController trong thư mục controller thêm các method	Error! Bookmark not defined.
Hình 8.5. Gọi api lấy danh sách books.....	188

Hình 8.6. Gọi api lấy thông tin sách dựa vào id.....	188
Hình 8.7. Thêm extend xem Json trên trình duyệt.....	189
Hình 8.8. Tuỳ chỉnh lại giao diện list.html cho phù hợp	190
Hình 8.9. Các hàm lấy danh sách books và Delete book.....	191
Hình 8.10. Giao diện hiển thị danh sách Books bằng API.....	191
Hình 8.11. Giao diện xoá sách bằng API	192
Hình 0.1. Cài đặt Java JDK X64 Installer	195
Hình 0.2. Install Java SE to path	196
Hình 0.3. Successfully installed Java JDK.....	196
Hình 0.4. Download Edition Laragon - full	197
Hình 0.5. Setup Laragon path	198
Hình 0.6. Setup Laragon runs extremely.....	198
Hình 0.7. Ready to Install Laragon.....	199
Hình 0.8. Completing the Laragon Setup	200
Hình 0.9. Download IntelliJ IDEA Community Edition.....	201
Hình 0.10.Next Setup IntelliJ IDEA	201
Hình 0.11. Installation Options IntelliJ IDEA	202
Hình 0.12. Completing IntelliJ IDEA Community	202
Hình 0.13. IntelliJ IDEA use agreement	203
Hình 0.14. Giao diện bắt đầu của IntelliJ IDEA.....	203

HƯỚNG DẪN

MÔ TẢ MÔN HỌC

Học phần này bao gồm hai nội dung chính. Nội dung đầu tiên là kiến thức tổng quan về lập trình ứng dụng bằng ngôn ngữ Java. Nội dung thứ hai tập trung vào cách phát triển ứng dụng website bằng việc sử dụng Spring Framework.

NỘI DUNG MÔN HỌC

- **Bài 01:** BÀI KHỞI ĐỘNG: LẬP TRÌNH JAVA CORE
- **Bài 02:** LÀM QUEN VỚI SPRING FRAMEWORK
- **Bài 03:** VIẾT CHƯƠNG TRÌNH QUẢN LÝ SÁCH
- **Bài 04:** THAO TÁC VỚI CƠ SỞ DỮ LIỆU
- **Bài 05:** XÂY DỰNG CÁC CHỨC NĂNG THÊM, XOÁ, SỬA
- **Bài 06:** XÂY DỰNG CHỨC NĂNG XÁC THỰC NGƯỜI DÙNG
- **Bài 07:** XÂY DỰNG CHỨC NĂNG PHÂN QUYỀN NGƯỜI DÙNG
- **Bài 08:** LÀM QUEN VỚI API VÀ SOCKET

KIẾN THỨC TIỀN ĐỀ

Môn học thực hành lập trình ứng dụng Java yêu cầu sinh viên có kiến thức về cơ sở dữ liệu, kỹ thuật lập trình, lập trình hướng đối tượng, lập trình web và tiếp xúc nhiều với môi trường internet.

YÊU CẦU MÔN HỌC

Người học cần đi học đầy đủ, đọc các nội dung sẽ được học trước khi đến lớp, làm các bài tập về nhà và đảm bảo thời gian tự học ở nhà.

CÁCH TIẾP NHẬN NỘI DUNG MÔN HỌC

Để học tốt môn này, sinh viên cần thực hiện tất cả các bài thực hành, ôn tập các bài đã học, trả lời các câu hỏi và làm đầy đủ bài tập; đọc trước bài mới và tìm thêm các thông tin liên quan đến bài học.

Đối với mỗi bài học, sinh viên đọc trước mục tiêu và tóm tắt bài học, sau đó đọc nội dung bài học. Kết thúc mỗi ý của bài học, sinh viên trả lời câu hỏi ôn tập và kết thúc toàn bộ bài học, sinh viên làm các bài tập.

PHƯƠNG PHÁP ĐÁNH GIÁ MÔN HỌC

Môn học được đánh giá như sau:

- Điểm thực hành (100%): Hình thức thi thực hành, phù hợp với quy chế đào tạo và tình hình thực tế tại nơi tổ chức học tập.

BÀI 1 BÀI KHỞI ĐỘNG: LẬP TRÌNH JAVA CORE

1.1 Giới thiệu và nội dung mục tiêu bài học

- ✓ Hướng dẫn sinh viên ôn tập lập trình với ngôn ngữ Java bằng cách viết các chương trình trong môi trường Eclipse / Netbean / IntelliJ IDEA (community).
- ✓ Xây dựng các lớp, khởi tạo đối tượng, truy xuất các phương thức...
- ✓ Tìm hiểu về cách sử dụng thư viện trong các phiên bản Java JDK trước JDK 20.
 - **Lambda Expressions & phương thức foreach.**

Biểu thức **lambda** được xác định bằng cách đặt các tham số đầu vào (nếu có) bên trái của toán tử lambda "`->`" và đặt biểu thức hoặc khối câu lệnh bên phải của toán tử lambda.

Ví dụ: Một biểu thức lambda `(a, b) -> a + b` cho biết biểu thức lambda nhận hai đối số a và b và trả về tổng của a và b.

- **Phương thức tham chiếu (Method References):** cho phép truyền một tham chiếu của một phương thức hoặc constructor bằng cách sử dụng từ khóa "`::`".

Phương thức tham chiếu là một cú pháp viết tắt của biểu thức **Lambda** để gọi phương thức.

Ví dụ: Nếu biểu thức Lamda được viết như sau: `p -> System.out.println(p)`
Cách viết bằng phương thức tham chiếu sẽ là: `System.out::println`.

- Stream API

Stream (luồng) là một đối tượng mới trong Java giúp thực hiện các thao tác trên collection và array một cách dễ dàng và hiệu quả hơn. Dưới đây là một số phương thức phổ biến thường được sử dụng:

- Tạo một Stream.
- Chuyển đổi một Stream sang Collection.
- Sử dụng Stream.filter để lọc dữ liệu.
- Sử dụng Stream.allMatch(), Stream.anyMatch() và Stream.noneMatch() để kiểm tra điều kiện trong Stream.
- Sử dụng Stream.findAny() và Stream.findFirst() để tìm kiếm phần tử trong Stream.
- Sử dụng Stream.distinct() để loại bỏ các phần tử trùng lặp trong Stream.
- Sử dụng Stream.map() để ánh xạ và biến đổi dữ liệu trong Stream.
- Stream.max() và Stream.min() để tìm giá trị lớn nhất và nhỏ nhất.

 **Tham khảo:**

<https://docs.oracle.com/javase/10/docs/api/java/util/stream/Stream.html>

- Switch expression.
- Text block.
- Record.
- Sealed class.
- Pattern matching.

 **Ngôn ngữ lập trình Java**

- Kiểu dữ liệu nguyên thủy: `byte`, `short`, `Integer`, `long`, `float`, `double`, `char`, `boolean`
 - Kiểu dữ liệu không nguyên thủy: `var`, `String`, `List<T>`, `Collection<T>`, `List`, `ArrayList`, `ArrayDeque`...

- Ghi ra màn hình

```
System.out.print("Giá trị cần ghi ra màn hình");
System.out.println("Giá trị cần ghi và xuống dòng");
```

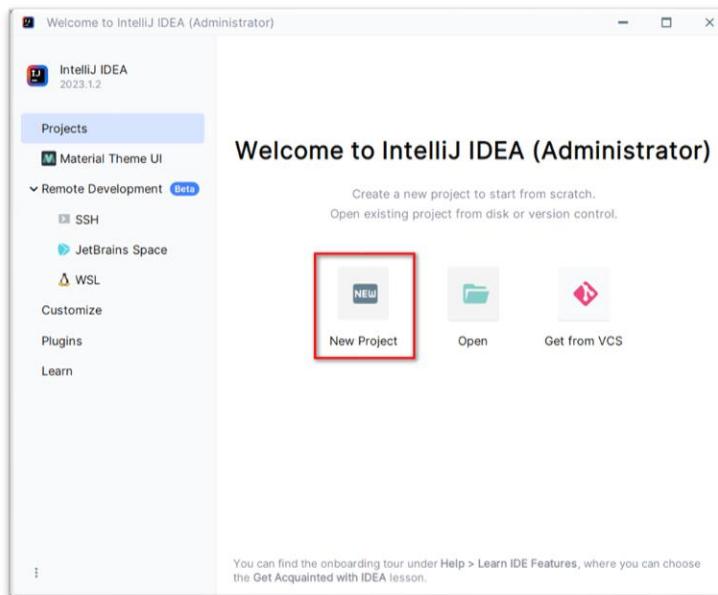
- Lớp Java User Input (Scanner)

```
Scanner x = new Scanner(System.in);
String gia_tri = x.nextLine();
```

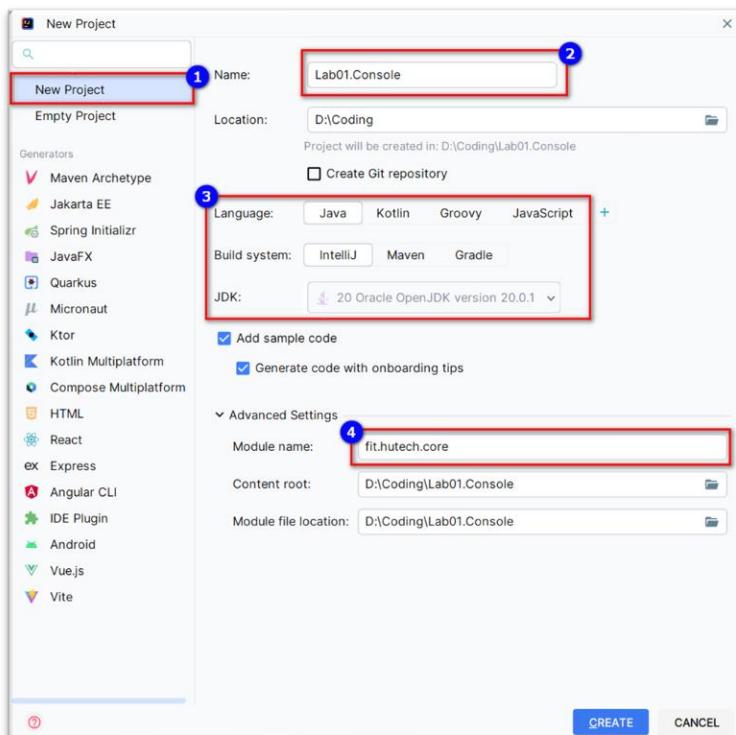
1.2 Bài tập

1.2.1. Khởi tạo Project Console

Mở phần mềm IntelliJ IDEA, Chọn Project.



Hình 1.1. Khởi tạo project console

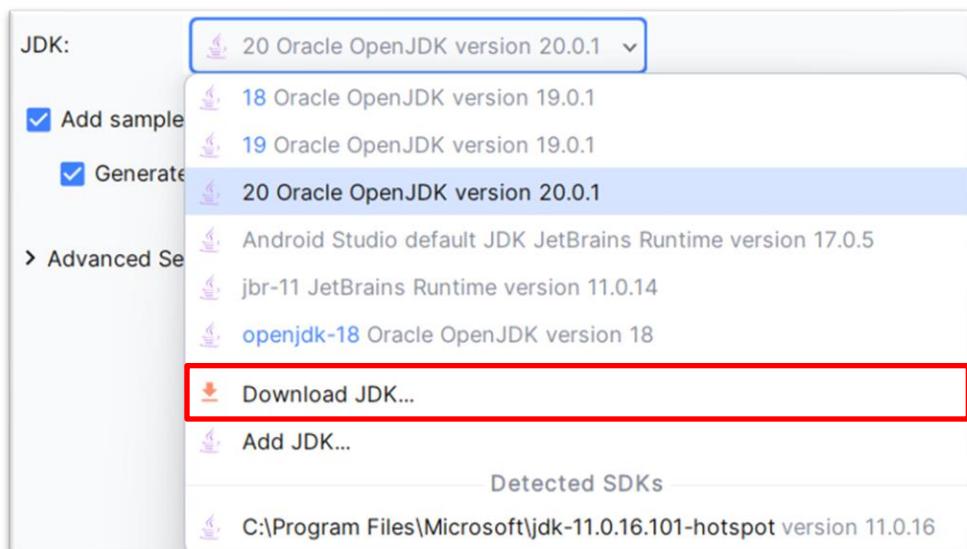


Hình 1.2. Thiết lập cài đặt New project

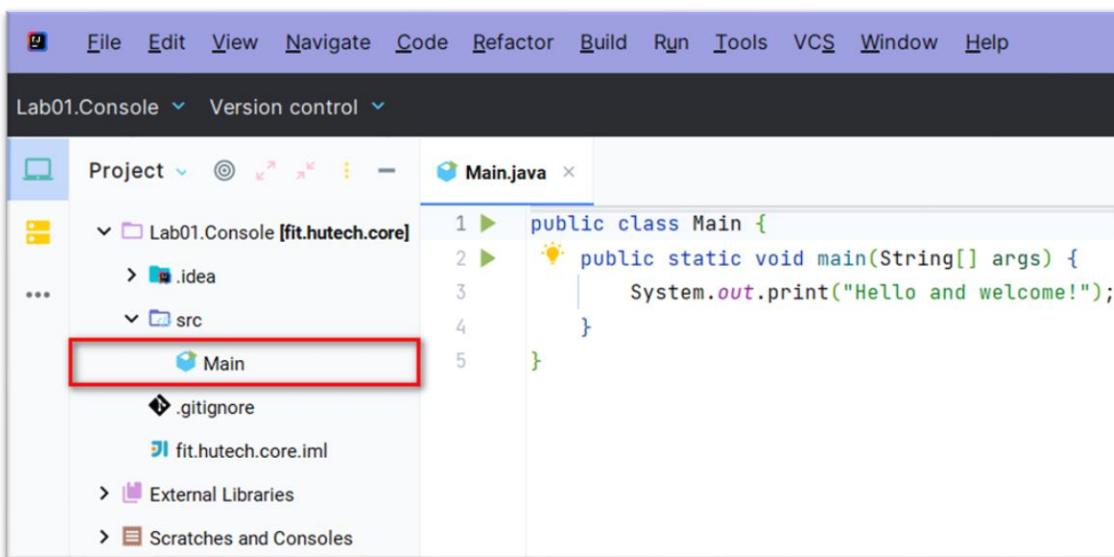
Dựa vào hình ảnh trên, hãy chọn "**New Project**" để bắt đầu khởi tạo một Project mới. Đặt tên cho dự án, chọn vị trí lưu trữ, chọn ngôn ngữ lập trình là **Java** và hệ thống xây dựng (build system) là **IntelliJ**. Chọn phiên bản **JDK 20** (tại thời điểm thực hiện bài thực hành).

Sau đó nhấn **Create** để khởi tạo Project.

Lưu ý: Nếu không có JDK tương ứng, sinh viên có thể tải JDK từ phần **Download JDK**.



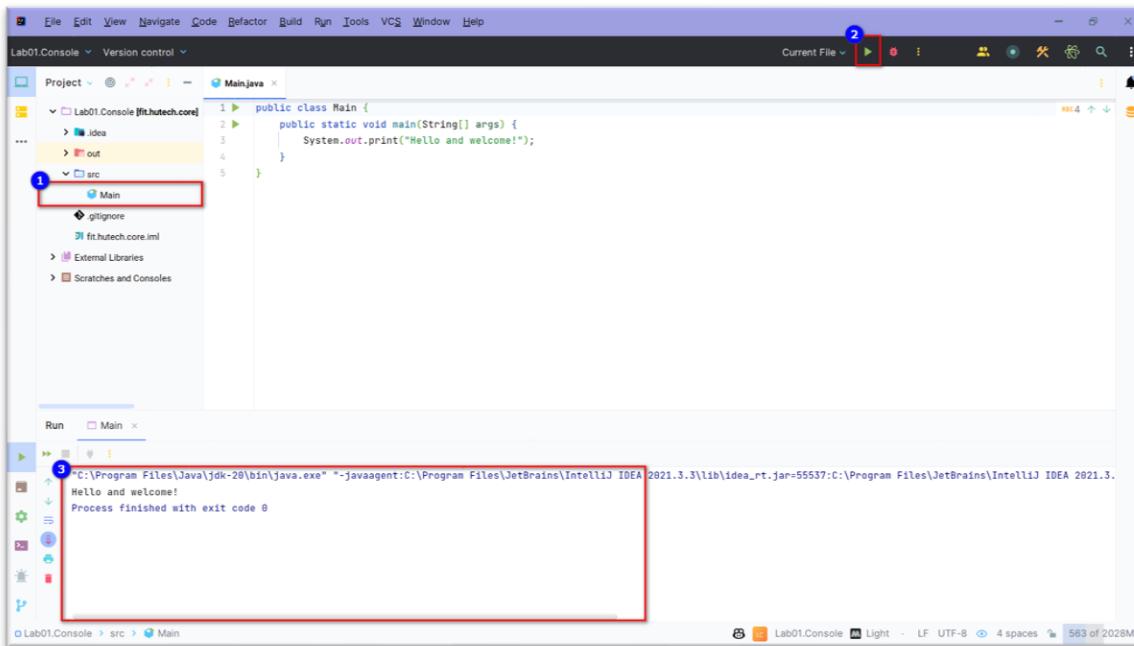
Hình 1.3. Download JDK new



Hình 1.4. Code file main.java project

Sau khi quá trình khởi tạo Project đã hoàn thành. Bên trái là cây thư mục của Project, trong khi bên phải là nội dung mã nguồn của file **Main.java**.

Tiếp theo, hãy chạy **Run** (chạy chương trình) và kiểm tra kết quả.



Hình 1.5. Run chương trình và xem kết quả

1.2.2. Bài tập 1: Viết chương trình menu cho phép quản lý sách

Cho biết thông tin của mỗi cuốn sách gồm: Mã Sách (kiểu số nguyên), Tên Sách (Kiểu chuỗi), Tác giả (Kiểu chuỗi) và Đơn giá (Kiểu double).

Áp dụng các tính chất hướng đối tượng (OOP), hãy viết một chương trình menu cho phép quản lý thông tin sách với các yêu cầu sau:

1. Thêm một cuốn sách.
2. Xóa một cuốn sách.
3. Thay đổi thông tin của một cuốn sách.
4. Xuất thông tin tất cả các cuốn sách.
5. Tìm kiếm cuốn sách có tên chứa từ "Lập trình" (không phân biệt hoa thường).
6. Lấy danh sách sách: Nhập số K và giá tiền P, trả về tối đa K cuốn sách có giá tiền ≤ P (**Lưu ý:** K và P là 2 số nguyên dương lớn hơn 0).

7. Nhập danh sách các tác giả từ bàn phím và hiển thị tất cả cuốn sách của những tác giả đó?

HƯỚNG DẪN

Bước 1:

- Tạo 1 Java class **Book** để lưu trữ thông tin của một cuốn sách.

```
public class Book {
    private int id;
    private String title;
    private String author;
    private long price;
}
```

- Thêm vào các phương thức **Getter, Setter**.

- Tạo các constructor: Chuẩn (không tham số) và đầy đủ tham số.

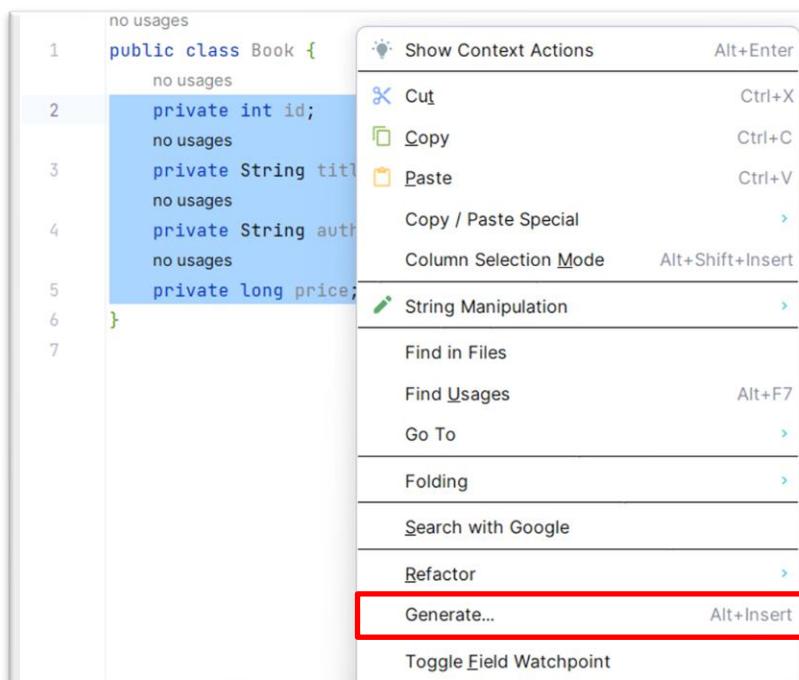
Để thực hiện việc này, có thể sử dụng các công cụ (Tool) để tự động sinh code:

Getter, Setter và Constructor.

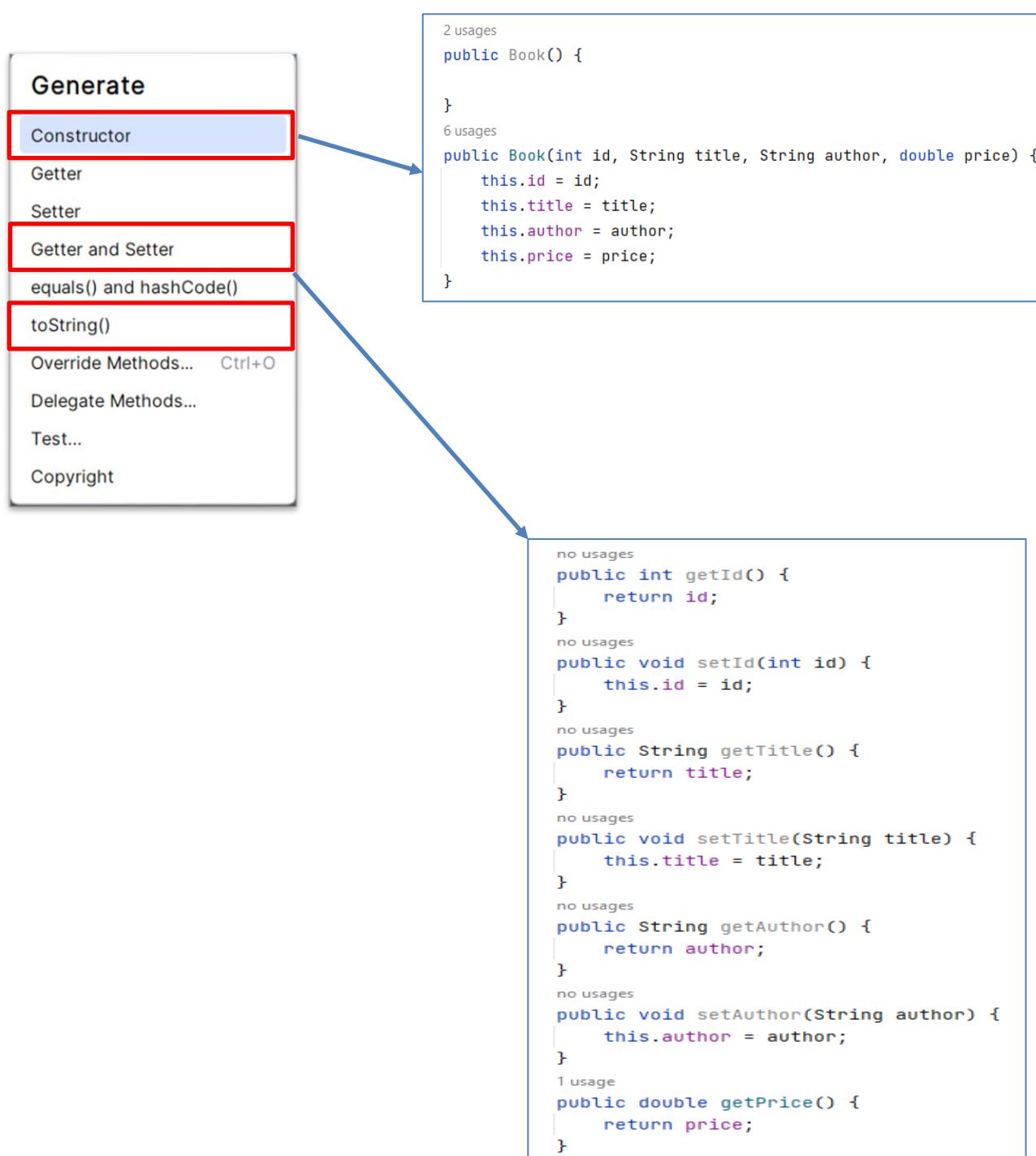
- Để sử dụng công cụ tự động sinh code thì sinh viên thực hiện các bước sau:

1. Quét các dữ liệu cần sinh ra các Getter, Setter và Constructor.

2. Nhấn chuột phải, sau đó chọn **Generate** hoặc nhấn tổ hợp phím **ALT + INSERT**



Hình 1.6. Sử dụng công cụ sinh code



Hình 1.7. Sử dụng Tool để Generation: Getter, Setter, Constructor

Bước 2: Viết các phương thức nhập, xuất thông tin cuốn sách.

- **input()**: Nhập thông tin của một cuốn sách (sử dụng thư viện **Scanner** để đọc dữ liệu nhập).
- **output()**: Hiển thị thông tin của một cuốn sách ra màn hình.

Hàm nhập thông tin của một cuốn sách sử dụng thư viện **Scanner**.

```
public void input() {
    Scanner sc = new Scanner(System.in);
    System.out.print("Nhập mã sách: ");
    id = Integer.parseInt(sc.nextLine());
    System.out.print("Nhập tên sách: ");
    title = sc.nextLine();
    System.out.print("Nhập tên tác giả: ");
    author = sc.nextLine();
    System.out.print("Nhập giá sách: ");
    price = sc.nextLong();
}
```

Hàm xuất thông tin cuốn sách (có sử dụng **Text Block** và thay thế các tham số tương ứng)

```
public void output() {
    var msg = """
        BOOK: id=%d, title=%s, author=%s, price=%d
        """.formatted(id, title, author, price);
    System.out.println(msg);
}
```

Bước 3: Viết hàm **main** thực hiện các chức năng quản lý sách, sử dụng:

- Switch expression
- Lambda expressions & phương thức foreach

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        List<Book> books = new ArrayList<>();
        var scanner = new Scanner(System.in);
        var msg = """
            Chương trình quản lý sách
            1. Thêm 1 cuốn sách
            2. Xoá 1 cuốn sách
            3. Thay đổi thông tin 1 cuốn sách
            4. Xuất thông tin tất cả các cuốn sách
            5. Tìm kiếm sách "Lập trình"
            6. Lấy sách tối đa theo giá
            7. Tìm kiếm sách theo tên tác giả
            0. Thoát
            Chọn chức năng:""";
```

```
int choice;

do {

    System.out.print(msg);
    choice = Integer.parseInt(scanner.nextLine());

    switch (choice) {
        case 1 -> {
            var book = new Book();
            book.input();
            books.add(book);
        }
        case 2 -> {
            System.out.print("Nhập mã sách cần xoá: ");
            var id = Integer.parseInt(scanner.nextLine());
            books.stream()
                .filter(book -> book.getId() == id)
                .findFirst()
                .ifPresent(books::remove);
            System.out.println("Đã xoá sách có mã " + id);
        }
        case 3 -> {
            System.out.print("Nhập mã sách: ");
            var id = scanner.nextInt();
            scanner.nextLine();
            books.stream()
                .filter(book -> book.getId() == id)
                .findFirst()
                .ifPresent(Book::input);
            System.out.println("Đã thay đổi thông tin sách!");
        }
        case 4 -> {
            System.out.println("Thông tin tất cả cuốn sách: ");
            books.forEach(System.out::println);
        }
        case 0 -> System.out.println("Đã thoát");
        default -> throw new IllegalStateException();
    }
} while (choice != 0);
}
```

Ví dụ: Thực hiện yêu cầu 5: Tìm cuốn sách có tên chứa từ "Lập trình" (không phân biệt hoa thường), ta có thể sử dụng Stream để truy vấn dữ liệu kết hợp với phương thức foreach và phương thức tham chiếu (method reference) để xuất kết quả trong lớp Book.

```
case 5 -> {
    System.out.println("Thông tin các sách có chứa \"Lập trình\": ");
    books.stream()
        .filter(book -> book.getTitle().contains("Lập trình"))
        .foreach(Book::output);
}
```

- Cách xuất **output()** ở trên là Phương thức tham chiếu (Book::output)

Gợi ý case 6: Kết hợp filter và limit trong stream

Gợi ý case 7: -Tập hợp tác giả cần tìm kiếm sẽ chuyển sang tập Set
 - filter tác giả mà nằm trong tập Set

1.2.3. Bài tập 2: Xây dựng chương trình kiểm định xe

Mỗi loại xe đều có ngày tháng năm sản xuất và 1 biển số duy nhất (Có 10 kí tự: 4 kí tự đầu là mã tỉnh và seri đăng ký - 5 kí tự sau là số từ 00000 đến 99999 Ví dụ: 62B6-88889). Ngoài các thông tin chung trên mỗi xe có các thuộc tính riêng:

Xe ô tô:

- Số chỗ ngồi.
- Có đăng ký kinh doanh vận tải Hoặc không đăng ký kinh doanh vận tải.

Xe tải:

- Có trọng tải của xe: (đơn vị tấn)

Viết chương trình menu để thực hiện các chức năng sau đây:

1. Thêm xe ô tô.
2. Thêm xe tải.
3. Xuất danh sách hiển thị thông tin tất cả các xe.
4. Lưu thông tin xe ra file.

5. Tìm xe ô tô có số chỗ ngồi nhiều nhất (nếu có)
6. Sắp xếp danh sách xe tải theo trọng tải tăng dần.
7. Xuất danh sách các biển số xe đẹp

Giả sử rằng: Biển số đẹp là biển số mà trong 5 số cuối có ít nhất 4 số giống nhau

VD: 62B6-88889, 60B6-88888, 79N1-99999...

8. Tính số tiền đăng kiểm định kỳ của từng xe đến thời điểm hiện tại.
9. Tính thời gian đăng kiểm định kỳ của từng xe sắp tới.
10. Tính tổng số tiền đã đăng kiểm.

Giả sử rằng tất cả các xe đều đã kiểm đúng ngày tháng định kỳ.

Lưu ý: Thông tin kiểm định định kỳ và mức giá đăng kiểm được quy định như sau:

- Về thời hạn đăng kiểm định kỳ:

1. Đối với những ô tô đã sản xuất dưới hoặc bằng 7 năm:

Ô tô chở người (số chỗ ≤ 9):

Không kinh doanh vận tải: Đăng kiểm định kỳ 2 năm.

Có kinh doanh vận tải: Đăng kiểm định kỳ 1 năm.

Ô tô chở người (số chỗ > 9): Đăng kiểm định kỳ 1 năm.

2. Đối với ô tô sản xuất trên 7 năm: Đăng kiểm định kỳ 6 tháng.
3. Đối với xe tải (sản xuất ≤ 20 năm): Đăng kiểm định kỳ 6 tháng.
4. Đối với xe tải (sản xuất > 20 năm): Đăng kiểm định kỳ 3 tháng.

- Về mức giá đăng kiểm:

1. Xe ô tô: chỗ ngồi ≤10 chỗ: 240,000đ
Chỗ ngồi >10 chỗ: 320,000đ
2. Xe tải có trọng tải > 20 tấn: 560,000đ
Từ 7-20 tấn: 350,000đ
và < 7: 320,000đ

HƯỚNG DẪN

Bước 1: Tạo các lớp để mô hình hóa các đối tượng trong bài toán như sau:

Lớp Xe: (có thể sử dụng abstract class):

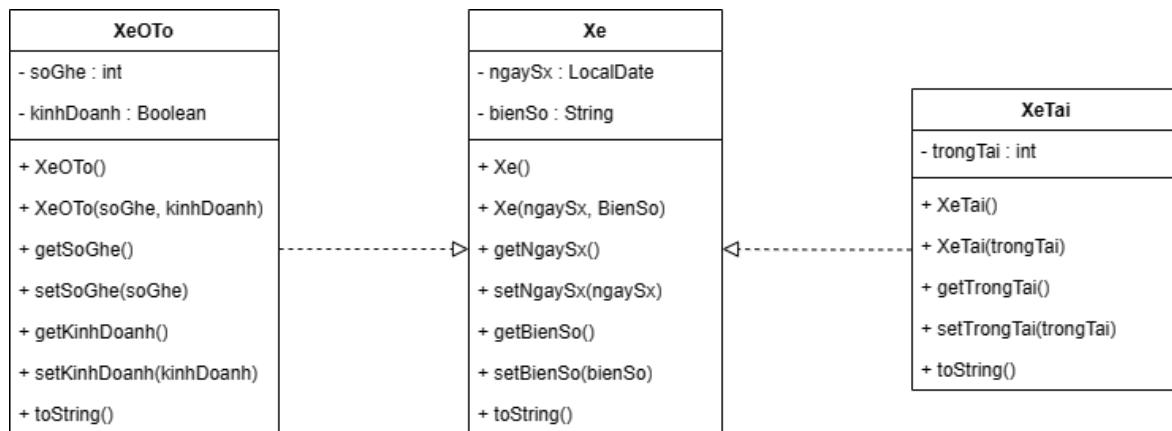
- Thuộc tính: ngày sản xuất (LocalDate), biển số (String)
- Phương thức: constructor (khởi tạo), getter/setter cho thuộc tính

Lớp XeOto (kế thừa từ lớp Xe):

- Thuộc tính: số ghế (int), có đăng ký kinh doanh vận tải hay không (boolean)
- Phương thức: constructor (khởi tạo), getter/setter cho thuộc tính

Lớp XeTai (kế thừa từ lớp Xe):

- Thuộc tính: trọng tải (int)
- Phương thức: constructor (khởi tạo), getter/setter cho thuộc tính.



Hình 1.8. Sơ đồ lớp mô tả quan hệ giữa các đối tượng

Bước 2: Sử dụng List<Xe> listXe để quản lý thông tin các xe. Và viết menu tương ứng.

- **Y/c 5:** Tìm xe ô tô có số chỗ ngồi nhiều nhất (nếu có).

```
ListXe.stream()
    .filter(Xe0To.class::isInstance)
    .map(xe -> (Xe0To) xe)
    .max(Comparator.comparingInt(Xe0To::getGhe))
    .stream()
    .findFirst()
    .ifPresentOrElse(
        xe -> System.out.println("Xe ô tô có nhiều ghế nhất: " + xe),
        () -> System.out.println("Không có xe ô tô nào"));
```

- **Y/c 6:** Sắp xếp xe tải có trọng tải tăng dần

Viết thêm hàm output ở lớp XeTai

```
ListXe.stream()
    .filter(XeTai.class::isInstance)
    .map(xe -> (XeTai) xe)
    .sorted(Comparator.comparingInt(XeTai::getTrongTai))
    .forEach(XeTai::output);
```

- **Y/c 7:** Tìm biển số xe đẹp

```
ListXe.stream()
    .map(Xe::getBienSo)
    .filter(bienSo -> bienSo.matches("[0-9]{3}.*"))
    .forEach(System.out::println);
```

1.3 Bài tập làm thêm

Bài 1: Viết chương trình quản lý các đối tượng trong nhà hàng Sushi: Đầu bếp, phục vụ, cửa hàng trưởng và tiếp thực. Các thành phần dữ liệu được quy định như sau:

- **Đầu bếp:** Họ tên, năm sinh, địa chỉ, số điện thoại, bằng cấp, vị trí công việc, số ngày công và bậc lương.
- **Phục vụ:** Họ tên, năm sinh, địa chỉ, số điện thoại, số ngày công và bậc lương.
- **Cửa hàng trưởng:** Họ tên, năm sinh, địa chỉ, số điện thoại, số năm kinh nghiệm, phụ cấp ăn uống, lương trong tháng
- **Tiếp thực:** Họ tên, năm sinh, địa chỉ, số điện thoại, phụ cấp sức khoẻ, số ngày công và bậc lương.

Biết rằng chỉ cửa hàng trưởng lãnh lương khoán, các nhân viên còn lại mức lương sẽ là ngày công * bậc lương. Phụ cấp ăn uống bằng 1/4 tiền lương, còn phụ cấp sức khoẻ sẽ là 20.000 mỗi ngày tính theo ngày công. Số tiền nhận được sẽ là tổng lương + phụ cấp

Ràng buộc dữ liệu: Số điện thoại phải là số có 10 chữ số, nhân viên khi đi làm phải đủ 18 tuổi trở lên (*Yêu cầu thiết lập các ràng buộc kiểm tra đầu vào*)

1. Nhập thông tin cho mỗi loại nhân viên
2. Xuất thông tin của tất cả nhân viên
3. In ra danh sách nhân viên kèm tổng lương và tổng số tiền nhận được cho từng loại đối tượng.
4. Tìm nhân viên có tổng số tiền nhận được lớn nhất và bé nhất
5. Tìm tiếp thực có mức lương cao nhất và thấp nhất
6. Tìm nhân viên có số ngày công cao nhất và thấp nhất
7. Tính và in ra tổng số tiền phụ cấp ăn uống cho tất cả nhân viên
8. Sắp xếp danh sách nhân viên theo tên và in ra kết quả
9. Lưu thông tin nhân viên vào file và có thể đọc lại từ file để hiển thị thông tin

Bài 2: Hãy viết trò chơi Con ong:

1. Khởi tạo danh sách đàn ong gồm 3 loại: Ong chúa, Ong thợ và Ong đực. Mỗi loại có 40 con ong.
2. Mỗi con ong được khởi tạo với 100 điểm máu.
3. Cung cấp chức năng tấn công đàn ong, khi tấn công mức máu sẽ giảm ngẫu nhiên từ 0 đến 80.
4. Hiển thị danh sách đàn ong sau mỗi lần tấn công kèm trạng thái (sống hoặc chết) và số điểm máu còn lại.
5. Đặt giới hạn điểm máu tối thiểu cho mỗi loại ong: Ong chúa (40), Ong thợ (50) và Ong đực (70). Nếu điểm máu của một con ong nhỏ hơn giới hạn tối thiểu, con ong đó sẽ chết.

Lưu ý: Lượng máu mỗi con ong được khởi tạo là 100. Sau mỗi lần tấn công, điểm máu sẽ giảm ngẫu nhiên từ 0 đến 80.

Hãy triển khai chương trình Trò chơi Con ong theo yêu cầu trên

TÓM TẮT

Qua bài học này, sinh viên đã ôn tập lập trình Java sử dụng các IDE như Eclipse, NetBeans và IntelliJ IDEA (Community Edition). Bài học bao gồm việc hướng dẫn sinh viên xây dựng các lớp, tạo đối tượng và truy xuất phương thức trong Java. Ngoài ra, sinh viên cũng tìm hiểu về việc sử dụng thư viện trong các phiên bản trước JDK 20, bao gồm lambda expressions, phương thức foreach, phương thức tham chiếu, Stream API, switch expression, text block và record.

Trong bài học này, sinh viên cũng sẽ tìm hiểu các kiểu dữ liệu trong Java, bao gồm kiểu dữ liệu nguyên thủy và kiểu dữ liệu không nguyên thủy. Sinh viên sẽ sử dụng lớp Scanner để nhập dữ liệu từ người dùng và sử dụng các phương thức để hiển thị thông tin trên màn hình.

Mục tiêu của bài học là cung cấp một hướng dẫn chi tiết để giúp sinh viên ôn tập và cải thiện kỹ năng lập trình Java của mình.

CÂU HỎI ÔN TẬP

- Khái niệm của Getter, Setter và Constructor trong Java là gì? Tại sao chúng cần phải được khởi tạo?
- Java Stream là gì? Đặc điểm và lợi ích của Java Stream là gì?
- Switch Expression là gì và tại sao nó được thêm vào trong các phiên bản Java mới? Cho ví dụ về cú pháp và cách sử dụng Switch Expression trong Java.
- Lambda Expression trong Java được sử dụng như thế nào?
- Text Block là gì và tại sao được giới thiệu trong Java?

BÀI 2 LÀM QUEN VỚI SPRING FRAMEWORK

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

- Hiểu về Spring Framework và vai trò của nó trong phát triển ứng dụng.
- Cài đặt môi trường và công cụ cần thiết để phát triển ứng dụng web bằng Spring Framework.
- Xây dựng ứng dụng web cơ bản bằng Spring Boot.

2.1 Cơ bản về Spring Framework và các cài đặt cần thiết

2.1.1 Yêu cầu

Máy tính phải cài đặt:

- Java JDK (Java Development Kit) [**Phiên bản ≥ 20**]
- Cài đặt Laragon (Quản lý CSDL MySQL)
- Cài đặt IntelliJ IDEA (Community)

(*Tham khảo cài đặt tại phần [PHỤ LỤC CÀI ĐẶT MÔI TRƯỜNG VÀ CÔNG CỤ CẦN THIẾT](#) ở CUỐI TRANG*)

Máy tính chạy trên nền tảng hệ điều hành:

- Windows 7.
- Windows 8.
- Windows 10.
- Windows 11.
- Windows Server 2019.
- Windows Server 2022.

2.2 Hướng Dẫn Thiết Lập phpMyAdmin

Laragon và phpMyAdmin là hai công cụ quan trọng, miễn phí và mã nguồn mở trong quá trình phát triển và quản lý ứng dụng web dựa trên ngôn ngữ lập trình PHP và cơ sở dữ liệu MySQL.

Laragon là một môi trường phát triển web dựa trên Windows, giúp người dùng dễ dàng cài đặt và quản lý các công cụ cần thiết như Apache, PHP và MySQL. Laragon cung cấp một giao diện đồ họa thân thiện và khả năng tùy chỉnh linh hoạt, cho phép người dùng dễ dàng tạo và quản lý các môi trường phát triển web.

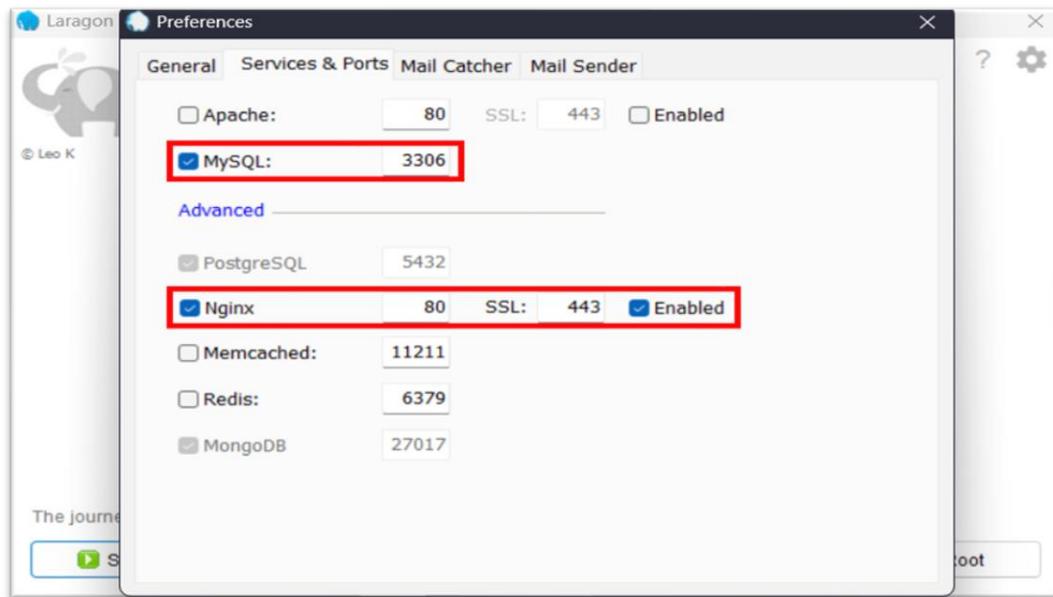
phpMyAdmin là một công cụ quản lý cơ sở dữ liệu MySQL thông qua giao diện web. Nó cung cấp các chức năng để **tạo, xóa và sửa đổi** các cơ sở dữ liệu, bảng, truy vấn và người dùng trong MySQL. phpMyAdmin giúp người dùng tương tác với cơ sở dữ liệu một cách trực quan và thuận tiện, giảm bớt việc phải sử dụng các câu lệnh SQL trực tiếp.

Để khởi động ứng dụng Laragon, sinh viên có thể nhấp vào **biểu tượng cài đặt** ở góc bên phải như hình minh họa dưới đây.



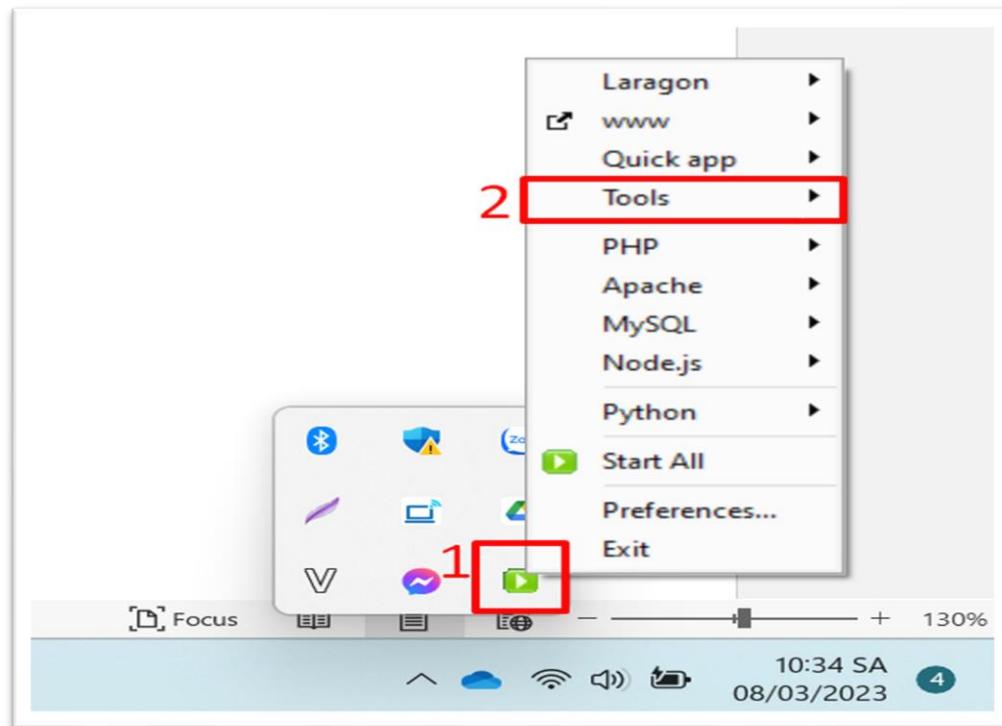
Hình 2.1. Khởi động ứng dụng Laragon

Khi lựa chọn cấu hình, hãy tích chọn **MySQL** và **Nginx**. Trong trường hợp gặp lỗi do trùng cổng với các dịch vụ khác, hãy thử thay đổi cổng sử dụng.

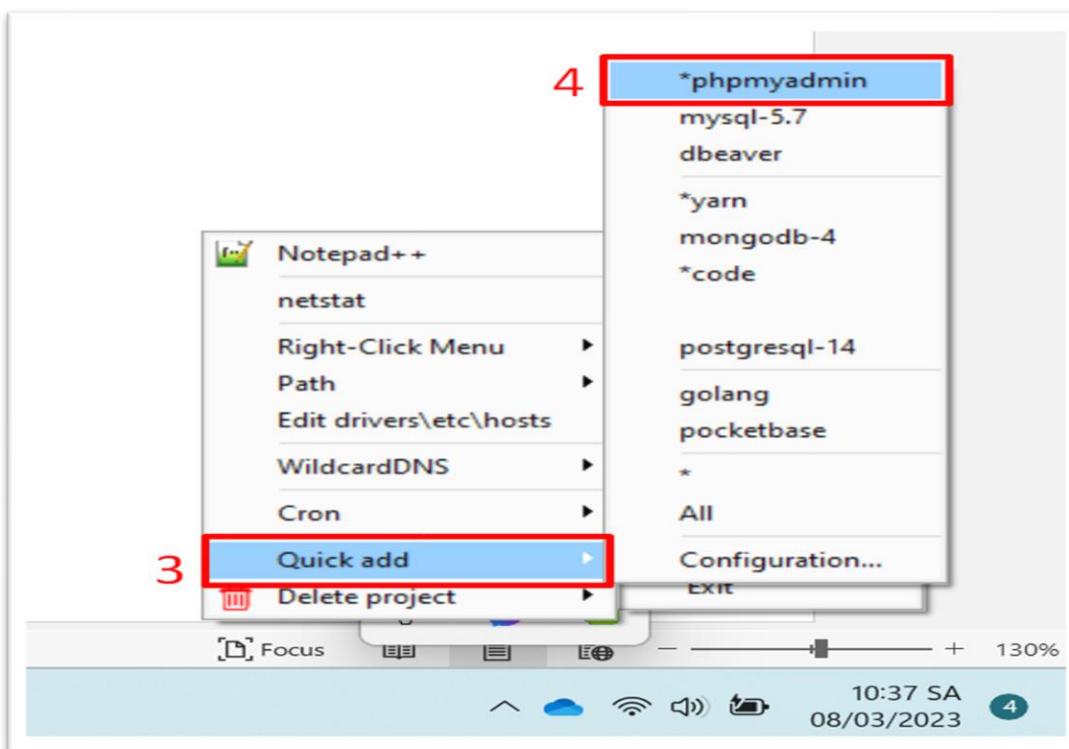


Hình 2.2. Cấu hình MySQL, Nginx cho Laragon

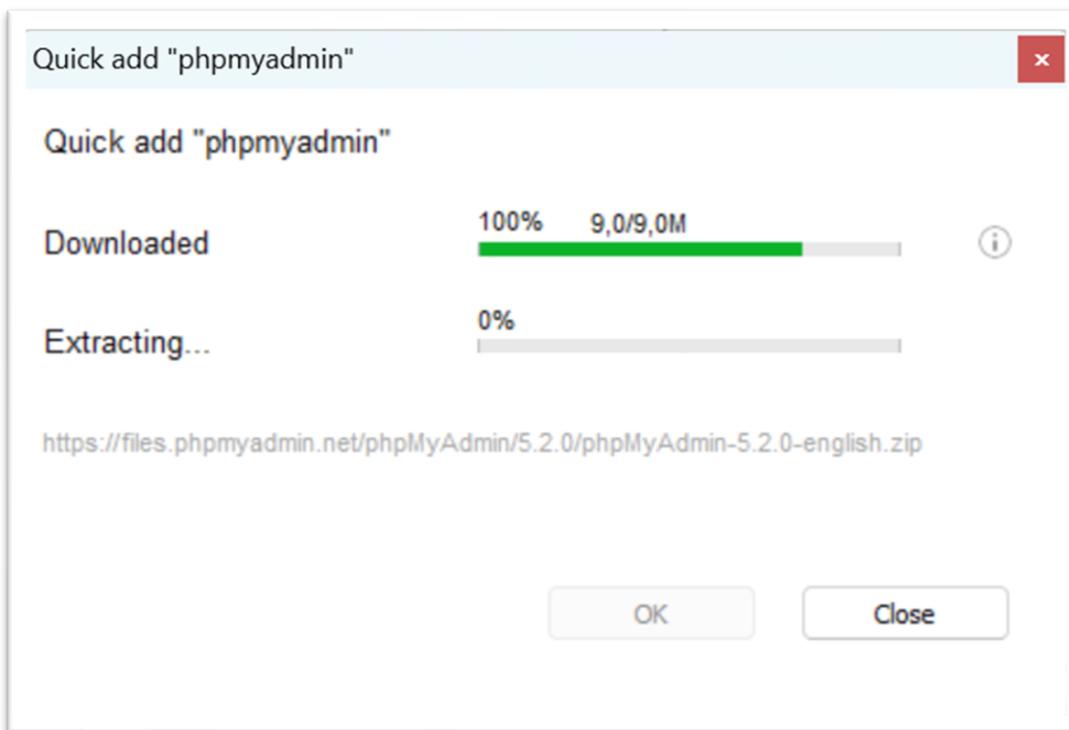
Sau khi làm xong các bước trên, ta tiến hành cài đặt công cụ **phpMyAdmin** theo các bước như bên dưới.



Hình 2.3. Cài đặt chọn phpMyAdmin 1

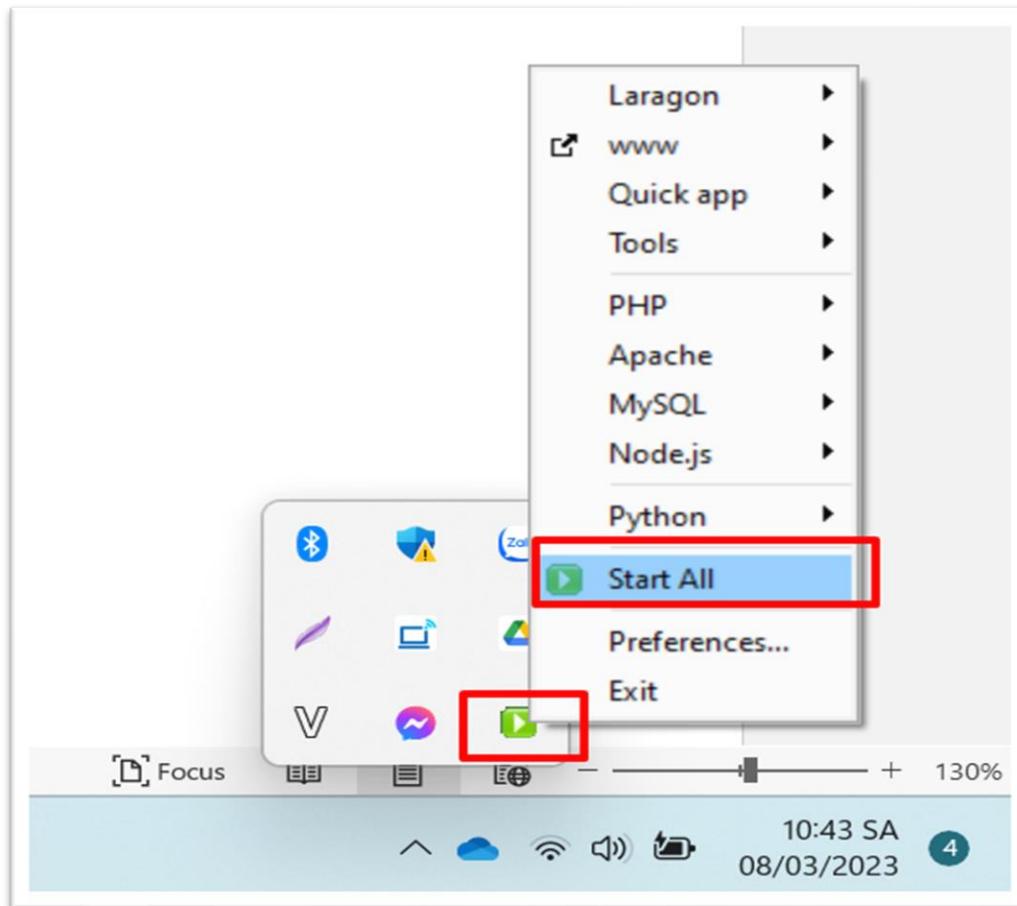


Hình 2.4. Cài đặt chọn phpMyAdmin 2



Hình 2.5. Quá trình Quick add "phpMyAdmin"

Sau khi thiết lập xong ấn **Start All** để khởi động các dịch vụ.

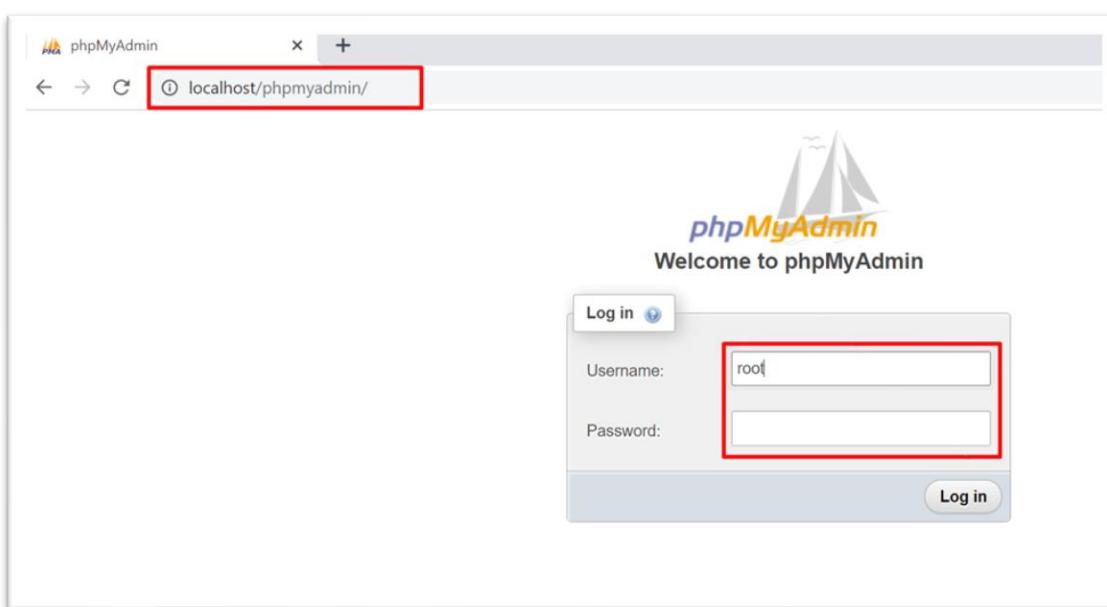


Hình 2.6. Khởi động dịch vụ PhpMyAdmin

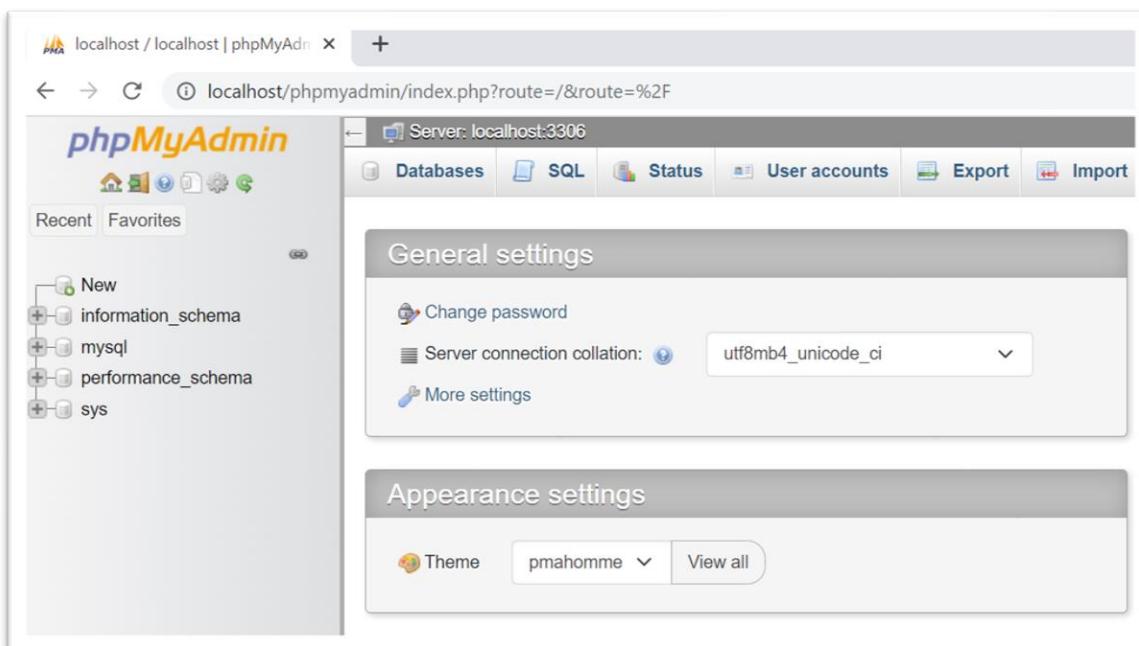
Như vậy bạn đã cài đặt và thiết lập thành công công cụ phpmyadmin trên máy tính. Bây giờ truy cập vào đường dẫn: <http://localhost/phpmyadmin/>

Đăng nhập vào **phpMyAdmin**. Theo mặc định tên người dùng là **root** và mật khẩu [để trống](#)

⚠ **Ghi chú:** Ngoài phpMyAdmin, sinh viên cũng có thể sử dụng nhiều trình quản lý cơ sở dữ liệu khác như HeidiSQL, TablePlus, DataGrip, Dbeaver, Navicat... Các công cụ này cung cấp giao diện đồ họa và nhiều tính năng hữu ích giúp quản lý cơ sở dữ liệu dễ dàng và thuận tiện



Hình 2.7. Giao diện đăng nhập phpMyAdmin



Hình 2.8. Giao diện trang Index PhpMyAdmin

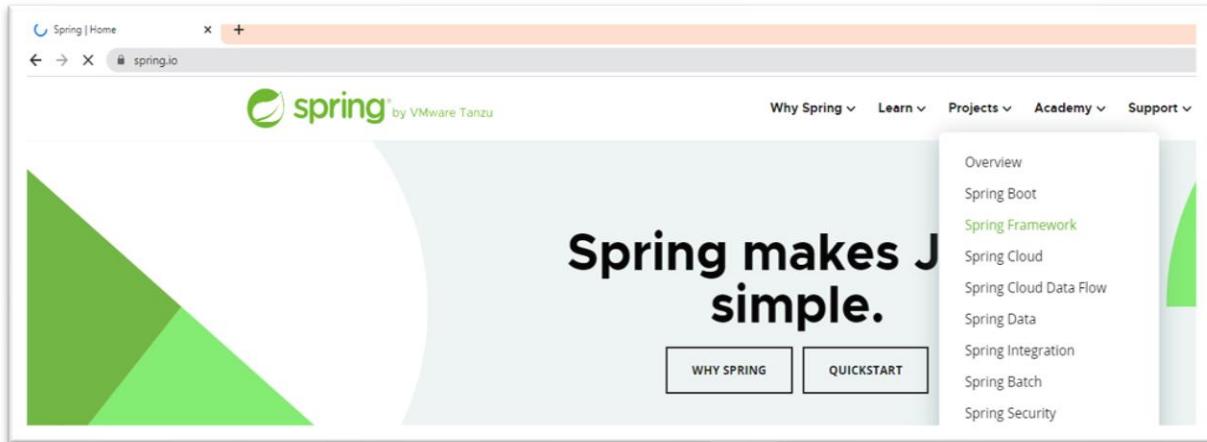
Sau khi hoàn tất quá trình cài đặt và khởi động thành công, sinh viên sẽ tiếp tục truy cập vào **phpMyAdmin**.

2.3 Download Template Spring từ spring.io

Spring.io cung cấp một Template Project Spring, là một dự án mẫu sẵn có để phát triển ứng dụng sử dụng framework Spring. Template này cung cấp các bộ khung cơ bản giúp giảm thời gian và công sức cho lập trình viên khi bắt đầu phát triển một dự án mới.

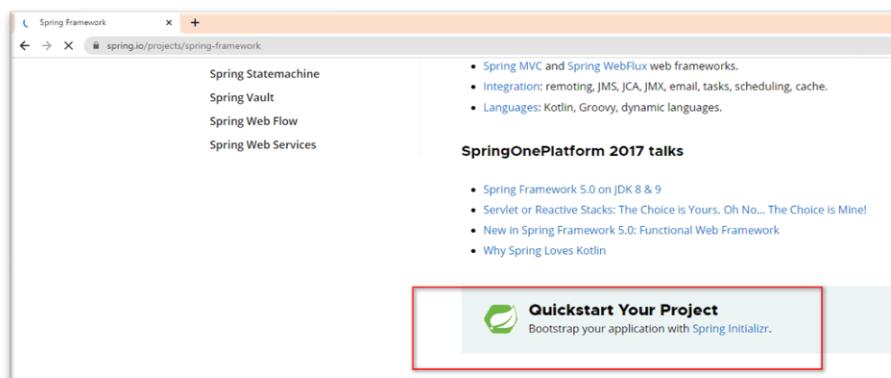
Template Spring bao gồm các module quan trọng như Spring Boot, Spring Security, Spring Data, Spring MVC, Spring Cloud và nhiều module khác.

Để truy cập vào trang web chính thức, vui lòng truy cập: <https://spring.io/> và chọn mục **Project** trong menu.



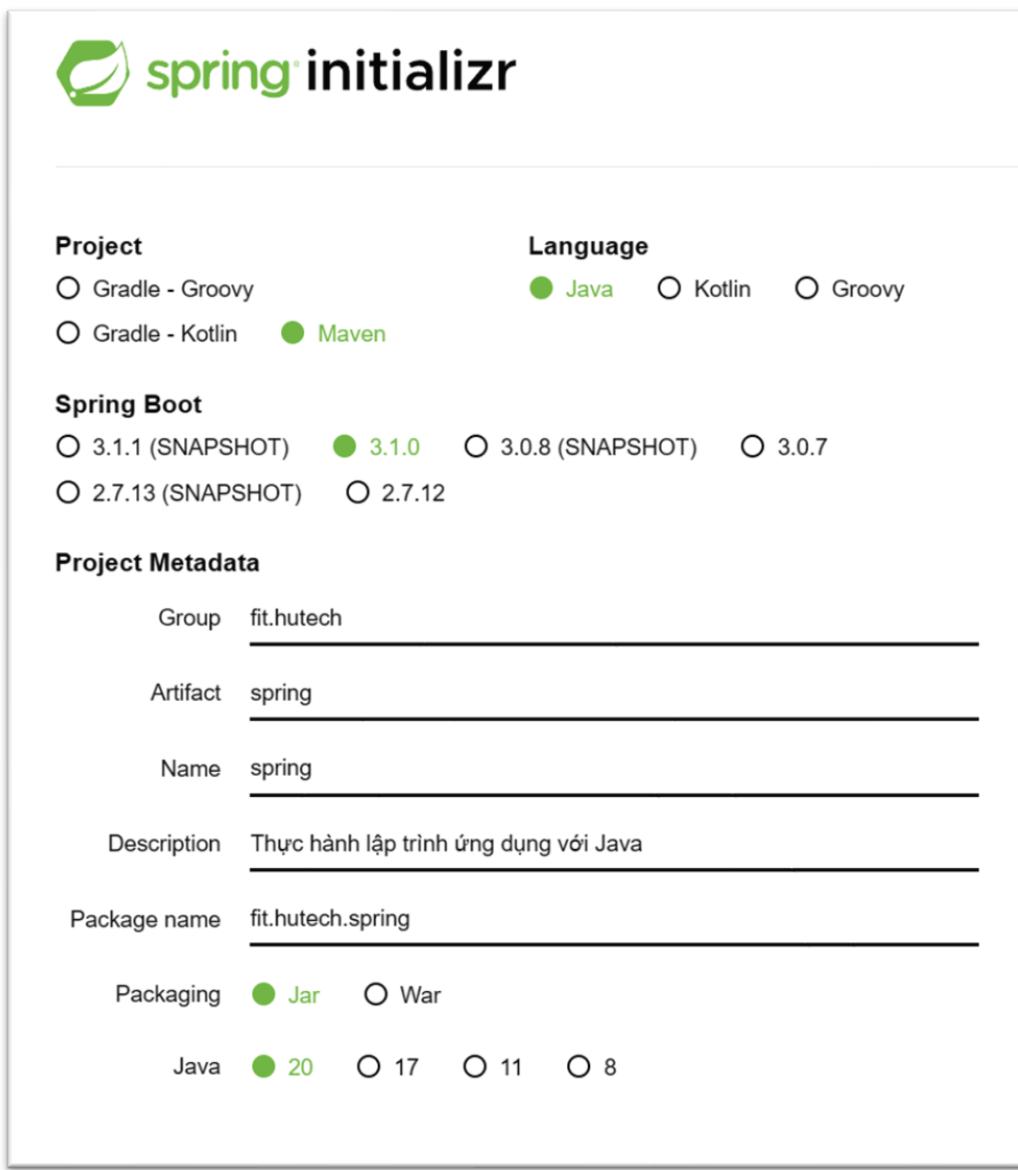
Hình 2.9. Open Spring Framework tại spring.io

Tiếp theo, lướt xuống dưới chọn **Spring Initializr**.



Hình 2.10. Khởi động Quickstart Your Project

Lựa chọn cấu hình, tích chọn các lựa chọn như ảnh bên dưới



Hình 2.11. Lựa chọn cấu hình cho spring initializr

Ý nghĩa lựa chọn quan trọng trong Spring Initializr:

- **Project:** Lựa chọn loại dự án Spring Boot
- **Language:** Chọn ngôn ngữ lập trình cho dự án, bao gồm Java hoặc Kotlin.
- **Spring Boot Version:** Chọn phiên bản Spring Boot mà dự án sẽ sử dụng.
- **Group:** Nhập tên nhóm cho dự án, thường là domain name ngược lại.
- **Artifact:** Nhập tên đại diện cho module chính trong dự án.
- **Name:** Nhập tên dự án.

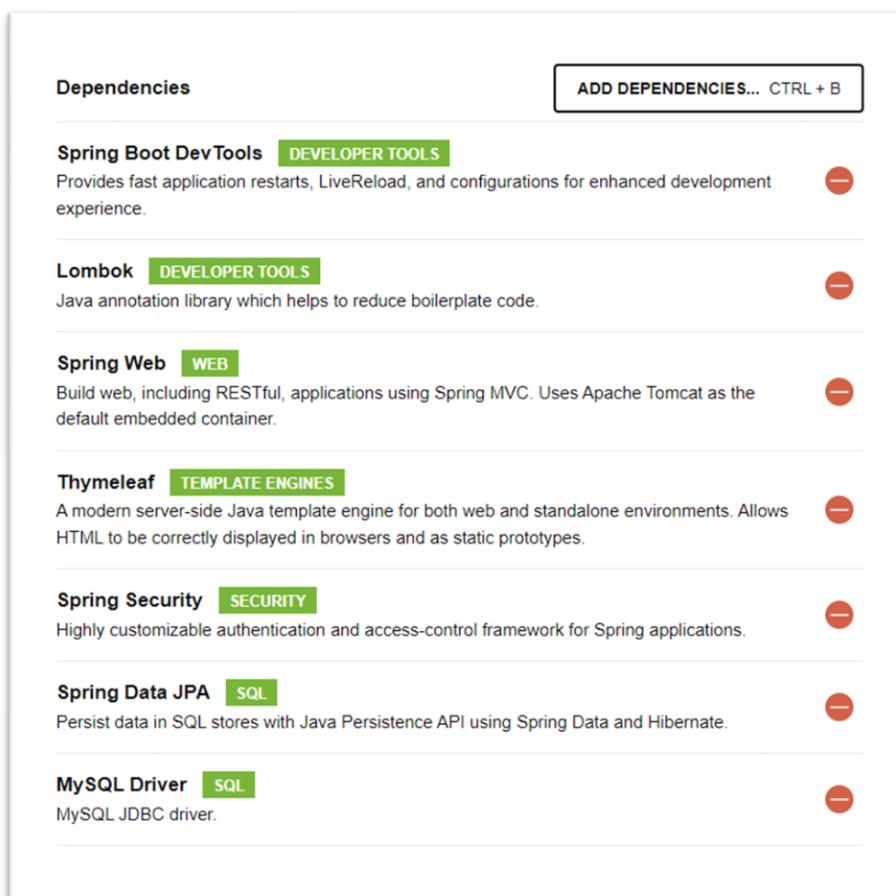
- **Description:** Cung cấp mô tả ngắn về dự án.
- **Package Name:** Nhập tên gói (package) cho các class trong dự án.
- **Packaging:** Lựa chọn định dạng đóng gói cho ứng dụng
- **Java:** Chọn phiên bản Java mà dự án sẽ sử dụng.
- **Dependencies:** Chọn các phụ thuộc (dependencies) mà dự án cần

Tiếp theo nhấn Tiếp, nhấn **Add Dependencies...**



Hình 2.12. Tiến hành thêm dependencies vào dự án

Cửa sổ mở ra, tìm và thêm các **Dependencies** như ảnh bên dưới.



Hình 2.13. Thêm các Dependencies trong spring.io

Ý nghĩa của các dependencies trên:

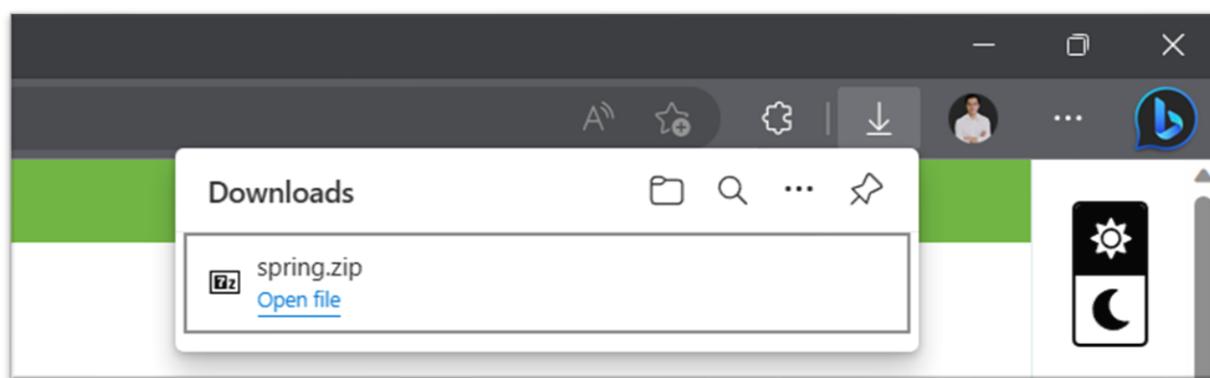
- **Spring Boot Dev Tools:** Công cụ hỗ trợ phát triển cho Spring Boot.
- **Lombok:** Thư viện giảm thiểu việc viết mã lặp lại trong Java.
- **Spring Web:** Module để phát triển ứng dụng web trong Spring Framework.
- **Thymeleaf:** Template engine cho giao diện người dùng trong ứng dụng web.
- **Spring Security:** Framework quản lý bảo mật cho ứng dụng web.
- **Spring Data JPA:** Module để làm việc với cơ sở dữ liệu quan hệ.
- **MySQL Driver:** Trình điều khiển MySQL.

Sau đó, nhấn **GENERATE** để tải về



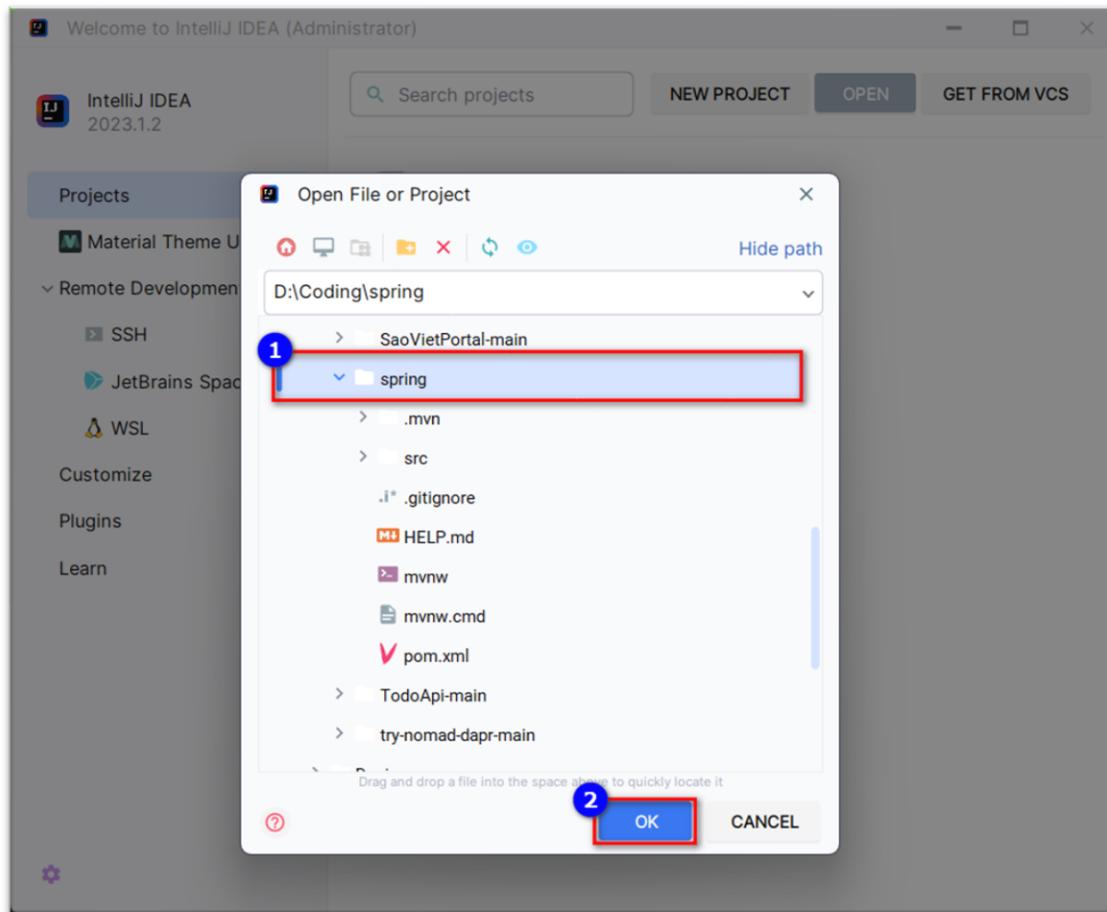
Hình 2.14. Generate dự án Spring Boot

Một file **spring.zip** được tải về máy tính,

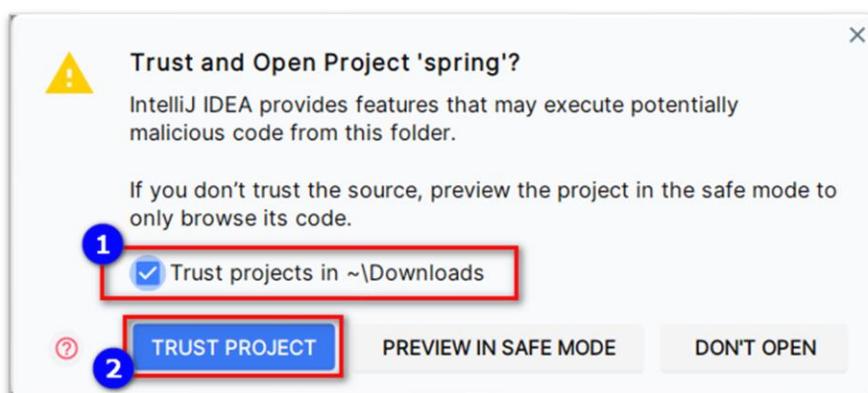


Hình 2.15. Tệp tin dự án được generate từ Spring Initializr

Tiến hành giải nén thư mục **spring.zip** template đã được tải về. Sau khi giải nén thành công, ta có thể thấy thư mục **spring** xuất hiện. Tiếp theo, mở **IntelliJ IDEA** và chọn **Open**. Duyệt đến thư mục **spring** và chọn nó. **IntelliJ IDEA** sẽ tải và mở dự án **spring**.

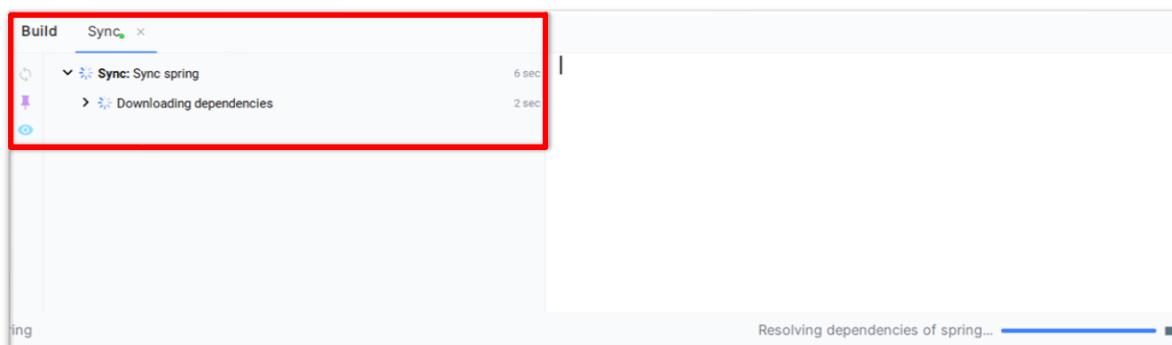


Hình 2.16. Giải nén và mở Spring Template bằng IntelliJ IDEA.



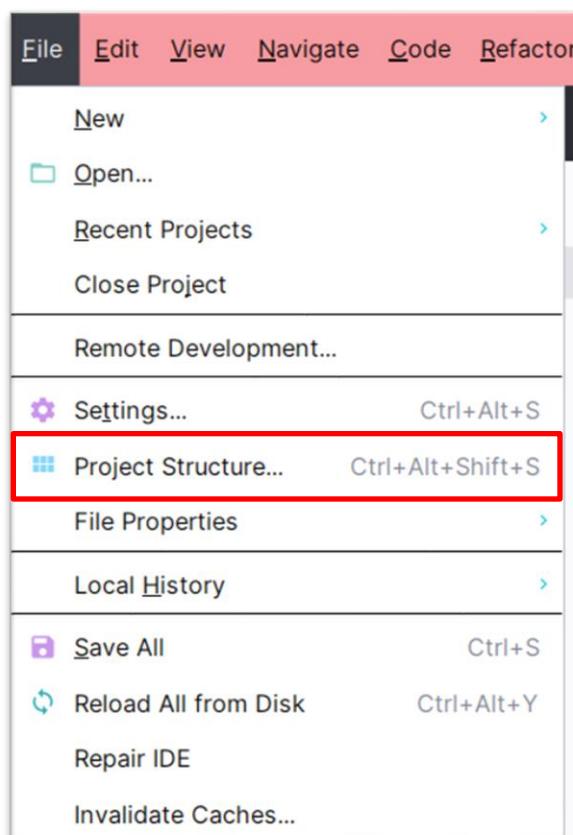
Hình 2.17. Thiết lập tin cậy thư mục chứa dự án

Ta sẽ tiến hành tải và cài đặt các **Dependencies** cần thiết. Hãy kiên nhẫn chờ đợi cho quá trình cài đặt hoàn tất, bao gồm việc cài đặt đường dẫn tới thư mục JDK đã được cài đặt ở bước trước.



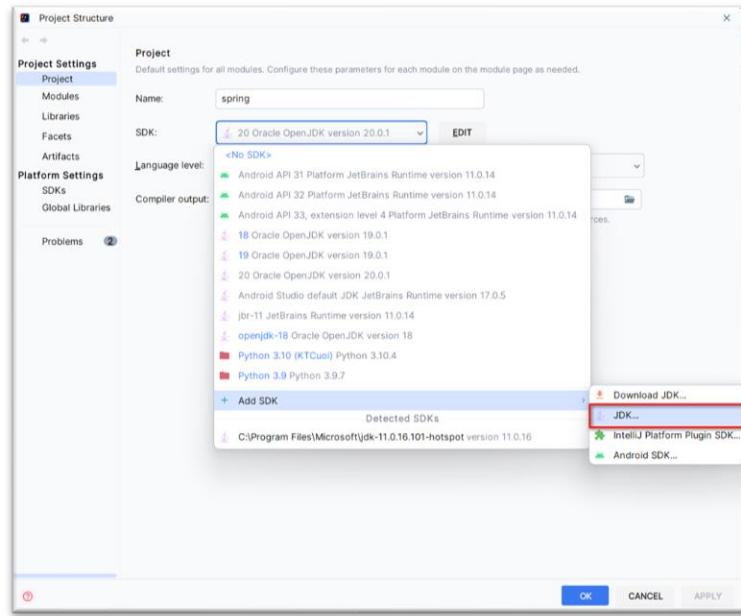
Hình 2.18. Bước cài đặt các Dependencies cần thiết

Tại thanh menu, chọn **File** → chọn **Project Structure...** hoặc nhấn tổ hợp phím **CTRL + ALT + SHIFT + S**. Để thiết lập cấu trúc dự án.



Hình 2.19. Lựa chọn Project Structure

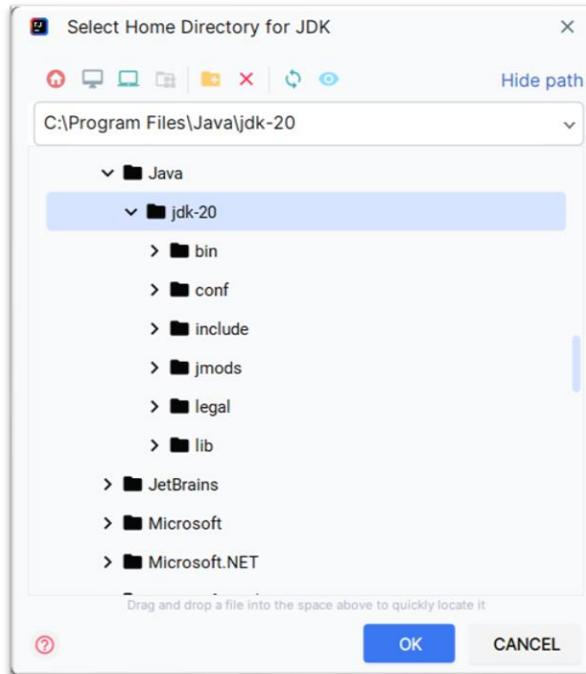
Tiếp theo, làm theo hướng dẫn bên dưới để Add đến đường dẫn chứa file JDK.



Hình 2.20. Lựa chọn phiên bản JDK

Cửa sổ xuất hiện, chọn thư mục **jdk-20** theo đường dẫn:

C:\Program Files\Java\jdk-20 .



Hình 2.21. Lựa chọn phiên bản JDK - 20

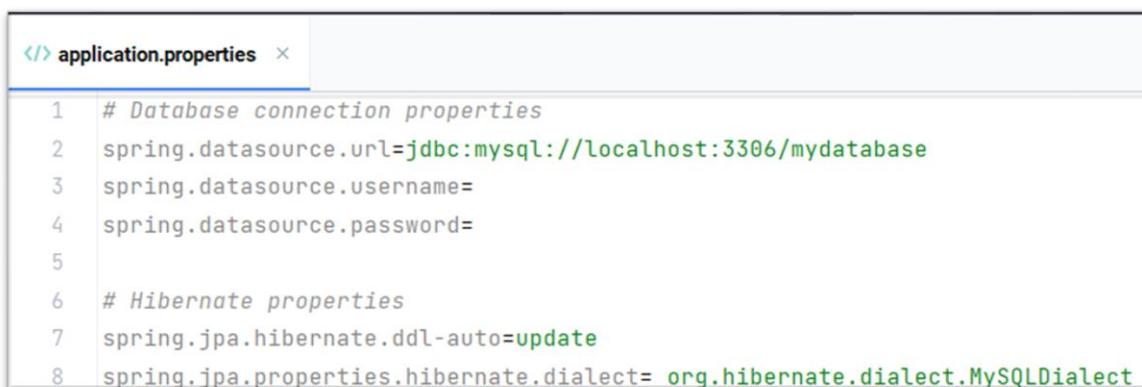
Thêm nội dung sau vào file **application.properties**.

application.properties có thể được đặt trong thư mục **src/main/resources** của ứng dụng web Java. Nó được sử dụng để cấu hình các thông số như cơ sở dữ liệu, cổng kết nối, tên đăng nhập và mật khẩu.

```
# Database connection properties
spring.datasource.url=jdbc:mysql://localhost:3306/mydatabase
spring.datasource.username=
spring.datasource.password=

# Hibernate properties
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=
org.hibernate.dialect.MySQLDialect
```

Tìm đến file **application.properties** tại đường dẫn **src/main/resources**.

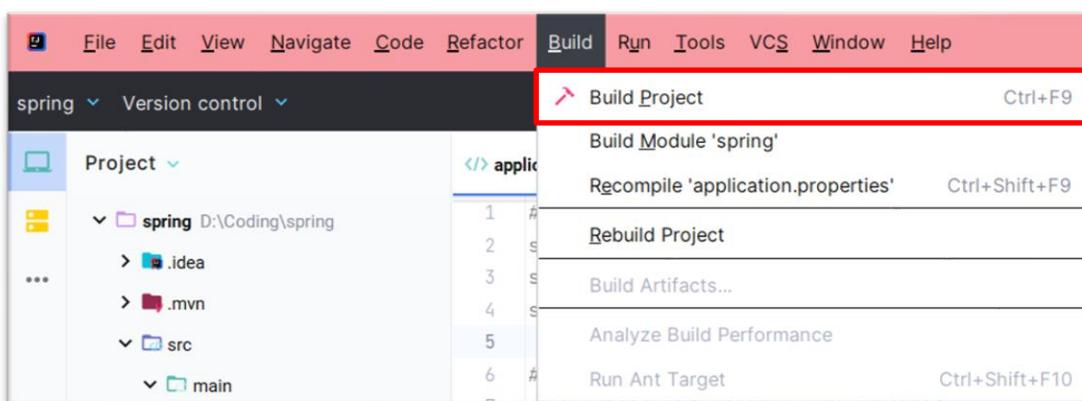


```
# Database connection properties
spring.datasource.url=jdbc:mysql://localhost:3306/mydatabase
spring.datasource.username=
spring.datasource.password=

# Hibernate properties
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect= org.hibernate.dialect.MySQLDialect
```

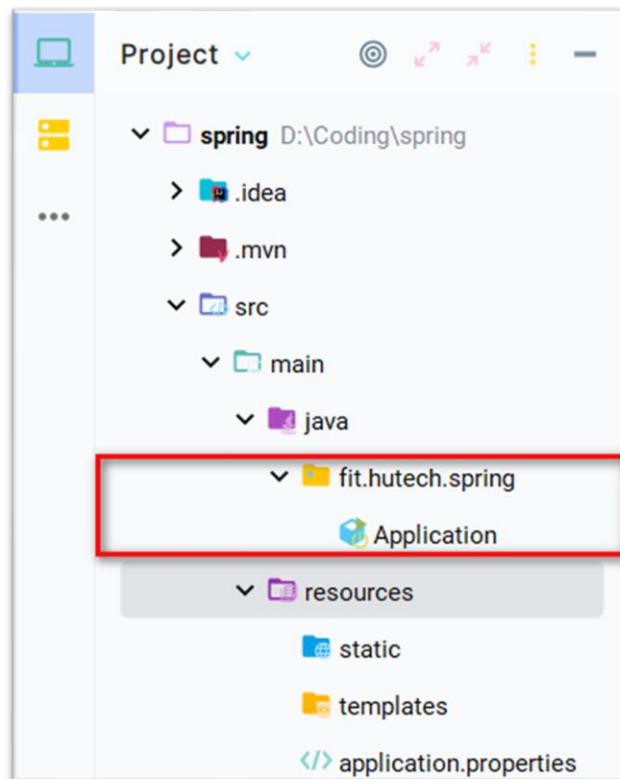
Hình 2.22. Bổ sung đoạn code cấu hình cho *application.properties*

Tiếp theo, chọn **Build → Build Project** hoặc nhấn tổ hợp phím **CTRL + F9** như ảnh bên dưới.



Hình 2.23. Build project chương trình

Tiến hành **Run** chương trình tại file **application.java** theo ảnh hướng dẫn bên dưới.
Hoặc nhấn tổ hợp phím **SHIFT + F10**.



Hình 2.24. Tiến hành chạy ứng dụng

```
Application.java x
```

```
1 package fit.hutech.spring;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class Application {
8     public static void main(String[] args) {
9         SpringApplication.run(Application.class, args);
10    }
11 }
12 }
```

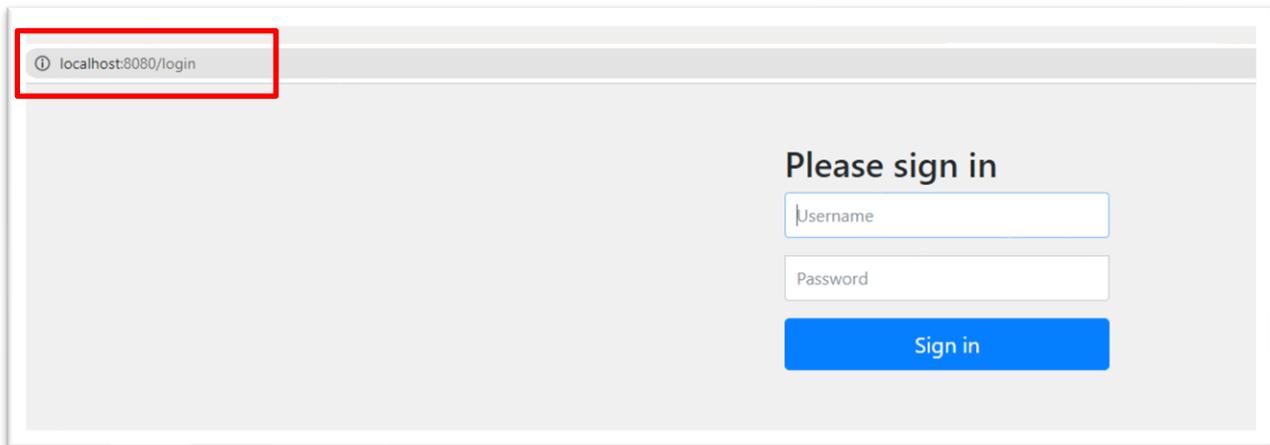
The screenshot shows the IntelliJ IDEA code editor with the 'Application.java' file open. The code is as follows:

```
1 package fit.hutech.spring;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class Application {
8     public static void main(String[] args) {
9         SpringApplication.run(Application.class, args);
10    }
11 }
12 }
```

Lines 7 and 8 are highlighted with green arrows, indicating the main entry point of the application.

Hình 2.25. Run chương trình tại file Application

Sau đó, thực hiện mở trình duyệt trên máy tính, truy cập vào đường dẫn mặc định: <http://localhost:8080/>, kết quả xuất hiện giao diện đăng nhập như sau:



Hình 2.26. Giao diện mặc định đăng nhập

Trang mặc định, được cấu hình khởi chạy từ **Spring Security**.

Để tắt **Spring Security** để thực hiện các bài lab phía dưới ta thực hiện như sau:

Chỉnh sửa file **src/resources/application.properties**, thêm phần **#Spring Security**.

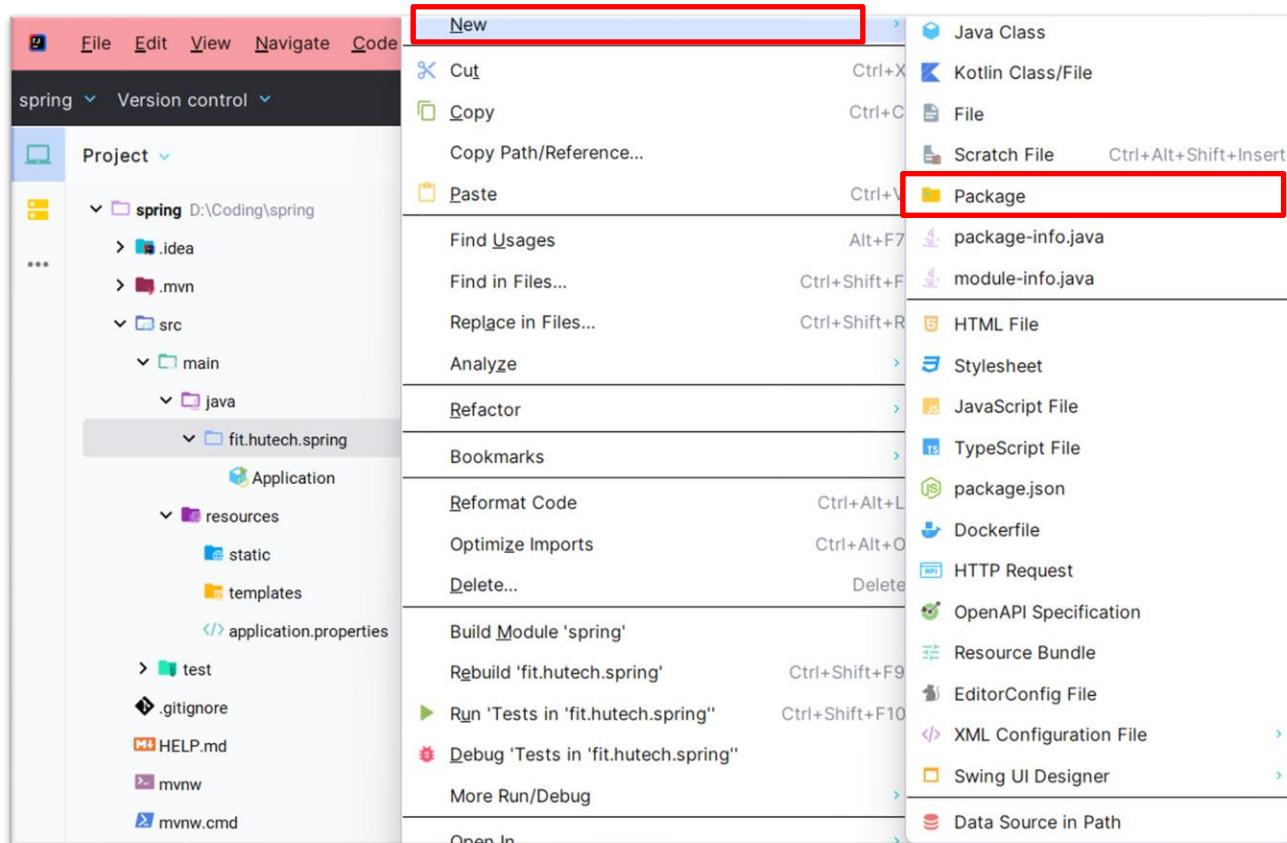
```
# Database connection properties
spring.datasource.url=jdbc:mysql://localhost:3306/mydatabase
spring.datasource.username=
spring.datasource.password=

# Hibernate properties
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=
org.hibernate.dialect.MySQLDialect

#Spring Security
spring.autoconfigure.exclude=org.springframework.boot.autoconfigure.security.servlet.SecurityAutoConfiguration
```

2.4 Chạy trang web đầu tiên

Tạo package **controllers** tại **src/main/java/fit.hutech.spring**. Nhấn chuột phải vào **fit.hutech.spring**, chọn **New → Package**.



New Package

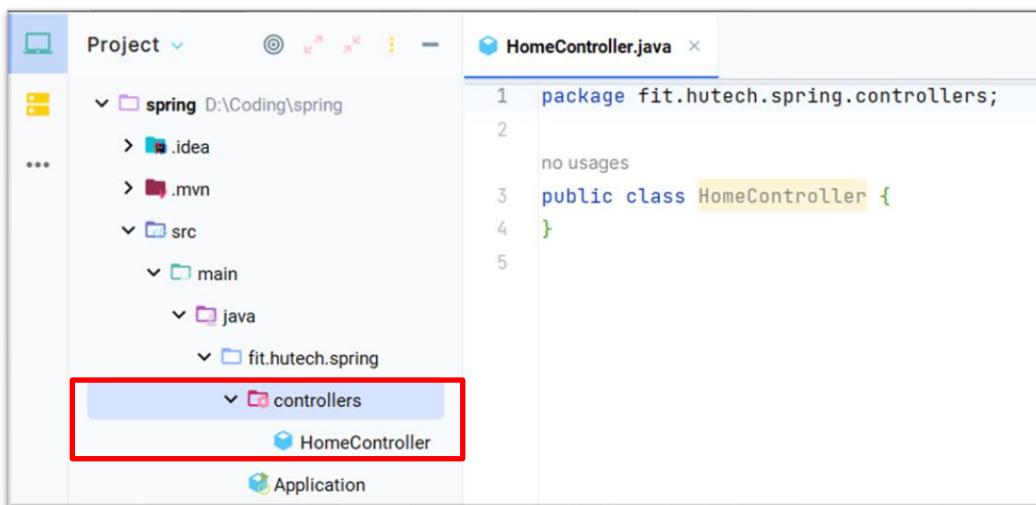
fit.hutech.spring.controllers

Hình 2.27. Tạo package controller

Tạo file **HomeController.java** đặt tại đường dẫn:

src/main/java/fit.hutech.spring/controllers.

Nhấn chuột phải vào **controllers**. Sau đó chọn **New → Java Class → Class →**
Nhập tên class là **HomeController**. Cuối cùng nhấn **ENTER** để khởi tạo.



Hình 2.28.Tạo file HomeController.java trong package controllers

Chỉnh sửa nội dung **HomeController.java** như ảnh bên dưới:

```
package fit.hutech.spring.controllers;

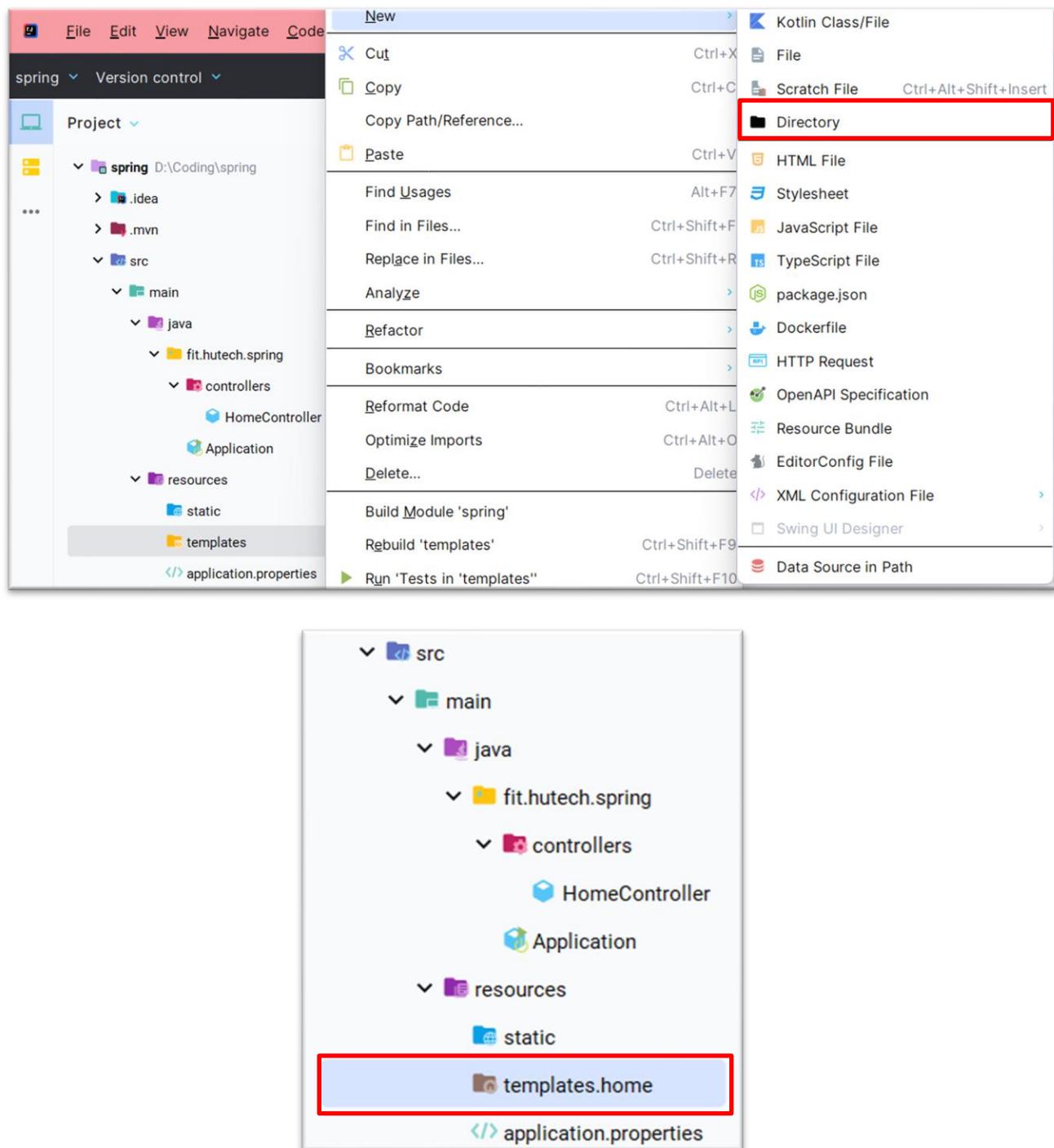
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("/")
public class HomeController {

    @GetMapping
    public String home() {
        return "home/index";
    }

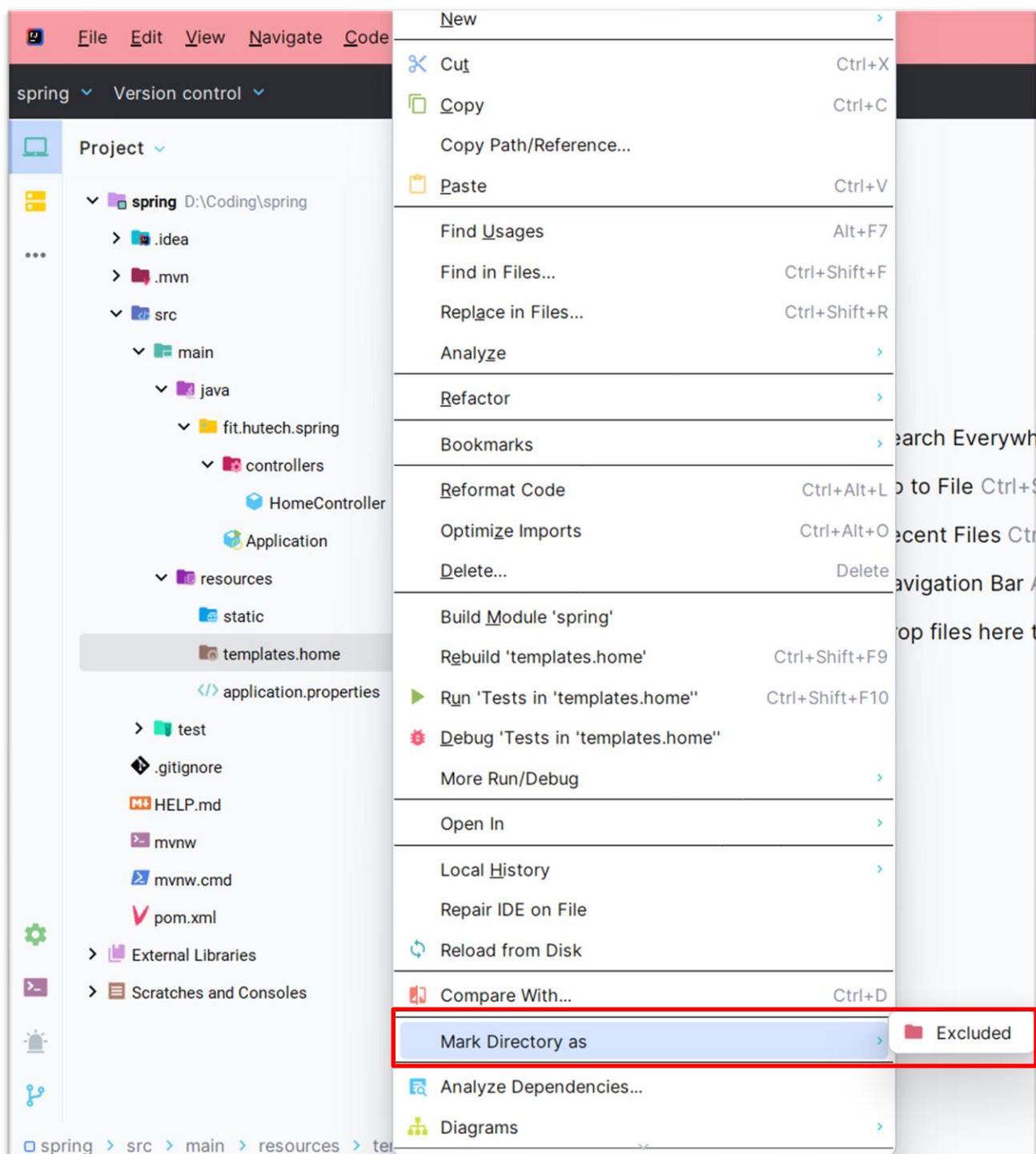
    @GetMapping("/contact")
    public String contact() {
        return "home/contact";
    }
}
```

Tạo thư mục **home** đặt tại **src/main/resources/templates**. Chuột phải vào **templates**, sau đó chọn **New → Directory**.



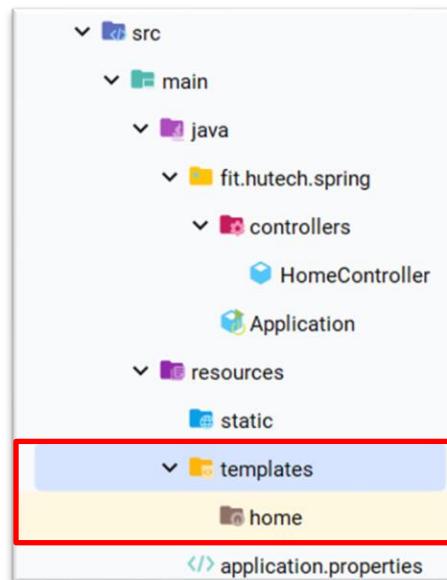
Hình 2.29. Tạo thư mục home

Để đảm bảo rằng thư mục **Home** hiển thị ngay dưới thư mục **Templates** như được thể hiện trong ảnh bên dưới, hãy thực hiện các bước sau. Nhấn chuột phải vào thư mục **templates.home**, sau đó chọn **Make Directory as → Exclude**.

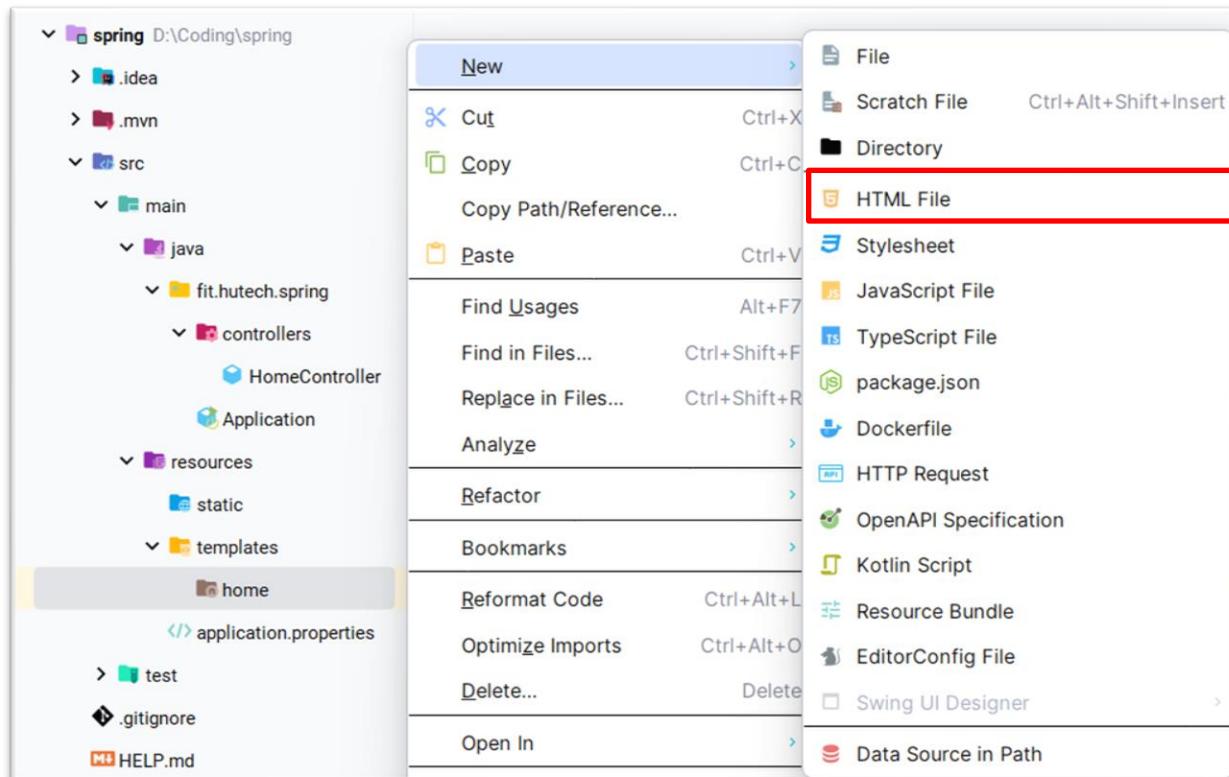


Hình 2.30. Hiển thị ngay phía dưới thư mục templates

Kết quả như ảnh bên dưới:



Tạo file **index.html** đặt tại **src/resources/templates/home**. Nhấn chuột phải vào thư mục **home**, sau đó chọn **New → HTML File** → đặt tên và ấn **ENTER**.

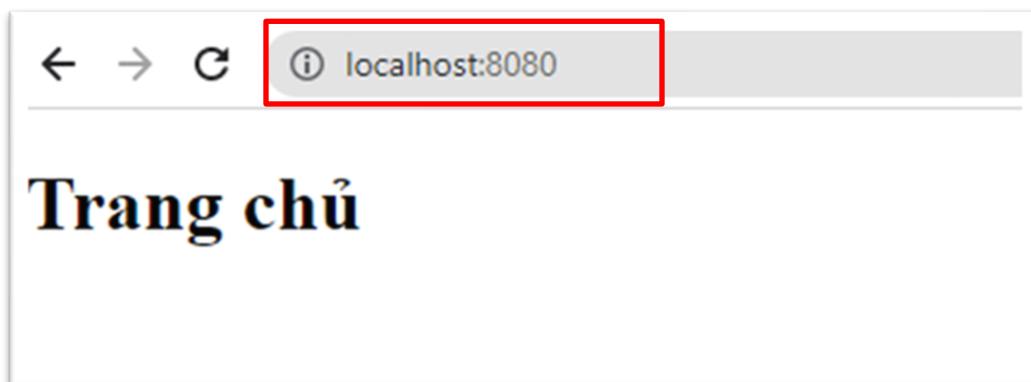


Hình 2.31. Tạo file index.html

Bổ sung đoạn code như bên dưới vào file **index.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>My App</title>
</head>
<body>
<h1>Trang chủ</h1>
</body>
</html>
```

Chạy ứng dụng web lên với địa chỉ **localhost:8080** và xem kết quả:



Hình 2.32. Giao diện trang chủ

2.5 Yêu cầu hoàn thành bài tập bổ sung sau

- Xây dựng các view có nội dung sau:

1. Trang chủ:

- ✓ Hiển thị dòng chữ: "Tôi là sinh viên Khoa Công nghệ thông tin HUTECH" trên màn hình trang chủ.

2. Danh sách môn học:

- ✓ Hiển thị danh sách các môn học thuộc chuyên ngành Công nghệ phần mềm Khoa Công nghệ thông tin Hutech.
- ✓ Thông tin về mỗi môn học bao gồm mã môn học, tên môn học và hình ảnh của môn học đó.

3. Trang liên hệ:

- ✓ Hiển thị một Form liên hệ trên trang liên hệ.
- ✓ Người dùng được yêu cầu nhập thông tin bao gồm Họ Tên, Email, Tin nhắn và Ngày Gửi (Ngày gửi sẽ tự động lấy ngày hiện tại).
- ✓ Các trường thông tin Họ Tên, Email và Tin nhắn được đánh dấu là bắt buộc (có thuộc tính required của thẻ input).
- ✓ Khi người dùng nhấn vào nút "Gửi thông tin", hệ thống sẽ xử lý và chuyển hướng người dùng sang một màn hình kết quả (trang "Kết quả").
- ✓ Trang "Kết quả" sẽ hiển thị thông tin liên hệ mà người dùng đã nhập.

Chú ý: Đảm bảo việc hiển thị các trường thông tin bắt buộc và xử lý đúng dữ liệu nhập vào từ người dùng.

TÓM TẮT

Spring là một framework được sử dụng để phát triển các ứng dụng web Java. Nó cung cấp các tính năng để xử lý các yêu cầu HTTP, định tuyến các URL đến các đối tượng xử lý và quản lý các biểu mẫu HTML và các tham số đầu vào khác được gửi từ trình duyệt web. Spring được xây dựng trên Spring Framework, một framework phổ biến cho phát triển ứng dụng Java.

Với Spring, các lập trình viên có thể phát triển các ứng dụng web Java một cách nhanh chóng và hiệu quả, bằng cách tận dụng các tính năng mạnh mẽ của framework để giảm thiểu công việc lập trình lặp đi lặp lại. Spring cũng hỗ trợ tích hợp với các công nghệ khác như Hibernate, JPA và Thymeleaf để đơn giản hóa việc xây dựng ứng dụng web.

Với Spring, các lập trình viên có thể xây dựng các ứng dụng web đa dạng, bao gồm các trang web tĩnh, trang web động, ứng dụng web phức tạp với nhiều chức năng, ứng dụng web thương mại điện tử và nhiều hơn nữa. Tóm lại, Spring là một trong những lựa chọn phổ biến cho phát triển ứng dụng web Java.

CÂU HỎI ÔN TẬP

1. Spring framework là gì? Tại sao nó lại quan trọng trong phát triển web Java?
2. Các công cụ hỗ trợ phát triển ứng dụng Spring Framework phổ biến là gì?
3. Các thành phần của Spring Framework gồm những gì?
4. Spring Boot có mối quan hệ như thế nào với Spring Framework?
5. Các tính năng chính của Spring Boot là gì?

BÀI 3 VIẾT CHƯƠNG TRÌNH QUẢN LÝ SÁCH

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

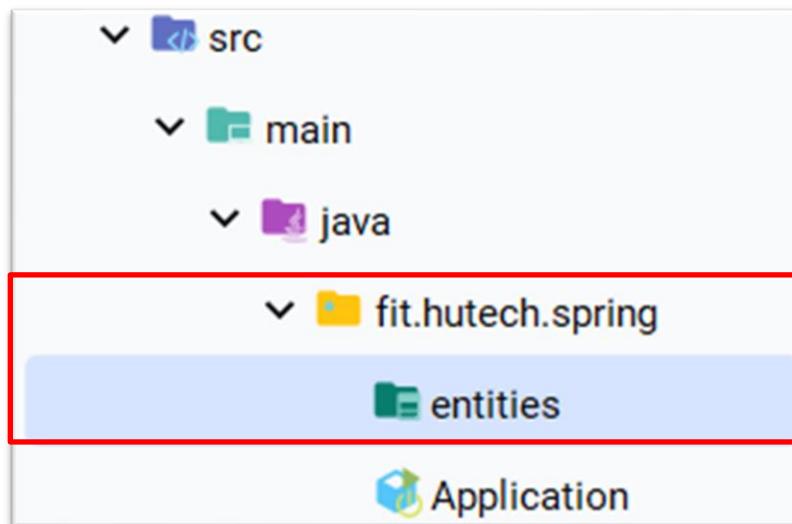
- Hiểu cách lập trình cơ bản để phát triển ứng dụng website bằng Java Spring Framework.
- Nắm vững cơ chế hoạt động của các thành phần chính như Entity, View và Controller.
- Thực hành thêm, xoá và sửa dữ liệu trong ứng dụng.
- Trở nên quen thuộc với việc sử dụng VCS (Version Control System) để lưu trữ mã nguồn.

Mô tả chức năng:

- ✓ **Thêm sách:** Cho phép người dùng nhập thông tin của một quyển sách gồm tiêu đề, tác giả, giá, danh mục của sách. Thông tin này sẽ được lưu vào một biến toàn cục trong ứng dụng
- ✓ **Xóa sách:** Cho phép người dùng chọn một quyển sách trong danh sách và xóa khỏi danh sách.
- ✓ **Sửa sách:** Cho phép người dùng chọn một quyển sách trong danh sách và cập nhật thông tin của nó.
- ✓ **Hiển thị sách:** Hiển thị danh sách tất cả các quyển sách có trong danh sách, bao gồm tiêu đề, tác giả, giá và danh mục của sách.

3.1 Xây dựng trang Book List

Tạo thêm 1 package với tên **entities** tại đường dẫn **fit.hutech.spring**



Hình 3.1. Thêm 1 Package với tên entities

Hãy tạo một file **Book.java** trong package **entities** và thêm các dữ liệu chứa thông tin về cuốn sách. Để tiện lợi cho việc khởi tạo **Getter**, **Setter** và **Constructor**, hãy sử dụng Lombok.

```
package fit.hutech.spring.entities;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class Book {
    private Long id;
    private String title;
    private String author;
    private Double price;
    private String category;
}
```

Hãy tạo một file **BookController.java** trong package **controllers** theo đường dẫn **src/main/java/fit.hutech.spring/controllers**. Sau đó, bổ sung đoạn code dưới đây vào file đó.

```
package fit.hutech.spring.controllers;

import fit.hutech.spring.entities.Book;
import fit.hutech.spring.services.BookService;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;

import java.util.List;

@Controller
@RequestMapping("/books")
@RequiredArgsConstructor
public class BookController {
    private final BookService bookService;

    @GetMapping
    public String showAllBooks(@NotNull Model model) {
        model.addAttribute("books", bookService.getAllBooks());
        return "book/list";
    }
}
```

Khởi tạo **AppConfig.java** tại đường dẫn **src/main/java/fit.hutech.spring** để khởi tạo danh sách các biến có thể truy cập từ toàn bộ ứng dụng. Sau đó, bổ sung đoạn mã code dưới đây vào file đó.

```
package fit.hutech.spring;

import fit.hutech.spring.entities.Book;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.util.ArrayList;
import java.util.List;

@Configuration
public class AppConfig {
```

```
@Bean
public List<Book> getBooks() {
    var books = new ArrayList<>();

    books.add(new Book(1L, "Lập trình Web Spring Framework", "Ánh
Nguyễn", 29.99, "Công nghệ thông tin"));
    books.add(new Book(2L, "Lập trình ứng dụng Java", "Huy Cường",
45.63, "Công nghệ thông tin"));
    books.add(new Book(3L, "Lập trình Web Spring Boot", "Xuân
Nhân", 12., "Công nghệ thông tin"));
    books.add(new Book(4L, "Lập trình Web Spring MVC", "Ánh
Nguyễn", .12, "Công nghệ thông tin"));

    return books;
}
```

Tạo thêm 1 file **BookService.java** trong package **services** theo đường dẫn sau: **src/main/java/ fit.hutech.spring/services**, và bổ sung những dòng code bên dưới vào như bên dưới.

```
package fit.hutech.spring.services;

import fit.hutech.spring.entities.Book;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
@RequiredArgsConstructor
public class BookService {
    private final List<Book> books;

    public List<Book> getAllBooks() {
        return books;
    }
}
```

3.1.1 Tạo view layout trong ứng dụng

Tải thư viện Bootstrap từ <https://getbootstrap.com/>

Compiled CSS and JS

Download ready-to-use compiled code for **Bootstrap v5.3.0-alpha1** to easily drop into your project, which includes:

- Compiled and minified CSS bundles (see [CSS files comparison](#))
- Compiled and minified JavaScript plugins (see [JS files comparison](#))

This doesn't include documentation, source files, or any optional JavaScript dependencies like Popper.

[Download](#)

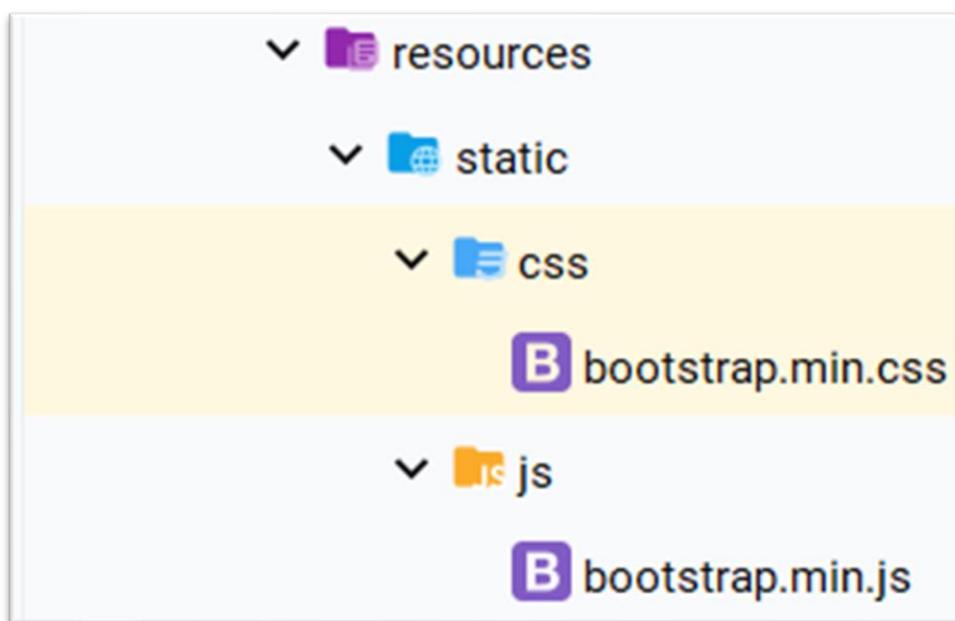
Hình 3.2. Tải thư viện Bootstrap

Sau khi tải xuống file từ trang **Bootstrap**, bạn cần giải nén và nó sẽ tạo ra hai thư mục là **css** và **js**. Hãy thực hiện các bước sau:

1. Tạo thư mục **css** trong đường dẫn **src/main/resources/static/css**.
2. Sao chép file **bootstrap.min.css** từ thư mục đã giải nén vào thư mục **css** mà bạn vừa tạo.
3. Tạo thư mục **js** trong đường dẫn **src/main/resources/static/js**.
4. Sao chép file **bootstrap.min.js** từ thư mục đã giải nén vào thư mục **js** mà bạn vừa tạo.

Sau khi hoàn thành, ta có thể sử dụng các tệp tin **bootstrap.min.css** và **bootstrap.min.js** từ thư mục **src/main/resources/static** trong ứng dụng của mình để sử dụng các tài nguyên Bootstrap.

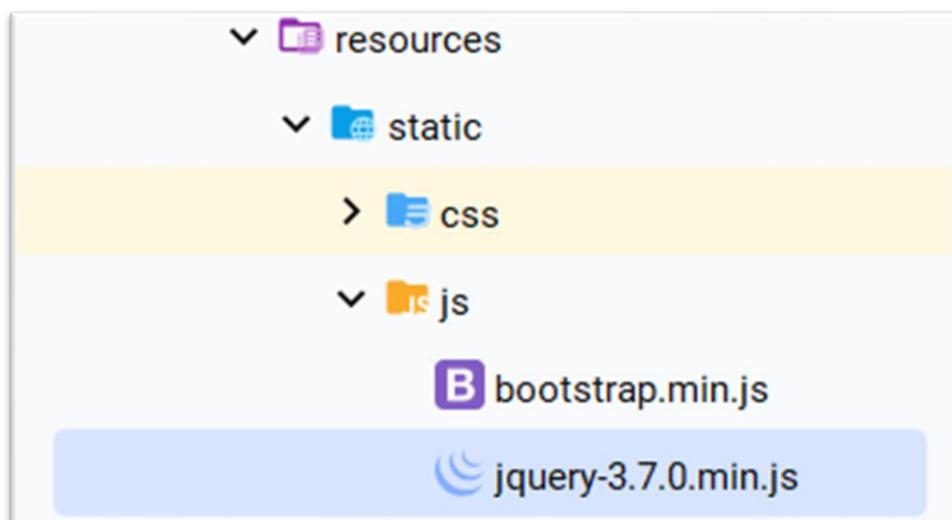
Dưới đây là kết quả sau khi thêm 2 file **bootstrap.min.css** và **bootstrap.min.js** vào project.



Hình 3.3. Thêm thư viện Bootstrap vào dự án

Sinh viên thực hiện tương tự tải và thêm thư viện Jquery tại <https://code.jquery.com/jquery-3.7.0.min.js> vào project?

Tạo 1 file **jquery-3.7.0.min.js** sau đó copy code trên đường dẫn vào file vừa tạo.



Hình 3.4. Thêm jquery-3.7.0.min.js

Tạo file custom css như sau. Khởi tạo **style.css** tại đường dẫn **src/main/resources/static/css**. Sau đó thêm nội dung dưới đây vào file.

```
nav{
    text-align: center;
}
.pagination{
    display: inline-block;
}

.footer {
    position: fixed;
    bottom: 0;
    width: 100%;
}
```

Tạo 1 file **layout.html** đặt tại đường dẫn **src/main/resources/templates**

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>My App</title>
    <link th:fragment="link-css" rel="stylesheet"
th:href="@{/css/bootstrap.min.css}">
    <link th:fragment="custom-css" rel="stylesheet"
th:href="@{/css/style.css}">
</head>
<body class="d-flex flex-column h-100">
<header th:fragment="header">
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
        <div class="container-fluid">
            <a class="navbar-brand" href="/">HUTECH</a>
            <button class="navbar-toggler" type="button" data-bs-
toggle="collapse"
                data-bs-target="#navbarSupportedContent"
                aria-controls="navbarSupportedContent"
                aria-expanded="false" aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse"
id="navbarSupportedContent">
                <ul class="navbar-nav me-auto mb-2 mb-lg-0">
```

```
<li class="nav-item">
    <a class="nav-link active" aria-current="page"
    href="/">Home</a>
</li>
<li class="nav-item"><a class="nav-link" href="/books">List
Book</a></li>
<li class="nav-item"><a class="nav-link"
    href="/books/add">Add Book</a></li>
</ul>
</div>
</div>
</nav>
</header>
<div class="container">
    <div th:fragment="content"></div>
</div>
<footer th:fragment="footer" class="footer mt-auto py-3 bg-light">
    <div class="container text-center">Copyright ©;
        <span th:text="#dates.year()"/></span>
        <a href="https://www.hutech.edu.vn/">HUTECH Education</a>
    </div>
    <script th:src="@{/js/bootstrap.min.js}"></script>
    <script th:src="@{/js/jquery-3.7.0.min.js}"></script>
</footer>
</body>
</html>
```

Tạo view hiển thị danh sách, tại thư mục **templates** tạo thêm 1 thư mục tên là **book**. Tạo thêm file có tên **list.html** trong thư mục **book** nằm tại đường dẫn **src/main/resources/templates/book**.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>My Book List</title>
    <th:block th:replace="~{layout::link-css}"></th:block>
    <th:block th:replace="~{layout::custom-css}"></th:block>
</head>
<body>
    <th:block th:replace="~{layout::header}"></th:block>
    <div class="container">
```

```

<table class="table">
  <tr>
    <th>ID</th>
    <th>Title</th>
    <th>Author</th>
    <th>Price</th>
    <th>Category</th>
    <th>Action</th>
  </tr>
  <tr th:each="book : ${books}">
    <td th:text="${book.getId()}"></td>
    <td th:text="${book.getTitle()}"></td>
    <td th:text="${book.getAuthor()}"></td>
    <td th:text="${book.getPrice()}"></td>
    <td th:text="${book.getCategory()}"></td>
    <td colspan="2">
      <a class="btn btn-primary"
          th:href="@{/books/edit/{id}(id=${book.getId()})}">Edit</a>
      <a class="btn btn-danger"
          th:href="@{/books/delete/{id}(id=${book.getId()})}"
          onclick="return confirm('Are you sure you want to delete
this book?')">Delete</a>
    </td>
  </tr>
</table>
</div>

<th:block th:replace="~{layout::footer}"></th:block>
</body>
</html>

```

Build và chạy ứng dụng xem kết quả, truy cập địa chỉ localhost:8080/books.

ID	Title	Author	Price	Category	Action
1	Lập trình Web Spring Framework	Ánh Nguyễn	29.99	Công nghệ thông tin	<button>Edit</button> <button>Delete</button>
2	Lập trình Ứng dụng Java	Huy Cường	45.63	Công nghệ thông tin	<button>Edit</button> <button>Delete</button>
3	Lập trình Web Spring Boot	Xuân Nhàn	12.0	Công nghệ thông tin	<button>Edit</button> <button>Delete</button>
4	Lập trình Web Spring MVC	Ánh Nguyễn	0.12	Công nghệ thông tin	<button>Edit</button> <button>Delete</button>

Hình 3.5. Kết quả, trang Book List

3.2 Xây dựng trang thêm sách

3.2.1 Xây dựng phương thức thêm sách

Tìm đến thư mục **controllers** và trong file **BookController.java** bổ sung phương thức thêm sách như sau.

```
@GetMapping("/add")
public String addBookForm(@NotNull Model model) {
    model.addAttribute("book", new Book());
    return "book/add";
}

@PostMapping("/add")
public String addBook(@ModelAttribute("book") Book book) {
    if(bookService.getBookById(book.getId()).isEmpty())
        bookService.addBook(book);
    return "redirect:/books";
}
```

Annotation **@GetMapping("/add")** trong Spring Framework được sử dụng để ánh xạ một phương thức xử lý yêu cầu **HTTP GET** tới một địa chỉ URL, cụ thể là **/add**. Khi một yêu cầu **HTTP GET** được gửi đến địa chỉ URL **/add**, phương thức **addBookForm()** sẽ được gọi để xử lý yêu cầu và trả về tên view là **book/add**.

Tương tự, annotation **@PostMapping("/add")** trong Spring Framework được sử dụng để ánh xạ một phương thức xử lý yêu cầu **HTTP POST** tới một địa chỉ URL, cụ thể là **/add**.

Với việc sử dụng các annotation này, bạn có thể xác định cách xử lý các yêu cầu **HTTP GET** và **POST** tới địa chỉ URL **/add** trong ứng dụng Spring.

Sau khi thêm phương thức thêm sách vào **Book controller**. Sau đó tìm đến thư mục **services** và thêm các phương thức sau vào file **BookService.java** như bên dưới.

```
public Optional<Book> getBookById(Long id) {
    return books.stream()
        .filter(book -> book.getId().equals(id))
        .findFirst();
}
```

```
public void addBook(Book book) {
    books.add(book);
}
```

Phương thức **getBookById(Long id)**: Phương thức này nhận vào một tham số là id kiểu Long và trả về một đối tượng Optional<Book>.

Phương thức **addBook(Book book)**: Phương thức này thêm một đối tượng Book vào danh sách books.

3.3 Thêm View hiển thị

Tạo file **add.html** đặt tại: **src/main/resources/templates/book**. Sau đó bổ sung các đoạn code dưới đây.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <th:block th:replace="~{layout::link-css}"></th:block>
    <th:block th:replace="~{layout::custom-css}"></th:block>
    <title>Add Book</title>
</head>
<body>
    <th:block th:replace="~{layout::header}"></th:block>
    <div class="container">
        <h1>Add Book</h1>
        <form th:action="@{/books/add}" th:object="${book}" method="post">
            <div class="col-6 mb-3">
                <label class="form-label" for="id">Id:</label>
                <input class="form-control" type="text" th:field="*{id}" id="id">
            </div>
            <div class="col-6 mb-3">
                <label class="form-label" for="title">Title:</label>
                <input class="form-control" type="text" th:field="*{title}" id="title">
            </div>
            <div class="col-6 mb-3">
                <label class="form-label" for="author">Author:</label>
                <input class="form-control" type="text" th:field="*{author}" id="author">
            </div>
        <div class="col-6 mb-3">
```

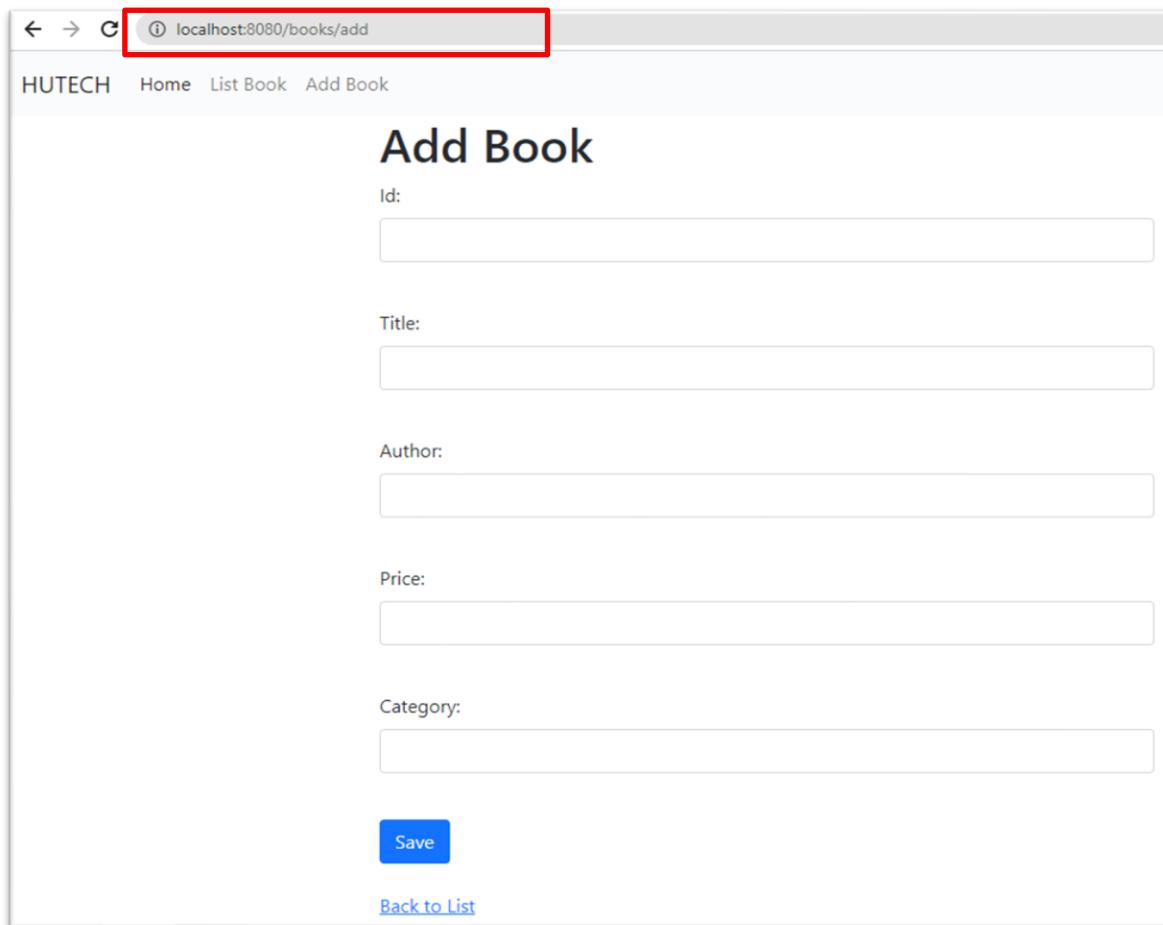
```
<label class="form-label" for="price">Price:</label>
<input class="form-control" type="text" th:field="*{price}"
id="price">
</div>
<div class="col-6 mb-3">
    <label class="form-label" for="category">Category:</label>
    <input class="form-control" type="text" th:field="*{category}"
id="category">
</div>
<input type="submit" class="btn btn-primary" value="Save">
</form>
<br>
<a th:href="@{/books}">Back to List</a>
</div>
<th:block th:replace="~{layout::footer}"></th:block>
</body>
</html>
```

Chỉnh sửa nội dung trang chủ **index.html** tại đường dẫn sau:

src/resources/templates/home/index.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>ỨNG DỤNG QUẢN LÝ SÁCH</title>
    <th:block th:replace="~{layout::link-css}"></th:block>
    <th:block th:replace="~{layout::custom-css}"></th:block>
</head>
<body>
<th:block th:replace="~{layout::header}"></th:block>
<div class="container">
    <nav class="navbar navbar-light bg-light">
        <div class="container-fluid">
            <a class="navbar-brand" href="#">
                TRANG CHỦ ỨNG DỤNG QUẢN LÝ SÁCH
            </a>
        </div>
    </nav>
</div>
<th:block th:replace="~{layout::footer}"></th:block>
</body>
</html>
```

Build và chương trình tại địa chỉ **localhost:8080/books/add**.



The screenshot shows a web browser window with the following details:

- Address Bar:** localhost:8080/books/add (highlighted with a red box).
- Header:** HUTECH Home List Book Add Book
- Title:** Add Book
- Fields:**
 - Id:** An input field.
 - Title:** An input field.
 - Author:** An input field.
 - Price:** An input field.
 - Category:** An input field.
- Buttons:**
 - A blue **Save** button.
 - A link labeled **Back to List**.

Hình 3.6. Giao diện trang thêm book

Để thêm một cuốn sách, ta sẽ nhập thông tin đầy đủ vào các trường dữ liệu và nhấn nút Save và xem kết quả. Sau khi chạy, bạn sẽ nhận được thông báo về kết quả thêm sách.

HUTECH Home List Book Add Book

Add Book

Id:

Title:

Author:

Price:

Category:

Save

[Back to List](#)

Hình 3.7. Thêm 1 cuốn sách

Kết quả nhận được là một cuốn sách vừa thêm vào cuối danh sách. Như vậy bạn đã hoàn thành chức năng thêm sách mới.

HUTECH Home List Book Add Book

ID	Title	Author	Price	Category	Action
1	Lập trình Web Spring Framework	Ánh Nguyễn	29.99	Công nghệ thông tin	Edit Delete
2	Lập trình Ứng dụng Java	Huy Cường	45.63	Công nghệ thông tin	Edit Delete
3	Lập trình Web Spring Boot	Xuân Nhân	12.0	Công nghệ thông tin	Edit Delete
4	Lập trình Web Spring MVC	Ánh Nguyễn	0.12	Công nghệ thông tin	Edit Delete
5	Lập trình Java	Nguyễn Xuân Nhân	90.0	Công nghệ thông tin	Edit Delete

Hình 3.8. Trang danh sách Book sau khi thêm Book mới

3.4 Xây dựng trang sửa sách

Thêm 1 file **edit.html** đặt trong thư mục **src/resources/templates/book**. Sau đó bổ sung các đoạn code dưới đây.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <th:block th:replace="~{layout::link-css}"></th:block>
    <th:block th:replace="~{layout::custom-css}"></th:block>
    <title>Edit Book</title>
</head>
<body>
<th:block th:replace="~{layout::header}"></th:block>
<div class="container">
    <h1>Edit Book</h1>
    <form th:action="@{/books/edit}" th:object="${book}" method="post">
        <input type="hidden" th:field="*{id}">
        <div class="col-6 mb-3">
            <label class="form-label" for="title">Title:</label>
            <input class="form-control" type="text" th:field="*{title}" id="title">
        </div>
        <div class="col-6 mb-3">
            <label class="form-label" for="author">Author:</label>
            <input class="form-control" type="text" th:field="*{author}" id="author">
        </div>
        <div class="col-6 mb-3">
            <label class="form-label" for="price">Price:</label>
            <input class="form-control" type="text" th:field="*{price}" id="price">
        </div>
        <div class="col-6 mb-3">
            <label class="form-label" for="category">Category:</label>
            <input class="form-control" type="text" th:field="*{category}" id="category">
        </div>
        <input type="submit" value="Save">
    </form>
    <br>
    <a th:href="@{/books}">Back to List</a>
```

```
</div>
<th:block th:replace="~{layout::footer}"></th:block>
</body>
</html>
```

Bổ sung các phương thức cập nhật sách vào file **BookController.java**

Thêm 1 hàm:

editbookForm(), với 1 annotation **@PostMapping("/edit/{id}")**

Tiếp, Thêm 1 hàm **editbook()**, với 1 annotation **@PostMapping("/edit")**

```
@GetMapping("/edit/{id}")
public String editBookForm(@NotNull Model model, @PathVariable long id)
{
    var book = bookService.getBookById(id).orElse(null);
    model.addAttribute("book", book != null ? book : new Book());
    return "book/edit";
}

@PostMapping("/edit")
public String editBook(@ModelAttribute("book") Book book) {
    bookService.updateBook(book);
    return "redirect:/books";
}
```

Thêm phương thức sửa sách vào file **BookService.java**

Thêm 1 hàm: **updateBook()** nhằm cập nhật thông tin sách

```
public void updateBook(Book book) {
    var bookOptional = getBookById(book.getId());
    if (bookOptional.isPresent()) {
        Book bookUpdate = bookOptional.get();
        bookUpdate.setTitle(book.getTitle());
        bookUpdate.setAuthor(book.getAuthor());
        bookUpdate.setPrice(book.getPrice());
        bookUpdate.setCategory(book.getCategory());
    }
}
```

Chạy ứng dụng với tại địa chỉ localhost:8080/books

ID	Title	Author	Price	Category	Action
1	Lập trình Web Spring Framework	Ánh Nguyễn	29.99	Công nghệ thông tin	<button>Edit</button> <button>Delete</button>
2	Lập trình Ứng dụng Java	Huy Cường	45.63	Công nghệ thông tin	<button>Edit</button> <button>Delete</button>
3	Lập trình Web Spring Boot	Xuân Nhân	12.0	Công nghệ thông tin	<button>Edit</button> <button>Delete</button>
4	Lập trình Web Spring MVC	Ánh Nguyễn	0.12	Công nghệ thông tin	<button>Edit</button> <button>Delete</button>

Hình 3.9. Giao diện danh sách các cuốn với nút Edit

Chọn 1 cuốn sách muốn **Edit**, xem đường dẫn như bên dưới, chỉnh sửa thông tin và kiểm tra kết quả sau đó nhấn nút **save**, như vậy bạn đã hoàn thành chức năng Edit Book.

localhost:8080/books/edit/1

HUTECH Home List Book Add Book

Edit Book

Title: Lập trình Web Spring Framework

Author: Ánh Nguyễn

Price: 99.99

Category: Công nghệ thông tin

[Save](#)

[Back to List](#)

Hình 3.10. Giao diện trang chỉnh sửa Book

Kết quả sau khi hoàn tất chỉnh sửa thông tin một cuốn sách.

ID	Title	Author	Price	Category	Action
1	Lập trình Web Spring Framework	Ánh Nguyễn	99.99	Công nghệ thông tin	<button>Edit</button> <button>Delete</button>
2	Lập trình Ứng dụng Java	Huy Cường	45.63	Công nghệ thông tin	<button>Edit</button> <button>Delete</button>
3	Lập trình Web Spring Boot	Xuân Nhân	12.0	Công nghệ thông tin	<button>Edit</button> <button>Delete</button>
4	Lập trình Web Spring MVC	Ánh Nguyễn	0.12	Công nghệ thông tin	<button>Edit</button> <button>Delete</button>

Hình 3.11. Kết quả sau khi chỉnh sửa thành công, trả về Book List

3.5 Thực hiện chức năng xóa sách

Thêm 1 phương thức tên **deleteBook()** bên trong file **BookController.java** để xóa cuốn sách dựa trên ID

Chỉnh sửa file **BookController.java** như bên dưới với **@PostMapping("delete/{id}")**, **Id** của cuốn sách cần xoá sẽ được truyền vào hàm **deleteBook()**.

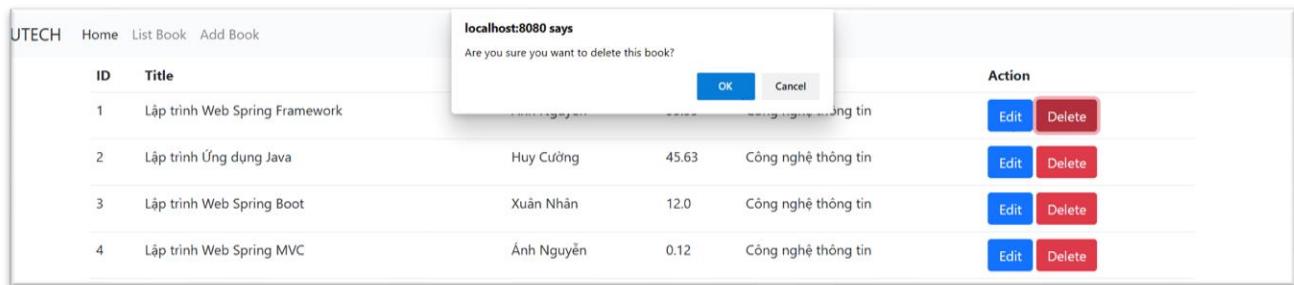
```
@GetMapping("/delete/{id}")
public String deleteBook(@PathVariable long id) {
    if (bookService.getBookById(id).isPresent())
        bookService.deleteBookById(id);
    return "redirect:/books";
}
```

Tiếp tục trong file **BookService.java** ta thêm phương thức xoá sách

```
public void deleteBookById(Long id) {
    getBookById(id).ifPresent(books::remove);
}
```

Sau khi đã build và chạy ứng dụng, bạn có thể thực hiện các bước sau để kiểm tra kết quả xoá cuốn sách từ giao diện Book List:

Truy cập vào giao diện Book List trong ứng dụng của bạn. Tìm và chọn cuốn sách mà bạn muốn xoá. Sau khi chọn cuốn sách, một thông báo xác nhận xoá sẽ xuất hiện dưới dạng một cửa sổ Alert hoặc hộp thoại tương tự. Thông báo này sẽ yêu cầu bạn xác nhận việc xoá cuốn sách.



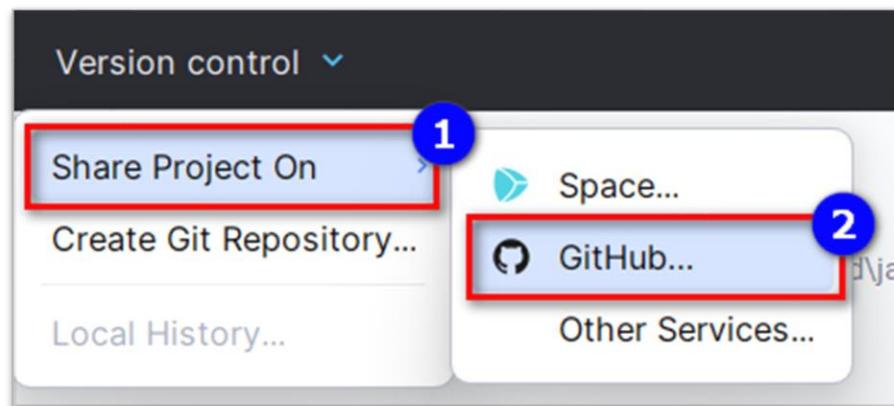
Hình 3.12. Giao diện delete Book

3.6 Hướng dẫn lấy và đưa mã nguồn lên Github

Để đưa mã nguồn lên Github, yêu cầu sinh viên phải có tài khoản Github. Để tạo tài khoản Github sinh viên vui lòng truy cập trang tạo tài khoản tại đường dẫn <https://github.com/signup?source=login>.

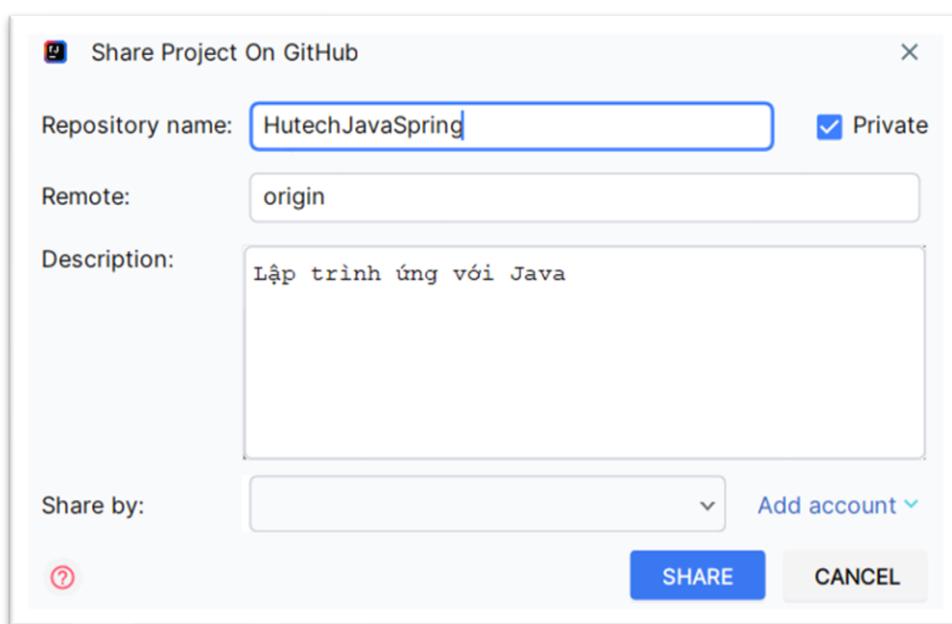
Đầu tiên, khởi tạo kho lưu trữ mã nguồn (repository) trên Github theo các bước như nhau:

Bước 1: Tại IntelliJ IDEA, chọn **Version Control (VCS)** → **Share Project On** → **Github**.



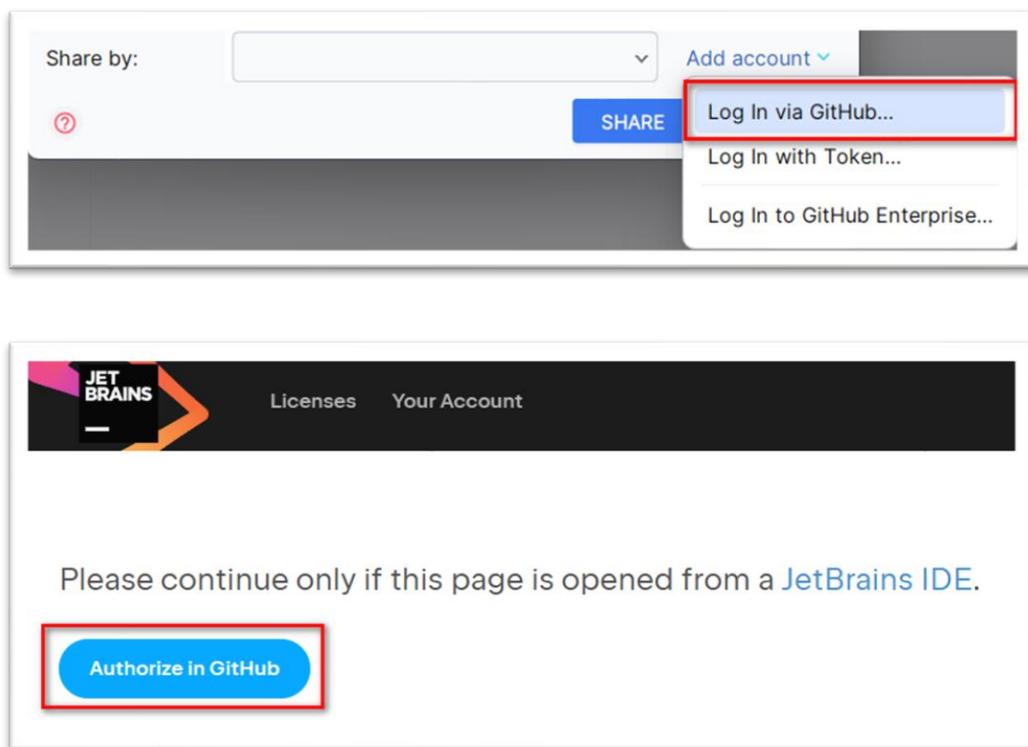
Hình 3.13. Chia sẻ mã nguồn lên Github

Bước 2: Nhập các thông tin cơ bản về kho lưu trữ mã nguồn



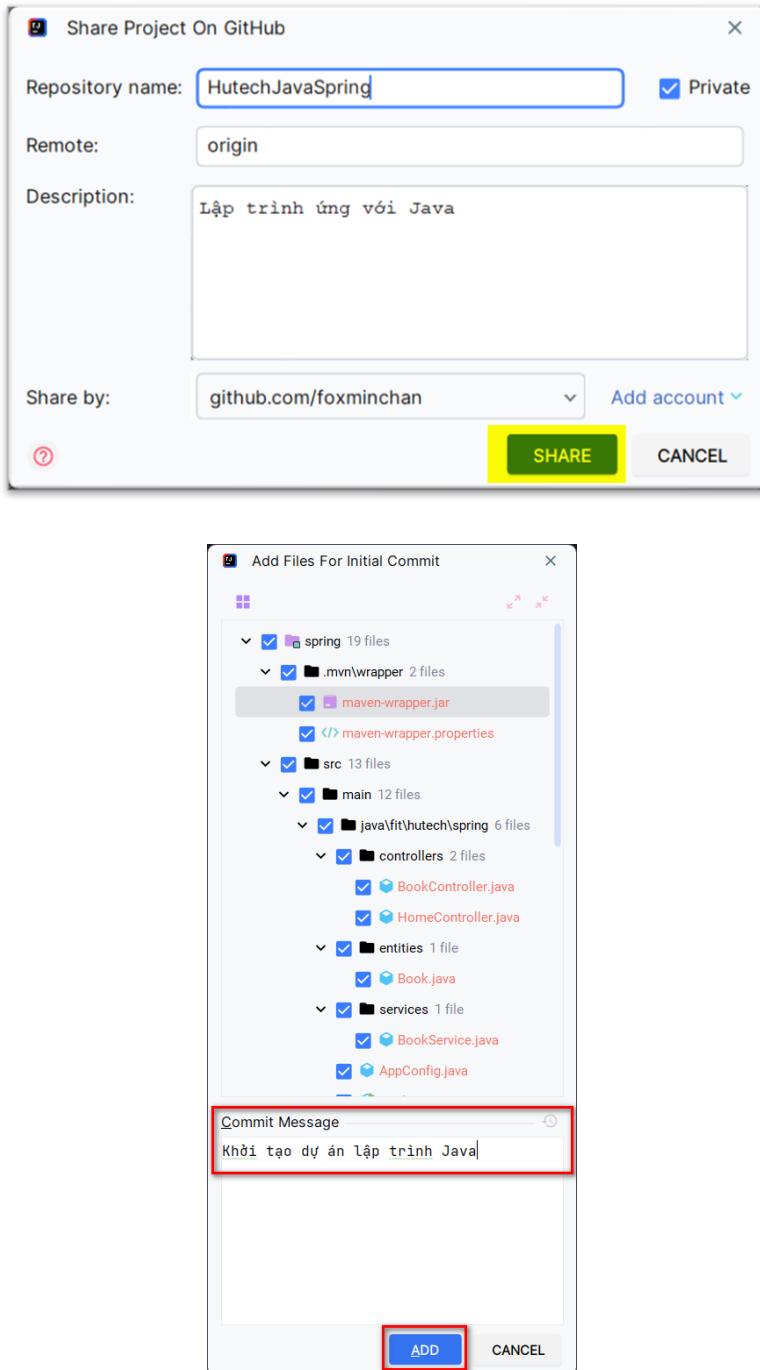
Hình 3.14. Nhập các thông tin cơ bản của kho lưu trữ

Bước 3: Kết nối IntelliJ IDEA với Github. Chọn **Add account** → **Log In via Github...**. Sau đó chọn **Authorize in GitHub**.



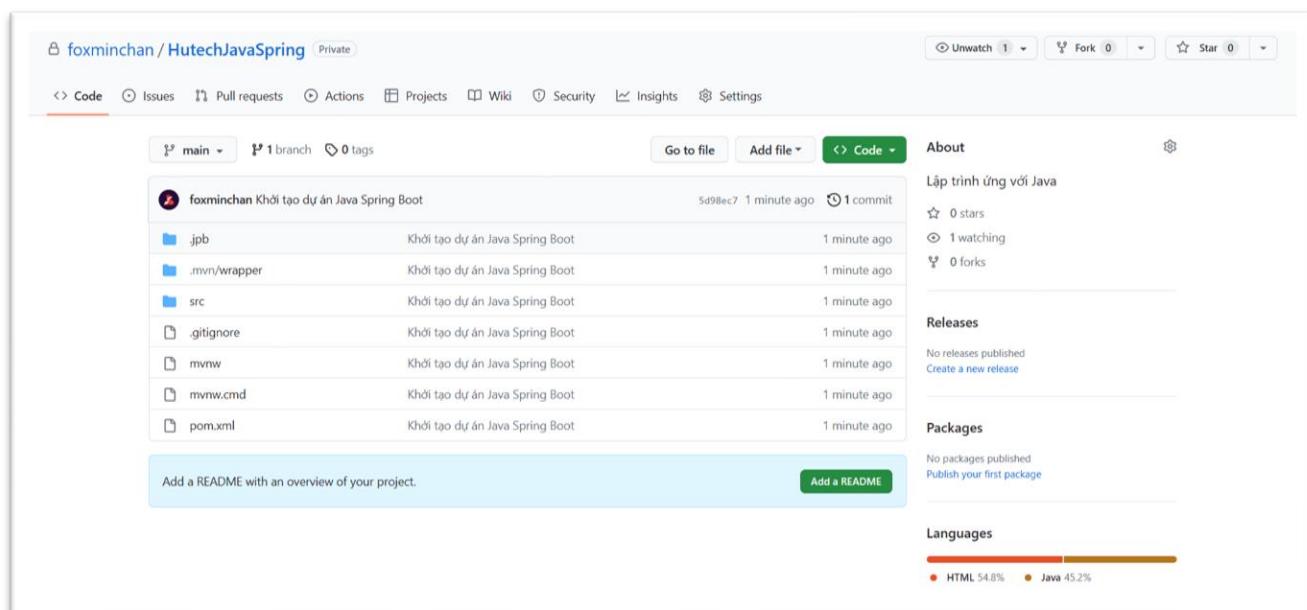
Hình 3.15: Kết nối IntelliJ IDEA với Github

Bước 4: Nhấn **share**, sau đó nhập *nội dung commit* và nhấn **Add** để đưa mã nguồn lên Github



Hình 3.16. Đẩy mã nguồn lên Github

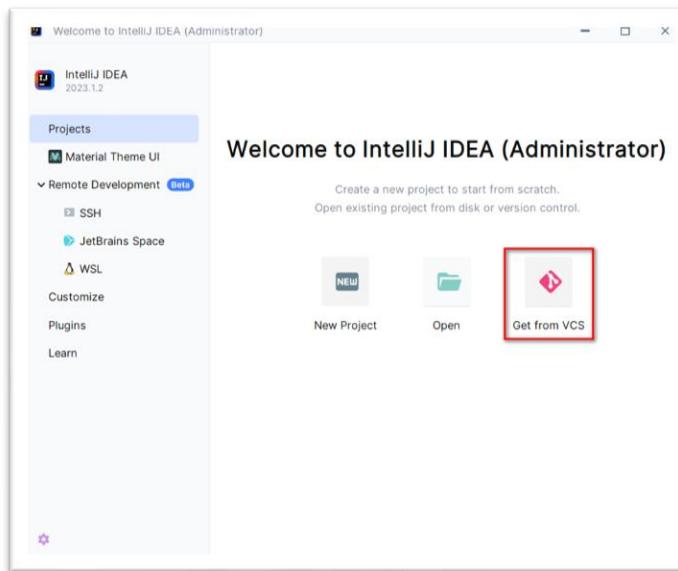
Bước 5: Kiểm tra kết quả vừa thực hiện được



Hình 3.17. Kết quả sau khi đưa mã nguồn lên Github

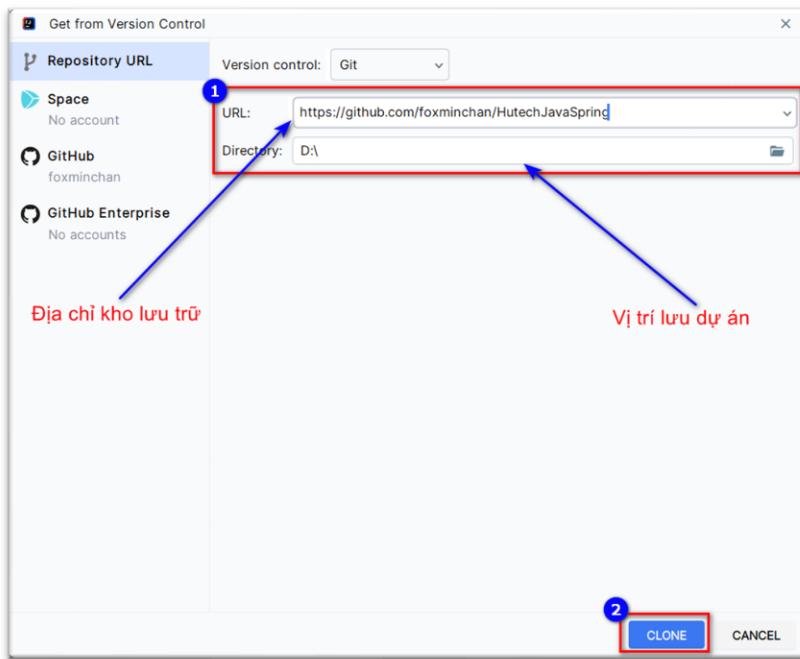
Để sao chép (clone) mã nguồn từ **Github** về máy tính thì sinh viên thực hiện theo các bước như sau:

Bước 1: Chọn Get from VCS

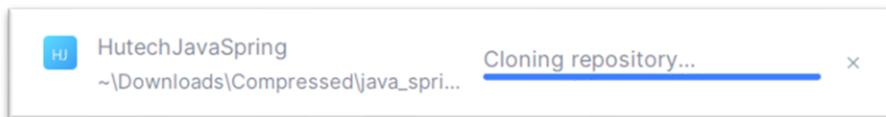


Hình 3.18. Lấy mã nguồn từ Github về máy tính

Bước 2: Nhập thông tin gồm địa chỉ repository và vị trí lưu project. Sau đó ấn **Clone**



Hình 3.19. Nhập thông tin kho lưu trữ cần lấy mã nguồn



Hình 3.20. Quá trình lấy mã nguồn

Thông qua các hướng dẫn trên, sinh viên đã được hướng dẫn về cách đẩy mã nguồn lên Github và sao chép mã nguồn về máy tính. Sinh viên có thể áp dụng các hướng dẫn này sau mỗi buổi học để đưa mã nguồn lên Github và tải về vào đầu buổi học tiếp theo. Điều này giúp tránh tình trạng không có mã nguồn để thực hiện các bài thực hành sau này.

TÓM TẮT

Java Spring Framework là một framework phát triển ứng dụng web được viết bằng Java. Nó cung cấp các công cụ và thư viện để phát triển ứng dụng web hiệu quả và nhanh chóng. Spring Framework có nhiều tính năng mạnh mẽ bao gồm Dependency Injection, AOP, MVC, JDBC, Security và nhiều hơn nữa.

Một trong những tính năng quan trọng của Spring Framework là cách tiếp cận quan điểm đối với cấu hình. Nó cung cấp các giá trị mặc định hợp lý và tự động cấu hình, giảm thiểu nhu cầu cấu hình thủ công. Điều này cho phép các nhà phát triển tập trung hơn vào việc viết logic kinh doanh và ít hơn vào việc thiết lập cơ sở hạ tầng ứng dụng.

Hơn nữa, Spring Framework ủng hộ việc tuân thủ quy ước hơn là cấu hình. Bằng cách tuân thủ một số quy ước về đặt tên và cấu trúc, bạn có thể giảm thiểu lượng cấu hình cần thiết.

Spring Framework hỗ trợ các loại ứng dụng khác nhau, bao gồm các ứng dụng web, dịch vụ RESTful và ứng dụng xử lý hàng loạt. Nó cũng cung cấp hỗ trợ đáng kể cho việc xây dựng các dịch vụ nhỏ, cho phép

CÂU HỎI ÔN TẬP

1. Làm thế nào để xử lý các yêu cầu HTTP GET, POST, PUT, DELETE trong một Controller?
2. Các đặc điểm chính của một Java Bean là gì? Vai trò của Java Bean?
3. Tại sao Dependency Injection (DI) là một khái niệm quan trọng trong Spring Framework?
4. Các tính năng chính của Thymeleaf là gì và tại sao nó trở thành lựa chọn phổ biến cho việc phát triển giao diện người dùng trong Java?
5. Lombok có hỗ trợ việc tạo Builder Pattern trong Java không? Nếu có, làm thế nào để sử dụng nó?

BÀI 4 THAO TÁC VỚI CƠ SỞ DỮ LIỆU

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

- Làm quen thao tác thêm, xoá sửa dữ liệu trên cơ sở dữ liệu
- Làm quen với trình Quản trị cơ sở dữ liệu (DBSM) MySQL phpMyAdmin
- Kết nối cơ sở dữ liệu Project với phpMyAdmin
- Làm cơ sở xây dựng ứng dụng website quản lý sách với các chức năng giao tiếp cơ bản với Cơ sở dữ liệu: chức năng thêm, chức năng xóa, chức năng sửa.

Mô tả chức năng:

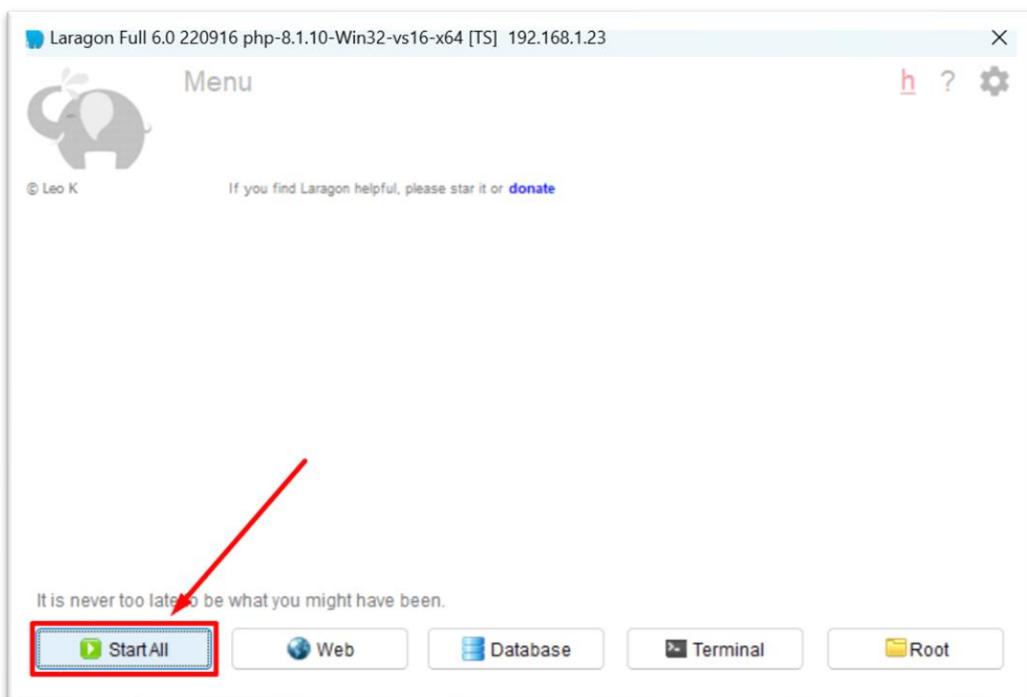
- ✓ **Thêm sách:** Cho phép người dùng nhập thông tin của một quyển sách gồm tiêu đề, tác giả, giá, danh mục của sách. Thông tin này sẽ được lưu vào cơ sở dữ liệu.
- ✓ **Xóa sách:** Cho phép người dùng chọn một quyển sách trong cơ sở dữ liệu hiển thị và xóa khỏi danh sách.
- ✓ **Sửa sách:** Cho phép người dùng chọn một quyển sách trong cơ sở dữ liệu hiển thị và cập nhật thông tin của nó.
- ✓ **Hiển thị sách:** Hiển thị danh sách tất cả các quyển sách có trong cơ sở dữ liệu, bao gồm tiêu đề, tác giả, giá và danh mục của sách.
- ✓ **Phân trang và sắp xếp:** Để cải thiện trải nghiệm người dùng và xử lý hiệu quả với số lượng sách lớn, ứng dụng của bạn hỗ trợ phân trang. Điều này cho phép hiển thị các quyển sách theo từng trang, giới hạn số lượng sách được hiển thị trên mỗi trang. Để cải thiện khả năng tìm kiếm và sắp xếp dữ liệu, ứng dụng của bạn hỗ trợ chức năng sắp xếp sách. Người dùng có thể sắp xếp danh sách sách theo mã, tên, giá...
- ✓ **Giỏ hàng:** Lưu và quản lý các sản phẩm đã chọn trước khi tiến hành thanh toán vào Session.

4.1 Khởi động phpMyAdmin – Kết nối CSDL

Mở chương trình quản lý cơ sở dữ liệu phù hợp, trong hướng dẫn này dùng chương trình quản lý cơ sở dữ liệu **phpMyAdmin** (*Tham khảo lại Lab thực hành 01 cách cài đặt phpMyAdmin*).

Khởi động phần mềm **Laragon**, nhấn **Start All** để khởi động để bật các dịch vụ Server.

Lưu ý: Nếu như gặp sự cố khi khởi động webserver và không thể truy cập vào nó do xung đột cổng (port). Sinh viên vui lòng thay đổi cổng (port) của webserver trong setting.



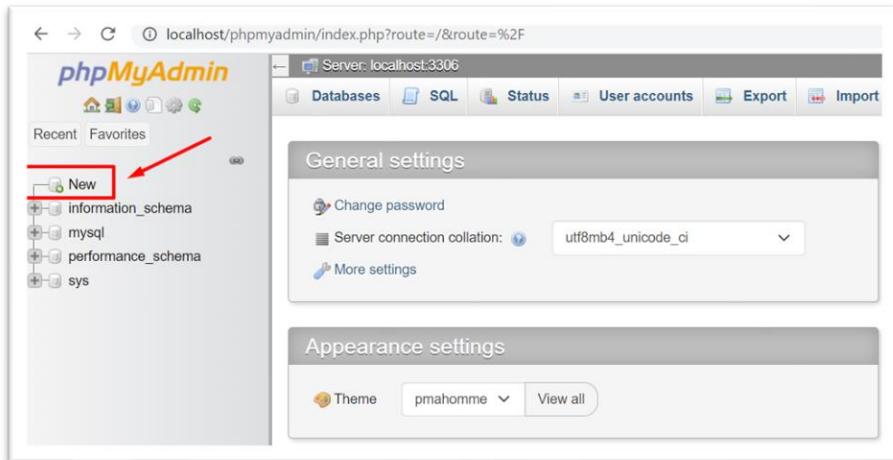
Hình 4.1. Khởi động phpMyAdmin

Truy cập địa chỉ website: <http://localhost/phpmyadmin/> , nhập tài khoản

Username: root

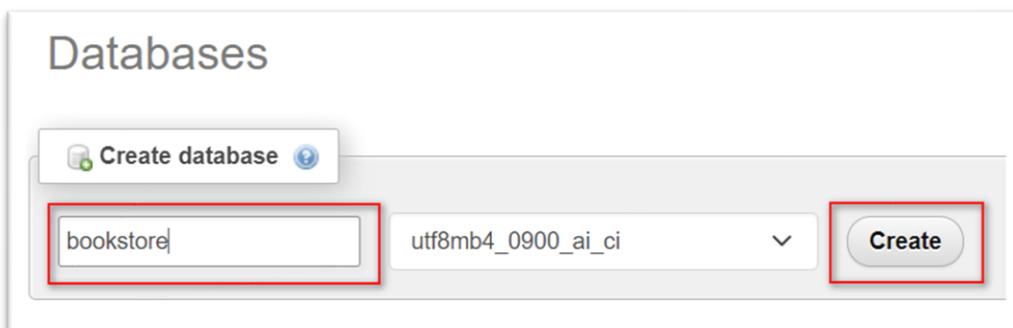
Password: "để trống"

Tiếp theo, tạo CSDL có tên **bookstore**, Chọn **New** ở góc bên trái.



Hình 4.2. Tạo mới CSDL trong phpMyAdmin

Đặt tên cơ sở dữ liệu và tạo, nhớ chọn đúng **utf8mb4_0900_ai_ci**



Hình 4.3. Đặt tên cho CSDL trong phpMyAdmin mới và tạo

Tiếp theo, mở dự án lên và tìm đến file **application.properties** và cấu hình lại như bên dưới:

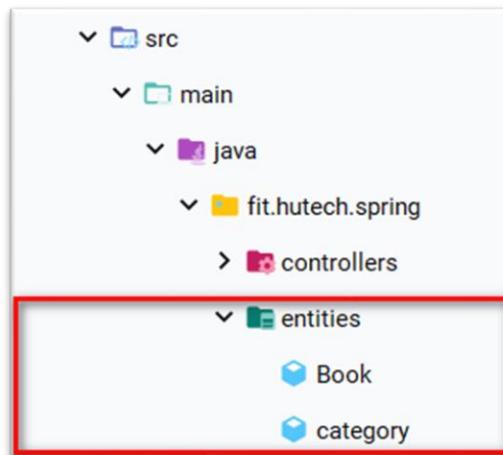
```
# Database connection properties
spring.datasource.url=jdbc:mysql://localhost:3306/bookstore
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# Hibernate properties
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect

#Spring Security
spring.autoconfigure.exclude=org.springframework.boot.autoconfigure.security.servlet.SecurityAutoConfiguration
```

4.2 Khởi tạo các đối tượng

Tạo 2 file model **Book.java** và **Category.java** đặt tại thư mục **entities** với đường dẫn sau **src/main/java/fit.hutech.spring/entities**.



Hình 4.4. Tạo 2 file Book.java và Category.java trong thư mục entity

Tiến hành thêm đoạn code khai báo các thuộc tính như bên dưới cho 2 file **Book.java** và **Category.java**. Ngoài ra, sinh viên có thể sử dụng công cụ **JPA Buddy** để hỗ trợ hướng dẫn các triển khai phổ biến nhất của JPA.

Book.java

```
package fit.hutech.spring.entities;

import jakarta.persistence.*;
import lombok.*;
import org.hibernate.Hibernate;

import java.util.Objects;

@Getter
@Setter
@ToString
@RequiredArgsConstructor
@AllArgsConstructor
@Entity
@Builder
@Table(name = "book")
public class Book {
```

```

@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;

@Column(name = "title", length = 50, nullable = false)
private String title;

@Column(name = "author", length = 50, nullable = false)
private String author;

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

@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "category_id", referencedColumnName = "id")
@ToString.Exclude
private Category category;

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || Hibernate.getClass(this) !=
Hibernate.getClass(o)) return false;
    Book book = (Book) o;
    return getId() != null && Objects.equals(getId(),
book.getId());
}

@Override
public int hashCode() {
    return getClass().hashCode();
}
}

```

Category.java

```

package fit.hutech.spring.entities;

import jakarta.persistence.*;
import lombok.*;
import org.hibernate.Hibernate;

import java.util.ArrayList;
import java.util.List;

```

```
import java.util.Objects;

@Getter
@Setter
@ToString
@RequiredArgsConstructor
@AllArgsConstructor
@Builder
@Entity
@Table(name = "category")
public class Category {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long id;

    @Column(name = "name", length = 50, nullable = false)
    private String name;

    @OneToMany(mappedBy = "category", cascade = CascadeType.ALL)
    @ToString.Exclude
    private List<Book> books = new ArrayList<>();

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || Hibernate.getClass(this) != Hibernate.getClass(o)) return false;
        Category category = (Category) o;
        return getId() != null && Objects.equals(getId(),
category.getId());
    }

    @Override
    public int hashCode() {
        return getClass().hashCode();
    }
}
```

Tiếp theo tạo thêm 2 file IBookRepository.java và ICategoryRepository.java đặt tại thư mục **src/main/java/fit.hutech.spring/repositories**

IBookRepository.java và **ICategoryRepository.java** là các file interface trong một ứng dụng phần mềm quản lý sách. Chúng được sử dụng để tương tác với

các đối tượng sách (books) và các đối tượng danh mục (categories) trong cơ sở dữ liệu.

IBookRepository.java cung cấp các phương thức để thêm, sửa, xóa, lấy danh sách các cuốn sách trong cơ sở dữ liệu và phân trang.

Code nội dung vào file **IBookRepository.java**

```
package fit.hutech.spring.repositories;

import fit.hutech.spring.entities.Book;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.repository.PagingAndSortingRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface IBookRepository extends
PagingAndSortingRepository<Book, Long>, JpaRepository<Book, Long> {
    default List<Book> findAllBooks(Integer pageNo,
                                    Integer pageSize,
                                    String sortBy) {
        return findAll(PageRequest.of(pageNo,
                                      pageSize,
                                      Sort.by(sortBy)))
            .getContent();
    }
}
```

ICategoryRepository.java cung cấp các phương thức để thêm, sửa, xóa, lấy danh sách các danh mục trong cơ sở dữ liệu.

Code nội dung vào file **ICategoryRepository.java**

```
package fit.hutech.spring.repositories;

import fit.hutech.spring.entities.Category;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface ICategoryRepository extends
JpaRepository<Category, Long> {
```

Tạo file **BookService.java** được đặt tại thư mục theo đường dẫn sau đây:

src/main/java/fit.hutech.spring/services

```
package fit.hutech.spring.services;

import fit.hutech.spring.entities.Book;
import fit.hutech.spring.repositories.IBookRepository;
import jakarta.validation.constraints.NotNull;
import lombok.RequiredArgsConstructor;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Sort;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Isolation;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;
import java.util.Objects;
import java.util.Optional;

@Service
@RequiredArgsConstructor
@Transactional(isolation = Isolation.SERIALIZABLE,
    rollbackFor = {Exception.class, Throwable.class})
public class BookService {
    private final IBookRepository bookRepository;

    public List<Book> getAllBooks(Integer pageNo,
        Integer pageSize,
        String sortBy) {
        return bookRepository.findAllBooks(pageNo, pageSize, sortBy);
    }

    public Optional<Book> getBookById(Long id) {
        return bookRepository.findById(id);
    }

    public void addBook(Book book) {
        bookRepository.save(book);
    }

    public void updateBook(@NotNull Book book) {
        Book existingBook = bookRepository.findById(book.getId())
            .orElse(null);
        Objects.requireNonNull(existingBook).setTitle(book.getTitle());
        existingBook.setAuthor(book.getAuthor());
        existingBook.setPrice(book.getPrice());
        existingBook.setCategory(book.getCategory());
        bookRepository.save(existingBook);
    }
}
```

```

    }

    public void deleteBookById(Long id) {
        bookRepository.deleteById(id);
    }
}

```

Trong lớp **BookService.java**, thường sẽ có các phương thức để **thêm, sửa, xóa, lấy danh sách các cuốn sách**, cùng với việc tìm kiếm các cuốn sách theo các tiêu chí khác nhau (ví dụ: tên sách, tác giả, danh mục). Lớp này thường sử dụng **IBookRepository.java** để tương tác với cơ sở dữ liệu và thực hiện các tác vụ liên quan đến sách.

Các phương thức trong lớp **BookController.java** thường được thiết kế để xử lý các yêu cầu từ người dùng *như thêm, sửa, xóa, hiển thị thông tin chi tiết, phân trang và tìm kiếm*.

Tạo file **BookController.java** đặt tại **src/main/java/fit.hutech.spring**

```

package fit.hutech.spring.controllers;

import fit.hutech.spring.entities.Book;
import fit.hutech.spring.services.BookService;
import jakarta.validation.constraints.NotNull;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

@Controller
@RequestMapping("/books")
@RequiredArgsConstructor
public class BookController {
    private final BookService bookService;

    @GetMapping
    public String showAllBooks(
        @NotNull Model model,
        @RequestParam(defaultValue = "0") Integer pageNo,
        @RequestParam(defaultValue = "20") Integer pageSize,
        @RequestParam(defaultValue = "id") String sortBy) {
        model.addAttribute("books", bookService.getAllBooks(pageNo,
pageNo, pageSize, sortBy));
        model.addAttribute("currentPage", pageNo);
    }
}

```

```
        model.addAttribute("categories",
categoryService.getAllCategories());
        model.addAttribute("totalPages",
bookService.getAllBooks(pageNo, pageSize, sortBy).size() / pageSize);
        return "book/list";
    }
}
```

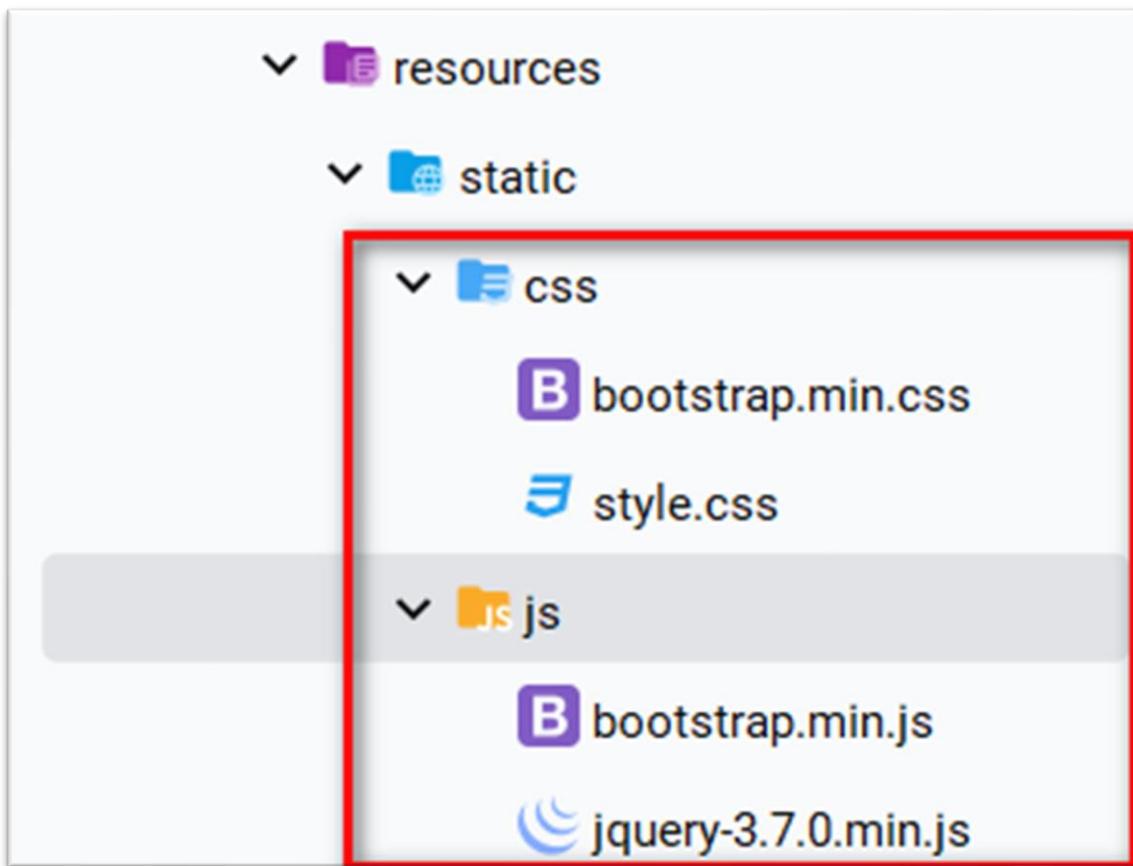
4.3 Khởi tạo các thành phần giao diện

Tạo 1 file **layout.html** đặt tại đường dẫn src/main/resources/templates. Mục đích của file layout.html là cung cấp một layout chung mà các trang giao diện khác có thể kế thừa và sử dụng. Thông qua file layout.html, bạn có thể định nghĩa các phần tử giao diện chung như header, footer, menu, hoặc các thành phần khác mà bạn muốn xuất hiện trên tất cả các trang giao diện của ứng dụng.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>My App</title>
    <link th:fragment="link-css" rel="stylesheet"
th:href="@{/css/bootstrap.min.css}">
    <link th:fragment="custom-css" rel="stylesheet"
th:href="@{/css/style.css}">
</head>
<body class="d-flex flex-column h-100">
<header th:fragment="header">
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
        <div class="container-fluid">
            <a class="navbar-brand" href="/">HUTECH</a>
            <button class="navbar-toggler" type="button" data-bs-
toggle="collapse"
                data-bs-target="#navbarSupportedContent"
                aria-controls="navbarSupportedContent"
                aria-expanded="false" aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse"
id="navbarSupportedContent">
                <ul class="navbar-nav me-auto mb-2 mb-lg-0">
```

```
<li class="nav-item">
    <a class="nav-link active" aria-current="page"
    href="/">Home</a>
    </li>
    <li class="nav-item"><a class="nav-link" href="/books">List
Book</a></li>
    <li class="nav-item"><a class="nav-link"
    href="/books/add">Add Book</a></li>
    </ul>
    </div>
</div>
</nav>
</header>
<div class="container">
    <div th:fragment="content"></div>
</div>
<footer th:fragment="footer" class="footer mt-auto py-3 bg-light">
    <div class="container text-center">Copyright ©;
    <span th:text="#dates.year()"/></span>
    <a href="https://www.hutech.edu.vn/">HUTECH Education</a>
    </div>
    <script th:src="@{/js/bootstrap.min.js}"></script>
    <script th:src="@{/js/jquery-3.7.0.min.js}"></script>
</footer>
</body>
</html>
```

Kiểm tra đường dẫn đặt hai file tĩnh **css** và **js** của **bootstrap** và **jquery** (như đã học ở bài lab 1, 2 đã làm).



Hình 4.5. Thư mục css và js

Tạo view hiển thị danh sách, tại thư mục **templates** tạo thêm 1 thư mục tên là **book**. Tạo thêm file có tên **list.html** trong thư mục **book** nằm tại đường dẫn **src/main/resources/templates/book**.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>My Book List</title>
  <th:block th:replace="~{layout::link-css}"></th:block>
  <th:block th:replace="~{layout::custom-css}"></th:block>
</head>
<body>
```

```
<th:block th:replace="~{layout::header}"></th:block>
<div class="container">
    <div class="row">
        <div class="col-md-12">
            <nav class="navbar navbar-light">
                <div class="container-fluid">
                    <h2>My Book List</h2>
                    <form class="d-flex" th:action="@{/books/search}"
method="get">
                        <input class="form-control me-2" type="search"
placeholder="Search" aria-label="Search" name="keyword">
                        <button class="btn btn-outline-success"
type="submit">Search</button>
                    </form>
                </div>
            </nav>
        </div>
    </div>
    <table class="table">
        <thead>
            <tr>
                <th><a th:href="@{/books(pageNo=${currentPage}, sortBy=
'id')}">Id</a></th>
                <th><a th:href="@{/books(pageNo=${currentPage}, sortBy=
'title')}">Title</a></th>
                <th><a th:href="@{/books(pageNo=${currentPage}, sortBy=
'author')}">Author</a></th>
                <th><a th:href="@{/books(pageNo=${currentPage}, sortBy=
'price')}">Price</a></th>
                <th><a th:href="@{/books(pageNo=${currentPage}, sortBy=
'category')}">Category</a></th>
                <th>Action</th>
            </tr>
        </thead>
        <tbody>
            <tr th:each="book : ${books}">
                <td th:text="${book.getId()}"></td>
                <td th:text="${book.getTitle()}"></td>
                <td th:text="${book.getAuthor()}"></td>
                <td th:text="${book.getPrice()}"></td>
                <td th:text="${book.getCategory().getName()}"></td>
                <td colspan="2">
                    <a class="btn btn-primary"
th:href="@{/books/edit/{id}(id=${book.getId()})}">Edit</a>
                    <a class="btn btn-danger"
th:href="@{/books/delete/{id}(id=${book.getId()})}">
```

```
        onclick="return confirm('Are you sure you want to delete  
this book?')">Delete</a>  
    </td>  
  </tr>  
  </tbody>  
</table>  
</div>  
<nav aria-label="Page navigation example">  
  <ul class="pagination justify-content-center pagination-sm"  
th:each="i : ${#numbers.sequence(0, totalPages)}">  
    <li class="page-item" th:classappend="${currentPage == i} ?  
'active'">  
      <a class="page-link" th:href="@{/books(pageNo=${i})}"  
th:text="${i}"></a>  
    </li>  
  </ul>  
</nav>  
  
<th:block th:replace="~{layout::footer}"></th:block>  
</body>  
</html>
```

Tạo file **HomeController.java** đặt tại thư mục **controllers** với đường dẫn sau **src/main/java/fit.hutech.spring/controllers**

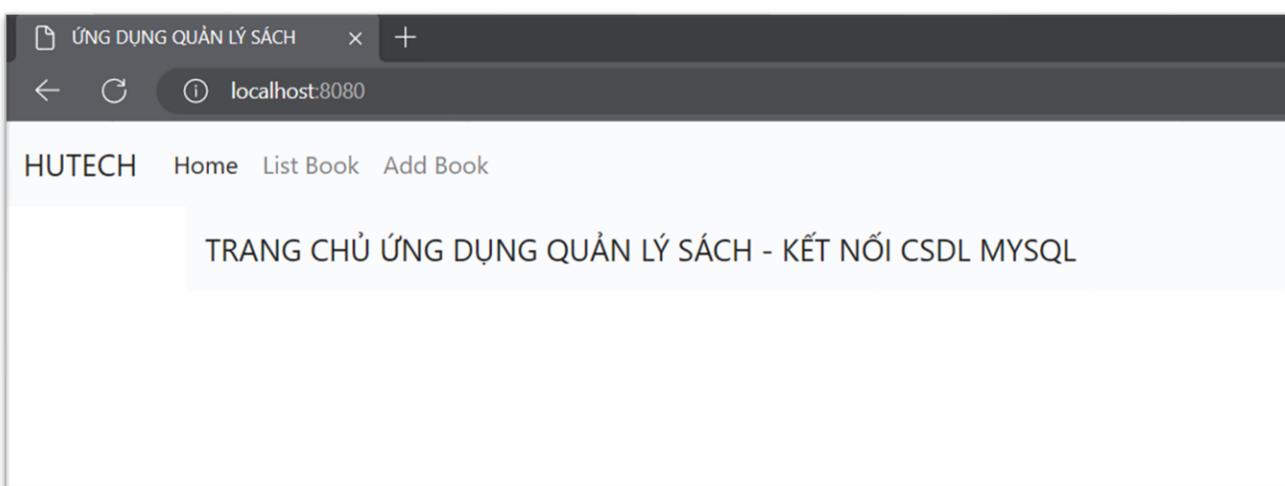
```
package fit.hutech.spring.controllers;  
  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RequestMapping;  
  
@Controller  
@RequestMapping("/")  
public class HomeController {  
    @GetMapping  
    public String home() {  
        return "home/index";  
    }  
}
```

Tạo file **index.html** đặt tại **src/main/resources/templates/home**.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>ỨNG DỤNG QUẢN LÝ SÁCH</title>
    <th:block th:replace="~{layout::link-css}"></th:block>
    <th:block th:replace="~{layout::custom-css}"></th:block>
</head>
<body>
<th:block th:replace="~{layout::header}"></th:block>
<div class="container">
    <nav class="navbar navbar-light bg-light">
        <div class="container-fluid">
            <a class="navbar-brand" href="#">
                TRANG CHỦ ỨNG DỤNG QUẢN LÝ SÁCH - KẾT NỐI CSDL MYSQL
            </a>
        </div>
    </nav>
</div>
<th:block th:replace="~{layout::footer}"></th:block>
</body>
</html>
```

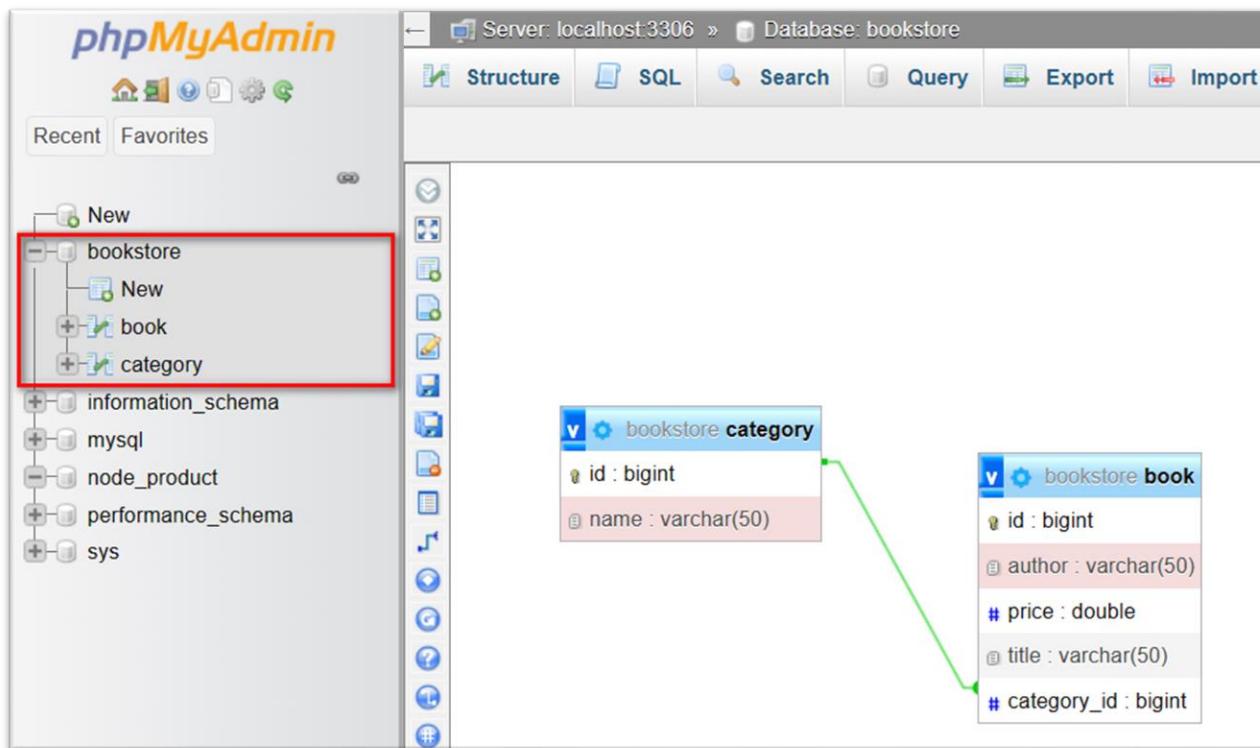
4.4Build và chạy ứng dụng

Build và chạy ứng dụng tại địa chỉ **localhost:8080**



Hình 4.6. Build và chạy ứng dụng

Kiểm tra CSDL thông qua **phpMyAdmin** <http://localhost/phpmyadmin>, hệ thống tự động tạo các bảng trong CSDL **bookstore** từ ánh xạ của entities.



Hình 4.7. Add database bookstore

Tiếp theo, chúng ta thêm dữ liệu mẫu vào bảng Category, bảng Book trong cơ sở dữ liệu vừa tạo.

Thêm dữ liệu cho bảng Category

	<input type="checkbox"/>				id	name
	<input type="checkbox"/>				1	Công nghệ phần mềm
	<input type="checkbox"/>				2	An toàn thông tin
	<input type="checkbox"/>				3	Hệ thống thông tin
	<input type="checkbox"/>				4	Mạng máy tính
	<input type="checkbox"/>				5	Khoa học dữ liệu

Hình 4.8. Thêm dữ liệu cho bảng Category

Thêm dữ liệu cho bảng Book

	<input type="text"/> Edit	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	id	author	price	title	category_id
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	1	Nguyễn Đình Ánh	12	Phát triển phần mềm mã nguồn mở	1
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	2	Cao Tùng Anh	10	Cơ sở dữ liệu phân tán	3
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	3	Văn Thiên Hoàng	11	Điều tra tấn công	2
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	4	Hàn Minh Châu	13	Mạng máy tính nâng cao	4
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	5	Nguyễn Thị Hải Bình	12	Phân tích dữ liệu truyền thông xã hội	5

Hình 4.9. Thêm dữ liệu cho bảng Book

Truy cập trang danh sách tại địa chỉ **localhost:8080/books**

My Book List					
ID	Title	Author	Price	Category	Action
1	Phát triển phần mềm mã nguồn mở	Nguyễn Đình Ánh	12.0	Công nghệ phần mềm	<input type="button" value="Edit"/> <input type="button" value="Delete"/>
2	Cơ sở dữ liệu phân tán	Cao Tùng Anh	10.0	Hệ thống thông tin	<input type="button" value="Edit"/> <input type="button" value="Delete"/>
3	Điều tra tấn công	Văn Thiên Hoàng	11.0	An toàn thông tin	<input type="button" value="Edit"/> <input type="button" value="Delete"/>
4	Mạng máy tính nâng cao	Hàn Minh Châu	13.0	Mạng máy tính	<input type="button" value="Edit"/> <input type="button" value="Delete"/>
5	Phân tích dữ liệu truyền thông xã hội	Nguyễn Thị Hải Bình	12.0	Khoa học dữ liệu	<input type="button" value="Edit"/> <input type="button" value="Delete"/>

Hình 4.10. Truy cập trang danh sách

4.5 Xây dựng giỏ hàng

Giỏ hàng đóng vai trò quan trọng trong quá trình phát triển ứng dụng thương mại điện tử. Đó là nơi mà người dùng có thể thêm sản phẩm vào để mua và quản lý các mục đã chọn trước khi tiến hành thanh toán. Dưới đây là hướng dẫn về cách lưu giỏ hàng vào Session. Sinh viên cũng có thể sử dụng Cookie và LocalStorage để lưu thông tin của giỏ hàng.

Đầu tiên ta thêm chức năng thêm vào giỏ hàng ở tệp **list.html**. Thêm đoạn code bên dưới vào bên cạnh đoạn code thiết kế nút xoá:

```
<form th:action="@{/books/add-to-cart}" method="post" class="d-inline">
    <input type="hidden" name="id" th:value="${book.getId()}">
    <input type="hidden" name="name">
```

```



```

Kết quả sau khi điều chỉnh file **list.html**:

```

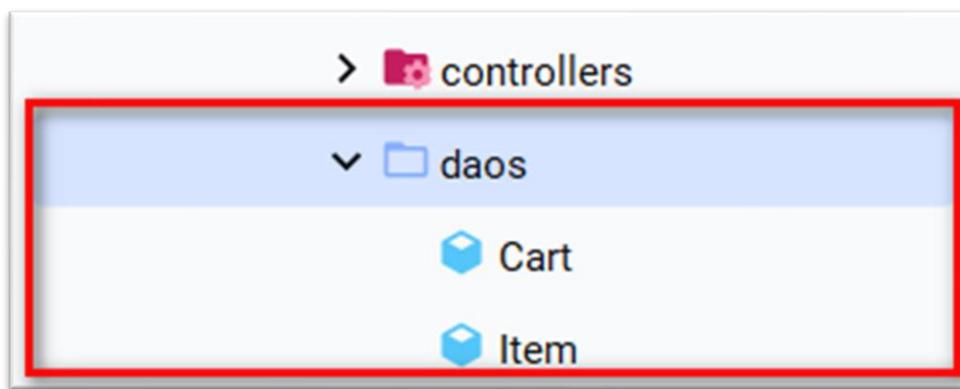
38     <tbody>
39         <tr th:each="book : ${books}">
40             <td th:text="${book.getId()}"/></td>
41             <td th:text="${book.getTitle()}"/></td>
42             <td th:text="${book.getAuthor()}"/></td>
43             <td th:text="${book.getPrice()}"/></td>
44             <td th:text="${book.getCategory().getName()}"/></td>
45             <td colspan="2">
46                 <a class="btn btn-primary" th:href="@{/books/edit/{id}(id=${book.getId()})}">Edit</a>
47                 <a class="btn btn-danger" th:href="@{/books/delete/{id}(id=${book.getId()})}"
48                     onclick="return confirm('Are you sure you want to delete this book?')">Delete</a>
49             <form th:action="@{/books/add-to-cart}" method="post" class="d-inline">
50                 <input type="hidden" name="id" th:value="${book.getId()}">
51                 <input type="hidden" name="name" th:value="${book.getTitle()}">
52                 <input type="hidden" name="price" th:value="${book.getPrice()}">
53                 <button type="submit" class="btn btn-success"
54                     onclick="return confirm('Are you sure you want to add this book to cart?')">
55                     Add to cart</button>
56             </form>
57         </td>
58     </tr>
59 </tbody>

```

Hình 4.11. Khởi tạo nút thêm vào giỏ hàng

Tạo thư mục **daos**, sao đó tạo 2 file model **Item.java** và **Cart.java** đặt tại thư mục **daos** với đường dẫn sau **src/main/java/fit.hutech.spring/daos**.

- **Item:** Chứa thông tin một sản phẩm trong giỏ hàng.
- **Cart:** Chứa thông tin giỏ hàng, lưu thông tin danh sách các sản phẩm trong giỏ hàng và các behavior của chúng.



Hình 4.12. Khởi tạo Item và Cart

Sau khi hoàn thành tạo 2 file **Item.java** và **Cart.java** trong thư mục daos với đường dẫn **src/main/java/fit.hutech.spring.daos**. Ta bổ sung code cho từng file với nội dung dưới đây:

Item.java

```
package fit.hutech.spring.daos;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class Item {
    private Long bookId;
    private String bookName;
    private Double price;
    private int quantity;
}
```

Cart.java

```
package fit.hutech.spring.daos;

import lombok.Data;

import java.util.ArrayList;
```

```
import java.util.List;
import java.util.Objects;

@Data
public class Cart {
    private List<Item> cartItems = new ArrayList<>();

    public void addItems(Item item) {
        boolean isExist = cartItems.stream()
            .filter(i -> Objects.equals(i.getBookId(),
item.getBookId()))
            .findFirst()
            .map(i -> {
                i.setQuantity(i.getQuantity() +
item.getQuantity());
                return true;
            })
            .orElse(false);

        if (!isExist) {
            cartItems.add(item);
        }
    }

    public void removeItems(Long bookId) {
        cartItems.removeIf(item -> Objects.equals(item.getBookId(),
bookId));
    }

    public void updateItems(int bookId, int quantity) {
        cartItems.stream()
            .filter(item -> Objects.equals(item
                .getBookId(), bookId))
            .forEach(item -> item.setQuantity(quantity));
    }
}
```

Tạo file service **CartService.java** đặt tại thư mục **services** theo đường dẫn sau **src/main/java/fit.hutech.spring/services**. Với mục đích thực hiện các thao tác với giỏ hàng trong biến Session, bao gồm lấy giỏ hàng, cập nhật giỏ hàng, xóa giỏ hàng và tính toán tổng số lượng và tổng giá trị của các mặt hàng trong giỏ hàng.

```
package fit.hutech.spring.services;

import fit.hutech.spring.entities.Cart;
```

```
import fit.hutech.spring.entities.Item;
import jakarta.servlet.http.HttpSession;
import jakarta.validation.constraints.NotNull;
import org.springframework.stereotype.Service;

import java.util.Optional;

@Service
public class CartService {
    private static final String CART_SESSION_KEY = "cart";

    public Cart getCart(@NotNull HttpSession session) {
        return Optional.ofNullable((Cart)
session.getAttribute(CART_SESSION_KEY))
            .orElseGet(() -> {
                Cart cart = new Cart();
                session.setAttribute(CART_SESSION_KEY, cart);
                return cart;
            });
    }

    public void updateCart(@NotNull HttpSession session, Cart cart) {
        session.setAttribute(CART_SESSION_KEY, cart);
    }

    public void removeCart(@NotNull HttpSession session) {
        session.removeAttribute(CART_SESSION_KEY);
    }

    public int getSumQuantity(@NotNull HttpSession session) {
        return getCart(session).getCartItems().stream()
            .mapToInt(Item::getQuantity)
            .sum();
    }

    public double getSumPrice(@NotNull HttpSession session) {
        return getCart(session).getCartItems().stream()
            .mapToDouble(item -> item.getPrice() *
item.getQuantity())
            .sum();
    }
}
```

Tại **BookController.java** bổ sung phương thức thêm sản phẩm vào giỏ hàng bổ sung phương thức thêm sản phẩm vào giỏ hàng và lưu trữ giỏ hàng trong biến Session như sau:

```
@PostMapping("/add-to-cart")
public String addToCart(HttpSession session,
                        @RequestParam long id,
                        @RequestParam String name,
                        @RequestParam double price,
                        @RequestParam(defaultValue = "1") int quantity)
{
    var cart = cartService.getCart(session);
    cart.addItems(new Item(id, name, price, quantity));
    cartService.updateCart(session, cart);
    return "redirect:/books";
}
```

Để thiết kế trang giao diện (view) cho giỏ hàng, bạn có thể tạo một file **cart.html** và đặt nó trong thư mục **book** theo đường dẫn **/src/main/resources/templates/book**. Sau đó bổ sung đoạn code bên dưới:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Cart</title>
    <th:block th:replace="~{layout::link-css}"></th:block>
    <th:block th:replace="~{layout::custom-css}"></th:block>
</head>
<body>
<th:block th:replace="~{layout::header}"></th:block>
<div class="container" th:if="${not #lists.isEmpty(cart.cartItems)}">
    <table class="table">
        <thead>
            <tr>
                <th scope="col">Book ID</th>
                <th scope="col">Book Name</th>
                <th scope="col">Quantity</th>
                <th scope="col">Price</th>
                <th scope="col">Total</th>
                <th scope="col">Action</th>
            </tr>
        <tbody>
            <tr th:each="item : cart.cartItems">
                <td>${item.id}</td>
                <td>${item.name}</td>
                <td>${item.quantity}</td>
                <td>${item.price}</td>
                <td>${item.total}</td>
                <td><a href="#" th:href="~/cart/delete/${item.id}"/>Delete</td>
            </tr>
        </tbody>
    </table>
</div>
<th:block th:replace="~{layout::footer}"></th:block>
</body>
</html>
```

```
</tr>
</thead>
<tbody>
<tr th:each="item : ${cart.cartItems}">
    <td th:text="${item.getBookId()}"></td>
    <td th:text="${item.getBookName()}"></td>
    <td>
        <label>
            <input type="number" min="1"
                th:value="${item.getQuantity()}"
                th:attr="data-id=${item.getBookId()}"
                class="form-control quantity">
        </label>
    </td>
    <td th:text="${item.getPrice()}"></td>
    <td th:text="${item.getPrice() * item.getQuantity()}"></td>
    <td>
        <a class="btn btn-danger"
            th:href="@{/cart/removeFromCart/{id}(id=${item.getBookId()})}">
            Remove
        </a>
    </td>
</tr>
</tbody>
</table>
<div class="row">
    <div class="col-md-12">
        <h3>Total: <span th:text="${totalPrice}"></span></h3>
    </div>
    <div class="col-md-12 text-center">
        <a class="btn btn-success"
            th:href="@{/cart/checkout}">Checkout</a>
        <a class="btn btn-danger"
            th:href="@{/cart/clearCart}">Clear Cart</a>
    </div>
</div>
<div class="container" th:if="#lists.isEmpty(cart.cartItems)">
    <div class="container">
        <div class="row">
            <div class="col-md-6 offset-md-3 text-center">
                <h3 class="mt-5">Your cart is empty</h3>
                <p class="lead">Please add some books to your cart</p>
                <a class="btn btn-primary" href="/books">Go to list
book</a>
            </div>
        </div>
    </div>
```

```

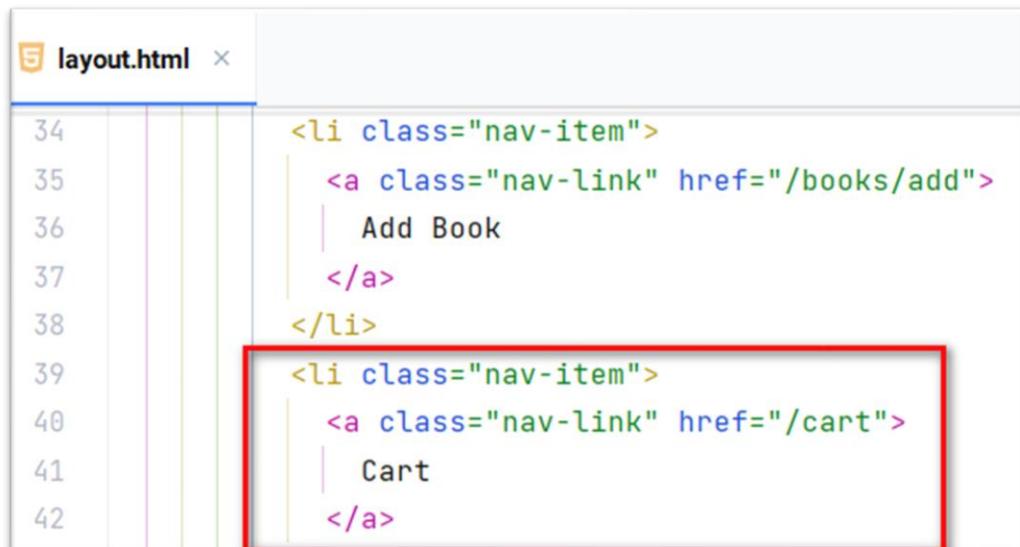
        </div>
    </div>
</div>
<th:block th:replace="~{layout::footer}"></th:block>
<script th:src="@{/js/cart.js}"></script>
</body>
</html>

```

Đoạn code trên là trang giao diện giỏ hàng trong ứng dụng. Nó hiển thị danh sách sản phẩm trong giỏ hàng, cho phép người dùng cập nhật số lượng và xóa sản phẩm. Ngoài ra, nó cũng hiển thị tổng giá trị của giỏ hàng. Nếu không tồn tại giỏ hàng trang sẽ hiển thị ra thông báo.

Thêm đoạn code sau vào file **layout.html** với mục đích tạo menu trỏ tới trang giỏ hàng. Lưu ý tab **Cart** nằm bên cạnh tab **Add book**.

```
<li class="nav-item"><a class="nav-link" href="/cart">Cart</a></li>
```



Hình 4.13. Thêm tab Cart

Tạo một tệp **cart.js** trong thư mục **/src/main/resources/static/js** với mục đích bắt sự kiện cập nhật số lượng của sản phẩm trong giỏ hàng. Sau đó thêm đoạn code JavaScript dưới đây:

```

$(document).ready(function () {
    $('.quantity').change(function () {
        let quantity = $(this).val();
        let id = $(this).attr('data-id');
        $.ajax({

```

```

        url: '/cart/updateCart/' + id + '/' + quantity,
        type: 'GET',
        success: function () {
            location.reload();
        }
    });
});

```

Khởi tạo file **CartController.java** và thêm các phương thức sau:

- **showCart:** Hàm này được gọi khi truy cập vào đường dẫn **/cart** bằng phương thức **GET**. Nó nhận vào một đối tượng **HttpSession** và một đối tượng **Model**, sau đó thêm các thuộc tính **cart**, **totalPrice** và **totalQuantity** vào **model**. Sau đó, nó trả về tên của view **book/cart** để hiển thị thông tin giỏ hàng.
- **removeFromCart:** Hàm này được gọi khi truy cập vào đường dẫn **/removeFromCart/{id}** bằng phương thức **GET**, trong đó **{id}** là một số nguyên duy nhất đại diện cho sản phẩm cần xóa khỏi giỏ hàng. Hàm này xóa sản phẩm có id tương ứng khỏi giỏ hàng và sau đó chuyển hướng người dùng đến trang **/cart**.
- **updateCart:** Hàm này được gọi khi truy cập vào đường dẫn **/updateCart/{id}/{quantity}** bằng phương thức **GET**, trong đó **{id}** là một số nguyên duy nhất đại diện cho sản phẩm trong giỏ hàng và **{quantity}** là số lượng mới của sản phẩm đó. Hàm này cập nhật số lượng của sản phẩm trong giỏ hàng và sau đó trả về tên của view **book/cart** để hiển thị thông tin giỏ hàng.
- **clearCart:** Hàm này được gọi khi truy cập vào đường dẫn **/clearCart** bằng phương thức **GET**. Hàm này xóa toàn bộ giỏ hàng của người dùng bằng cách gọi phương thức **removeCart()** của đối tượng **cartService**.

```

package fit.hutech.spring.controllers;

import fit.hutech.spring.services.CartService;
import jakarta.servlet.http.HttpSession;
import jakarta.validation.constraints.NotNull;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Controller;

```

```
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("/cart")
@RequiredArgsConstructor
public class CartController {
    private final CartService cartService;

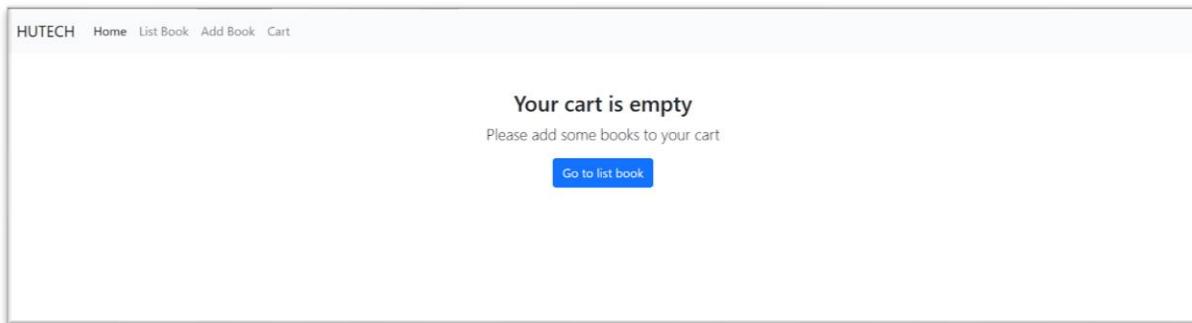
    @GetMapping
    public String showCart(HttpSession session,
                          @NotNull Model model) {
        model.addAttribute("cart", cartService.getCart(session));
        model.addAttribute("totalPrice",
        cartService.getSumPrice(session));
        model.addAttribute("totalQuantity",
        cartService.getSumQuantity(session));
        return "book/cart";
    }

    @GetMapping("/removeFromCart/{id}")
    public String removeFromCart(HttpSession session,
                                @PathVariable Long id) {
        var cart = cartService.getCart(session);
        cart.removeItems(id);
        return "redirect:/cart";
    }

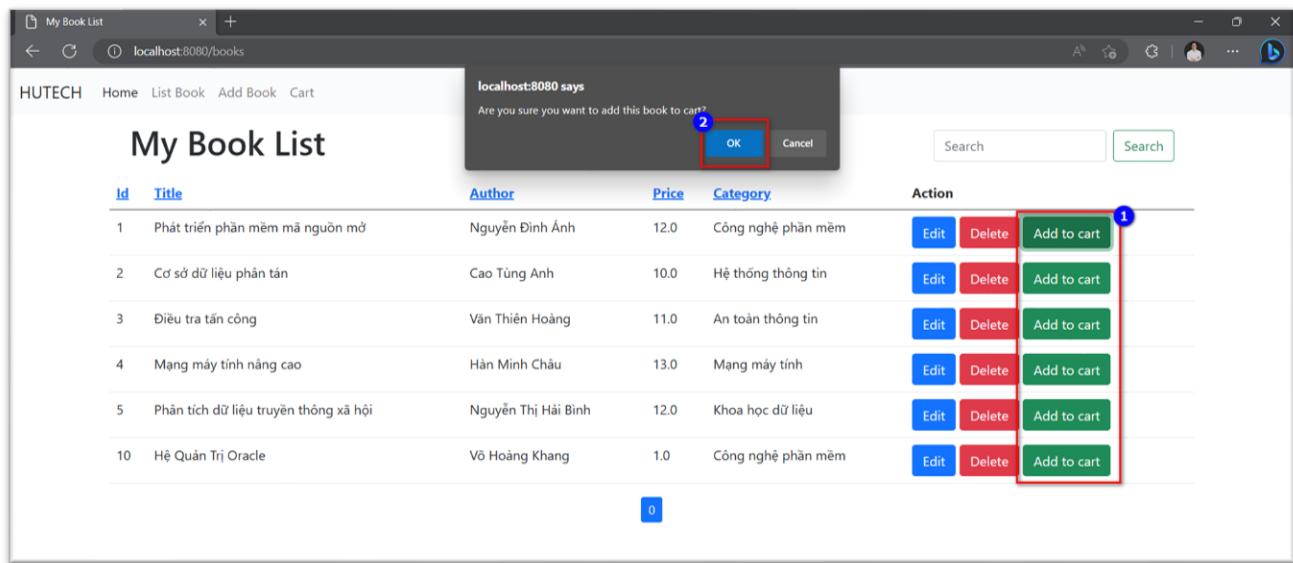
    @GetMapping("/updateCart/{id}/{quantity}")
    public String updateCart(HttpSession session,
                            @PathVariable Long id,
                            @PathVariable int quantity) {
        var cart = cartService.getCart(session);
        cart.updateItems(id, quantity);
        return "book/cart";
    }

    @GetMapping("/clearCart")
    public String clearCart(HttpSession session) {
        cartService.removeCart(session);
        return "redirect:/cart ";
    }
}
```

Sau đó **Build** và chạy ứng dụng kiểm tra kết quả:



Hình 4.14. Trang mặc định khi chưa có sản phẩm nào thêm vào giỏ hàng
Tiến hành thêm một vài sản phẩm vào giỏ hàng



Hình 4.15. Thêm một vài sản phẩm vào giỏ hàng
Kết quả thu được sau khi thêm vào giỏ hàng:

Book ID	Book Name	Quantity	Price	Total	Action
1	Phát triển phần mềm mã nguồn mở	1	12.0	12.0	<button>Remove</button>
3	Điều tra tấn công	1	11.0	11.0	<button>Remove</button>
2	Cơ sở dữ liệu phân tán	2	10.0	20.0	<button>Remove</button>
Total: 43.0					<button>Checkout</button> <button>Clear Cart</button>

Hình 4.16. Kết quả sau khi thêm vào giỏ hàng

TÓM TẮT

Java Spring Framework là một framework phát triển ứng dụng web được viết bằng Java. Nó cung cấp các công cụ và thư viện để phát triển ứng dụng web hiệu quả và nhanh chóng. Spring Framework có nhiều tính năng mạnh mẽ bao gồm Dependency Injection, AOP, MVC, JDBC, Security và nhiều hơn nữa.

phpMyAdmin là một công cụ quản lý cơ sở dữ liệu MySQL được viết bằng PHP. Nó cung cấp cho người dùng một giao diện web để quản lý các cơ sở dữ liệu MySQL. PHPMyAdmin cho phép người dùng thực hiện các tác vụ như tạo, xóa, sửa đổi các bảng dữ liệu, quản lý người dùng và phân quyền truy cập.

Khi sử dụng Java Spring Framework và phpMyAdmin cùng nhau, người lập trình có thể phát triển các ứng dụng web mạnh mẽ với cơ sở dữ liệu MySQL và sử dụng các tính năng mạnh mẽ của Spring Framework để tăng hiệu suất và tăng tính bảo mật của ứng dụng.

CÂU HỎI ÔN TẬP

1. Hibernate là gì và tại sao nó được sử dụng trong phát triển ứng dụng?
2. phpMyAdmin là gì và chức năng của nó là gì trong việc quản lý cơ sở dữ liệu MySQL?
3. Làm thế nào để tạo một bảng dữ liệu sử dụng Hibernate và MySQL?
4. Làm thế nào để thực hiện tạo, sửa đổi và xóa bảng dữ liệu trong PhpMyAdmin?
5. Hibernate hỗ trợ những quan hệ giữa các bảng như thế nào?
6. Cách cấu hình Hibernate để kết nối với cơ sở dữ liệu MySQL?
7. Các phương thức quan trọng của đối tượng HttpSession trong Spring Boot là gì và cách sử dụng chúng?
8. Làm thế nào để lưu trữ và truy xuất dữ liệu trong HttpSession trong ứng dụng Spring Boot?

BÀI 5 XÂY DỰNG CÁC CHỨC NĂNG THÊM, XOÁ, SỬA

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

- ✓ Hiểu cách sử dụng phpMyAdmin, một công cụ quản trị cơ sở dữ liệu MySQL.
- ✓ Kết nối thành công cơ sở dữ liệu của dự án với phpMyAdmin.
- ✓ Xây dựng một ứng dụng website quản lý sách với các chức năng cơ bản để tương tác với cơ sở dữ liệu. Các chức năng bao gồm thêm sách mới, xóa sách và chỉnh sửa thông tin sách.

Mô tả chức năng:

- ✓ **Thêm sách:** Cho phép người dùng nhập thông tin của một quyển sách bao gồm tiêu đề, tác giả, giá và danh mục. Thông tin này sẽ được lưu vào cơ sở dữ liệu.
- ✓ **Xóa sách:** Người dùng có thể chọn một quyển sách từ danh sách hiển thị và xóa nó khỏi danh sách.
- ✓ **Sửa sách:** Chức năng này cho phép người dùng chọn một quyển sách trong danh sách hiển thị và cập nhật thông tin liên quan đến nó.
- ✓ **Hiển thị sách:** Ứng dụng sẽ hiển thị danh sách toàn bộ các quyển sách có trong danh sách, bao gồm tiêu đề, tác giả, giá và danh mục tương ứng.
- ✓ **Tìm kiếm:** Để giúp người dùng tìm kiếm sách dễ dàng, ứng dụng cung cấp chức năng tìm kiếm theo từ khóa.
- ✓ **Checkout:** Thông tin giỏ hàng của người dùng sẽ được lưu vào cơ sở dữ liệu.

5.1 Viết trang thêm sách – Add Book

Đầu tiên, chúng ta sẽ tạo một file mới có tên **CategoryService.java** trong thư mục **src/main/java/fit.hutech.spring/services**.

Trong lớp **CategoryService.java**, chúng ta sẽ định nghĩa các phương thức để thao tác với cơ sở dữ liệu, bao gồm thêm, sửa, xóa và truy vấn dữ liệu. Lớp này sẽ cung cấp các chức năng để quản lý danh mục sản phẩm và đảm bảo tính nhất quán của dữ liệu trong toàn bộ ứng dụng web.

Dưới đây là cách viết code cho lớp **CategoryService.java**:

```
package fit.hutech.spring.services;

import fit.hutech.spring.entities.Category;
import fit.hutech.spring.repositories.ICategoryRepository;
import jakarta.validation.constraints.NotNull;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Isolation;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;
import java.util.Objects;
import java.util.Optional;

@Service
@RequiredArgsConstructor
@Transactional(isolation = Isolation.SERIALIZABLE,
    rollbackFor = {Exception.class, Throwable.class})
public class CategoryService {
    private final ICategoryRepository categoryRepository;

    public List<Category> getAllCategories() {
        return categoryRepository.findAll();
    }

    public Optional<Category> getCategoryById(Long id) {
        return categoryRepository.findById(id);
    }

    public void addCategory(Category category) {
        categoryRepository.save(category);
    }
}
```

```

public void updateCategory(@NotNull Category category) {
    Category existingCategory = categoryRepository
        .findById(category.getId())
        .orElse(null);
    Objects.requireNonNull(existingCategory)
        .setName(category.getName());

    categoryRepository.save(existingCategory);
}

public void deleteCategoryById(Long id) {
    categoryRepository.deleteById(id);
}
}

```

4.1.1. Thêm các phương thức thêm sách vào Controller

Trong file **BookController.java**, chúng ta có thể thêm các phương thức sau để thực hiện việc thêm sách vào cơ sở dữ liệu.

- Phương thức **GET** để hiển thị form thêm sách
- Phương thức **POST** để xử lý việc thêm sách

Thêm code như ảnh bên dưới:

```

package fit.hutech.spring.controllers;

import fit.hutech.spring.entities.Book;
import fit.hutech.spring.entities.Item;
import fit.hutech.spring.services.BookService;
import fit.hutech.spring.services.CartService;
import fit.hutech.spring.services.CategoryService;
import jakarta.servlet.http.HttpSession;
import jakarta.validation.Valid;
import jakarta.validation.constraints.NotNull;
import lombok.RequiredArgsConstructor;
import org.springframework.context.support.DefaultMessageSourceResolvable;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.*;

```

```
@Controller
@RequestMapping("/books")
@RequiredArgsConstructor
public class BookController {
    private final BookService bookService;

    private final CategoryService categoryService;

    private final CartService cartService;

    @GetMapping
    public String showAllBooks(@NotNull Model model,
                               @RequestParam(defaultValue = "0") Integer pageNo,
                               @RequestParam(defaultValue = "20") Integer pageSize,
                               @RequestParam(defaultValue = "id") String sortBy) {
        model.addAttribute("books", bookService.getAllBooks(pageNo, pageSize, sortBy));
        model.addAttribute("currentPage", pageNo);
        model.addAttribute("totalPages",
bookService.getAllBooks(pageNo, pageSize, sortBy).size() / pageSize);
        model.addAttribute("categories",
categoryService.getAllCategories());
        return "book/list";
    }

    @GetMapping("/add")
    public String addBookForm(@NotNull Model model) {
        model.addAttribute("book", new Book());
        model.addAttribute("categories",
categoryService.getAllCategories());
        return "book/add";
    }

    @PostMapping("/add")
    public String addBook(
            @Valid @ModelAttribute("book") Book book,
            @NotNull BindingResult bindingResult,
            Model model) {
        if (bindingResult.hasErrors()) {
            var errors = bindingResult.getAllErrors()
                .stream()

.map(DefaultMessageSourceResolvable::getDefaultMessage)
```

```

        .toArray(String[]::new);
        model.addAttribute("errors", errors);
        model.addAttribute("categories",
categoryService.getAllCategories());
        return "book/add";
    }
    bookService.addBook(book);
    return "redirect:/books";
}

@PostMapping("/add-to-cart")
public String addToCart(HttpSession session,
    @RequestParam long id,
    @RequestParam String name,
    @RequestParam double price,
    @RequestParam(defaultValue = "1") int
quantity) {
    var cart = cartService.getCart(session);
    cart.addItems(new Item(id, name, price, quantity));
    cartService.updateCart(session, cart);
    return "redirect:/books";
}
}

```

Trong đoạn code trên, chúng ta có hai phương thức được sử dụng để thêm thông tin về một cuốn sách (Book) vào cơ sở dữ liệu.

Phương thức **addBookForm()** trả về một trang HTML cho phép người dùng thêm mới một đối tượng Book vào cơ sở dữ liệu. Đây là một phương thức xử lý yêu cầu **GET**, được kết nối với đường dẫn **/add**. Khi người dùng truy cập đường dẫn này, trang HTML cho phép nhập thông tin về cuốn sách sẽ được hiển thị.

Phương thức **addBook()** được sử dụng để xử lý dữ liệu đăng ký từ trang HTML **/add** mà phương thức **addBookForm** trả về. Đây là một phương thức xử lý yêu cầu **POST**, cũng được ánh xạ với đường dẫn **/add**. Khi người dùng gửi thông tin của cuốn sách từ trang HTML, phương thức **addBook()** sẽ được gọi để xử lý dữ liệu và thêm thông tin về cuốn sách vào cơ sở dữ liệu.

Các annotation **@GetMapping** và **@PostMapping** được sử dụng để ánh xạ các yêu cầu HTTP **GET** và **POST** tương ứng đến phương thức xử lý của controller. Trong trường hợp này, phương thức **addBookForm** xử lý yêu cầu **GET** tới đường dẫn **/add**,

và phương thức **addBook** xử lý yêu cầu **POST** tới đường dẫn **/add**. Điều này đảm bảo rằng các yêu cầu **GET** và **POST** liên quan đến việc thêm sách đều được xử lý chính xác.

5

```
import fit.hutech.spring.services.CategoryService;
```

Hình 5.1. Import service vào controller

Sử dụng để import và sử dụng các phương thức và thuộc tính được định nghĩa trong class **CategoryService** trong các lớp khác của ứng dụng.

```
20 @RequiredArgsConstructor
21 public class BookController {
22     private final BookService bookService;
23
24     private final CategoryService categoryService;
25
26     private final CartService cartService;
```

Hình 5.2. Inject service vào controller

@RequiredArgsConstructor được sử dụng để chú thích một annotation trong một class.

Annotation **@RequiredArgsConstructor** thực hiện việc constructor inject đối tượng implement (thực thi) của interface **CategoryService** vào thuộc tính **categoryService** của lớp hiện tại. Điều này cho phép đối tượng đó được sử dụng trong các phương thức của lớp đó một cách dễ dàng và tiện lợi. Cơ chế này giúp tăng tính chuyên nghiệp và hiệu quả trong quá trình lập trình.

Tạo mới hoặc chỉnh sửa file **add.html** được đặt tại đường dẫn sau đây **src/main/resources/templates/book**. Người dùng sẽ chọn một danh mục từ danh sách. Bổ sung đoạn code như bên dưới:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
```

```
scale=1.0">
    <th:block th:replace="~{layout::link-css}"></th:block>
    <th:block th:replace="~{layout::custom-css}"></th:block>
    <title>Add Book</title>
</head>
<body>
    <th:block th:replace="~{layout::header}"></th:block>
    <div class="container">
        <h1>Add Book</h1>
        <form th:action="@{/books/add}" th:object="${book}" method="post">
            <div class="col-6 mb-3">
                <label class="form-label" for="title">Title:</label><span
                class="text-danger">*</span>
                <input class="form-control" required type="text"
                th:field="*{title}" id="title" placeholder="Enter title" autofocus >
            </div>
            <div class="col-6 mb-3">
                <label class="form-label" for="author">Author:</label>
                <input class="form-control" type="text" th:field="*{author}"
                id="author" placeholder="Enter author">
            </div>
            <div class="col-6 mb-3">
                <label class="form-label" for="price">Price:</label><span
                class="text-danger">*</span>
                <input class="form-control" required type="text"
                th:field="*{price}" id="price" placeholder="Enter price">
            </div>
            <div class="col-6 mb-3">
                <label class="form-label" for="category">Category:</label><span
                class="text-danger">*</span>
                <select class="form-control" id="category" name="category.id">
                    <option value="">-- Select Category --</option>
                    <option th:each="category : ${categories}"
                th:value="${category.id}"
                    th:text="${category.name}"></option>
                </select>
            </div>
            <input type="submit" class="btn btn-primary" value="Save">
        </form>
        <br>
        <a th:href="@{/books}">Back to List</a>
    </div>
    <th:block th:replace="~{layout::footer}"></th:block>
</body>
</html>
```

Build, chạy ứng dụng tại địa chỉ **localhost:8080/books/add**

Add Book

Title:^{*}

Author:

Price:^{*}

Category:^{*}

[Save](#)

[Back to List](#)

Hình 5.3. Giao diện add Book

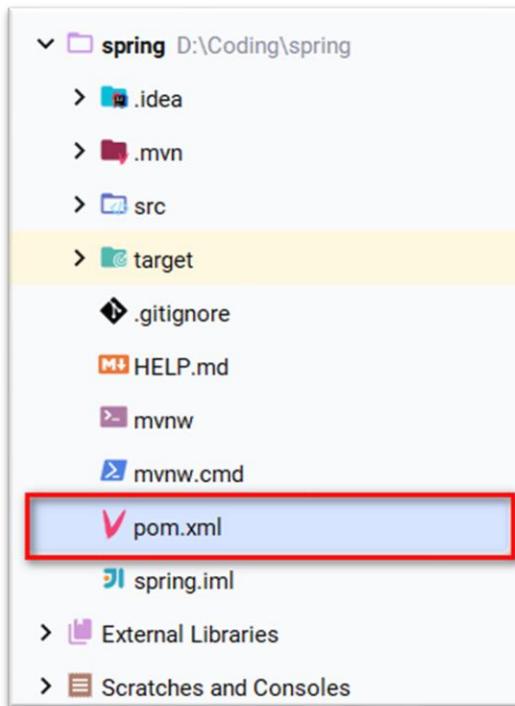
Nhấn **Save** và xem kết quả.

My Book List					
Id	Title	Author	Price	Category	Action
1	Phát triển phần mềm mã nguồn mở	Nguyễn Đình Ánh	12.0	Công nghệ phần mềm	Edit Delete
2	Cơ sở dữ liệu phân tán	Cao Tùng Anh	10.0	Hệ thống thông tin	Edit Delete
3	Điều tra tấn công	Vân Thiên Hoàng	11.0	An toàn thông tin	Edit Delete
4	Mạng máy tính nâng cao	Hàn Minh Châu	13.0	Mạng máy tính	Edit Delete
5	Phân tích dữ liệu truyền thông xã hội	Nguyễn Thị Hải Bình	12.0	Khoa học dữ liệu	Edit Delete
8	Công cụ và môi trường phát triển phần mềm	Nguyễn Huy Cường	15.0	Công nghệ phần mềm	Edit Delete

Hình 5.4. Danh sách Book sau khi Add Book mới

5.2 Thêm ràng buộc khi lưu sách

Mở file **pom.xml** thêm **dependency**



Hình 5.5. Thêm ràng buộc vào file pom.xml

Trong Spring, khi sử dụng các phụ thuộc (dependencies), việc xác thực và kiểm tra tính hợp lệ của dữ liệu được thực hiện tự động trước khi thực hiện các thao tác trong controller của ứng dụng. Nếu dữ liệu không hợp lệ, Spring sẽ trả về các thông báo lỗi tương ứng, giúp bạn xử lý vấn đề một cách chính xác.

Để thực hiện điều này, Spring tích hợp sẵn các cơ chế xác thực và kiểm tra tính hợp lệ. Bạn có thể định nghĩa các quy tắc xác thực cho các trường dữ liệu trong đối tượng mà bạn muốn xử lý. Khi một yêu cầu được gửi đến controller, Spring sẽ kiểm tra các dữ liệu đầu vào dựa trên các quy tắc này.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

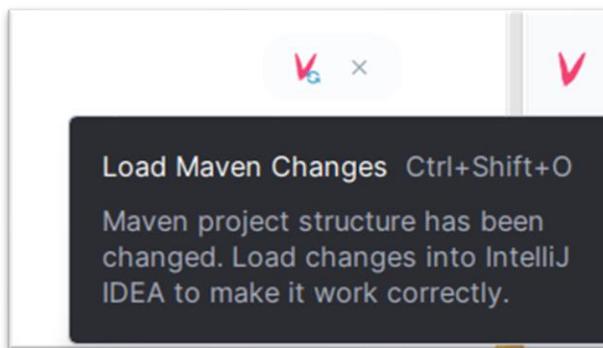
Đoạn code trên là một phần của tệp pom.xml trong dự án Spring Boot. Nó định nghĩa một dependency vào module **spring-boot-starter-validation** của

Spring Boot. Cụ thể, nó sử dụng groupId là **org.springframework.boot** và artifactId là **spring-boot-starter-validation**. Khi bạn thêm phụ thuộc này vào dự án, Spring Boot sẽ tự động tích hợp mô-đun xác thực và kiểm tra tính hợp lệ dữ liệu.

Module **spring-boot-starter-validation** cung cấp các công cụ và cơ chế để thực hiện xác thực và validation trong ứng dụng Spring. Bằng cách sử dụng nó, bạn có thể xác thực các dữ liệu đầu vào và kiểm tra tính hợp lệ của chúng trước khi xử lý trong controller.

Nếu dữ liệu không đáp ứng các quy tắc xác thực đã định nghĩa, Spring sẽ tự động tạo ra các thông báo lỗi tương ứng để bạn có thể xử lý vấn đề một cách chính xác.

Tại góc phải màn hình của file **pom.xml**, cập nhật maven hoặc nhấn tổ hợp phím **CTRL + SHIFT + O**

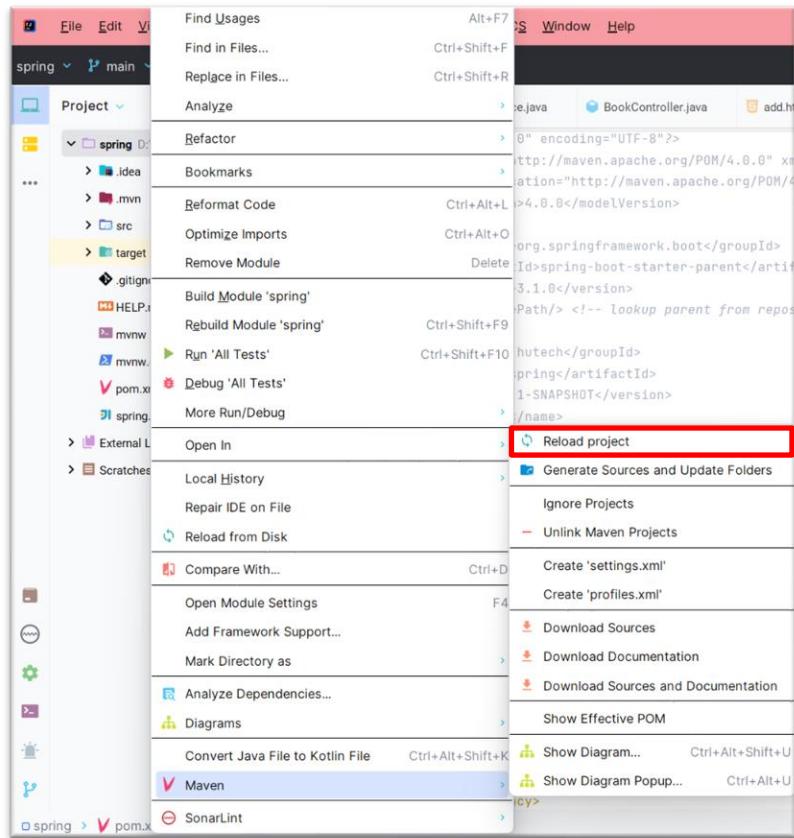


Hình 5.6. Cập nhật maven

Nếu vẫn còn có lỗi xuất hiện, thì rất có thể là **dependencies** chưa cập nhập.

Vậy để cập nhập làm theo các bước sau.

- Bước 1: Chọn **chuột phải vào dự án (Project)** của mình
- Bước 2: Tìm và chọn vào **Maven**
- Bước 3: Chọn **Reload project**



Hình 5.7. Reload project Maven

Tạo 1 thư mục Validator đặt tại **src/main/java/fit.hutech.spring**

Tạo file **ValidCategoryIdValidator.java** được đặt tại đường dẫn sau đây
src/main/java/fit.hutech.spring/validators

Hàm **isValid()** Kiểm tra các đối tượng "Category" được sử dụng trong hệ thống là hợp lệ và có tồn tại Id không.

```
package fit.hutech.spring.validators;

import fit.hutech.spring.entities.Category;
import fit.hutech.spring.validators.annotations.ValidCategoryId;
import jakarta.validation.ConstraintValidator;
import jakarta.validation.ConstraintValidatorContext;
public class ValidCategoryIdValidator implements
ConstraintValidator<ValidCategoryId, Category> {
    @Override
    public boolean isValid(Category category,
ConstraintValidatorContext context) {
        return category != null && category.getId() != null;
    }
}
```

Tiếp theo, Tạo thư mục **annotation** (*thư mục con của validators*) đặt tại đường dẫn **src/main/java/fit.hutech.spring/validators**

Tạo file ValidCategoryId.java được đặt tại đường dẫn sau đây **src/main/java/fit.hutech.spring/validators/annotations**

```
package fit.hutech.spring.validators.annotations;

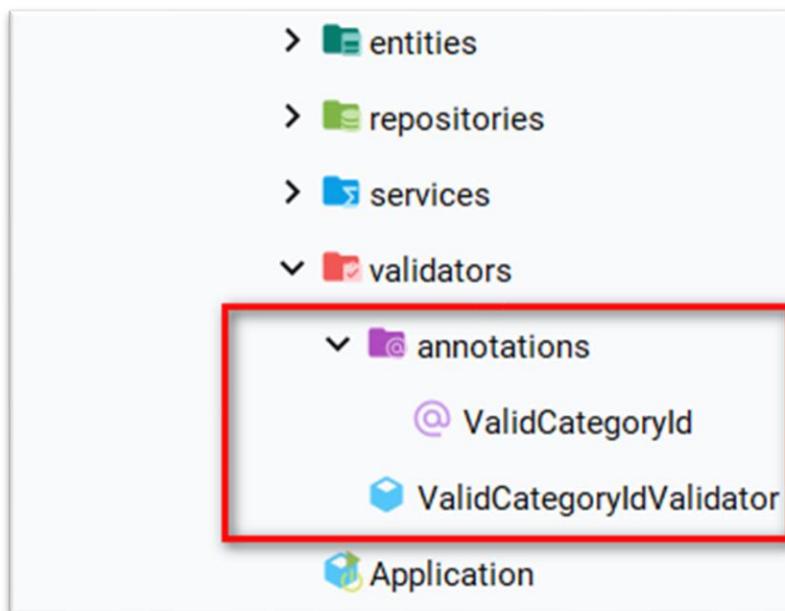
import fit.hutech.spring.validators.ValidCategoryIdValidator;
import jakarta.validation.Constraint;
import jakarta.validation.Payload;

import java.lang.annotation.Documented;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;

import static java.lang.annotation.ElementType.FIELD;
import static java.lang.annotation.ElementType.TYPE;
import static java.lang.annotation.RetentionPolicy.RUNTIME;

@Target({TYPE, FIELD})
@Retention(RUNTIME)
@Constraint(validatedBy = ValidCategoryIdValidator.class)
@Documented
public @interface ValidCategoryId {
    String message() default "Invalid Category Id";
    Class<?>[] groups() default {};
    Class<? extends Payload>[] payload() default {};
}
```

Cấu trúc thư mục **Validators** chứa **annotations** sẽ như bên dưới.



Hình 5.8. Cấu trúc thư mục Validator

Chỉnh sửa file **Book.java** được đặt tại đường dẫn sau đây:
src/main/java/fit.hutech.spring/entities.

Thêm các **annotation** để ràng buộc việc nhập liệu.

Annotation được hiểu là một dạng chú thích hoặc một dạng siêu dữ liệu (metadata) được dùng để cung cấp thông tin dữ liệu cho mã nguồn Java. Các chú thích không có ảnh hưởng trực tiếp đến hoạt động của code mà chúng chú thích.

```

package fit.hutech.spring.entities;

import fit.hutech.spring.validators.annotations.ValidCategoryId;
import jakarta.persistence.*;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.Positive;
import jakarta.validation.constraints.Size;
import lombok.*;
import org.hibernate.Hibernate;

import java.util.Objects;

@Getter
@Setter

```

```
@ToString  
@RequiredArgsConstructor  
@AllArgsConstructor  
@Builder  
@Entity  
@Table(name = "book")  
public class Book {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
  
    @Column(name = "title", length = 50, nullable = false)  
    @Size(min = 1, max = 50, message = "Title must be between 1 and 50  
characters")  
    @NotBlank(message = "Title must not be blank")  
    private String title;  
  
    @Column(name = "author", length = 50, nullable = false)  
    @Size(min = 1, max = 50, message = "Author must be between 1 and 50  
characters")  
    @NotBlank(message = "Author must not be blank")  
    private String author;  
  
    @Column(name = "price")  
    @Positive(message = "Price must be greater than 0")  
    private Double price;  
  
    @ManyToOne(fetch = FetchType.LAZY)  
    @JoinColumn(name = "category_id", referencedColumnName = "id")  
    @ValidCategoryId  
    @ToString.Exclude  
    private Category category;  
  
    @Override  
    public boolean equals(Object o) {  
        if (this == o) return true;  
        if (o == null || Hibernate.getClass(this) !=  
Hibernate.getClass(o)) return false;  
        Book book = (Book) o;  
        return getId() != null && Objects.equals(getId(),  
book.getId());  
    }  
  
    @Override  
    public int hashCode() {  
        return getClass().hashCode();  
    }
```

```
}
```

Ta thực hiện ràng buộc tương tự cho **Category** tại **Category.java**.

```
package fit.hutech.spring.entities;

import jakarta.persistence.*;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.Size;
import lombok.*;
import org.hibernate.Hibernate;

import java.util.ArrayList;
import java.util.List;
import java.util.Objects;

@Getter
@Setter
@ToString
@RequiredArgsConstructor
@AllArgsConstructor
@Builder
@Entity
@Table(name = "category")
public class Category {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long id;

    @Column(name = "name", length = 50, nullable = false)
    @Size(min = 1, max = 50, message = "Name must be between 1 and 50 characters")
    @NotBlank(message = "Name must not be blank")
    private String name;

    @OneToMany(mappedBy = "category", cascade = CascadeType.ALL)
    @ToString.Exclude
    private List<Book> books = new ArrayList<>();

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || Hibernate.getClass(this) !=
Hibernate.getClass(o)) return false;
```

```

        Category category = (Category) o;
        return getId() != null && Objects.equals(getId(),
category.getId());
    }

    @Override
    public int hashCode() {
        return getClass().hashCode();
    }
}

```

Trong **BookController.java** ta bổ sung đoạn code xác thực dữ liệu vào phương thức thêm sách như sau:

```

@PostMapping("/add")
public String addBook(
    @Valid @ModelAttribute("book") Book book,
    @NotNull BindingResult bindingResult,
    Model model) {
    if (bindingResult.hasErrors()) {
        var errors = bindingResult.getAllErrors()
            .stream()
            .map(DefaultMessageSourceResolvable::getDefaultMessage)
            .toArray(String[]::new);
        model.addAttribute("errors", errors);
        model.addAttribute("categories",
categoryService.getAllCategories());
        return "book/add";
    }
    bookService.addBook(book);
    return "redirect:/books";
}

```

Chỉnh sửa file **Add.html** đặt tại **src/main/templates/book/add.html**

```

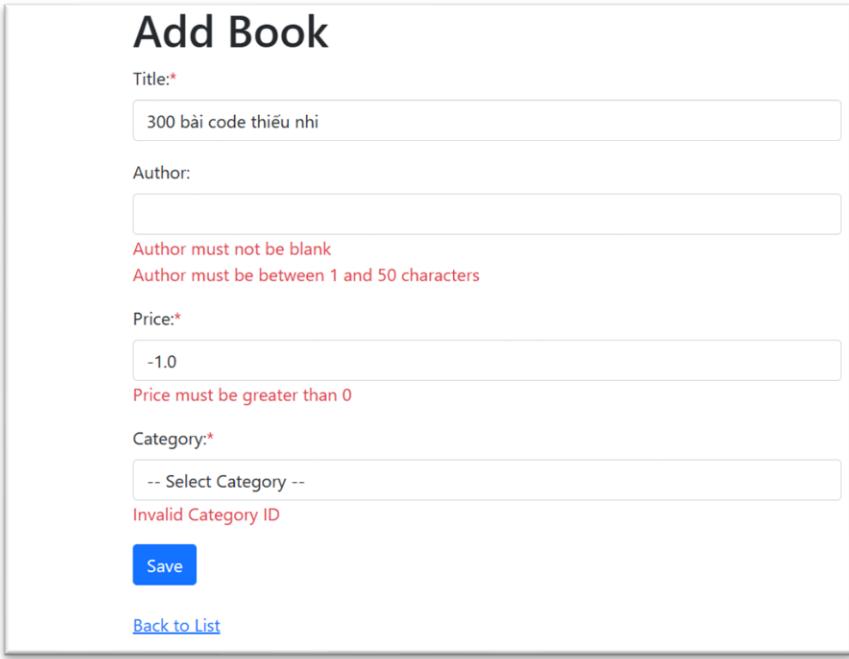
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <th:block th:replace="~{layout::link-css}"></th:block>
    <th:block th:replace="~{layout::custom-css}"></th:block>
    <title>Add Book</title>
</head>
<body>

```

```
<th:block th:replace="~{layout::header}"></th:block>
<div class="container">
    <h1>Add Book</h1>
    <form th:action="@{/books/add}" th:object="${book}" method="post">
        <div class="col-6 mb-3">
            <label class="form-label" for="title">Title:</label><span
class="text-danger">*</span>
            <input class="form-control" type="text" placeholder="Enter title"
th:field="*{title}" id="title" required autofocus >
            <span class="text-danger" th:if="#{#fields.hasErrors('title')}"
th:errors="*{title}"></span>
        </div>
        <div class="col-6 mb-3">
            <label class="form-label" for="author">Author:</label>
            <input class="form-control" type="text" th:field="*{author}"
id="author" placeholder="Enter author">
            <span class="text-danger" th:if="#{#fields.hasErrors('author')}"
th:errors="*{author}"></span>
        </div>
        <div class="col-6 mb-3">
            <label class="form-label" for="price">Price:</label><span
class="text-danger">*</span>
            <input class="form-control" required type="text"
th:field="*{price}" id="price" placeholder="Enter price">
            <span class="text-danger" th:if="#{#fields.hasErrors('price')}"
th:errors="*{price}"></span>
        </div>
        <div class="col-6 mb-3">
            <label class="form-label" for="category">Category:</label><span
class="text-danger">*</span>
            <select class="form-control" id="category" name="category.id">
                <option value="">-- Select Category --</option>
                <option th:each="category : ${categories}"
th:value="${category.getId()}"
                    th:text="${category.getName()}"></option>
            </select>
            <span class="text-danger"
th:if="#{#fields.hasErrors('category')}"
th:errors="*{category}"></span>
        </div>
        <input type="submit" class="btn btn-primary" value="Save" />
    </form>
    <br>
    <a th:href="@{/books}">Back to List</a>
</div>
<th:block th:replace="~{layout::footer}"></th:block>
```

```
</body>
</html>
```

Build, chạy ứng dụng và truy cập tới trang <http://localhost:8080/books/add>.
Không nhập nhưng vẫn nhấn nút **SAVE** để kiểm tra ràng buộc



The screenshot shows a web form titled 'Add Book'. It has four input fields: 'Title' (containing '300 bài code thiếu nhi'), 'Author' (empty), 'Price' (containing '-1.0'), and 'Category' (containing '-- Select Category --'). Below each field, there is red validation feedback: 'Author must not be blank' and 'Author must be between 1 and 50 characters' for the Author field; 'Price must be greater than 0' for the Price field; and 'Invalid Category ID' for the Category field. At the bottom left is a blue 'Save' button, and at the bottom right is a link 'Back to List'.

Hình 5.9. Kiểm tra ràng buộc

5.3 Chức năng Edit và delete Book

5.3.1 Delete Book

Tại **BookService.java** ta bổ sung phương thức xoá sách dựa theo Id của nó theo đoạn code dưới đây:

```
public void deleteBookById(Long id) {
    bookRepository.deleteById(id);
}
```

Sau đó, tiến hành thêm phương thức xoá sách dựa theo Id của nó vào file **BookController.java** theo đoạn code dưới đây:

```
@GetMapping("/delete/{id}")
public String deleteBook(@PathVariable long id) {
    bookService.getBookById(id)
        .ifPresentOrElse(
            book -> bookService.deleteBookById(id),
            () -> { throw new IllegalArgumentException("Book not
found"); });
    return "redirect:/books";
}
```

5.3.2 Edit Book

Tạo 2 phương thức sau vào trong file **BookController.java**:

Trong **phương thức GET**, truyền vào đường dẫn có định dạng **/edit/{id}** để lấy thông tin của đầu sách cần chỉnh sửa. Sau đó, chúng ta thêm đối tượng book và danh sách các category vào Model và trả về trang view để hiển thị thông tin đầu sách cần chỉnh sửa.

Trong **phương thức POST**, chúng ta cũng truyền vào đường dẫn có định dạng **/edit** để định tuyến đến phương thức cập nhật sách, tiếp chúng ta cập nhật thông tin của đầu sách và chuyển hướng người dùng đến trang danh sách đầu sách.

```
@GetMapping("/edit/{id}")
public String editBookForm(@NotNull Model model, @PathVariable long id)
{
    var book = bookService.getBookById(id);
    model.addAttribute("book", book.orElseThrow(() -> new
IllegalArgumentException("Book not found")));
    model.addAttribute("categories",
categoryService.getAllCategories());
    return "book/edit";
}

@PostMapping("/edit")
public String editBook(@Valid @ModelAttribute("book") Book book,
                      @NotNull BindingResult bindingResult,
                      Model model) {
    if (bindingResult.hasErrors()) {
        var errors = bindingResult.getAllErrors()
            .stream()
            .map(DefaultMessageSourceResolvable::getDefaultMessage)
            .toArray(String[]::new);
        model.addAttribute("errors", errors);
        model.addAttribute("categories",
categoryService.getAllCategories());
        return "book/edit";
    }
    bookService.updateBook(book);
    return "redirect:/books";
}
```

Trong **BookService.java** ta tiễn hành thêm phương thức cập nhật sách.

```
public void updateBook(@NotNull Book book) {
    Book existingBook = bookRepository.findById(book.getId())
        .orElse(null);
    Objects.requireNonNull(existingBook).setTitle(book.getTitle());
    existingBook.setAuthor(book.getAuthor());
    existingBook.setPrice(book.getPrice());
    existingBook.setCategory(book.getCategory());
    bookRepository.save(existingBook);
}
```

Tiễn hành khởi tạo view **edit.html** nằm tại đường dẫn:

/src/main/resources/templates

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <th:block th:replace="~{layout::link-css}"></th:block>
    <th:block th:replace="~{layout::custom-css}"></th:block>
    <title>Edit Book</title>
</head>
<body>
<th:block th:replace="~{layout::header}"></th:block>
<div class="container">
    <h1>Edit Book</h1>
    <form th:action="@{/books/edit}" th:object="${book}" method="post">
        <input type="hidden" th:field="*{id}">
        <div class="col-6 mb-3">
            <label class="form-label" for="title">Title:</label>
            <input class="form-control" type="text" th:field="*{title}" id="title" required autofocus>
            <span class="text-danger" th:if="#fields.hasErrors('title')"
th:errors="*{title}"></span>
        </div>
        <div class="col-6 mb-3">
            <label class="form-label" for="author">Author:</label>
            <input class="form-control" type="text" th:field="*{author}" id="author">
            <span class="text-danger" th:if="#fields.hasErrors('author')"
th:errors="*{author}"></span>
        </div>
    </form>
</div>
```

```
<div class="col-6 mb-3">
    <label class="form-label" for="price">Price:</label>
    <input class="form-control" type="text" th:field="*{price}"
id="price">
    <span class="text-danger" th:if="#{#fields.hasErrors('price')}"
th:errors="*{price}"></span>
</div>
<div class="col-6 mb-3">
    <label class="form-label" for="category">Category:</label><span
class="text-danger">*</span>
    <select class="form-control" id="category" name="category.id">
        <option value="">-- Select Category --</option>
        <option th:each="category : ${categories}"
th:value="${category.getId()}"
            th:text="${category.getName()}"
            th:selected="${category.getId() ==
book.category.getId()}></option>
    </select>
</div>
<input type="submit" value="Save">
</form>
<br>
<a th:href="@{/books}">Back to List</a>
</div>
<th:block th:replace="~{layout::footer}"></th:block>
</body>
</html>
```

5.4 Xây dựng chức năng tìm kiếm theo từ khóa

Chức năng tìm kiếm theo từ khóa là một tính năng quan trọng trong nhiều ứng dụng và trang web hiện đại. Nó cho phép người dùng nhập từ khóa hoặc cụm từ mà họ quan tâm và tìm kiếm thông tin liên quan. Khi người dùng thực hiện tìm kiếm, hệ thống sẽ tìm trong cơ sở dữ liệu hoặc nội dung trang web để tìm các mục phù hợp với từ khóa đã nhập.

Chức năng tìm kiếm theo từ khóa cung cấp sự thuận tiện và tăng khả năng tìm kiếm thông tin nhanh chóng và chính xác. Nó đã trở thành một yếu tố không thể thiếu trong việc cung cấp trải nghiệm người dùng tốt hơn và giúp họ tiếp cận với thông tin một cách dễ dàng hơn.

Sinh viên có thể sử dụng các công cụ như **Apache Lucene**, **Apache Solr** và **Elasticsearch** để thực hiện chức năng tìm kiếm trong các ứng dụng và trang web. Các công cụ này được phát triển nhằm cung cấp khả năng tìm kiếm mạnh mẽ và hiệu quả.

Tại **BookController.java** ta bổ sung phương thức tìm kiếm sách theo đoạn code dưới đây:

```
@GetMapping("/search")
public String searchBook(
    @NotNull Model model,
    @RequestParam String keyword,
    @RequestParam(defaultValue = "0") Integer pageNo,
    @RequestParam(defaultValue = "20") Integer pageSize,
    @RequestParam(defaultValue = "id") String sortBy) {
    model.addAttribute("books", bookService.searchBook(keyword));
    model.addAttribute("currentPage", pageNo);
    model.addAttribute("totalPages",
        bookService
            .getAllBooks(pageNo, pageSize, sortBy)
            .size() / pageSize);
    model.addAttribute("categories",
        categoryService.getAllCategories());
    return "book/list";
}
```

Sau đó, tiến hành thêm phương thức tìm kiếm theo từ khóa vào file **BookService.java** như sau:

```
public List<Book> searchBook(String keyword) {
    return bookRepository.searchBook(keyword);
}
```

Tiến hành tạo câu truy vấn tìm kiếm theo từ khóa (keyword) tại file **IBookRepository.java** như sau:

```
package fit.hutech.spring.repositories;

import fit.hutech.spring.entities.Book;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.PagingAndSortingRepository;
import org.springframework.stereotype.Repository;
```

```

import java.util.List;

@Repository
public interface IBookRepository extends
PagingAndSortingRepository<Book, Long>, JpaRepository<Book, Long> {
    @Query("""
        SELECT b FROM Book b
        WHERE b.title LIKE %?1%
        OR b.author LIKE %?1%
        OR b.category.name LIKE %?1%
        """)
    List<Book> searchBook(String keyword);
}

```

Build và chạy ứng dụng kiểm tra kết quả:

My Book List					Ánh	Search
<u>Id</u>	<u>Title</u>	<u>Author</u>	<u>Price</u>	<u>Category</u>	Action	
1	Phát triển phần mềm mã nguồn mở	Nguyễn Đình Ánh	12.0	Công nghệ phần mềm	Edit Delete	
2	Cơ sở dữ liệu phân tán	Cao Tùng Anh	10.0	Hệ thống thông tin	Edit Delete	

Hình 5.10. Kết quả tìm kiếm bằng từ khóa

5.5 Xây dựng chức năng Checkout

Đầu tiên, ta khởi tạo 2 file **ItemInvoice.java** và **Invoice.java** trong thư mục **entities** với đường dẫn **/src/main/java/fit.hutech.spring.entities**. Ta bổ sung code cho từng file với nội dung dưới đây:

Invoice.java

```

package fit.hutech.spring.entities;

import jakarta.persistence.*;
import jakarta.validation.constraints.Positive;
import lombok.*;
import org.hibernate.Hibernate;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.Objects;

@Getter

```

```

@Setter
@ToString
@RequiredArgsConstructor
@AllArgsConstructor
@Builder
@Entity
@Table(name = "invoices")
public class Invoice {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long id;

    @Column(name = "invoice_date")
    private Date invoiceDate = new Date();

    @Column(name = "total")
    @Positive(message = "Total must be positive")
    private Double price;

    @OneToMany(mappedBy = "invoice", cascade = CascadeType.ALL)
    @ToString.Exclude
    private List<ItemInvoice> itemInvoices = new ArrayList<>();

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || Hibernate.getClass(this) != Hibernate.getClass(o)) return false;
        Invoice invoice = (Invoice) o;
        return getId() != null && Objects.equals(getId(),
        invoice.getId());
    }

    @Override
    public int hashCode() {
        return getClass().hashCode();
    }
}

```

ItemInvoice.java

```
package fit.hutech.spring.entities;
```

```
import jakarta.persistence.*;
import jakarta.validation.constraints.Positive;
import lombok.*;
import org.hibernate.Hibernate;

import java.util.Objects;

@Getter
@Setter
@ToString
@RequiredArgsConstructor
@AllArgsConstructor
@Builder
@Entity
@Table(name = "item_invoice")
public class ItemInvoice {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "quantity")
    @Positive(message = "Quantity must be positive")
    private int quantity;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "book_id", referencedColumnName = "id")
    @ToString.Exclude
    private Book book;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "invoice_id", referencedColumnName = "id")
    @ToString.Exclude
    private Invoice invoice;

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || Hibernate.getClass(this) != Hibernate.getClass(o)) return false;
        ItemInvoice that = (ItemInvoice) o;
        return getId() != null && Objects.equals(getId(),
        that.getId());
    }

    @Override
    public int hashCode() {
```

```

        return getClass().hashCode();
    }
}

```

Ta thêm rằng buộc 1-n giữa **Book** và **Item Invoice** trong file **Book.java** được đặt tại đường dẫn **/src/main/java/fit.hutech.spring.entities**. Với nội dung như sau:

```

package fit.hutech.spring.entities;

import fit.hutech.spring.validators.annotations.ValidCategoryId;
import jakarta.persistence.*;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.Positive;
import jakarta.validation.constraints.Size;
import lombok.*;
import org.hibernate.Hibernate;

import java.util.ArrayList;
import java.util.List;
import java.util.Objects;

@Getter
@Setter
@ToString
@RequiredArgsConstructor
@AllArgsConstructor
@Builder
@Entity
@Table(name = "book")
public class Book {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "title", length = 50, nullable = false)
    @Size(min = 1, max = 50, message = "Title must be between 1 and 50 characters")
    @NotBlank(message = "Title must not be blank")
    private String title;

    @Column(name = "author", length = 50, nullable = false)
    @Size(min = 1, max = 50, message = "Author must be between 1 and 50 characters")
    @NotBlank(message = "Author must not be blank")
}

```

```
private String author;

@Column(name = "price")
@Positive(message = "Price must be greater than 0")
private Double price;

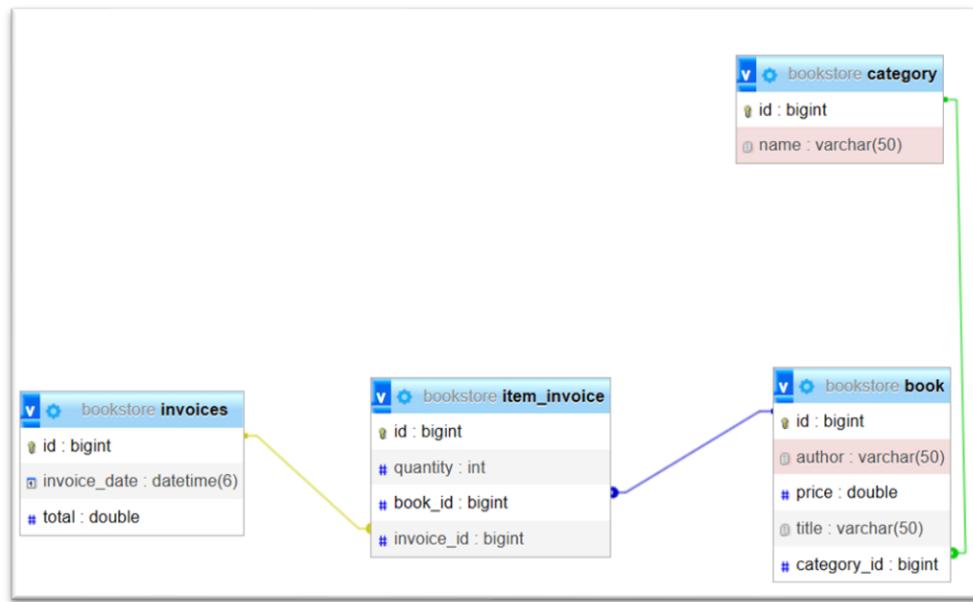
@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "category_id", referencedColumnName = "id")
@ValidCategoryId
@ToString.Exclude
private Category category;

@OneToMany(mappedBy = "book", cascade = CascadeType.ALL)
@ToString.Exclude
private List<ItemInvoice> itemInvoices = new ArrayList<>();

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || Hibernate.getClass(this) !=
Hibernate.getClass(o)) return false;
    Book book = (Book) o;
    return getId() != null && Objects.equals(getId(),
book.getId());
}

@Override
public int hashCode() {
    return getClass().hashCode();
}
}
```

Nhấn Build và chạy ứng dụng. Sau đó truy cập phpMyAdmin xem kết quả:



Hình 5.11. Diagram sau khi tạo ra item và invoice

Tiến hành khởi tạo 2 repository đó là **IInvoiceRepository.java** và **IIItemInvoiceRepository.java** nằm tại thư mục **repositories** với đường dẫn: **/src/main/java/fit.hutech.spring.repositories**

Sau đó bổ sung các đoạn code sau:

IInvoiceRepository.java

```

package fit.hutech.spring.repositories;

import fit.hutech.spring.entities.Invoice;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface IInvoiceRepository extends JpaRepository<Invoice,
Long>{
}
    
```

IIItemInvoiceRepository.java

```
package fit.hutech.spring.repositories;

import fit.hutech.spring.entities.ItemInvoice;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface IItemInvoiceRepository extends
JpaRepository<ItemInvoice, Long>{}
```

Sau đó, tại **CartService.java** nằm tại thư mục **services** ta thêm các phương thức lưu thông tin giỏ hàng của người dùng vào cơ sở dữ liệu như sau:

```
package fit.hutech.spring.services;

import fit.hutech.spring.daos.Cart;
import fit.hutech.spring.daos.Item;
import fit.hutech.spring.entities.Invoice;
import fit.hutech.spring.entities.ItemInvoice;
import fit.hutech.spring.repositories.IBookRepository;
import fit.hutech.spring.repositories.IInvoiceRepository;
import fit.hutech.spring.repositories.IItemInvoiceRepository;
import jakarta.servlet.http.HttpSession;
import jakarta.validation.constraints.NotNull;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Isolation;
import org.springframework.transaction.annotation.Transactional;

import java.util.Date;
import java.util.Optional;

@Service
@RequiredArgsConstructor
@Transactional(isolation = Isolation.SERIALIZABLE,
    rollbackFor = {Exception.class, Throwable.class})
public class CartService {
    private static final String CART_SESSION_KEY = "cart";

    private final IInvoiceRepository invoiceRepository;

    private final IItemInvoiceRepository itemInvoiceRepository;

    private final IBookRepository bookRepository;
```

```
public Cart getCart(@NotNull HttpSession session) {
    return Optional.ofNullable(Cart)
        .get(session.getAttribute(CART_SESSION_KEY))
        .orElseGet(() -> {
            Cart cart = new Cart();
            session.setAttribute(CART_SESSION_KEY, cart);
            return cart;
        });
}

public void updateCart(@NotNull HttpSession session, Cart cart) {
    session.setAttribute(CART_SESSION_KEY, cart);
}

public void removeCart(@NotNull HttpSession session) {
    session.removeAttribute(CART_SESSION_KEY);
}

public int getSumQuantity(@NotNull HttpSession session) {
    return getCart(session).getCartItems().stream()
        .mapToInt(Item::getQuantity)
        .sum();
}

public double getSumPrice(@NotNull HttpSession session) {
    return getCart(session).getCartItems().stream()
        .mapToDouble(item -> item.getPrice() *
item.getQuantity())
        .sum();
}

public void saveCart(@NotNull HttpSession session) {
    var cart = getCart(session);
    if (cart.getCartItems().isEmpty()) return;

    var invoice = new Invoice();
    invoice.setInvoiceDate(new Date(new Date().getTime()));
    invoice.setPrice(getSumPrice(session));
    invoiceRepository.save(invoice);

    cart.getCartItems().forEach(item -> {
        var items = new ItemInvoice();
        items.setInvoice(invoice);
        items.setQuantity(item.getQuantity());
        items.setBook(bookRepository.findById(item.getBookId()))
    });
}
```

```
        .orElseThrow());
    itemInvoiceRepository.save(items);
}

removeCart(session);
}
```

Tại **CartController.java** ta bổ sung phương thức xử lý cho việc checkout giỏ hàng như sau:

```
@GetMapping("/checkout")
public String checkout(HttpSession session) {
    cartService.saveCart(session);
    return "redirect:/cart";
}
```

Sau đó build và chạy thử ứng dụng. Sinh viên thêm một vài cuốn sách vào giỏ hàng và ấn nút Check out. Sau đó nhận xét kết quả xảy ra?

TÓM TẮT

Trong Java Spring framework, Spring Data JPA cung cấp các chức năng CRUD (Create, Read, Update, Delete) để tương tác với cơ sở dữ liệu. Nó giúp đơn giản hóa việc quản lý dữ liệu trong ứng dụng.

Để thêm mới dữ liệu, chúng ta có thể sử dụng các phương thức của Spring Data JPA như `save()` hoặc `saveAll()` để lưu đổi tượng mới vào cơ sở dữ liệu. Để xoá dữ liệu, chúng ta cần truy vấn đến đổi tượng cần xoá và sử dụng phương thức `delete()` để xoá đổi tượng đó khỏi cơ sở dữ liệu. Để sửa đổi dữ liệu, chúng ta có thể truy vấn đến đổi tượng cần sửa đổi, thực hiện các thay đổi trên đổi tượng đó và sử dụng phương thức `save()` để cập nhật thông tin đã thay đổi vào cơ sở dữ liệu..

Validation trong Spring là quá trình kiểm tra tính hợp lệ của dữ liệu đầu vào trước khi nó được lưu vào cơ sở dữ liệu. Điều này đảm bảo rằng dữ liệu được lưu trữ trong cơ sở dữ liệu là chính xác và đáng tin cậy. Spring cung cấp các annotation và interfaces như `@Validated`, `@NotNull`, `@NotEmpty`, `@Size` để thực hiện validation trên các đối tượng. Validation có thể được áp dụng trước khi thực hiện các thao tác CRUD bằng cách sử dụng các chú thích và validator trong Spring Data JPA. Khi dữ liệu không hợp lệ được tìm thấy, Spring sẽ tạo ra các thông báo lỗi tương ứng để thông báo cho người dùng. Các thông báo lỗi này có thể được tùy chỉnh và xử lý theo yêu cầu của ứng dụng.

CÂU HỎI ÔN TẬP

1. Spring Data JPA là gì và nó được sử dụng để làm gì trong phát triển ứng dụng?
2. Làm thế nào để tùy chỉnh quy tắc validation trong Spring cho phù hợp với yêu cầu của ứng dụng?
3. Làm thế nào để tạo các truy vấn dữ liệu sử dụng Spring Data JPA?
4. Có những annotation quan trọng nào trong Spring Data JPA và chúng được sử dụng như thế nào?

BÀI 6 XÂY DỰNG CHỨC NĂNG XÁC THỰC NGƯỜI DÙNG

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

Bài này nhằm giúp người học tiếp cận với các khái niệm và kỹ thuật cần thiết để triển khai chức năng đăng nhập và tạo tài khoản người dùng một cách an toàn và linh hoạt. Mục tiêu của chúng ta là phát triển một chức năng đăng ký cho người dùng, đảm bảo tính toàn vẹn và bảo mật thông tin cá nhân. Chức năng đăng ký này cho phép người dùng tạo tài khoản mới bằng cách đặt mật khẩu và xác nhận mật khẩu. Ngoài ra, người dùng cũng sẽ có thể nhập thông tin cá nhân của mình và xác thực thông tin này. Điều này giúp chúng ta đảm bảo rằng chỉ có những người dùng chính xác mới có thể truy cập và sử dụng các tính năng của hệ thống.

Qua việc triển khai chức năng đăng nhập và tạo tài khoản người dùng, chúng ta có thể xây dựng một hệ thống đáng tin cậy và an toàn, đồng thời tạo điều kiện thuận lợi cho người dùng tiếp cận và sử dụng các dịch vụ của chúng ta.

Mô tả chức năng:

Thêm chức năng đăng ký vào tạo tài khoản người dùng trên Spring Boot là một mục tiêu quan trọng để đáp ứng nhu cầu của người dùng và cải thiện trải nghiệm của họ khi sử dụng ứng dụng. Với chức năng đăng ký, người dùng có thể tạo tài khoản cá nhân và lưu trữ thông tin của mình trên hệ thống của chúng ta.

Thêm vào đó, việc thêm chức năng đăng ký sẽ giúp chúng ta thu thập thông tin từ người dùng. Chúng ta có thể sử dụng thông tin đăng ký của người dùng để tùy chỉnh trải nghiệm của họ, cung cấp các tính năng mới, và nâng cao chất lượng dịch vụ của chúng ta.

Vì vậy, mục tiêu của chúng ta khi thêm chức năng đăng ký vào tạo tài khoản người dùng trên Spring Boot là tạo ra một trải nghiệm dễ sử dụng và bảo mật cho người dùng, thu thập thông tin và cải thiện tính năng của ứng dụng của chúng ta, và đáp ứng nhu cầu của người dùng.

6.1 Xây dựng thêm table mới là User

Để tạo lớp **User.java** trong thư mục **Entities** với đường dẫn **src/main/java/fit.hutech.spring/entities**, chúng ta sẽ xác định thông tin về người dùng và quyền truy cập. Lớp **User.java** sẽ đại diện cho thông tin người dùng bao gồm tên đăng nhập, mật khẩu, địa chỉ email và các thông tin cá nhân khác. Ngoài ra, lớp này cũng có thể bao gồm các phương thức để xử lý các hoạt động liên quan đến người dùng như đăng ký, đăng nhập, cập nhật thông tin người dùng và khôi phục mật khẩu.

Việc tạo lớp **User.java** và định nghĩa các thuộc tính và phương thức liên quan sẽ giúp chúng ta xây dựng một hệ thống quản lý người dùng chuyên nghiệp và linh hoạt. Lớp này sẽ đảm bảo tính bảo mật và toàn vẹn của thông tin người dùng, đồng thời cung cấp các chức năng quan trọng như đăng ký và quản lý tài khoản người dùng.

Đường dẫn đến lớp **User.java** trong thư mục **Entities** sẽ là **src/main/java/fit.hutech.spring/entities**, và thông qua việc triển khai các phương thức liên quan, chúng ta có thể xử lý các hoạt động quan trọng về người dùng một cách chuyên nghiệp và hiệu quả.

Thêm nội dung code vào file **User.java** như bên dưới

```
package fit.hutech.spring.entities;

import fit.hutech.spring.validators.annotations.ValidUsername;
import jakarta.persistence.*;
import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.Pattern;
import jakarta.validation.constraints.Size;
import lombok.*;
import org.hibernate.Hibernate;
import org.hibernate.validator.constraints.Length;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

import java.util.Collection;
import java.util.HashSet;
import java.util.Set;
```

```
import java.util.Objects;

@Getter
@Setter
@ToString
@RequiredArgsConstructor
@AllArgsConstructor
@Builder
@Entity
@Table(name = "user")
public class User implements UserDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "username", length = 50, unique = true)
    @NotBlank(message = "Username is required")
    @Size(min = 1, max = 50, message = "Username must be between 1 and
50 characters")
    @ValidUsername
    private String username;

    @Column(name = "password", length = 250)
    @NotBlank(message = "Password is required")
    private String password;

    @Column(name = "email", length = 50, unique = true)
    @NotBlank(message = "Email is required")
    @Size(min = 1, max = 50, message = "Email must be between 1 and 50
characters")
    @Email
    private String email;

    @Column(name = "phone", length = 10, unique = true)
    @Length(min = 10, max = 10, message = "Phone must be 10
characters")
    @Pattern(regexp = "[0-9]*$", message = "Phone must be number")
    private String phone;

    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL)
    @ToString.Exclude
    private Set<Invoice> invoices = new HashSet<>();

    @Override
```

```
public Collection<? extends GrantedAuthority> getAuthorities() {
    return Collections.emptyList();
}

@Override
public String getPassword() {
    return password;
}

@Override
public String getUsername() {
    return username;
}

@Override
public boolean isAccountNonExpired() {
    return true;
}

@Override
public boolean isAccountNonLocked() {
    return true;
}

@Override
public boolean isCredentialsNonExpired() {
    return true;
}

@Override
public boolean isEnabled() {
    return true;
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || Hibernate.getClass(this) !=
Hibernate.getClass(o)) return false;
    User user = (User) o;
    return getId() != null && Objects.equals(getId(),
user.getId());
}

@Override
public int hashCode() {
```

```
        return getClass().hashCode();
    }
}
```

```
package com.example.demo.entity;

import com.example.demo.validator.annotation.ValidUsername;
import jakarta.persistence.*;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.Size;
import lombok.Data;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
```

Hình 6.1. Thông báo lỗi chưa định nghĩa lớp ValidUsername

ValidUsername dùng để kiểm tra tính hợp lệ của tên người dùng trong ứng dụng. Có thể được định nghĩa là một chuỗi ký tự chỉ chứa các ký tự chữ cái, số và dấu gạch dưới, không được bắt đầu bằng số hoặc dấu gạch dưới, và không quá dài hoặc quá ngắn.

Tiến hành tạo thêm file tên là **ValidUsername.java** trong thư mục annotations theo đường dẫn sau:

src/main/java/fit.hutech.spring/validators/annotations

```
package fit.hutech.spring.validators.annotations;

import fit.hutech.spring.validators.ValidUsernameValidator;
import jakarta.validation.Constraint;
import jakarta.validation.Payload;

import java.lang.annotation.Retention;
import java.lang.annotation.Target;

import static java.lang.annotation.ElementType.FIELD;
import static java.lang.annotation.ElementType.TYPE;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
```

```

@Target({TYPE, FIELD})
@Retention(RUNTIME)
@Constraint(validatedBy = ValidUsernameValidator.class)
public @interface ValidUsername {
    String message() default "Username already exists";
    Class<?>[] groups() default {};
    Class<? extends Payload>[] payload() default {};
}

```

Trong tệp **ValidUsername.java**, đã thông báo rằng thiếu class **ValidUsernameValidator**. Để giải quyết vấn đề này, chúng ta cần thêm một tệp **ValidUsernameValidator.java** vào thư mục validator với đường dẫn sau:

src/main/java/fit.hutech.spring/validators

```

package fit.hutech.spring.validators;

import fit.hutech.spring.services.UserService;
import fit.hutech.spring.validators.annotations.ValidUsername;
import jakarta.validation.ConstraintValidator;
import jakarta.validation.ConstraintValidatorContext;
import lombok.RequiredArgsConstructor;

@RequiredArgsConstructor
public class ValidUsernameValidator implements
ConstraintValidator<ValidUsername, String> {
    private final UserService userService;

    @Override
    public boolean isValid(String username, ConstraintValidatorContext
context) {
        return userService.findByUsername(username).isEmpty();
    }
}

```

Interface **IUserRepository** là một thành phần trong Java, thường được sử dụng trong kiến trúc phần mềm hướng đối tượng (OOP) để truy xuất và lưu trữ dữ liệu liên quan đến người dùng. Nó được định nghĩa một tập hợp các phương thức để thao tác với dữ liệu người dùng, bao gồm các hoạt động như tìm kiếm, thêm, cập nhật và xóa người dùng. Các phương thức này có thể được cài đặt trong các lớp cụ thể khác nhau tuỳ thuộc vào nguồn dữ liệu mà ứng dụng sử dụng.

Tại 1 file tên **IUserRepository.java** trong thư mục theo đường dẫn **src/main/java/fit.hutech.spring/repositories**

```
package fit.hutech.spring.repositories;

import fit.hutech.spring.entities.User;
import org.springframework.data.jpa.repository.JpaRepository;

@Repository
public interface IUserRepository extends JpaRepository<User, String> {
    User findByUsername(String username);
}
```

Tại file **Invoice.java** ta bổ sung đoạn code khởi tạo quan hệ 1-n giữa **User** với **Invoice** như sau:

```
@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "user_id", referencedColumnName = "id")
@ToString.Exclude
private User user;
```

Tiếp theo, điều chỉnh **UserService.java** được đặt tại đường dẫn sau **src/main/java/fit.hutech.spring/services**

Class **UserService** sẽ implement interface **UserDetailsService**, cung cấp phương thức **loadUserByUsername** để tìm kiếm thông tin người dùng theo tên đăng nhập.

```
package fit.hutech.spring.services;

import fit.hutech.spring.entities.User;
import fit.hutech.spring.repositories.IUserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.userdetails.UserDetails;
import
org.springframework.security.core.userdetails.UserDetailsService;
import jakarta.validation.constraints.NotNull;
import lombok.extern.slf4j.Slf4j;
import
org.springframework.security.core.userdetails.UsernameNotFoundException;
;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Isolation;
import org.springframework.transaction.annotation.Transactional;
```

```

@Service
@Slf4j
public class UserService implements UserDetailsService {
    @Autowired
    private IUserRepository userRepository;

    @Transactional(isolation = Isolation.SERIALIZABLE,
                  rollbackFor = {Exception.class, Throwable.class})
    public void save(@NotNull User user) {
        user.setPassword(new BCryptPasswordEncoder()
                         .encode(user.getPassword()));
        userRepository.save(user);
    }

    public Optional<User> findByUsername(String username) throws
    UsernameNotFoundException {
        return userRepository.findByUsername(username);
    }

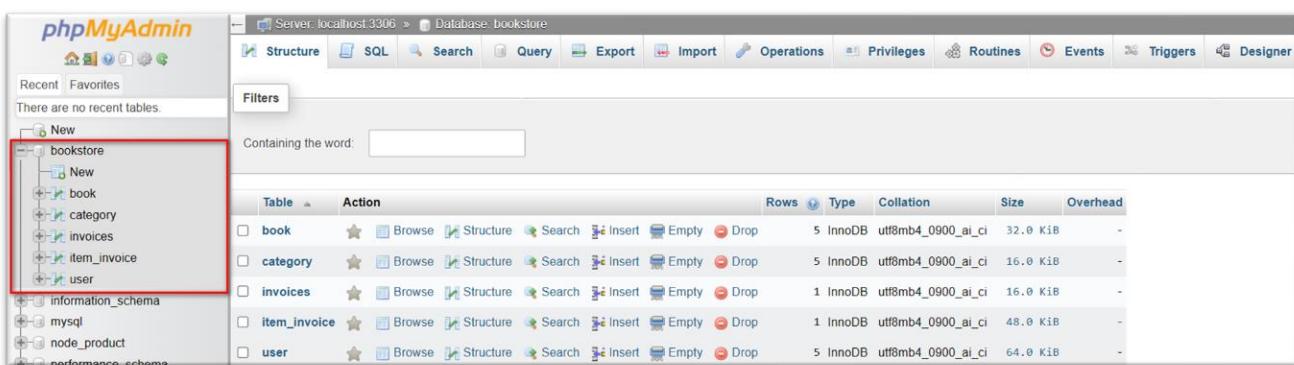
    @Override
    public UserDetails loadUserByUsername(String username) throws
    UsernameNotFoundException {
        return userRepository.findByUsername(username);
    }
}

```

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 các table đã thêm tương ứng User.

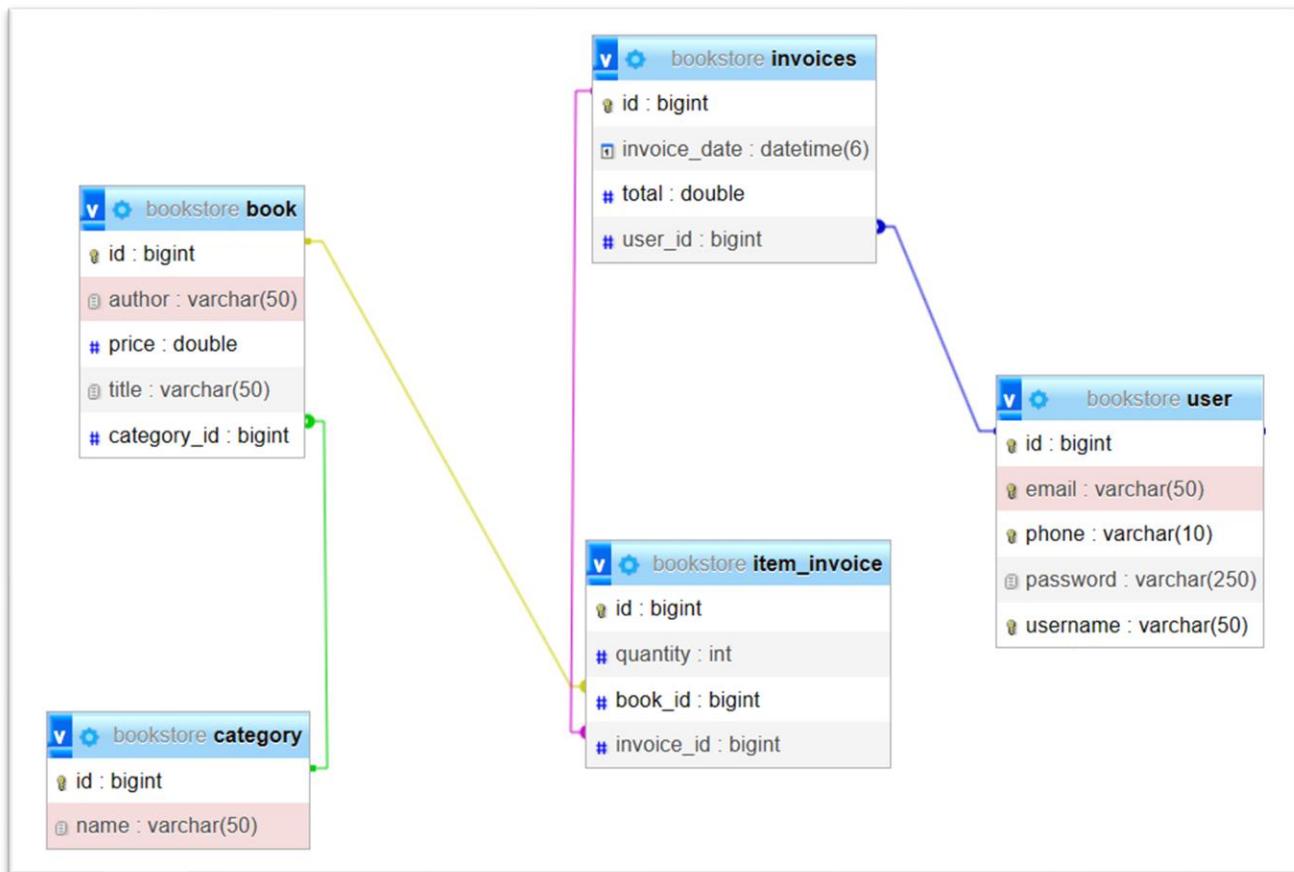


Hình 6.2. Kiểm tra database sau khi thêm table User

The screenshot shows the phpMyAdmin interface for a database named 'bookstore'. On the left, there's a tree view of schemas: 'bookstore' (selected), 'New', 'information_schema', 'mysql', 'node_product', and 'performance_schema'. A red box highlights the 'bookstore' schema. A blue circle with the number '1' is placed over the 'bookstore' node. At the top right, there are tabs for Structure, SQL, Search, Query, Import, Operations, Privileges, Routines, Events, Triggers, and Designer. A blue circle with the number '2' is placed over the 'Designer' tab.

Table	Action	Rows	Type	Collation	Size	Overhead
book	★ Browse Structure Search Insert Empty Drop	5	InnoDB	utf8mb4_0900_ai_ci	32.0 Kib	-
category	★ Browse Structure Search Insert Empty Drop	5	InnoDB	utf8mb4_0900_ai_ci	16.0 Kib	-
invoices	★ Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_0900_ai_ci	16.0 Kib	-
item_invoice	★ Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_0900_ai_ci	48.0 Kib	-
user	★ Browse Structure Search Insert Empty Drop	5	InnoDB	utf8mb4_0900_ai_ci	64.0 Kib	-

Hình 6.3. Mở Designer và xem kết quả



Hình 6.4. Sơ đồ Designer của database bookstore

Tiếp theo, chúng ta sẽ tạo một class mới có tên **UserService.java** trong thư mục services theo đường dẫn **src/main/java/fit.hutech.spring/services**.

Lớp **UserService** đóng vai trò quan trọng trong việc tổ chức và quản lý các hoạt động liên quan đến người dùng và vai trò của họ trong hệ thống. Viết code cho lớp này sẽ đảm bảo tính nhất quán và hiệu quả trong việc quản lý người dùng và vai trò của họ.

```
package fit.hutech.spring.services;

import fit.hutech.spring.entities.User;
import fit.hutech.spring.repositories.IUserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import jakarta.validation.constraints.NotNull;
import lombok.extern.slf4j.Slf4j;
import org.springframework.transaction.annotation.Isolation;
import org.springframework.transaction.annotation.Transactional;

@Service
@Slf4j
public class UserService {
    @Autowired
    private IUserRepository userRepository;

    @Transactional(isolation = Isolation.SERIALIZABLE,
        rollbackFor = {Exception.class, Throwable.class})
    public void Save(@NotNull User user) {
        userRepository.save(user);
    }
}
```

Tạo file **UserController.java** được đặt tại đường dẫn sau đây:

src/main/java/fit.hutech.spring/controllers

```
package fit.hutech.spring.controllers;

import fit.hutech.spring.entities.User;
import fit.hutech.spring.services.UserService;
import jakarta.validation.Valid;
import jakarta.validation.constraints.NotNull;
import lombok.RequiredArgsConstructor;
import org.springframework.context.support.DefaultMessageSourceResolvable;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
```

```
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;

@Controller
@RequestMapping("/")
@RequiredArgsConstructor
public class UserController {
    private final UserService userService;

    @GetMapping("/login")
    public String login() {
        return "user/login";
    }

    @GetMapping("/register")
    public String register(@NotNull Model model) {
        model.addAttribute("user", new User());
        return "user/register";
    }

    @PostMapping("/register")
    public String register(@Valid @ModelAttribute("user") User user,
                          @NotNull BindingResult bindingResult,
                          Model model) {
        if (bindingResult.hasErrors()) {
            var errors = bindingResult.getAllErrors()
                .stream()
                .map(DefaultMessageSourceResolvable::getDefaultMessage)
                .toArray(String[]::new);
            model.addAttribute("errors", errors);
            return "user/register";
        }
        userService.save(user);
        return "redirect:/login";
    }
}
```

Tạo thư mục **utils** đặt tại **src/main/java/fit.hutech.spring/utils**

Tạo file **SecurityConfig.java** đặt tại **src/main/java/fit.hutech.spring/Utils** dán đoạn code bên dưới vào.

SecurityConfig cung cấp các cấu hình bảo mật cho ứng dụng web. Cụ thể:

- ⊕ Phương thức **userDetailsService()** trả về một CustomUserDetailService, được sử dụng để cung cấp thông tin chi tiết về người dùng.
- ⊕ Phương thức **passwordEncoder()** trả về một BCryptPasswordEncoder, được sử dụng để mã hóa mật khẩu của người dùng.
- ⊕ Phương thức **authenticationProvider()** sẽ trả về một DaoAuthenticationProvider, được sử dụng để định nghĩa cách xác thực người dùng.
- ⊕ Phương thức **securityFilterChain(HttpSecurity http)** trả về một SecurityFilterChain, được sử dụng để cấu hình các hành vi bảo mật trong ứng dụng.

```

package fit.hutech.spring.utils;

import fit.hutech.spring.services.UserService;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import jakarta.validation.constraints.NotNull;
import
org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.core.userdetails.UserDetailsService;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
@EnableWebSecurity
@EnableMethodSecurity(securedEnabled = true, jsr250Enabled = true)
public class SecurityConfig {
    @Bean
    public UserDetailsService userDetailsService() {
        return new UserService();
    }
}

```

```
@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}

@Bean
public DaoAuthenticationProvider authenticationProvider() {
    var auth = new DaoAuthenticationProvider();
    auth.setUserDetailsService(userDetailsService());
    auth.setPasswordEncoder(passwordEncoder());
    return auth;
}

@Bean
public SecurityFilterChain securityFilterChain(@NotNull
HttpSecurity http) throws Exception {
    return http.authorizeHttpRequests(auth -> auth
        .requestMatchers( "/css/**", "/js/**", "/",
"/register", "/error")
        .permitAll()
        .requestMatchers( "/books/edit",
"/books/delete")
        .authenticated()
        .requestMatchers( "/books", "/books/add")
        .authenticated()
        .requestMatchers( "/api/**")
        .authenticated()
        .anyRequest().authenticated()
    )
    .logout(Logout ->
        Logout.logoutUrl("/logout")
        .logoutSuccessUrl("/login")
        .deleteCookies("JSESSIONID")
        .invalidateHttpSession(true)
        .clearAuthentication(true)
        .permitAll()
    )
    .formLogin(formLogin ->
        formLogin.loginPage("/login")
        .loginProcessingUrl("/login")
        .defaultSuccessUrl("/")
        .failureUrl("/login?error=true")
        .permitAll()
    )
    .rememberMe(rememberMe ->
        rememberMe.key("hutech")
    )
}
```

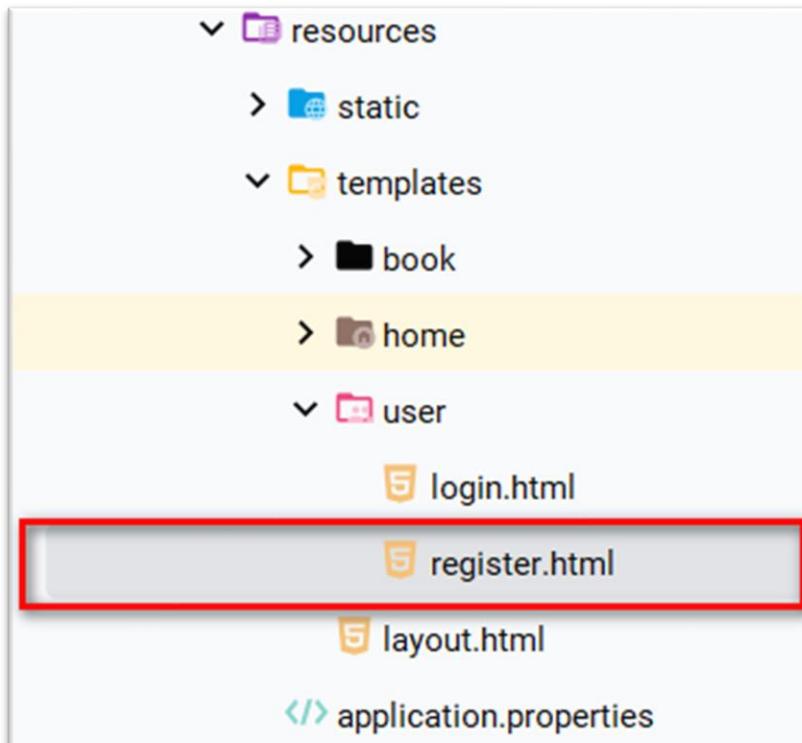
```

        .rememberMeCookieName("hutech")
        .tokenValiditySeconds(24 * 60 * 60)
        .userDetailsService(userDetailsService())
    )
    .exceptionHandling(exceptionHandling ->
        exceptionHandling.accessDeniedPage("/403"))
    .sessionManagement(sessionManagement ->
        sessionManagement.maximumSessions(1)
        .expiredUrl("/login"))
)
.httpBasic(httpBasic -> httpBasic.realmName("hutech"))
.build();
}
}

```

6.2 Thiết kế giao diện đăng nhập và đăng ký

- Tạo thư mục *user* tại đường dẫn ***src/main/resources/templates***
- Tạo 2 view *login.html* và *register.html* đặt tại thư mục
src/main/resources/templates/user



Hình 6.5. Thêm 2 file *login.html* và *register.html*

Thêm nội dung code cho file **login.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Login</title>
    <th:block th:replace="~{layout::link-css}"></th:block>
    <th:block th:replace="~{layout::custom-css}"></th:block>
</head>
<body style="background: #f6f7fc;">
<th:block th:replace="~{layout::header}"></th:block>
<div class="container py-5 h-100">
    <div class="row justify-content-center align-items-center h-100">
        <div class="col-12 col-lg-6 col-md-8 col-xl-5">
            <div class="card shadow" style="border-radius: 1rem;">
                <div class="card-body p-5 text-center">
                    <h3>Login to <strong>Bookstore</strong></h3>
                    <p class="mb-4 text-secondary">Enter your username
and password to access your account.</p>
                    <form th:action="@{/login}" method="post">
                        <div th:if="${param.error}" class="alert alert-
danger">
                            Invalid username and password.
                        </div>
                        <div th:if="${param.logout}" class="alert
alert-success">
                            You have been logged out.
                        </div>
                        <div class="form-group mb-4">
                            <label for="username"></label>
                            <input type="text" class="form-control"
required id="username" name="username" placeholder="Username">
                        </div>
                        <div class="form-group mb-4">
                            <label for="password"></label>
                            <input type="password" class="form-control"
required id="password" name="password" placeholder="Password">
                        </div>
                        <div class="form-check d-flex justify-content-
start mb-4">
                            <input type="checkbox" class="form-check-
input" name="remember-me" id="remember-me">
                        </div>
                    </form>
                </div>
            </div>
        </div>
    </div>
</div>
```

```

        <label class="form-check-label"
for="remember-me"> Remember me</label>
    </div>
    <div class="d-grid gap-2 form-action">
        <button type="submit" class="btn btn-
primary btn-lg btn-block">Login</button>
        <p class="mt-3 mb-0">Don't have an account?
<a class="text-info text-center" th:href="${'/register'}">Sign up?</a>
</p>
    </div>
</form>
</div>
</div>
</div>
</div>
<th:block th:replace=~{layout::footer}></th:block>
</body>
</html>

```

Thêm nội dung code cho file **register.html**

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Sign Up</title>
    <th:block th:replace=~{layout :: link-css}></th:block>
    <th:block th:replace=~{layout::custom-css}></th:block>
</head>
<body style="background: #f6f7fc;">
<th:block th:replace=~{layout :: header}></th:block>
<div class="container py-5 h-100">
    <div class="row justify-content-center align-items-center h-100">
        <div class="col-12 col-lg-6 col-md-8 col-xl-5">
            <div class="card shadow" style="border-radius: 1rem;">
                <div class="card-body p-5 text-center">
                    <h3>Sign up to <strong>Bookstore</strong></h3>
                    <p class="mb-4 text-secondary">Welcome to
Bookstore, please sign up to continue.</p>
                    <form th:action="@{/register}" method="post">
                        <div th:if="${errors}" class="alert alert-
danger justify-content-center" role="alert">

```

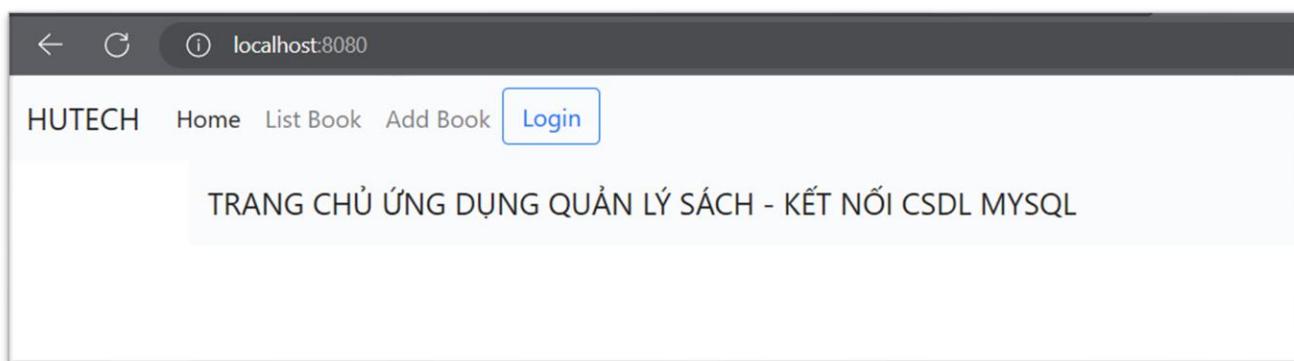
```
<ul>
    <li th:each="error : ${errors}"
th:text="${error}" class="text-danger text-start"></li>
</ul>
</div>
<div class="form-group mb-4">
    <label for="email"></label>
    <input type="email" class="form-control"
required id="email" name="email" placeholder="Enter your email">
</div>
<div class="form-group mb-4">
    <label for="username"></label>
    <input type="text" class="form-control"
required id="username" name="username" placeholder="Enter your
username">
</div>
<div class="form-group mb-4">
    <label for="password"></label>
    <input type="password" class="form-control"
required id="password" name="password" placeholder="Enter your
password">
</div>
<div class="form-group mb-4">
    <label for="phone"></label>
    <input type="tel" class="form-control"
required id="phone" name="phone" placeholder="Enter your phone">
</div>
<div class="d-grid gap-2 form-action">
    <button type="submit" class="btn btn-
primary btn-lg btn-block">Sign up</button>
    <p class="mt-3 mb-0">Already have an
account? <a class="text-info text-center" th:href="${'/login'}">Login
in?</a></p>
</div>
</form>
</div>
</div>
</div>
</div>
<th:block th:replace="~{layout :: footer}"></th:block>
</body>
</html>
```

Chỉnh sửa file **layout.html**

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/extras/spring-security"
      lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>My App</title>
    <link th:fragment="link-css" rel="stylesheet"
th:href="@{/css/bootstrap.min.css}">
    <link th:fragment="custom-css" rel="stylesheet"
th:href="@{/css/style.css}">
</head>
<body class="d-flex flex-column h-100">
<header th:fragment="header">
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
        <div class="container-fluid">
            <a class="navbar-brand" href="/">HUTECH</a>
            <button class="navbar-toggler" type="button" data-bs-
toggle="collapse"
                    data-bs-target="#navbarSupportedContent"
                    aria-controls="navbarSupportedContent"
                    aria-expanded="false" aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse"
id="navbarSupportedContent">
                <ul class="navbar-nav me-auto mb-2 mb-lg-0">
                    <li class="nav-item">
                        <a class="nav-link active" aria-current="page"
href="/">Home</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="/books">
                            List Book
                        </a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="/books/add">
                            Add Book
                        </a>
                    </li>
                    <li sec:authorize="isAuthenticated()">
                        <form th:action="@{/logout}" method="post">
```

```
<button class="btn btn-outline-danger"
type="submit">Logout</button>
</form>
</li>
<li sec:authorize="!isAuthenticated()">
    <a class="btn btn-outline-primary" href="/login">
        Login
    </a>
</li>
</ul>
</div>
</div>
</nav>
</header>
<div class="container">
    <div th:fragment="content"></div>
</div>
<footer th:fragment="footer" class="footer mt-auto py-3 bg-light">
    <div class="container text-center">Copyright ©;
        <span th:text="#dates.year(#dates.createNow())"></span>
        <a href="https://www.hutech.edu.vn/">HUTECH Education</a>
    </div>
    <script th:src="@{/js/bootstrap.min.js}"></script>
    <script th:src="@{/js/jquery-3.7.0.min.js}"></script>
</footer>
</body>
</html>
```

Build và chạy ứng dụng tại địa chỉ **localhost:8080**



Hình 6.6. Giao diện web chưa đăng nhập

Kiểm tra bằng cách tạo tài khoản. Theo các bước sau:

HUTECH Home List Book Add Book **Login**

TRANG CHỦ ỨNG DỤNG QUẢN LÝ SÁCH - KẾT NỐI CSDL MYSQL

Hình 6.7. Tạo tài khoản mới

Kiểm tra **valid dữ liệu** bằng cách xoá các thuộc tính **required** trong các thẻ **input**, sau đó trở lại trang Sign Up không nhập và ấn nút **sign up**

HUTECH Home List Book Add Book **Login**

Sign up to Bookstore

Welcome to Bookstore, please sign up to continue.

- Email must be between 1 and 50 characters
- Phone must be 10 characters
- Email is required
- Password is required
- Username is required
- Username must be between 1 and 50 characters

Enter your email

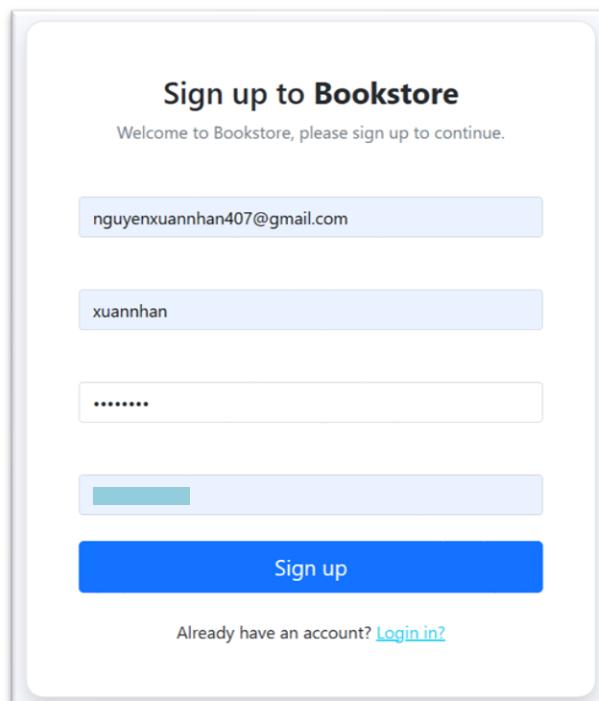
Enter your username

Enter your password

Enter your phone

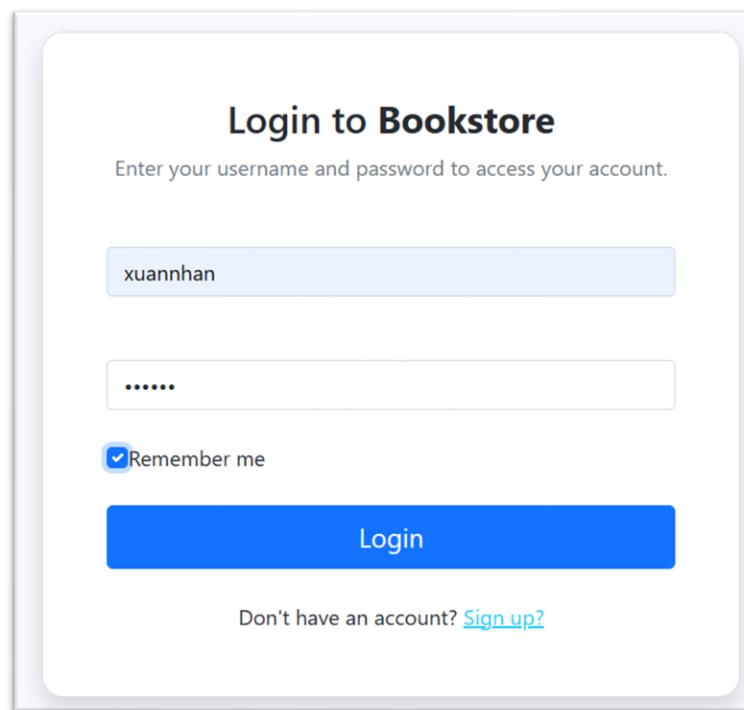
Hình 6.8. Kiểm tra valid dữ liệu khi không điền dữ liệu

Tiến hành đăng ký thử tài khoản làm mẫu:



The image shows a sign-up form titled "Sign up to Bookstore". It includes fields for email (nguyễnxuannhan407@gmail.com), username (xuannhan), password (represented by a series of dots), and a confirmation password field (represented by a blue bar). A large blue "Sign up" button is at the bottom. Below the button, a link says "Already have an account? [Login in?](#)".

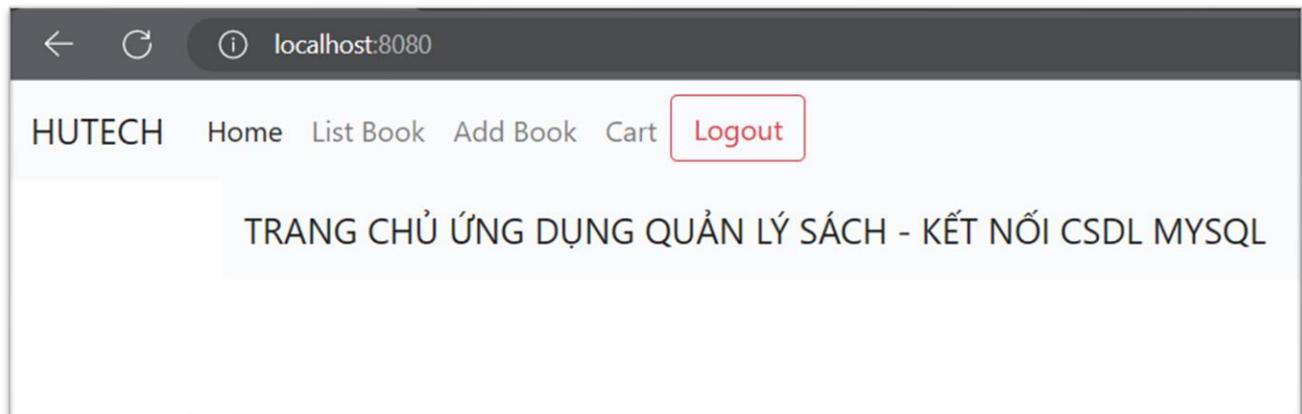
Hình 6.9. Đăng ký thử tài khoản mẫu



The image shows a login form titled "Login to Bookstore". It asks for the user's "username and password to access your account". There are fields for the username (xuannhan) and password (dots). A "Remember me" checkbox is checked. A large blue "Login" button is at the bottom. Below the button, a link says "Don't have an account? [Sign up?](#)".

Hình 6.10. Đăng nhập bằng tài khoản vừa tạo

Đăng nhập thành công, nút **Login** chuyển sang trạng thái **Logout**.



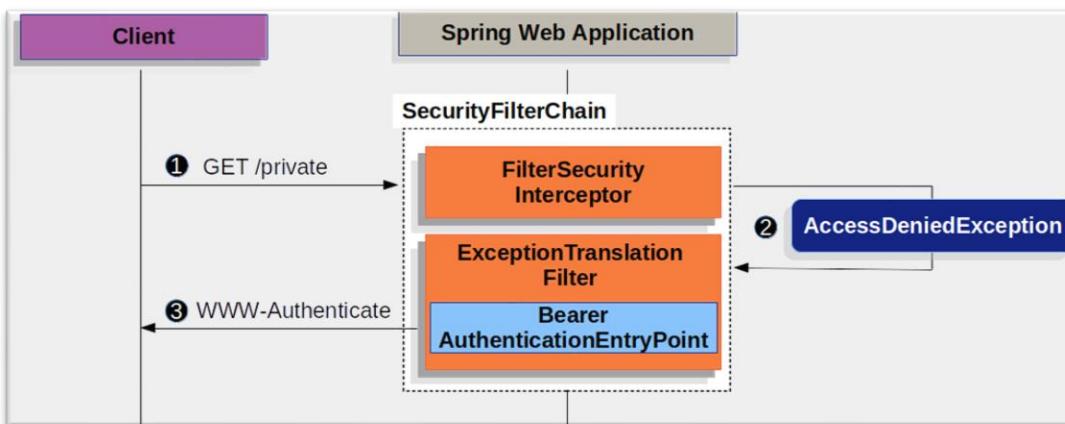
Hình 6.11. Đăng nhập thành công, xuất hiện nút Logout

		Extra options						
		← T →	↓	id	email	phone	password	username
<input type="checkbox"/>		Edit		Copy		Delete	1	nguyenxuannhan407@gmail.com
					0896401096		\$2a\$10\$6Sw8XDL1Q4OX0lce24qApe/RFZS0lnAmqpetXfTvIY1...	xuannhan
		<input type="checkbox"/> Check all	With selected:					

Hình 6.12. Kiểm tra thông tin tài khoản đã có trong Database

6.3 Xây dựng chức năng xác thực bằng OAuth2

Trong thế giới kỹ thuật ngày nay, việc xác thực người dùng đã trở thành một yếu tố quan trọng trong việc phát triển ứng dụng web. Việc xác thực giúp đảm bảo rằng chỉ những người dùng có quyền truy cập mới có thể sử dụng các tính năng và dịch vụ của ứng dụng một cách an toàn. Một trong những phương thức xác thực phổ biến được sử dụng là **OAuth2**.



Hình 6.13. Cách hoạt động của OAuth2

Trong phần này, ta sẽ xây dựng chức năng xác thực bằng OAuth2 với việc sử dụng tài khoản Google để đăng nhập vào ứng dụng. OAuth2 là một giao thức xác thực mở rộng, cho phép người dùng cấp quyền truy cập cho ứng dụng mà không cần chia sẻ mật khẩu của họ. Thay vào đó, người dùng có thể xác thực bằng tài khoản Google của mình.

Đầu tiên thêm dependency **oauth2 client** vào file **pom.xml** (Các bước thêm dependency vào project vui lòng xem lại lab 5)

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-oauth2-client</artifactId>
</dependency>
```

Tạo package với tên **constants** theo đường dẫn:

/src/main/java/fit.hutech.spring/constants

Tiếp tục tạo lớp **Enum** nằm trong thư mục **constants** với tên là **Provider** với nội dung như sau:

```
package fit.hutech.spring.constants;

import lombok.AllArgsConstructor;

@AllArgsConstructor
public enum Provider {
    LOCAL("Local"),
    GOOGLE("Google");
    public final String value;
}
```

Tại **User.java** ta bổ sung trường provider chứa phương pháp xác thực người dùng với đoạn code dưới đây:

```
@Column(name = "provider", length = 50)
private String provider;
```

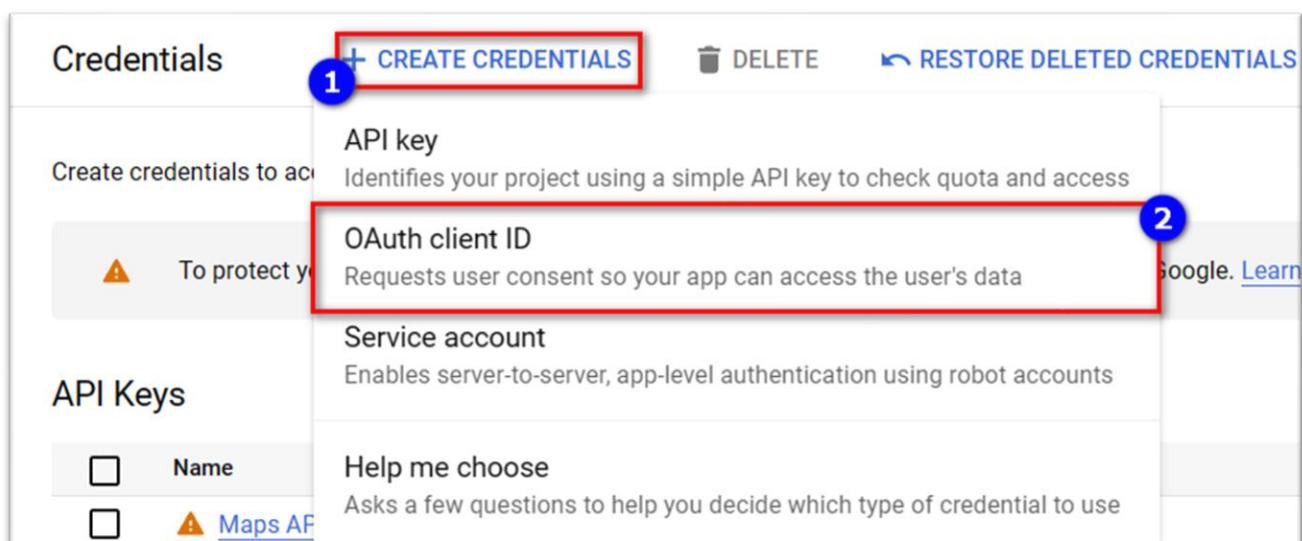
Kết quả sau khi khởi tạo:

	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/>	1	id 🌐	bigint		No	None		AUTO_INCREMENT	Change Drop More	
<input type="checkbox"/>	2	email 📩	varchar(50)	utf8mb4_0900_ai_ci	Yes	NULL			Change Drop More	
<input type="checkbox"/>	3	phone 📞	varchar(10)	utf8mb4_0900_ai_ci	Yes	NULL			Change Drop More	
<input type="checkbox"/>	4	password	varchar(250)	utf8mb4_0900_ai_ci	Yes	NULL			Change Drop More	
<input type="checkbox"/>	5	username 🔑	varchar(50)	utf8mb4_0900_ai_ci	No	None			Change Drop More	
<input type="checkbox"/>	6	provider	varchar(50)	utf8mb4_0900_ai_ci	Yes	NULL			Change Drop More	

Hình 6.14. Thiết lập Provider cho Oauth2

Cấu hình **Google API** cho việc đăng nhập sử dụng giao thức **OAuth2**. Để lấy thông tin như ClientId và Secret ta thực hiện theo các bước sau đây:

1. Truy cập vào trang <https://console.cloud.google.com/apis/credentials>.
2. Đăng nhập bằng tài khoản Google của mình.
3. Chọn **CREATE CREDENTIALS → OAuth client ID**



Hình 6.15. Khởi tạo OAuth client ID

4. Tại mục **Application Type** chọn **Web application**. Sau đó điền các thông tin như hình.

The screenshot shows the 'Create OAuth client ID' page in the Google Cloud Platform API & Services section. The left sidebar lists various configuration options like Enabled APIs & services, Library, Credentials, OAuth consent screen, and Page usage agreements. The main area is titled 'Create OAuth client ID' with a back arrow. It contains fields for 'Application type' (set to 'Web application'), 'Name' (set to 'Bookstore'), and 'Authorized JavaScript origins' (set to 'http://localhost:8080'). Below these are sections for 'Authorized redirect URIs' (set to 'http://localhost:8080/login/oauth2/code/google') and a note about it taking 5 minutes to hours for settings to take effect. At the bottom are 'CREATE' and 'CANCEL' buttons. A red box highlights the 'Application type' field, another highlights the 'Authorized JavaScript origins' field, a third highlights the 'Authorized redirect URIs' field, and a fourth highlights the 'CREATE' button.

API APIs & Services

← Create OAuth client ID

Enabled APIs & services

Library

Credentials

OAuth consent screen

Page usage agreements

1 Application type * Web application

Name * Bookstore

The name of your OAuth 2.0 client. This name is only used to identify the client in the console and will not be shown to end users.

2 URls 1 * http://localhost:8080

+ ADD URI

3 URls 1 * http://localhost:8080/login/oauth2/code/google

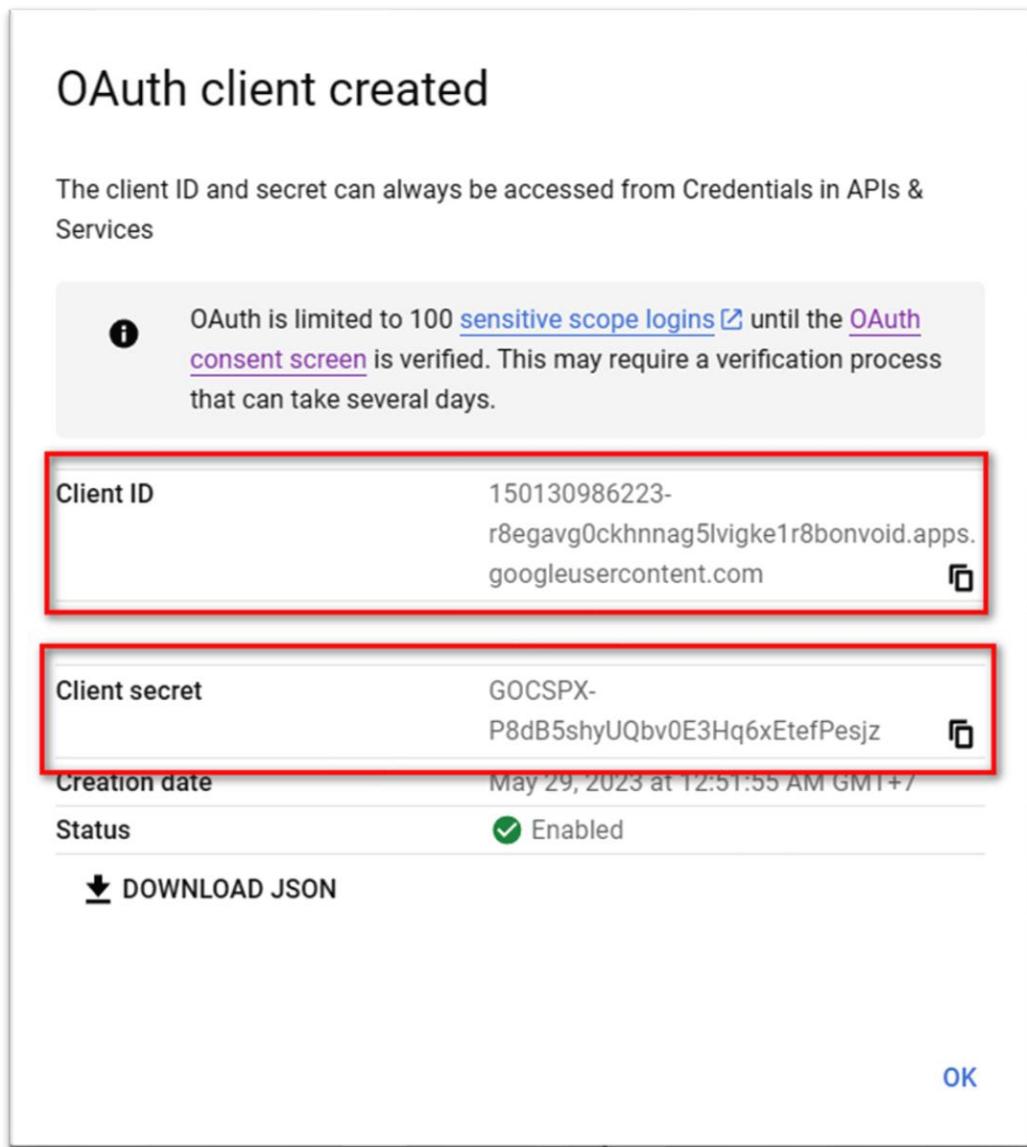
+ ADD URI

Note: It may take 5 minutes to a few hours for settings to take effect

4 CREATE CANCEL

Hình 6.16. Thiết lập các thông số cần thiết cho OAuth2

5. Copy ClientId và Secret



Hình 6.17. Lấy ClientID và Secret

6. Sau đó nội dung sau vào tệp **application.properties** được đặt tại đường dẫn **src/main/resources/application.properties**.

```
# Database connection properties
spring.datasource.url=jdbc:mysql://localhost:3306/bookstore
spring.datasource.username=root
spring.datasource.password=

# Hibernate properties
```

```
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=
org.hibernate.dialect.MySQLDialect

#Spring Security
spring.autoconfigure.exclude=org.springframework.boot.autoconfigure.security.servlet.SecurityAutoConfiguration
#Nhập thông tin Client Id
spring.security.oauth2.client.registration.google.client-id=
150130986223-
r8egavg0ckhnag5lvigke1r8bonvoid.apps.googleusercontent.com
#Nhập thông tin Secret
spring.security.oauth2.client.registration.google.client-secret=GOCSPX-
P8dB5shyUQbv0E3Hq6xEtefPesjz
```

Điều chỉnh file **login.html** lại như sau:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Login</title>
    <th:block th:replace="~{layout :: link-css}"></th:block>
    <th:block th:replace="~{layout::custom-css}"></th:block>
</head>
<body style="background: #f6f7fc;">
<th:block th:replace="~{layout :: header}"></th:block>
<div class="container py-5 h-100">
    <div class="row justify-content-center align-items-center h-100">
        <div class="col-12 col-lg-6 col-md-8 col-xl-5">
            <div class="card shadow" style="border-radius: 1rem;">
                <div class="card-body p-5 text-center">
                    <h3>Login to <strong>Bookstore</strong></h3>
                    <p class="mb-4 text-secondary">Enter your username and password to access your account.</p>
                    <a href="/oauth2/authorization/google" class="btn btn-outline-danger btn-floating m-1" role="button">
                        Login with Google
                    </a>
                    <hr class="my-4">
                    <form th:action="@{/login}" method="post">
                        <div th:if="${param.error}" class="alert alert-
```

```
danger">
    Invalid username and password.
</div>
<div th:if="${param.logout}" class="alert alert-success">
    You have been logged out.
</div>
<div class="form-group mb-4">
    <label for="username"></label>
    <input type="text" class="form-control" required id="username" name="username" placeholder="Username">
</div>
<div class="form-group mb-4">
    <label for="password"></label>
    <input type="password" class="form-control" required id="password" name="password" placeholder="Password">
</div>
<div class="form-check d-flex justify-content-start mb-4">
    <input type="checkbox" class="form-check-input" name="remember-me" id="remember-me">
    <label class="form-check-label" for="remember-me"> Remember me</label>
</div>
<div class="d-grid gap-2 form-action">
    <button type="submit" class="btn btn-primary btn-lg btn-block">Login</button>
    <p class="mt-3 mb-0">Don't have an account?</p>
<a class="text-info text-center" th:href="${'/register'}">Sign up?</a>
</p>
</div>
</form>
</div>
</div>
</div>
</div>
<th:block th:replace="~{layout :: footer}"></th:block>
</body>
</html>
```

Tại file **SecurityConfig.java** đặt tại src/main/java/fit.hutech.spring/Utils dán đoạn code bên dưới vào:

```
package fit.hutech.spring.utils;

import fit.hutech.spring.services.OAuthService;
import fit.hutech.spring.services.UserService;
import jakarta.validation.constraints.NotNull;
import lombok.RequiredArgsConstructor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import
org.springframework.security.config.annotation.method.configuration.Enable
MethodSecurity;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.Enable
WebSecurity;
import
org.springframework.security.core.userdetails.UserDetailsService;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import
org.springframework.security.oauth2.core.oidc.user.DefaultOidcUser;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
@EnableWebSecurity
@EnableMethodSecurity(securedEnabled = true, jsr250Enabled = true)
@RequiredArgsConstructor
public class SecurityConfig {

    private final OAuthService oAuthService;

    private final UserService userService;

    @Bean
    public UserDetailsService userDetailsService() {
        return new UserService();
    }
}
```

```
}

@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}

@Bean
public DaoAuthenticationProvider authenticationProvider() {
    var auth = new DaoAuthenticationProvider();
    auth.setUserDetailsService(userDetailsService());
    auth.setPasswordEncoder(passwordEncoder());
    return auth;
}

@Bean
public SecurityFilterChain securityFilterChain(@NotNull
HttpSecurity http) throws Exception {
    return http
        .authorizeHttpRequests(auth -> auth
            .requestMatchers("/css/**", "/js/**", "/",
"/oauth/**", "/register", "/error")
            .permitAll()
            .requestMatchers("/books/edit/**",
"/books/add", "/books/delete")
            .authenticated()
            .requestMatchers("/books", "/cart", "/cart/**")
            .authenticated()
            .requestMatchers("/api/**")
            .authenticated()
            .anyRequest().authenticated())
        .logout(logout -> logout
            .logoutUrl("/logout")
            .logoutSuccessUrl("/login")
            .deleteCookies("JSESSIONID")
            .invalidateHttpSession(true)
            .clearAuthentication(true)
            .permitAll())
        .formLogin(formLogin -> formLogin
            .LoginPage("/login")
            .loginProcessingUrl("/login")
            .defaultSuccessUrl("/")
            .failureUrl("/login?error")
            .permitAll())
        .oauth2Login(
            oauth2Login -> oauth2Login
```

```
        .loginPage("/login")
        .failureUrl("/login?error")
        .userInfoEndpoint(userInfoEndpoint ->
userInfoEndpoint
            .userService(oAuthService)
        )
        .successHandler(
            (request, response,
authentication) -> {
                var oidcUser =
(DefaultOidcUser) authentication.getPrincipal();

userService.saveOAuthUser(oidcUser.getEmail(), oidcUser.getName());
                    response.sendRedirect("/");
                }
            )
            .permitAll()
        ).rememberMe(rememberMe -> rememberMe
            .key("hutech")
            .rememberMeCookieName("hutech")
            .tokenValiditySeconds(24 * 60 * 60)
            .userDetailsService(userDetailsService())
        ).exceptionHandling(exceptionHandling ->
exceptionHandling
            .accessDeniedPage("/403")
        ).sessionManagement(sessionManagement ->
sessionManagement
            .maximumSessions(1)
            .expiredUrl("/login")
        ).httpBasic(httpBasic -> httpBasic
            .realmName("hutech")
        ).build();
    }
}
```

Tạo file **OauthService.java** nằm tại thư mục services, sau đó thêm nội dung đoạn code sau đây:

```
package fit.hutech.spring.services;

import
org.springframework.security.oauth2.client.userinfo.DefaultOAuth2UserService;
import
org.springframework.security.oauth2.client.userinfo.OAuth2UserRequest;
import
org.springframework.security.oauth2.core.OAuth2AuthenticationException;
import org.springframework.security.oauth2.core.user.OAuth2User;
import org.springframework.stereotype.Service;

@Service
public class OAuthService extends DefaultOAuth2UserService {
    @Override
    public OAuth2User loadUser(OAuth2UserRequest userRequest) throws
OAuth2AuthenticationException {
        return super.loadUser(userRequest);
    }
}
```

Phương thức **loadUser()** được ghi đè từ lớp cơ sở **DefaultOAuth2UserService**. Phương thức này nhận đối tượng **OAuth2UserRequest** làm tham số, đại diện cho yêu cầu người dùng đăng nhập thông qua **OAuth2**. Phương thức này được sử dụng để tải thông tin người dùng từ nhà cung cấp dịch vụ **OAuth2**.

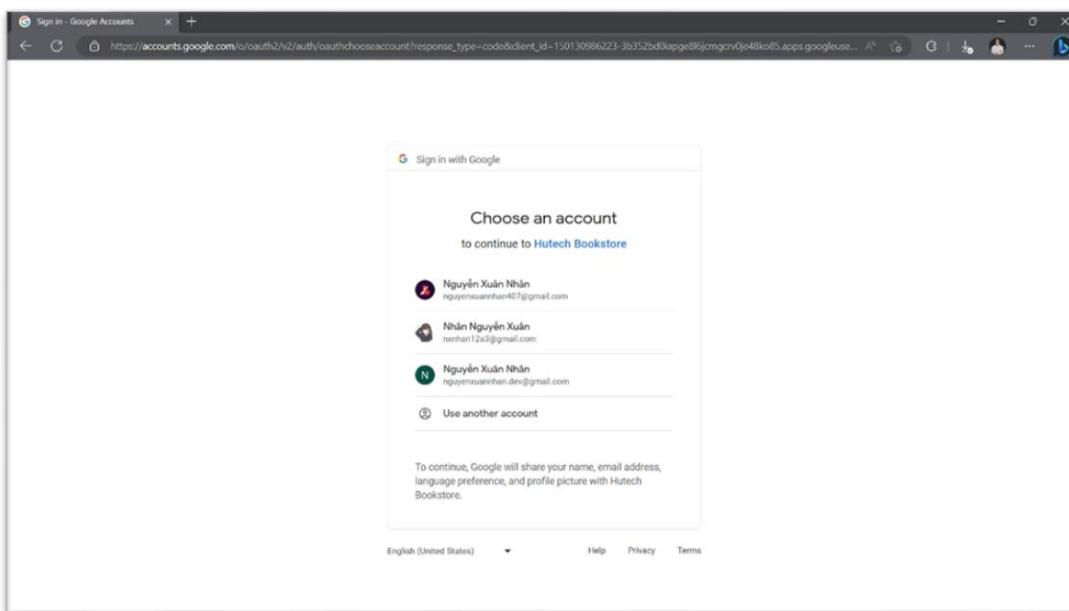
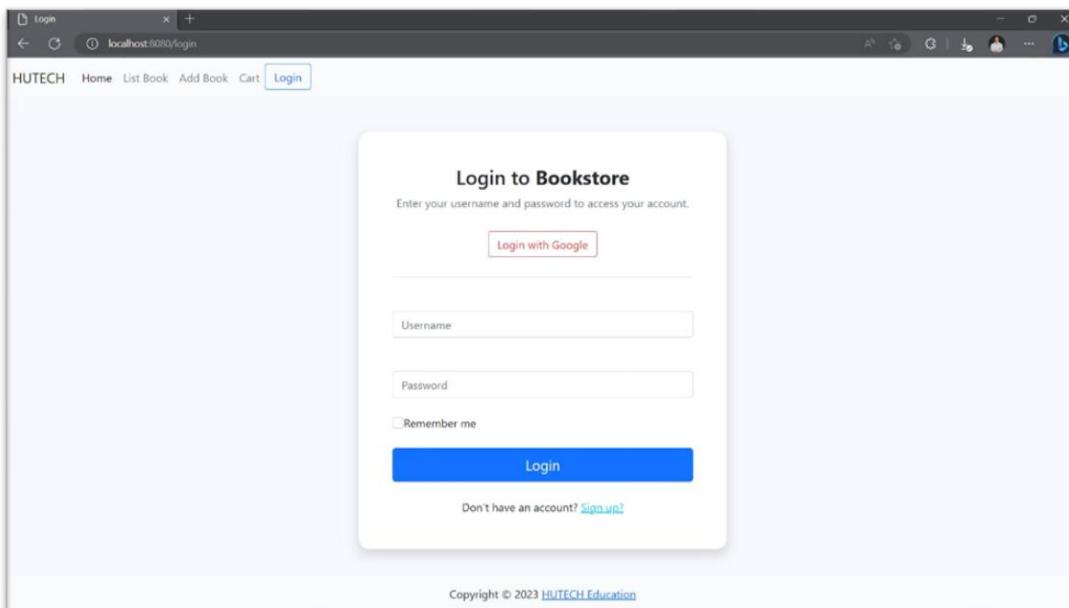
Trong trường hợp này, phương thức **loadUser()** chỉ đơn giản gọi phương thức **loadUser()** của lớp cơ sở **DefaultOAuth2UserService** bằng cách sử dụng từ khóa **super**. Điều này cho phép phương thức cơ sở xử lý việc tải thông tin người dùng từ nhà cung cấp dịch vụ **OAuth2** một cách mặc định.

Tại **UserService.java** nằm tại thư mục services, ta bổ sung thêm phương thức xử lý đăng nhập bằng OAuth2:

```
public void saveOauthUser(String email, @NotNull String username) {
    if(userRepository.findByUsername(username).isPresent())
        return;
    var user = new User();
    user.setUsername(username);
```

```
        user.setEmail(email);
        user.setPassword(new BCryptPasswordEncoder().encode(username));
        user.setProvider(Provider.GOOGLE.value);
        user.getRoles().add(roleRepository.findRoleById(Role.USER.value));
        userRepository.save(user);
    }
```

Build ứng dụng và kiểm tra kết quả:



Hình 6.18. Xác thực bằng OAuth2

TÓM TẮT

Trong Spring Security, xác thực (authentication) đóng vai trò quan trọng trong bảo mật ứng dụng. Quá trình xác thực được thực hiện để xác định xem một người dùng có quyền truy cập vào hệ thống hay không dựa trên thông tin xác thực mà người dùng cung cấp.

Trong ứng dụng Spring Boot, để xây dựng quá trình xác thực, bạn có thể sử dụng Spring Security. Spring Security cung cấp các cơ chế mạnh mẽ để xác thực người dùng, bao gồm xác thực dựa trên tên người dùng và mật khẩu, xác thực dựa trên mã thông báo (token-based authentication) và xác thực dựa trên các cơ chế bên ngoài như OAuth.

Để cấu hình xác thực trong Spring Security, bạn cần tạo một lớp cấu hình (configuration class) và cấu hình các bộ xác thực (authentication providers) và bộ phân quyền (authorization) cho ứng dụng. Bạn có thể sử dụng các thành phần như AuthenticationManagerBuilder để cấu hình xác thực dựa trên tên người dùng và mật khẩu, và sử dụng các annotation như @EnableWebSecurity để kích hoạt bảo mật trên ứng dụng.

CÂU HỎI ÔN TẬP

1. Làm thế nào để cấu hình xác thực dựa trên tên người dùng và mật khẩu trong Spring Security?
2. Làm thế nào để tích hợp Spring Security với OAuth để xác thực người dùng từ các dịch vụ bên ngoài như Facebook hoặc Google?
3. Spring Security cung cấp các tính năng bảo mật khác nhau như xử lý tấn công CSRF, XSS hay Session Fixation không? Làm thế nào để kích hoạt và cấu hình chúng?
4. Làm thế nào để tạo trang đăng nhập tùy chỉnh trong Spring Security?
5. Làm thế nào để cấu hình xác thực dựa trên tên người dùng và mật khẩu trong Spring Security?

BÀI 7 XÂY DỰNG CHỨC NĂNG PHÂN QUYỀN NGƯỜI DÙNG

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

Để tạo class **Role.java** trong thư mục **Entities** theo đường dẫn **src/main/java/fit.hutech.spring/entities**, sinh viên có thể làm như sau.

```
package fit.hutech.spring.entities;

import jakarta.persistence.*;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.Size;
import lombok.*;
import org.hibernate.Hibernate;
import org.springframework.security.core.GrantedAuthority;

import java.util.HashSet;
import java.util.Objects;
import java.util.Set;

@Getter
@Setter
@ToString
@RequiredArgsConstructor
@AllArgsConstructor
@Builder
@Entity
@Table(name = "role")
public class Role implements GrantedAuthority {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotBlank(message = "Name is required")
    @Column(name = "name", length = 50, nullable = false)
    @Size(max = 50, message = "Name must be less than 50 characters")
    private String name;

    @Size(max = 250, message = "Description must be less than 250
characters")
    @Column(name = "description", length = 250)
```

```

private String description;

@ManyToMany(mappedBy = "roles", cascade = CascadeType.ALL)
@ToString.Exclude
private Set<User> users = new HashSet<>();

@Override
public String getAuthority() {
    return name;
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || Hibernate.getClass(this) !=
Hibernate.getClass(o)) return false;
    Role role = (Role) o;
    return getId() != null && Objects.equals(getId(),
role.getId());
}

@Override
public int hashCode() {
    return getClass().hashCode();
}
}

```

Để thực hiện quan hệ n-n (many-to-many) giữa đối tượng **User** và đối tượng **Role** trong cơ sở dữ liệu, bạn có thể chỉnh sửa nội dung của tệp **User.java** như sau:

```

@ManyToMany(fetch=FetchType.EAGER)
@JoinTable(name = "user_role",
    joinColumns = @JoinColumn(name = "user_id"),
    inverseJoinColumns = @JoinColumn(name = "role_id"))
private Set<Role> roles = new HashSet<>();

```

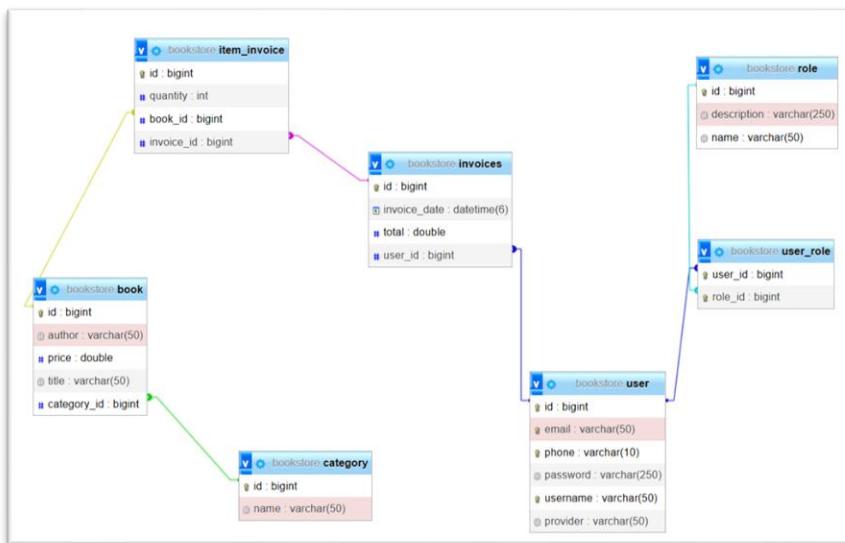
Thuộc tính **roles** trong đối tượng **User** được đánh dấu với **@ManyToMany** để thể hiện mối quan hệ nhiều-nhiều với đối tượng **Role**.

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

Hãy kiểm tra lại các lớp (class) và giao diện (interface) đã được thêm vào. Sau đó, tiến hành xây dựng (build) và khởi chạy lại dự án (project) bằng cách nhấn tổ hợp phím **CTRL + F9**.

Tiếp theo, hãy mở trình duyệt và truy cập vào **phpMyAdmin** bằng đường dẫn <http://localhost/phpmyadmin/>.

Tại đây, hãy kiểm tra cơ sở dữ liệu (database) để xem liệu có xuất hiện thêm bảng (table) "Role" mà chúng ta đã thêm vào hay không.



Hình 7.1. Database sẽ xuất table Role và user_role

Hãy thêm dữ liệu cho role **ADMIN** và **USER** vào bảng **Role**.

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

Hình 7.2. Thêm dữ liệu vào table role

Tạo 2 tài khoản như bên dưới để thử, ta sẽ gán tài khoản **hutech1** thành quyền **ADMIN**, tài khoản **hutech2** là quyền **USER**.

	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	id	username	email	phone	password
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	1	xuannhan	nguyenxuannhan407@gmail.com	0896401096	\$2a\$10\$6Sw8XDL1Q4OX0lce24qApe/RFZS0InAmqpetXfTvIY1...
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	2	hutech1	h1@hutech.edu.vn	0935120791	\$2a\$10\$x8uHES8RlLzBVwGO/RmEt xLXgoCQq9gmMxxJ1Z9Dp...
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	3	hutech2	h2@hutech.edu.vn	0948160792	\$2a\$10\$glw77E1Vc5jAgddy6kcGi.x3T3D6b89aNfK4y/pMBM...

Hình 7.3. Xem id của user

The screenshot shows the phpMyAdmin interface for the 'bookstore' database. The left sidebar shows the schema structure with 'user_role' selected. The main area displays the 'user_role' table. The 'Insert' tab is active, and the 'user_id' and 'role_id' fields are highlighted with a red box. A blue circle labeled '2' is positioned over the 'Insert' button, and another blue circle labeled '3' is positioned over the 'Go' button.

Hình 7.4. Điền id của user và role vào bảng user_role

The screenshot shows the phpMyAdmin interface for the 'bookstore' database. The left sidebar shows the schema structure with 'user_role' selected. The main area displays the 'user_role' table. The table has two rows. The first row has 'user_id' value 2 and 'role_id' value 1. The second row has 'user_id' value 3 and 'role_id' value 2. Both rows are highlighted with a red box.

Hình 7.5. Dữ liệu bảng user_role vừa set quyền

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

Tại thư mục **constants** nằm tại **/src/main/java/fit.hutech.spring/constants**. Ta tạo lớp **Enum** nằm trong thư mục **constants** với tên là **Role** với nội dung như sau:

```
package fit.hutech.spring.constants;

import lombok.AllArgsConstructor;
import lombok.Getter;

@AllArgsConstructor
public enum Role {
    ADMIN(1),
    USER(2);
    public final long value;
}
```

Khởi tạo file **IRoleRepository.java** đặt tại thư mục repositories theo đường dẫn **src/main/java/fit.hutech.spring/repositories** và bổ sung thêm các phương thức sau:

Phương thức **findRoleById** trong interface **IRoleRepository** là một phương thức dùng để tìm kiếm một đối tượng **Role** dựa trên giá trị của thuộc tính **id** của nó. Phương thức này nhận vào một tham số kiểu **Long** là **id** và trả về một đối tượng **Role** có **id** tương ứng.

Khi gọi phương thức này và truyền vào một giá trị **id**, nó sẽ truy vấn cơ sở dữ liệu và tìm kiếm đối tượng **Role** có **id** tương ứng. Nếu tìm thấy, phương thức sẽ trả về đối tượng **Role** đó, nếu không tìm thấy, phương thức sẽ trả về **null** hoặc giá trị tương đương để chỉ ra rằng không có đối tượng nào được tìm thấy.

```
package fit.hutech.spring.repositories;

import fit.hutech.spring.entities.Role;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface IRoleRepository extends JpaRepository<Role, Long>{
    Role findRoleById(Long id);
}
```

Tại file **UserService.java** đặt **src/main/java/fit.hutech.spring/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**.

```
package fit.hutech.spring.services;

import fit.hutech.spring.constants.Role;
import fit.hutech.spring.entities.User;
import fit.hutech.spring.repositories.IRoleRepository;
import fit.hutech.spring.repositories.IUserRepository;
import jakarta.validation.constraints.NotNull;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import
org.springframework.security.core.userdetails.UsernameNotFoundException
;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Isolation;
import org.springframework.transaction.annotation.Transactional;

@Service
@Slf4j
public class UserService implements UserDetailsService {
    @Autowired
    private IUserRepository userRepository;

    @Autowired
    private IRoleRepository roleRepository;

    @Transactional(isolation = Isolation.SERIALIZABLE,
                  rollbackFor = {Exception.class, Throwable.class})
    public void save(@NotNull User user) {
        user.setPassword(new BCryptPasswordEncoder()
                        .encode(user.getPassword()));
        userRepository.save(user);
    }

    @Transactional(isolation = Isolation.SERIALIZABLE,
                  rollbackFor = {Exception.class, Throwable.class})
    public void setDefaultRole(String username){
        userRepository.findByUsername(username)
```

```

        .getRoles()
        .add(roleRepository
            .findRoleById(Role.USER.value));
    }

@Override
public UserDetails loadUserByUsername(String username)
    throws UsernameNotFoundException {
    var user = userRepository.findByUsername(username);
    return org.springframework.security.core.userdetails.User
        .withUsername(user.getUsername())
        .password(user.getPassword())
        .authorities(user.getAuthorities())
        .accountExpired(false)
        .accountLocked(false)
        .credentialsExpired(false)
        .disabled(false)
        .build();
}
}

```

Tiến hành thay đổi phương thức **Register** tại file **UserController.java** như sau:

```

@PostMapping("/register")
public String register(@Valid @ModelAttribute("user") User user,
                      @NotNull BindingResult bindingResult,
                      Model model) {
    if (bindingResult.hasErrors()) {
        var errors = bindingResult.getAllErrors()
            .stream()
            .map(DefaultMessageSourceResolvable::getDefaultMessage)
            .toArray(String[]::new);
        model.addAttribute("errors", errors);
        return "user/register";
    }
    userService.save(user);
    userService.setDefaultRole(user.getUsername());
    return "redirect:/login";
}

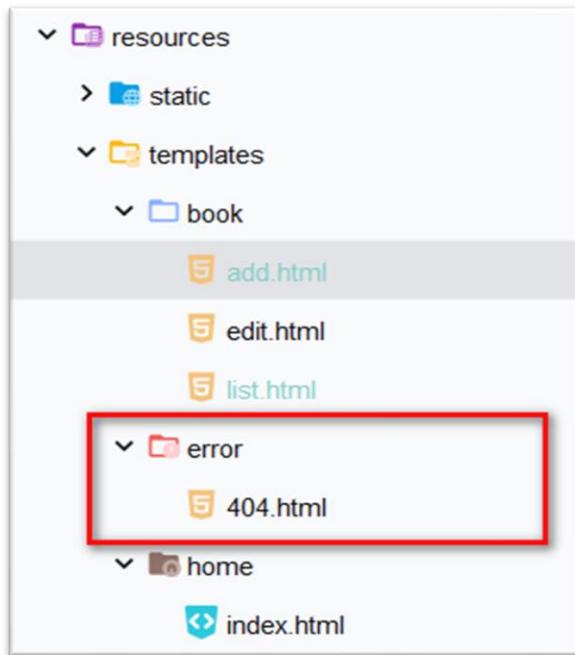
```

7.2 Tạo thông báo lỗi khi chuyển trang

➤ Thiết kế giao diện 404.html

Tạo 1 thư mục mới tên **errors** trong thư mục templates Tạo file **404.html** tại thư đường dẫn: **src/main/java/fit.hutech.spring/resources/templates/errors**

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 7.6. Thư mục và file 404.html được tạo

Dưới đây là nội dung của tệp **404.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Not Found</title>
  <th:block th:replace="~{layout::link-css}"></th:block>
  <th:block th:replace="~{layout::custom-css}"></th:block>
</head>
<body>
  <th:block th:replace="~{layout::header}"></th:block>
  <div class="d-flex align-items-center justify-content-center vh-25">
    <div class="text-center">
      <h1 class="display-1 fw-bold">404</h1>
      <p class="fs-3"> <span class="text-danger">Opps!</span> Page not
```

```

found.</p>
<p class="lead">
    The page you are looking for might have been removed.
</p>
<a href="/" class="btn btn-primary">Back to homepage</a>
</div>
</div>
<th:block th:replace=~{layout::footer}"></th:block>
</body>
</html>

```

Tương tự như trên ta tiến hành tạo view cho các mã lỗi **500** và **403** được đặt tại đường dẫn **errors**. Các mã lỗi khác sinh viên có thể tạo các file view tương ứng và tùy chỉnh giao diện cho từng mã lỗi đó.

403.html

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Access Denied</title>
    <th:block th:replace=~{layout::link-css}"></th:block>
    <th:block th:replace=~{layout::custom-css}"></th:block>
</head>
<body>
<th:block th:replace=~{layout::header}"></th:block>
<div class="d-flex align-items-center justify-content-center vh-25">
    <div class="text-center">
        <h1 class="display-1 fw-bold">403</h1>
        <p class="fs-3"> <span class="text-danger">Opps!</span> Access
Denied.</p>
        <p class="lead">
            You don't have permission to access this page.
        </p>
        <a href="/" class="btn btn-primary">Back to homepage</a>
    </div>
</div>
<th:block th:replace=~{layout::footer}"></th:block>
</body>
</html>

```

500.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Internal Server Error</title>
    <th:block th:replace="~{layout::link-css}"></th:block>
    <th:block th:replace="~{layout::custom-css}"></th:block>
</head>
<body>
<th:block th:replace="~{layout::header}"></th:block>
<div class="d-flex align-items-center justify-content-center vh-25">
    <div class="text-center">
        <h1 class="display-1 fw-bold">500</h1>
        <p class="fs-3"> <span class="text-danger">Opps!</span>
Internal Server Error.</p>
        <p class="lead">
            The server encountered an internal error and was unable to
            complete your request.
        </p>
        <small class="text-muted">Please check the server logs for more
            details.</small><br/>
        <a href="/" class="btn btn-primary">Back to homepage</a>
    </div>
</div>
<th:block th:replace="~{layout::footer}"></th:block>
</body>
</html>
```

Bằng cách tạo các view như vậy, bạn có thể cung cấp thông báo lỗi đẹp mắt và dễ hiểu cho người dùng khi gặp các mã lỗi khác nhau trong ứng dụng của bạn.

➤ 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.

```
# Database connection properties
spring.datasource.url=jdbc:mysql://localhost:3306/bookstore
spring.datasource.username=root
spring.datasource.password=

# Hibernate properties
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=
org.hibernate.dialect.MySQLDialect

#Spring Security
spring.autoconfigure.exclude=org.springframework.boot.autoconfigure.security.servlet.SecurityAutoConfiguration
spring.security.oauth2.client.registration.google.client-id=
150130986223-
r8egavg0ckhnag5lvigke1r8bonvoid.apps.googleusercontent.com
spring.security.oauth2.client.registration.google.client-secret=GOCSPX-
P8dB5shyUQbv0E3Hq6xEtefPesjz

server.error.path=/error
```

Tạo file **ExceptionController.java** đặt tại thư mục **controllers**.

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.

```
package fit.hutech.spring.controllers;

import jakarta.servlet.RequestDispatcher;
import jakarta.servlet.http.HttpServletRequest;
import org.springframework.boot.web.servlet.error.ErrorController;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;

import java.util.Optional;

@Controller
@RequestMapping("/error")
public class ExceptionController implements ErrorController {

    @GetMapping
    public String handleError(@NotNull HttpServletRequest request) {
```

```

        return Optional
            .ofNullable(request.getAttribute(
                RequestDispatcher.ERROR_STATUS_CODE))
            .map(status -> Integer.parseInt(status.toString()))
            .filter(status -> status == 404
                || status == 500
                || status == 403)
            .map(status -> "error/" + status)
            .orElse(null);
    }
}

```

7.3 Gán quyền cho User và Role

Tại tệp **User.java** ta thay đổi các phương thức **getAuthorities()** như sau:

```

@Override
public Collection<? extends GrantedAuthority> getAuthorities() {
    Set<Role> userRoles = this.getRoles();
    return userRoles.stream()
        .map(role -> new SimpleGrantedAuthority(role.getName()))
        .toList();
}

```

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

Resource	Quyền
/books/edit", "/books/delete"	ADMIN
/books", "/books/add", "/api/**", "/cart", "/cart/**"	ADMIN, USER

Hình 7.7. Phân quyền theo admin và user

```

package fit.hutech.spring.utils;

import fit.hutech.spring.services.OAuthService;
import fit.hutech.spring.services.UserService;
import jakarta.validation.constraints.NotNull;
import lombok.RequiredArgsConstructor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import

```

```
org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import
org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.core.userdetails.UserDetailsService;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import
org.springframework.security.oauth2.core.oidc.user.DefaultOidcUser;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
@EnableWebSecurity
@EnableMethodSecurity(securedEnabled = true, jsr250Enabled = true)
@RequiredArgsConstructor
public class SecurityConfig {

    private final OAuthService oAuthService;

    private final UserService userService;

    @Bean
    public UserDetailsService userDetailsService() {
        return new UserService();
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public DaoAuthenticationProvider authenticationProvider() {
        var auth = new DaoAuthenticationProvider();
        auth.setUserDetailsService(userDetailsService());
        auth.setPasswordEncoder(passwordEncoder());
        return auth;
    }
}
```

```
}

@Bean
public SecurityFilterChain securityFilterChain(@NotNull
HttpSecurity http) throws Exception {
    return http
        .authorizeHttpRequests(auth -> auth
            .requestMatchers("/css/**", "/js/**", "/",
"/oauth/**", "/register", "/error")
            .permitAll()
            .requestMatchers("/books/edit/**",
"/books/add", "/books/delete")
            .hasAnyAuthority("ADMIN")
            .requestMatchers("/books", "/cart", "/cart/**")
            .hasAnyAuthority("ADMIN", "USER")
            .requestMatchers("/api/**")
            .hasAnyAuthority("ADMIN", "USER")
            .anyRequest().authenticated())
        .logout(logout -> logout
            .logoutUrl("/logout")
            .logoutSuccessUrl("/login")
            .deleteCookies("JSESSIONID")
            .invalidateHttpSession(true)
            .clearAuthentication(true)
            .permitAll())
        .formLogin(formLogin -> formLogin
            .LoginPage("/login")
            .loginProcessingUrl("/login")
            .defaultSuccessUrl("/")
            .failureUrl("/login?error")
            .permitAll())
        .oauth2Login(
            oauth2Login -> oauth2Login
            .LoginPage("/login")
            .failureUrl("/login?error")
            .userInfoEndpoint(userInfoEndpoint ->
userInfoEndpoint
                .userService(oAuthService)
            )
            .successHandler(
                (request, response,
authentication) -> {
                    var oidcUser =
(DefaultOidcUser) authentication.getPrincipal();
                    userService.saveOAuthUser(oidcUser.getEmail(),
oidcUser.getName());
                }
            )
        )
    );
}
```

```

        response.sendRedirect("/");
    }

    .permitAll()
).rememberMe(rememberMe -> rememberMe
    .key("hutech")
    .rememberMeCookieName("hutech")
    .tokenValiditySeconds(24 * 60 * 60)
    .userDetailsService(userDetailsService())
).exceptionHandling(exceptionHandling ->
exceptionHandling
    .accessDeniedPage("/403")
).sessionManagement(sessionManagement ->
sessionManagement
    .maximumSessions(1)
    .expiredUrl("/login")
).httpBasic(httpBasic -> httpBasic
    .realmName("hutech")
).build();
}
}

```

```

@Bean
public SecurityFilterChain securityFilterChain(@NotNull HttpSecurity http) throws Exception {
    return http
        .authorizeHttpRequests(auth -> auth
            .requestMatchers(ignoredPathMatchers("/css/**", "/js/**", "/", "/oauth/**", "/register", "/error"))
            .permitAll()
            .requestMatchers(ignoredPathMatchers("/books/edit/**", "/books/add", "/books/delete"))
            .hasAnyAuthority(authorities: "ADMIN")
            .requestMatchers(ignoredPathMatchers("/books", "/cart", "/cart/**"))
            .hasAnyAuthority(authorities: "ADMIN", "USER")
            .requestMatchers(ignoredPathMatchers("/api/**"))
            .hasAnyAuthority(authorities: "ADMIN", "USER")
            .anyRequest().authenticated()
        );
}

```

Hình 7.8. Tiến hành phân quyền truy cập resource tại SecurityFilterChain

7.4 ĐIỀU CHỈNH PHÂN QUYỀN TẠI GIAO DIỆN list.html và layout.html

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

src/main/java/fit.hutech.spring/resources/templates/book

```
<!DOCTYPE html>
<html
    xmlns:th="http://www.thymeleaf.org"
    xmlns:sec="http://www.thymeleaf.org/extras/spring-security"
    lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>My Book List</title>
    <th:block th:replace="~{layout::link-css}"></th:block>
    <th:block th:replace="~{layout::custom-css}"></th:block>
</head>
<body>
<th:block th:replace="~{layout::header}"></th:block>
<div class="container">
    <div class="row">
        <div class="col-md-12">
            <nav class="navbar navbar-light">
                <div class="container-fluid">
                    <h1>My Book List</h1>
                    <form class="d-flex" th:action="@{/books/search}"
method="get">
                        <input class="form-control me-2" type="search"
placeholder="Search" aria-label="Search" name="keyword">
                        <button class="btn btn-outline-success"
type="submit">Search</button>
                    </form>
                </div>
            </nav>
        </div>
    </div>
    <table class="table">
        <thead>
            <tr>
                <th><a th:href="@{/books(pageNo=${currentPage}, sortBy=
'id')}">Id</a></th>
                <th><a th:href="@{/books(pageNo=${currentPage}, sortBy=
'title')}">Title</a></th>
                <th><a th:href="@{/books(pageNo=${currentPage}, sortBy=
'author')}">Author</a></th>
                <th><a th:href="@{/books(pageNo=${currentPage}, sortBy=
'price')}">Price</a></th>
                <th><a th:href="@{/books(pageNo=${currentPage}, sortBy=
'category')}">Category</a></th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>1</td>
                <td>The Great Gatsby</td>
                <td>F. Scott Fitzgerald</td>
                <td>$12.99</td>
                <td>Fiction</td>
            </tr>
            <tr>
                <td>2</td>
                <td>1984</td>
                <td>George Orwell</td>
                <td>$14.99</td>
                <td>Science Fiction</td>
            </tr>
            <tr>
                <td>3</td>
                <td>To Kill a Mockingbird</td>
                <td>Harper Lee</td>
                <td>$11.99</td>
                <td>Literature</td>
            </tr>
            <tr>
                <td>4</td>
                <td>The Catcher in the Rye</td>
                <td>J. D. Salinger</td>
                <td>$13.99</td>
                <td>Literature</td>
            </tr>
            <tr>
                <td>5</td>
                <td>The Hobbit</td>
                <td>J. R. R. Tolkien</td>
                <td>$15.99</td>
                <td>Fantasy</td>
            </tr>
        </tbody>
    </table>
</div>
</body>
```

```

<th>Action</th>
</tr>
</thead>
<tbody>
<tr th:each="book : ${books}">
    <td th:text="${book.getId()}"></td>
    <td th:text="${book.getTitle()}"></td>
    <td th:text="${book.getAuthor()}"></td>
    <td th:text="${book.getPrice()}"></td>
    <td th:text="${book.getCategory().getName()}"></td>
    <td colspan="2">
        <a class="btn btn-primary"
            sec:authorize="hasAnyAuthority('ADMIN')"
            th:href="@{/books/edit/{id}(id=${book.getId()})}">Edit</a>
        <a class="btn btn-danger"
            sec:authorize="hasAnyAuthority('ADMIN')"
            th:href="@{/books/delete/{id}(id=${book.getId()})}"
            onclick="return confirm('Are you sure you want to delete
this book?')">Delete</a>
        <form th:action="@{/books/add-to-cart}" method="post"
class="d-inline">
            <input type="hidden" name="id" th:value="${book.getId()}">
            <input type="hidden" name="name"
th:value="${book.getTitle()}">
            <input type="hidden" name="price"
th:value="${book.getPrice()}">
            <button type="submit" class="btn btn-success"
                onclick="return confirm('Are you sure you want to
add this book to cart?')">
                Add to cart</button>
        </form>
    </td>
</tr>
</tbody>
</table>
</div>
<nav aria-label="Page navigation example">
    <ul class="pagination justify-content-center pagination-sm"
th:each="i : ${#numbers.sequence(0, totalPages)}">
        <li class="page-item" th:classappend="${currentPage == i} ?
'active'">
            <a class="page-link" th:href="@{/books(pageNo=${i})}"
th:text="${i}"></a>
        </li>
    </ul>
</nav>

```

```
<th:block th:replace="~{layout::footer}"></th:block>
</body>
</html>
```

Tương tự ta cũng tiến hành phân quyền tại file **layout.html**

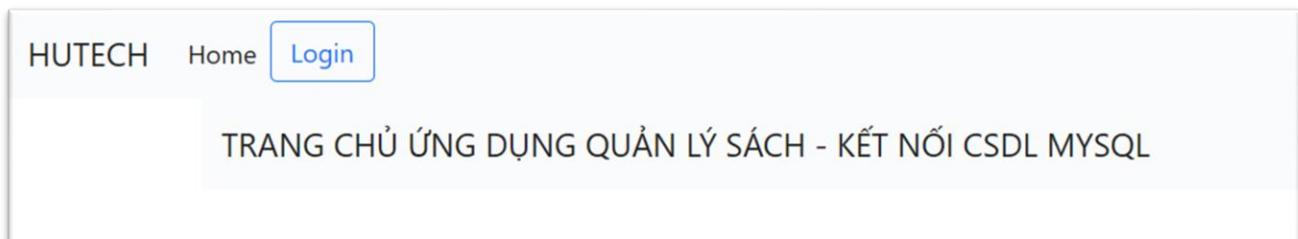
```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/extras/spring-security"
      lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>My App</title>
    <link th:fragment="link-css" rel="stylesheet"
th:href="@{/css/bootstrap.min.css}">
    <link th:fragment="custom-css" rel="stylesheet"
th:href="@{/css/style.css}">
</head>
<body class="d-flex flex-column h-100">
<header th:fragment="header">
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
        <div class="container-fluid">
            <a class="navbar-brand" href="/">HUTECH</a>
            <button class="navbar-toggler" type="button" data-bs-
toggle="collapse"
                   data-bs-target="#navbarSupportedContent"
                   aria-controls="navbarSupportedContent"
                   aria-expanded="false" aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse"
id="navbarSupportedContent">
                <ul class="navbar-nav me-auto mb-2 mb-lg-0">
                    <li class="nav-item">
                        <a class="nav-link active" aria-current="page"
href="/">Home</a>
                    </li>
                    <li class="nav-item" sec:authorize="hasAnyAuthority('ADMIN',
'USER')">
                        <a class="nav-link" href="/books">
                            List Book
                        </a>
                    </li>
                <ul>

```

```
<li class="nav-item"
sec:authorize="hasAnyAuthority('ADMIN')"
    <a class="nav-link" href="/books/add">
        Add Book
    </a>
</li>
<li class="nav-item" sec:authorize="hasAnyAuthority('ADMIN',
'USER')">
    <a class="nav-link" href="/cart">
        Cart
    </a>
</li>
<li sec:authorize="isAuthenticated()">
    <form th:action="@{/logout}" method="post">
        <button class="btn btn-outline-danger"
type="submit">Logout</button>
    </form>
</li>
<li sec:authorize="!isAuthenticated()">
    <a class="btn btn-outline-primary" href="/login">
        Login
    </a>
</li>
</ul>
</div>
</div>
</nav>
</header>
<div class="container">
    <div th:fragment="content"></div>
</div>
<footer th:fragment="footer" class="footer mt-auto py-3 bg-light">
    <div class="container text-center">Copyright ©;
        <span th:text="#dates.year(#dates.createNow())"></span>
        <a href="https://www.hutech.edu.vn/">HUTECH Education</a>
    </div>
    <script th:src="@{/js/bootstrap.min.js}"></script>
    <script th:src="@{/js/jquery-3.7.0.min.js}"></script>
</footer>
</body>
</html>
```

7.5 Kiểm tra kết quả phân quyền

Giao diện khi chưa đăng nhập, chỉ hiển thị nút Login và nút trở về trang Home



Hình 7.9: Giao diện khi chưa đăng nhập

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.

The screenshot shows the same interface as Figure 7.9, but with a 'Logout' button highlighted in red at the top right. Below it, the title 'My Book List' is displayed. A table lists five books with columns for Id, Title, Author, Price, Category, and Action. Each row has three buttons: 'Edit' (blue), 'Delete' (red), and 'Add to cart' (green). A search bar and a 'Search' button are also visible.

Id	Title	Author	Price	Category	Action
1	Phát triển phần mềm mã nguồn mở	Nguyễn Đình Ánh	12.0	Công nghệ phần mềm	<button>Edit</button> <button>Delete</button> <button>Add to cart</button>
2	Cơ sở dữ liệu phân tán	Cao Tùng Anh	10.0	Hệ thống thông tin	<button>Edit</button> <button>Delete</button> <button>Add to cart</button>
3	Điều tra tấn công	Văn Thiên Hoàng	11.0	An toàn thông tin	<button>Edit</button> <button>Delete</button> <button>Add to cart</button>
4	Mạng máy tính nâng cao	Hàn Minh Châu	13.0	Mạng máy tính	<button>Edit</button> <button>Delete</button> <button>Add to cart</button>
5	Phân tích dữ liệu truyền thông xã hội	Nguyễn Thị Hải Bình	12.0	Khoa học dữ liệu	<button>Edit</button> <button>Delete</button> <button>Add to cart</button>

Hình 7.10. Giao diện xem với quyền ADMIN

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

The screenshot shows a web application interface titled "My Book List". At the top, there is a navigation bar with links: HUTECH, Home, List Book, Cart, and Logout. On the right side of the header is a search bar with a "Search" button. Below the header, the main content area displays a table of books. The table has columns for Id, Title, Author, Price, Category, and Action. There are five rows of data, each representing a book. Each row includes an "Add to cart" button in the Action column. The books listed are:

Id	Title	Author	Price	Category	Action
1	Phát triển phần mềm mã nguồn mở	Nguyễn Đinh Ánh	12.0	Công nghệ phần mềm	Add to cart
2	Cơ sở dữ liệu phân tán	Cao Tùng Anh	10.0	Hệ thống thông tin	Add to cart
3	Điều tra dân số	Văn Thiên Hoàng	11.0	An toàn thông tin	Add to cart
4	Mạng máy tính nâng cao	Hàn Minh Châu	13.0	Mạng máy tính	Add to cart
5	Phân tích dữ liệu truyền thông xã hội	Nguyễn Thị Hải Bình	12.0	Khoa học dữ liệu	Add to cart

Below the table, there is a small blue square button with the number "0".

Hình 7.11. Giao diện xem với quyền USER

7.6 Thiết lập bảo mật bằng Json Web Token

TÓM TẮT

Bài học tập trung vào phần Spring Security authorize và JWT trong Java Spring Boot. Sau khi hoàn thành bài học này, bạn sẽ có khả năng triển khai hệ thống phân quyền bảo mật và an toàn trong ứng dụng web.

Trong bài học, bạn sẽ học về cách xác thực và ủy quyền người dùng sử dụng ứng dụng web. Spring Security authorize cung cấp khả năng xác thực người dùng thông qua các phương thức như đăng nhập và đăng ký. Bạn sẽ tìm hiểu cách sử dụng các công cụ và chức năng của Spring Security để xác thực và phân quyền người dùng.

JWT (JSON Web Token) là một phương thức xác thực phổ biến trong ứng dụng web. Bạn sẽ được hướng dẫn về cách tạo và xác thực JWT trong Spring Boot. JWT cho phép tạo ra các mã thông báo (token) chứa thông tin người dùng và các quyền truy cập. Bằng cách sử dụng JWT, bạn có thể giữ trạng thái xác thực giữa các yêu cầu từ người dùng.

Trong quá trình học, bạn sẽ hiểu rõ vai trò và trách nhiệm của người dùng và quản trị viên trong việc sử dụng ứng dụng web. Người dùng có vai trò thực hiện các hoạt động cơ bản của ứng dụng, trong khi quản trị viên có quyền kiểm soát và quản lý toàn bộ hệ thống.

CÂU HỎI ÔN TẬP

1. Spring Security authorize sử dụng cơ chế nào để kiểm tra quyền truy cập của người dùng?
2. Làm thế nào để tùy chỉnh các quy tắc phân quyền trong Spring Security authorize?
3. Spring Security JWT sử dụng cơ chế mã hóa nào để bảo vệ dữ liệu trong token?
4. Spring Security JWT có cung cấp khả năng giải quyết các vấn đề bảo mật như tấn công XSS hay không?
5. Làm thế nào để giới hạn thời gian sống của một JWT trong Spring Security?

BÀI 8 LÀM QUEN VỚI API VÀ SOCKET

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

Sau khi hoàn thành việc học về phân quyền trong Java Spring Boot, người học sẽ có khả năng nắm vững các kiến thức và kỹ năng cơ bản sau đây:

- ✓ Thực hiện thao tác lấy danh sách thông qua API.
- ✓ Thực hiện thao tác thêm sách thông qua API.
- ✓ Thực hiện thao tác xoá sách thông qua API.
- ✓ Thực hiện thao tác sửa sách thông qua API.
- ✓ Có hiểu biết về việc sử dụng Websocket và RabbitMQ để xây dựng chức năng chat.

Mô tả chức năng:

Chức năng lấy danh sách, thêm, xoá, sửa sách thông qua API trong Java Spring Boot bao gồm:

- Lấy danh sách sách: Tạo điểm cuối (endpoint) API sử dụng phương thức GET để truy vấn và trả về danh sách sách từ cơ sở dữ liệu.
- Thêm sách mới: Tạo điểm cuối API sử dụng phương thức POST để đọc dữ liệu sách từ yêu cầu và lưu vào cơ sở dữ liệu.
- Xoá sách: Tạo điểm cuối API sử dụng phương thức DELETE để xoá sách dựa trên mã sách hoặc ID sách được cung cấp.
- Sửa sách: Tạo điểm cuối API sử dụng phương thức PUT để cập nhật thông tin sách dựa trên mã sách hoặc ID sách và dữ liệu cập nhật từ yêu cầu.

Các chức năng trên có thể được triển khai bằng cách sử dụng Spring Boot và Spring Data JPA để tương tác với cơ sở dữ liệu. Đồng thời, chúng có thể được bảo mật bằng Spring Security để xác thực và phân quyền truy cập. Ngoài ra, cần xây dựng chức năng chat giữa các người dùng sử dụng Websocket và RabbitMQ.

8.1 Thêm authorize cho API

Authorize trong API là quá trình xác thực người dùng và cấp quyền truy cập cho các yêu cầu API. Điều này đồng nghĩa với việc chỉ những người dùng được ủy quyền mới có thể truy cập và thực thi các hoạt động trong API.

Tìm đến và mở file **SecurityConfig.java** theo đường dẫn trong thư mục `src/main/java/fit.hutech.spring/utils`.

Bổ sung đoạn code như bên dưới vào.



```

47  ...
48  @Bean
49  public SecurityFilterChain securityFilterChain(@NotNull HttpSecurity http) throws Exception {
50      return http
51          .authorizeHttpRequests(auth -> auth
52              .requestMatchers( "/css/**", "/js/**", "/", "/oauth/**", "/register", "/error") AuthorizedUrl
53              .permitAll() AuthorizationManagerRequestMat...
54              .requestMatchers( "/books/edit/**", "/books/add", "/books/delete") AuthorizedUrl
55              .hasAnyAuthority( ...authorities: "ADMIN") AuthorizationManagerRequestMat...
56              .requestMatchers( "/books", "/cart", "/cart/**") AuthorizedUrl
57              .hasAnyAuthority( ...authorities: "ADMIN", "USER") AuthorizationManagerRequestMat...
58              .requestMatchers( "/api/**") AuthorizedUrl
59              .hasAnyAuthority( ...authorities: "ADMIN", "USER") AuthorizationManagerRequestMat...
59          .anyRequest().authenticated()
    }

```

Hình 8.1. Thêm quyền authorize cho API

Định nghĩa rằng chỉ những người dùng có vai trò "ADMIN" hoặc "USER" mới được phép truy cập vào các API endpoint có đường dẫn bắt đầu là "/api/**".

Dấu " ** " được sử dụng làm đại diện cho bất kỳ phần tử nào trong đường dẫn của URL.

Lưu ý: Khi testing api gặp lỗi **401** thì do chưa xác thực, để khắc phục lỗi này vui lòng chuyển thành **permitAll()**.

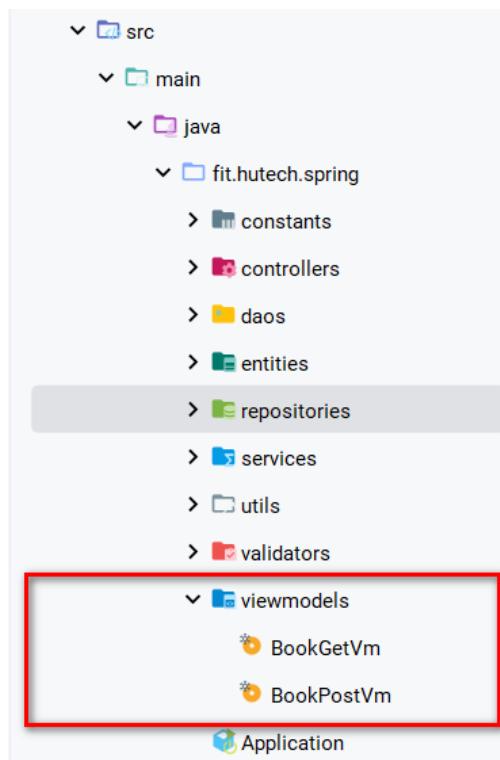
8.2 Tạo package viewmodels chứa các View Models

Package **viewmodels** là một gói (package) trong dự án Java, thường được sử dụng để chứa các record và các đối tượng (objects) được thiết kế để chuyển đổi và truyền dữ liệu giữa các thành phần khác nhau trong hệ thống. Mục tiêu chính của

gói **viewmodels** là tách biệt logic của dữ liệu và tương tác với người dùng. Nó cung cấp một cách để đóng gói dữ liệu và hành vi liên quan trong các đối tượng đặc biệt được gọi là view model. Đối tượng view model đại diện cho dữ liệu và các phương thức để truy cập và thao tác dữ liệu đó. Các bản ghi trong gói **viewmodels** thường chứa các thuộc tính (properties) tương ứng với các trường dữ liệu cần thiết để hiển thị hoặc truyền dữ liệu từ người dùng. Ngoài ra, chúng cũng có thể chứa các phương thức để kiểm tra tính hợp lệ của dữ liệu, thực hiện các xử lý đặc biệt hoặc chuẩn bị dữ liệu để hiển thị.

Gói **viewmodels** đóng vai trò quan trọng trong kiến trúc ứng dụng, đảm bảo sự tách biệt giữa giao diện người dùng và logic kinh doanh. Nó giúp giảm sự phụ thuộc trực tiếp giữa các thành phần và đơn giản hóa việc phát triển, bảo trì và kiểm thử ứng dụng.

Tạo một thư mục tên **viewmodels** được đặt tại đường dẫn **src/main/java/fit.hutech.spring/viewmodels**, trong thư mục **viewmodels** tạo thêm file **BookGetVm.java** và **BootPostVm.java** đặt trong thư mục **viewmodels**.



Hình 8.2. Tạo package ViewModel

Sau đó tại mỗi file ta bổ sung các thuộc tính sau:

BookGetVm.java

```
package fit.hutech.spring.viewmodel;

import fit.hutech.spring.entities.Book;
import jakarta.validation.constraints.NotNull;
import lombok.Builder;

@Builder
public record BookGetVm(Long id, String title, String author, Double price, String category) {
    public static BookGetVm from(@NotNull Book book) {
        return BookGetVm.builder()
            .id(book.getId())
            .title(book.getTitle())
            .author(book.getAuthor())
            .price(book.getPrice())
            .category(book.getCategory().getName())
            .build();
    }
}
```

BookPostVm.java

```
package fit.hutech.spring.viewmodel;

import fit.hutech.spring.entities.Book;
import jakarta.validation.constraints.NotNull;
import lombok.Builder;

@Builder
public record BookPostVm(String title, String author, Double price, Long categoryId) {
    public static BookPostVm from(@NotNull Book book) {
        return new BookPostVm(book.getTitle(), book.getAuthor(),
            book.getPrice(), book.getCategory().getId());
    }
}
```

8.3 Tạo ApiController

Tạo thêm 1 file có tên là **ApiController** trong thư mục controller và bổ sung các phương thức như sau:

- **@RestController**: Đánh dấu lớp là một điều khiển của Spring Boot để xử lý yêu cầu và trả về đáp ứng dưới dạng dữ liệu hoặc định dạng khác.
- **@CrossOrigin("*")**: Cho phép yêu cầu gọi từ bất kỳ nguồn nào, không bị hạn chế bởi chính sách cùng nguồn của trình duyệt.
- **@RequestMapping("/api/v1/books")**: Xác định đường dẫn URI mà ApiController lắng nghe để xử lý yêu cầu liên quan đến sách trong phiên bản API v1 của ứng dụng.

```
package fit.hutech.spring.controllers;

import fit.hutech.spring.services.BookService;
import fit.hutech.spring.services.CartService;
import fit.hutech.spring.services.CategoryService;
import fit.hutech.spring.viewmodel.BookGetVm;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/v1")
@CrossOrigin(origins = "*")
@RequiredArgsConstructor
public class ApiController {
    private final BookService bookService;

    private final CategoryService categoryService;

    private final CartService cartService;

    @GetMapping("/books")
    public ResponseEntity<List<BookGetVm>> getAllBooks(Integer pageNo,
    Integer pageSize, String sortBy) {
        return ResponseEntity.ok(bookService.getAllBooks(
            pageNo == null ? 0 : pageNo,
```

```
        pageSize == null ? 20 : pageSize,
        sortBy == null ? "id" : sortBy)
        .stream()
        .map(BookGetVm::from)
        .toList());
    }

    @GetMapping("/books/id/{id}")
    public ResponseEntity<BookGetVm> getBookById(@PathVariable Long id)
    {
        return ResponseEntity.ok(bookService.getBookById(id)
            .map(BookGetVm::from)
            .orElse(null));
    }

    @DeleteMapping("/books/{id}")
    public ResponseEntity<Void> deleteBookById(@PathVariable Long id) {
        bookService.deleteBookById(id);
        return ResponseEntity.ok().build();
    }

    @GetMapping("/books/search")
    public ResponseEntity<List<BookGetVm>> searchBooks(String keyword)
    {
        return ResponseEntity.ok(bookService.searchBook(keyword)
            .stream()
            .map(BookGetVm::from)
            .toList());
    }
}
```

Bài tập: Sinh viên thực hiện tiếp chức năng thêm, sửa, xoá Book bằng API

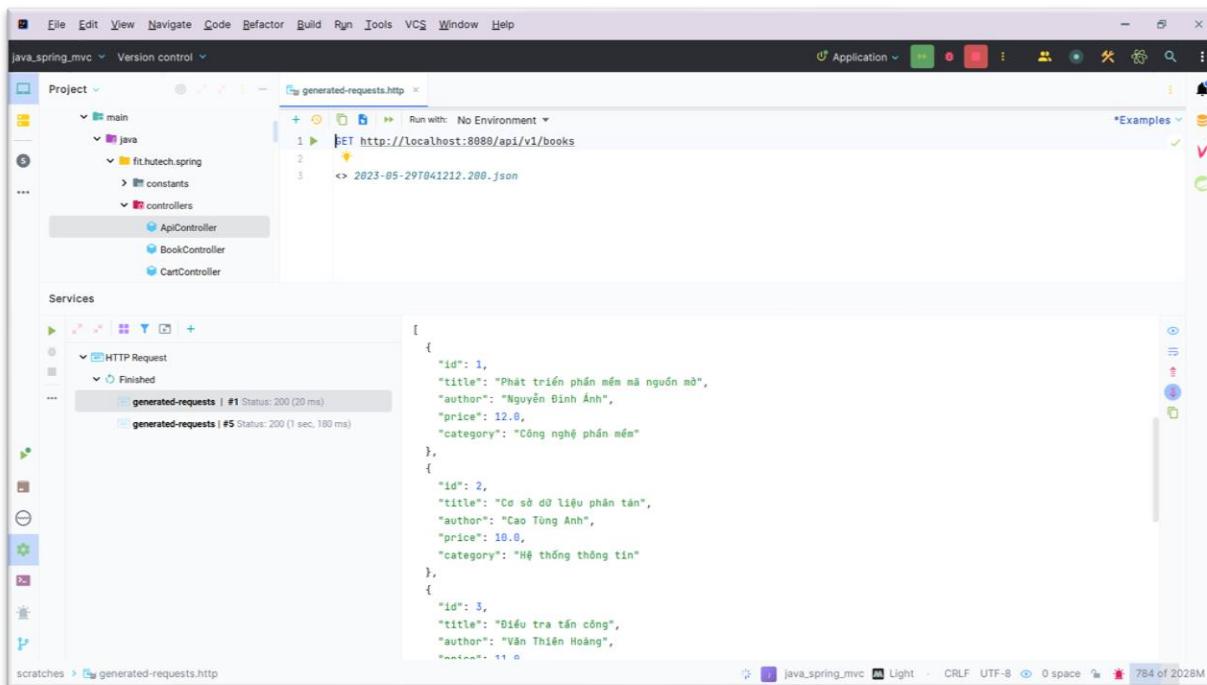
Sau khi tạo xong các API gồm: **Lấy danh sách Books, lấy sách theo ID sách, xoá sách dựa vào ID sách.** Kết quả khi truy vấn các đường dẫn API sẽ trả về các kết quả như bên dưới.

- 💡 Gọi API lấy danh sách Books <http://localhost:8080/api/v1/books>, kết quả trả về là chuỗi JSON như ảnh bên dưới.



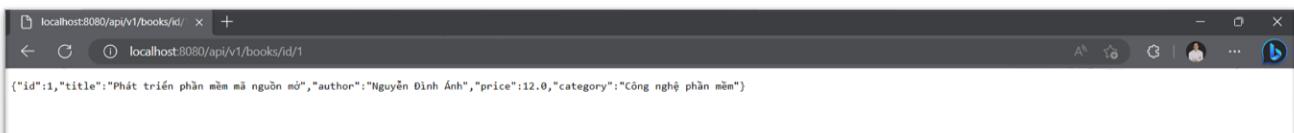
```

26
27 @GetMapping("/books")
28     public ResponseEntity<List<BookGetVm>> getAllBooks(Integer pageNo, Integer pageSize, String sortBy) {
29         return ResponseEntity.ok(bookService.getAllBooks(
30             pageNo == null ? 0 : pageNo,
31             pageSize == null ? 20 : pageSize,
32             sortBy == null ? "id" : sortBy).stream()
33             .map(BookGetVm::from).toList());
34     }
35 }
```



Hình 8.3. Gọi api lấy danh sách books

💡 Gọi API lấy thông tin của một đối tượng Book (quển sách) dựa vào id của Book đó <http://localhost:8080/api/v1/books/{id}>, ví dụ: cần tìm cuốn sách có id = 3, gọi API <http://localhost:8080/api/v1/books/3> kết quả trả về là chuỗi JSON chứa thông tin của Book có id=3 như ảnh bên dưới. Sinh viên có thể dùng trình duyệt để kiểm thử Api như bên dưới:

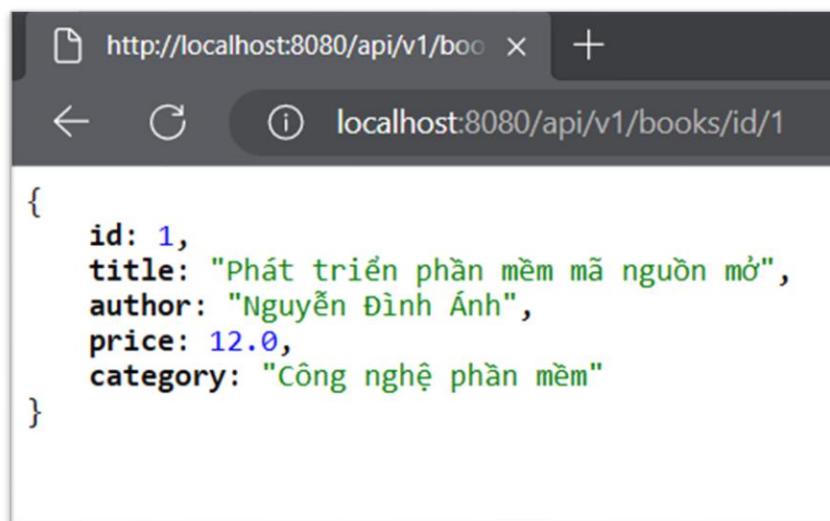


Hình 8.4. Gọi api lấy thông tin sách dựa vào id

Theo mặc định, file JSON khi xem trên trình duyệt sẽ như 1 file văn bản thông thường như bên trên. Nếu dùng trình duyệt Google Chrome bạn có thể thêm tiện ích **JSONVue** để xem JSON. Bạn có thể thêm tiện ích mở rộng này thì các file JSON sẽ được hiển thị rất đẹp và trực quan hơn nhiều trên cửa sổ trình duyệt Google Chrome.

Link add extension trên GG Chrome: [JSONVue](#)

Ví dụ như xem kết quả hiển thị danh sách books sẽ như bên dưới.



A screenshot of a Google Chrome browser window. The address bar shows the URL `localhost:8080/api/v1/books/id/1`. The main content area displays a JSON object representing a book:

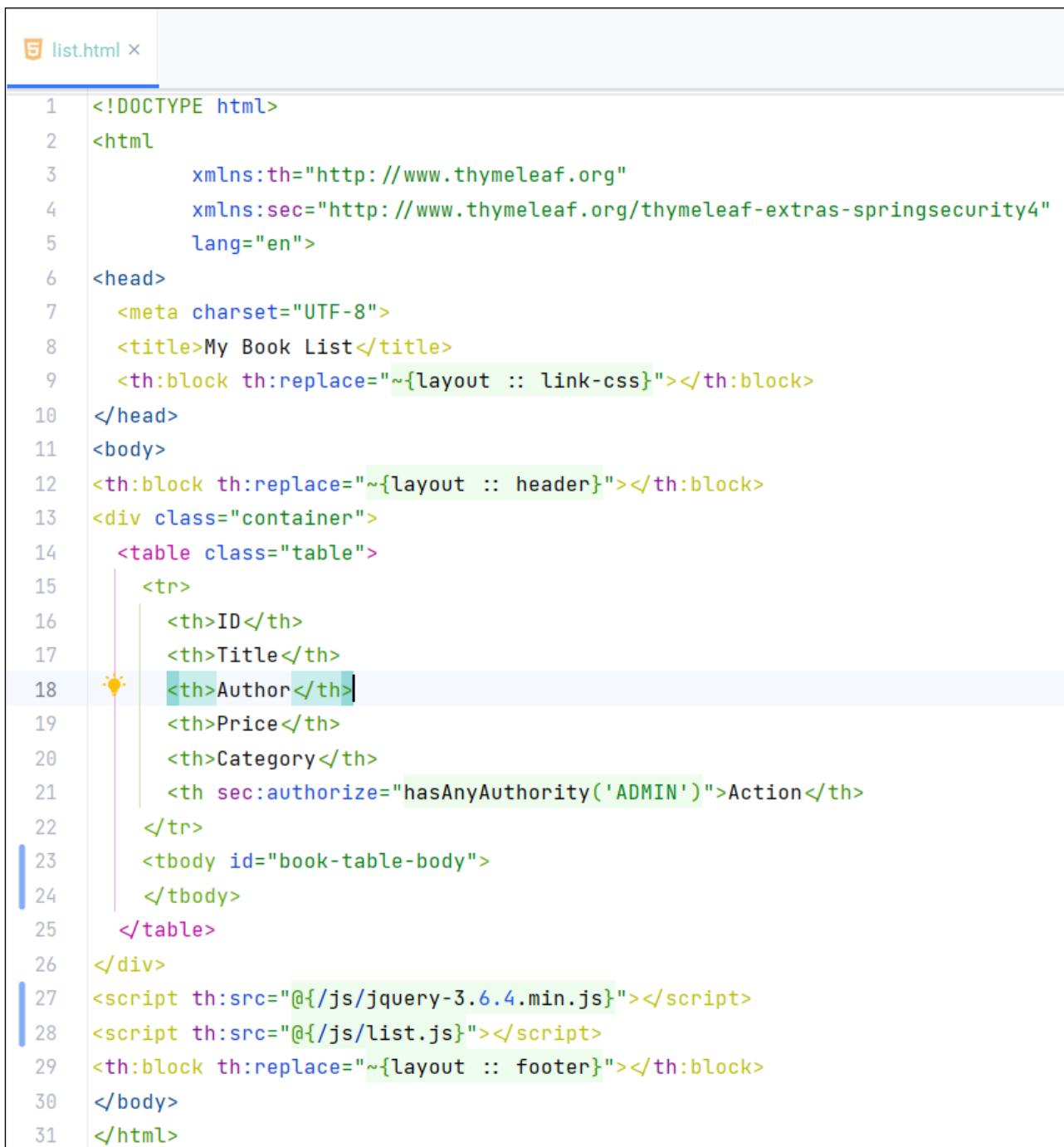
```
{  
    id: 1,  
    title: "Phát triển phần mềm mã nguồn mở",  
    author: "Nguyễn Đình Ánh",  
    price: 12.0,  
    category: "Công nghệ phần mềm"  
}
```

Hình 8.5. Thêm extend xem Json trên trình duyệt

Ngoài ra, sinh viên có thể sử dụng các công cụ khác như Postman, Insomnia, REST Client, Thunder Client... để kiểm thử API

8.4 Chỉnh sửa List.html cho phù hợp

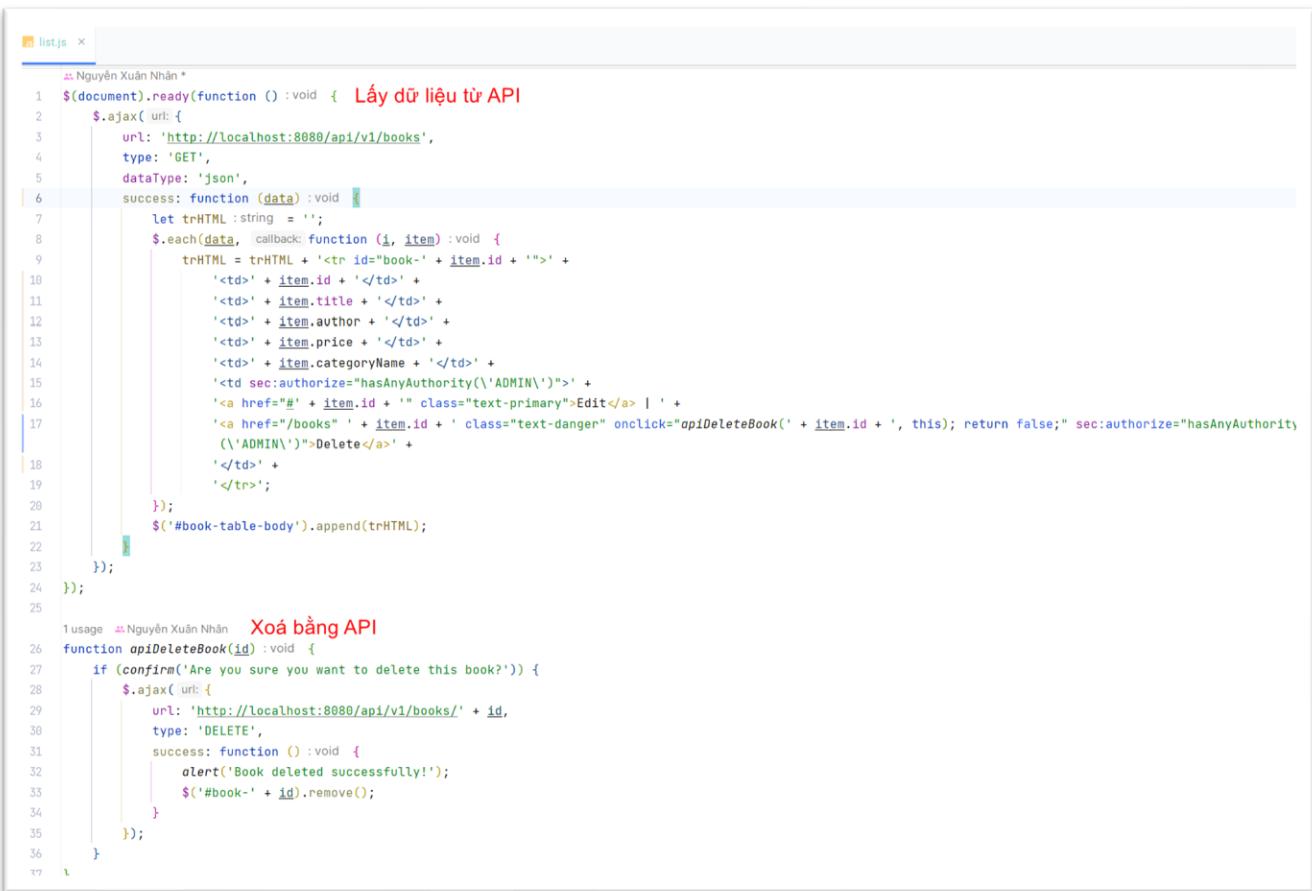
Chỉnh sửa giao diện **List.html** cho phù hợp với tính năng xây dựng API.



```
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('ADMIN')">Action</th>
22         </tr>
23         <tbody id="book-table-body">
24             </tbody>
25     </table>
26 </div>
27 <script th:src="@{/js/jquery-3.6.4.min.js}"></script>
28 <script th:src="@{/js/list.js}"></script>
29 <th:block th:replace="~{layout :: footer}"></th:block>
30 </body>
31 </html>
```

Hình 8.6. Tuỳ chỉnh lại giao diện list.html cho phù hợp



```

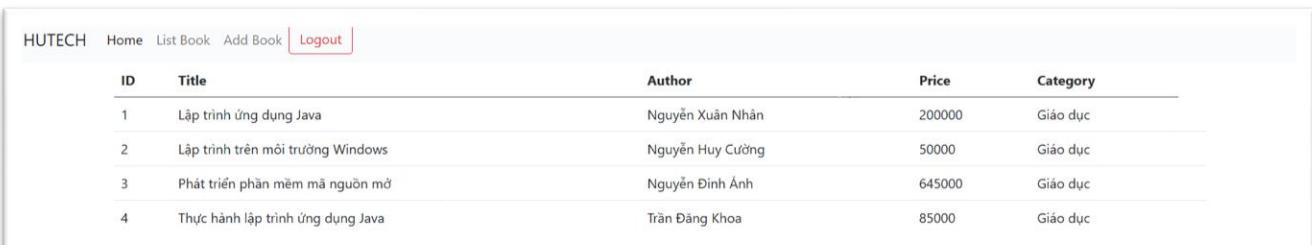
list.js x
1  as Nguyen Xuân Nhán *
2  $(document).ready(function () :void { Lấy dữ liệu từ API
3    $.ajax({ url: {
4      url: 'http://localhost:8080/api/v1/books',
5      type: 'GET',
6      dataType: 'json',
7      success: function (data) :void {
8        let trHTML :string = '';
9        $.each(data, callback: function (i, item) :void {
10          trHTML = trHTML + '<tr id="book-' + item.id + '">' +
11            '<td>' + item.id + '</td>' +
12            '<td>' + item.title + '</td>' +
13            '<td>' + item.author + '</td>' +
14            '<td>' + item.price + '</td>' +
15            '<td>' + item.categoryName + '</td>' +
16            '<td sec:authorize="hasAnyAuthority(['ADMIN\\'])">' +
17            '<a href="#" + item.id + '" class="text-primary">Edit</a> | ' +
18            '<a href="/books/" + item.id + ' class="text-danger" onclick="apiDeleteBook(' + item.id + ', this); return false;" sec:authorize="hasAnyAuthority(['ADMIN\\'])">Delete</a>' +
19            '</td>' +
20            '</tr>';
21        });
22        $('#book-table-body').append(trHTML);
23      });
24    });
25  });

1 usage as Nguyen Xuân Nhán Xoá bằng API
26  function apiDeleteBook(id) :void {
27    if (confirm('Are you sure you want to delete this book?')) {
28      $.ajax({ url: {
29        url: 'http://localhost:8080/api/v1/books/' + id,
30        type: 'DELETE',
31        success: function () :void {
32          alert('Book deleted successfully!');
33          $('#book-' + id).remove();
34        }
35      });
36    }
37  }

```

Hình 8.7. Các hàm lấy danh sách books và Delete book

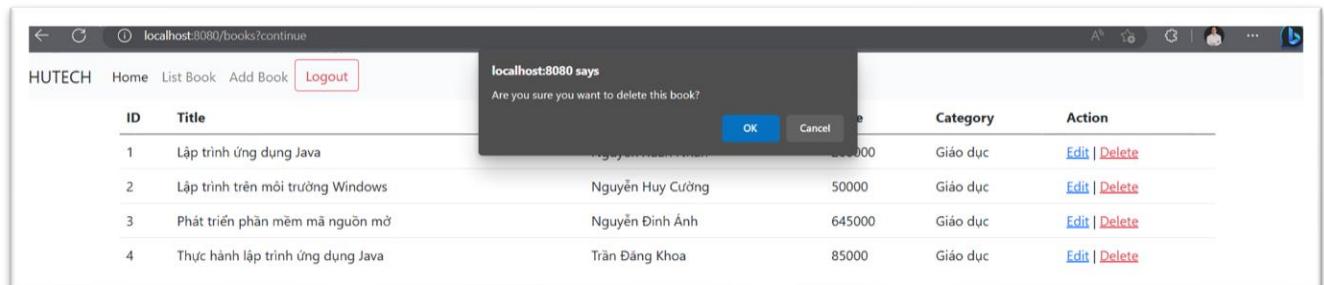
Giao diện lấy danh sách Books sử dụng dữ liệu Json từ API khi đăng nhập bằng người dùng có quyền là USER



ID	Title	Author	Price	Category
1	Lập trình ứng dụng Java	Nguyễn Xuân Nhán	200000	Giáo dục
2	Lập trình trên môi trường Windows	Nguyễn Huy Cường	50000	Giáo dục
3	Phát triển phần mềm mã nguồn mở	Nguyễn Đinh Ánh	645000	Giáo dục
4	Thực hành lập trình ứng dụng Java	Trần Đăng Khoa	85000	Giáo dục

Hình 8.8. Giao diện hiển thị danh sách Books bằng API

Giao diện Delete Book sử dụng API khi đăng nhập bằng người dùng có quyền là ADMIN



Hình 8.9. Giao diện xoá sách bằng API

8.5 Thực hiện

Chức

TÓM TẮT

API bao gồm năm phương thức chính để thực hiện các hoạt động liên quan đến dữ liệu:

- ✓ GET: Dùng để lấy thông tin hoặc dữ liệu từ nguồn tài nguyên.
- ✓ POST: Dùng để gửi dữ liệu mới lên nguồn tài nguyên để tạo mới hoặc cập nhật thông tin.
- ✓ PUT: Dùng để cập nhật dữ liệu của nguồn tài nguyên đã tồn tại.
- ✓ PATCH: Dùng để thay đổi dữ liệu tuy nhiên nó chỉ thay đổi những trường dữ liệu được yêu cầu thay đổi thay vì toàn bộ nguồn tài nguyên.
- ✓ DELETE: Dùng để xoá dữ liệu của nguồn tài nguyên.

Các phương thức này giúp đơn giản hóa việc truy cập và quản lý dữ liệu của API, bao gồm danh sách sách, xoá sách theo ID, và chỉnh sửa sách.

Lợi ích của việc sử dụng API cho các chức năng này bao gồm tiết kiệm thời gian và công sức, đơn giản hóa tích hợp dữ liệu và xử lý yêu cầu liên quan đến danh sách sách, đồng thời đảm bảo tính chính xác và bảo mật trong quá trình thao tác với dữ liệu sách.

CÂU HỎI ÔN TẬP

1. Lợi ích chính của việc sử dụng API trong phát triển ứng dụng là gì?
2. Lợi ích chính của việc sử dụng API cho các chức năng liên quan đến danh sách sách là gì?
3. API có mấy phương thức? liệt kê chi tiết ra?

TÀI LIỆU THAM KHẢO

- [1] <https://docs.oracle.com/en/java/>
- [2] <https://spring.io>
- [3] K. Siva Prasad Reddy, Sai Upadhyayula (2022). Beginning Spring Boot 3: Build Dynamic Cloud-Native Java Applications and Microservices 2nd ed. Edition. Apress
- [4] Greg L. Turnquist (2022). Learning Spring Boot 3.0 - Third Edition. Packt
- [5] Joseph B. Ottinger, Jeff Linwood, Dave Minter (2021). Beginning Hibernate 6: Java Persistence from Beginner to Pro. Apress
- [6] Andres Sacco (2022). Beginning Spring Data: Data Access and Persistence for Spring Framework 6 and Boot 3. Apress
- [7] Catalin Tudose (2023). Java Persistence with Spring Data and Hibernate. Manning
- [8] Thomas Vitale (2022). Cloud Native Spring in Action: With Spring Boot and Kubernetes. Manning
- [9] Ted Haggas (2021). Beginning IntelliJ IDEA: Integrated Development Environment for Java Programming. Apress
- [10] Marc Loy, Patrick Niemeyer, Daniel Leuck (2023). Learning Java, 6th Edition. O'Reilly Media, Inc.
- [11] Ben Weidig (2023). A Functional Approach to Java. O'Reilly Media, Inc.

PHỤ LỤC CÀI ĐẶT MÔI TRƯỜNG VÀ CÔNG CỤ CẦN THIẾT

Cài đặt Java JDK (Java Development Kit)

Java JDK (Java Development Kit) là một bộ công cụ để phát triển các ứng dụng Java. Java JDK cung cấp một số công cụ để biên dịch và thực thi mã Java, cũng như các thư viện và tài liệu hỗ trợ. **Lưu ý:** phiên bản cài đặt là **JDK20** hoặc mới hơn.

Để cài đặt Java JDK:

Google: JAVA JDK download

Hoặc truy cập đường dẫn sau:

<https://www.oracle.com/eg/java/technologies/downloads/>

The screenshot shows the Oracle Java Downloads page for JDK 20.0.1. It features a navigation bar with tabs for Linux, macOS, and Windows. Below the tabs is a table with three rows of download links. The second row, which corresponds to the X64 Installer option, is highlighted with a red border. The table columns are Product/file description, File size, and Download.

Product/file description	File size	Download
x64 Compressed Archive	180.81 MB	https://download.oracle.com/java/20/latest/jdk-20_windows-x64_bin.zip (sha256)
x64 Installer	159.95 MB	https://download.oracle.com/java/20/latest/jdk-20_windows-x64_bin.exe (sha256)
x64 MSI Installer	158.74 MB	https://download.oracle.com/java/20/latest/jdk-20_windows-x64_bin.msi (sha256)

Hình 0.1. Cài đặt Java JDK X64 Installer

Cài đặt Java JDK như hướng dẫn, chọn Next..



Hình 0.2. Install Java SE to path

Quá trình cài đặt thành công sẽ như bên dưới.



Hình 0.3. Successfully installed Java JDK

Cài đặt Laragon (Quản lý CSDL MySQL)

Laragon cung cấp tính năng quản lý CSDL (Cơ sở dữ liệu) MySQL thông qua trình quản lý cơ sở dữ liệu phpMyAdmin. Ngoài ra có thể sử dụng các phần mềm tương đương khác nhau XAMPP, WAMP, Open Server Panel...

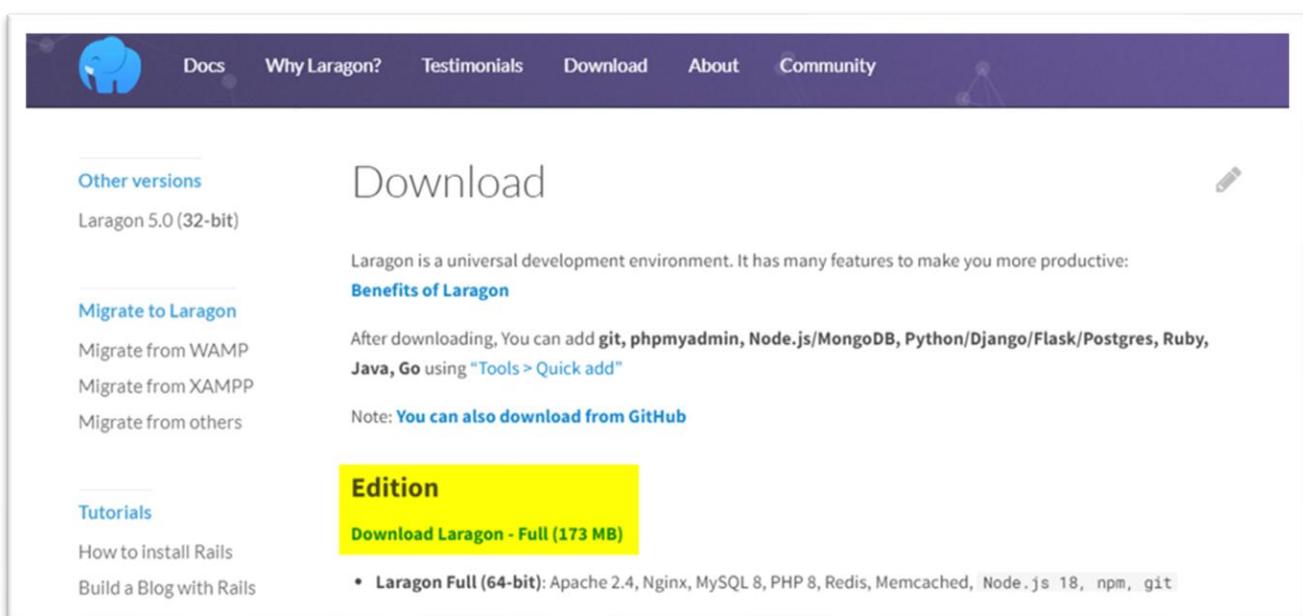
Khi bạn cài đặt và khởi chạy Laragon, nó cũng cài đặt một máy chủ MySQL đồng thời để bạn có thể tạo, quản lý và sử dụng các cơ sở dữ liệu MySQL.

Để cài đặt Laragon:

Google: download Laragon

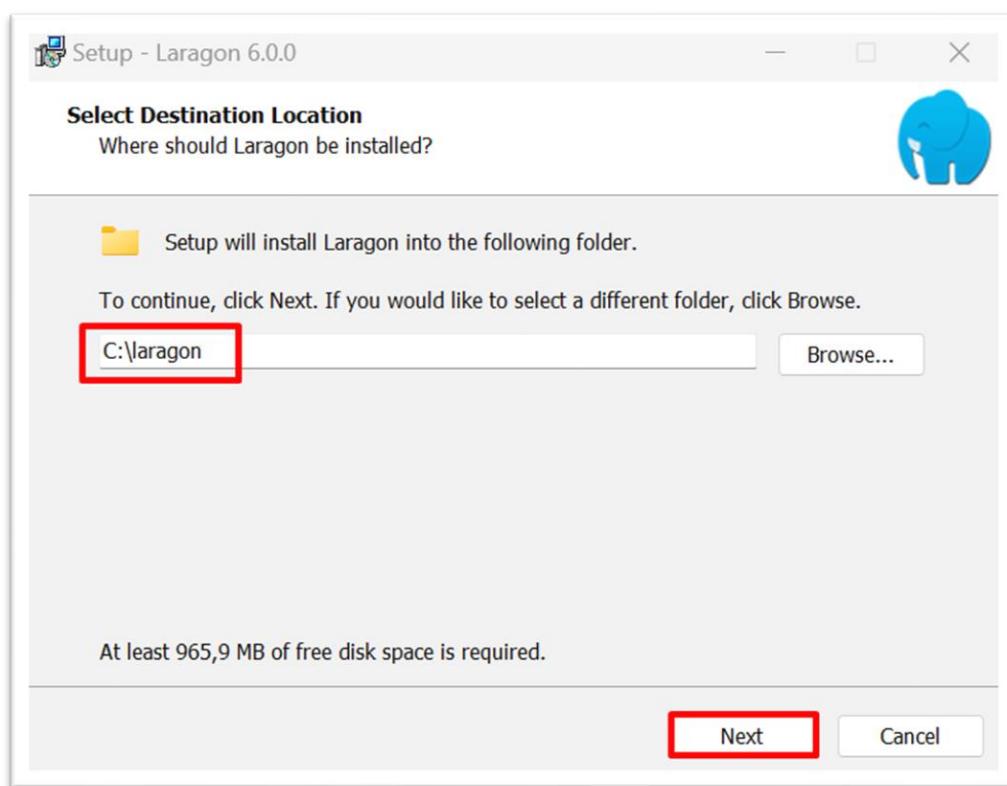
Hoặc truy cập đường dẫn sau: <https://laragon.org/download/index.html>

Tại trang Download, chọn phiên bản Download Laragon – Full (tầm 173 MB)



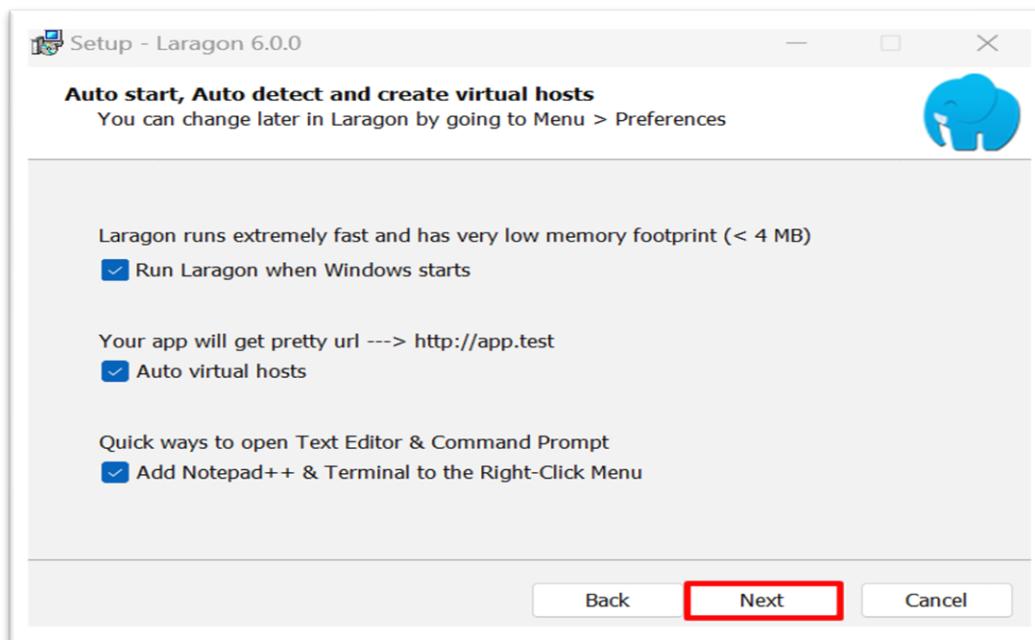
Hình 0.4. Download Edition Laragon - full

Tiếp tục, chọn nơi lưu trữ và nhấn Next

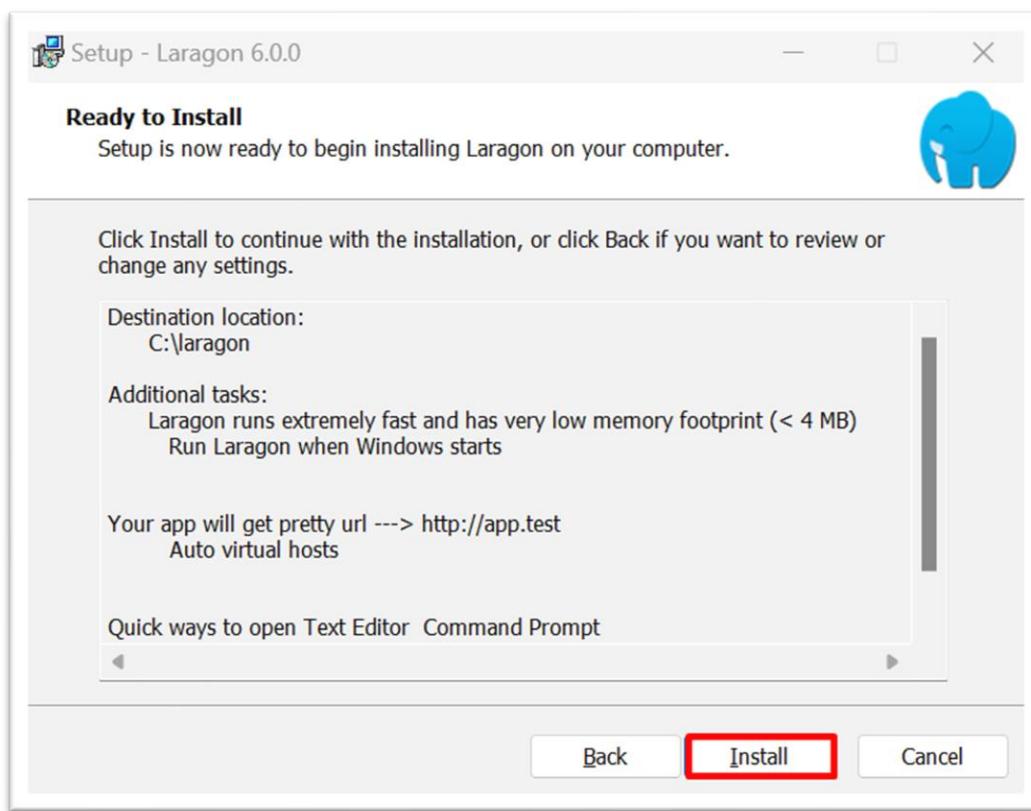


Hình 0.5. Setup Laragon path

Tích chọn 3 dấu tích và nhấn Next.



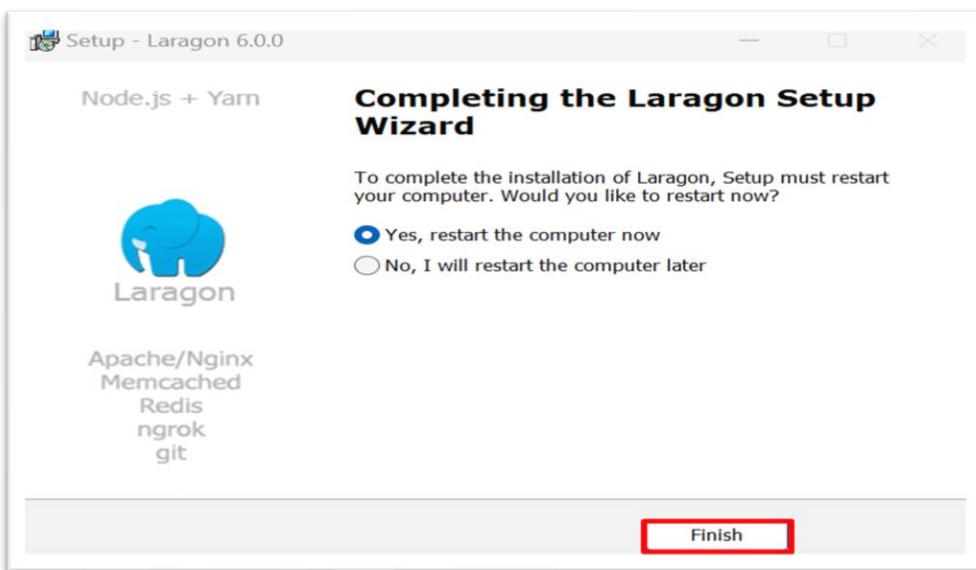
Hình 0.6. Setup Laragon runs extremely



Hình 0.7. Ready to Install Laragon

Quá trình cài đặt hoàn thành, máy tính sẽ khởi động lại.

Kiểm tra trên máy tính xuất hiện Laragon.



Hình 0.8. Completing the Laragon Setup

Cài đặt IntelliJ IDEA

IntelliJ IDEA là một IDE (Integrated Development Environment) cho lập trình viên phát triển các ứng dụng sử dụng ngôn ngữ lập trình Java, Scala, Kotlin và một số ngôn ngữ khác. IntelliJ IDEA được phát triển bởi JetBrains và được coi là một trong những IDE tốt nhất cho lập trình Java.

Ngoài ra, IntelliJ IDEA còn hỗ trợ tính năng đa nền tảng, cho phép lập trình viên phát triển ứng dụng trên nhiều hệ điều hành khác nhau bao gồm Windows, MacOS và Linux.

Ngoài ra có thể sử dụng các IDE khác như: Visual Studio Code, Netbean, Eclipse, Spring Tool Suite...

Để cài đặt IntelliJ IDEA Community:

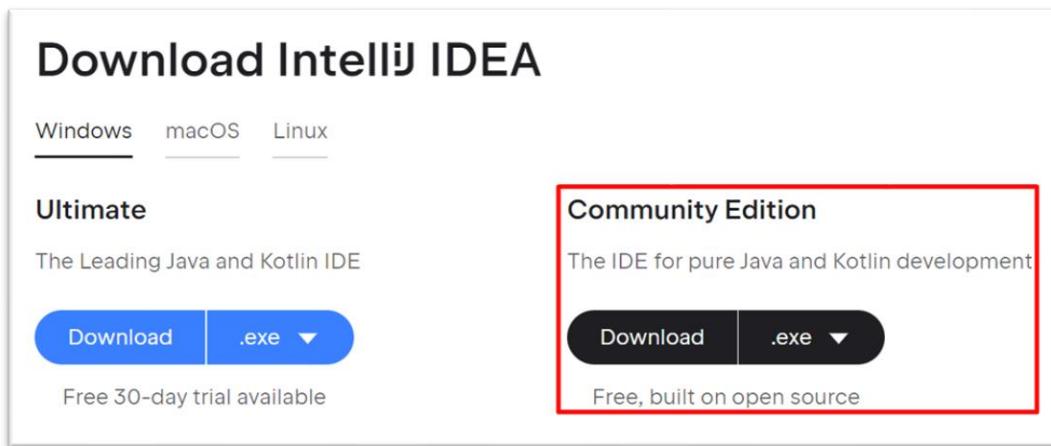
Google: Download IntelliJ IDEA Community

Hoặc truy cập đường dẫn sau:

<https://www.jetbrains.com/idea/download/#section=windows>

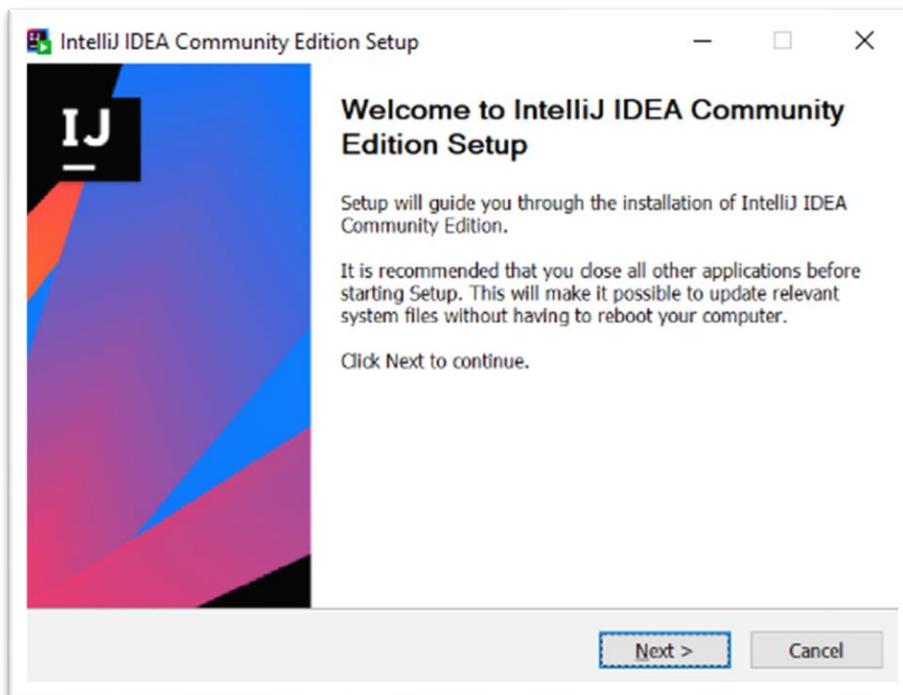
Tại trang download IntelliJ IDEA, chọn phiên bản **Community Edition**.

Phiên bản Community Edition này miễn phí cho người dùng.



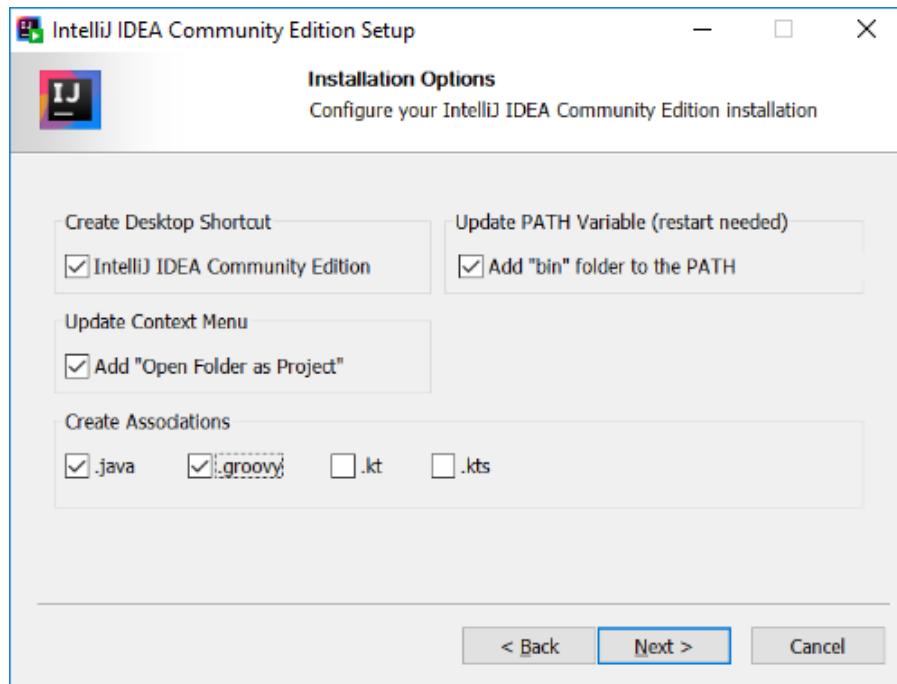
Hình 0.9. Download IntelliJ IDEA Community Edition

Sau khi tải về, nhấn cài đặt như hướng dẫn bên dưới. Tiếp theo, Nhấn **Next**



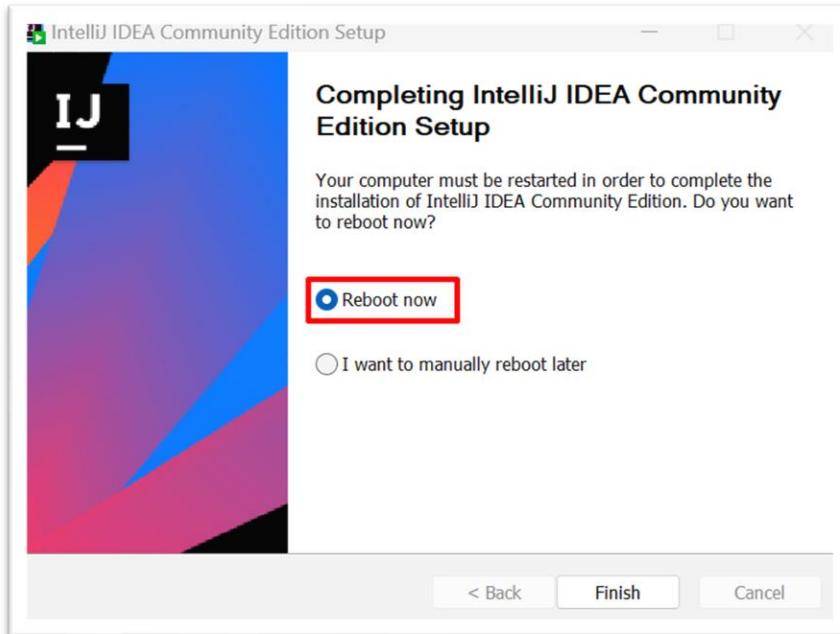
Hình 0.10. Next Setup IntelliJ IDEA

Ở bước này, tích chọn 4 lựa chọn như ảnh bên dưới, sau đó nhấn **Next**.



Hình 0.11. Installation Options IntelliJ IDEA

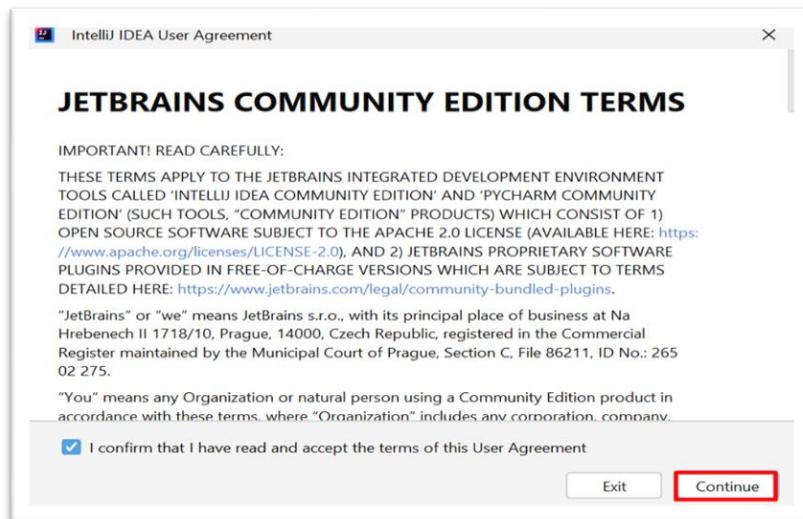
Tiếp theo, Tích chọn **Reboot now**, chọn **Finish** để hoàn tất cài đặt.



Hình 0.12. Completing IntelliJ IDEA Community

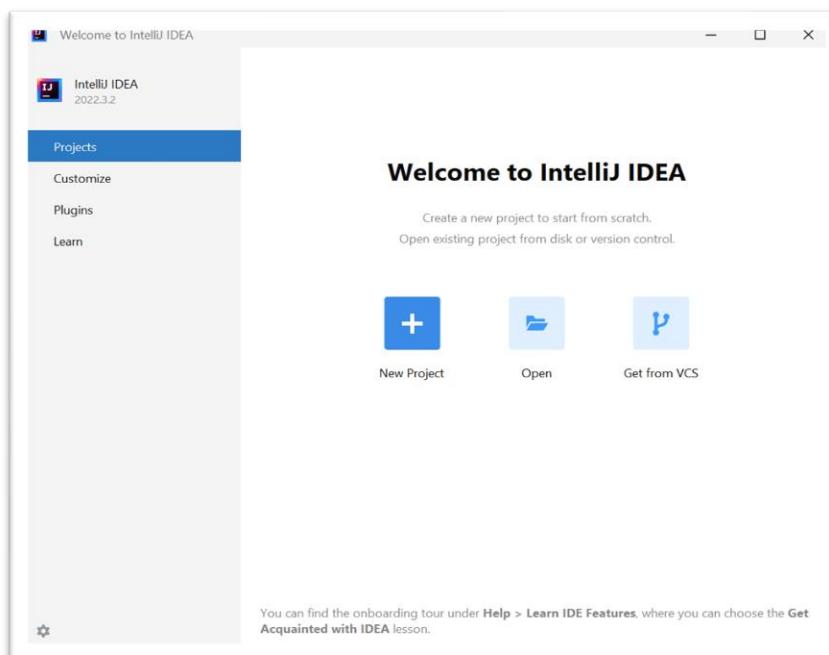
Sau khi hoàn tất cài đặt, tìm phần mềm trên máy và khởi động.

Chọn **Continue**,



Hình 0.13. IntelliJ IDEA use agreement

Giao diện bắt đầu của **IntelliJ IDEA**, chúng ta sẽ quay lại nó sau.



Hình 0.14. Giao diện bắt đầu của IntelliJ IDEA