

# MỤC LỤC

<b>MỤC LỤC .....</b>	<b>I</b>
<b>HƯỚNG DẪN.....</b>	<b>IV</b>
<b>BÀI 1. CHUẨN BỊ MÔI TRƯỜNG PHÁT TRIỂN ỨNG DỤNG VỚI SPRING BOOT VÀ CÁC THAO TÁC CƠ BẢN .....</b>	<b>1</b>
<b>1.1 CÀI ĐẶT INTELLIJ IDEA.....</b>	<b>2</b>
<b>1.2 CÀI ĐẶT JAVA 22 .....</b>	<b>4</b>
<b>1.3 TẠO DỰ ÁN SPRING BOOT.....</b>	<b>5</b>
<b>1.3.1 Download Template Spring Boot từ spring.io.....</b>	<b>5</b>
<b>1.3.2 Cấu hình dự án.....</b>	<b>7</b>
<b>1.4 CÁC THAO TÁC CƠ BẢN TRÊN SPRING BOOT .....</b>	<b>9</b>
<b>1.4.1 Khởi tạo trang Web đầu tiên.....</b>	<b>9</b>
<b>1.4.2 Khởi tạo trang web hiển thị tên sinh viên .....</b>	<b>13</b>
<b>1.5 YÊU CẦU BỔ SUNG.....</b>	<b>18</b>
<b>BÀI 2. TẠO CƠ SỞ DỮ LIỆU CHO WEBSITE BÁN HÀNG .....</b>	<b>22</b>
<b>2.1 CÀI ĐẶT LARAGON ĐỂ QUẢN LÝ CƠ SỞ DỮ LIỆU MYSQL.....</b>	<b>22</b>
<b>2.2 THÊM CÁC DEPENDENCY .....</b>	<b>24</b>
<b>2.2.1 Dependency cho Spring Data JPA .....</b>	<b>24</b>
<b>2.2.2 Dependency cho MySQL JDBC Driver .....</b>	<b>25</b>
<b>2.2.3 Dependency cho Lombok .....</b>	<b>25</b>
<b>2.3 TẠO CÁC ENTITY CLASS CHO CÁC BẢNG: .....</b>	<b>27</b>
<b>2.4 ÁNH XẠ CÁC ENTITY VỚI CƠ SỞ DỮ LIỆU.....</b>	<b>28</b>
<b>2.5 TẠO CÁC REPOSITORY INTERFACES CHO CÁC ENTITY .....</b>	<b>28</b>
<b>2.6 TRIỂN KHAI PHƯƠNG THỨC CRUD BẰNG JPARepository .....</b>	<b>29</b>
<b>2.7 BỔ SUNG CÁC YÊU CẦU CÒN THIẾU .....</b>	<b>29</b>
<b>2.8 HƯỚNG DẪN CODE MẪU .....</b>	<b>30</b>
<b>2.8.1 Entity class `Product` : .....</b>	<b>30</b>
<b>2.8.2 Entity class `Category` : .....</b>	<b>31</b>
<b>2.8.3 Entity class `Order` : .....</b>	<b>31</b>
<b>2.8.4 Entity class `OrderDetail` : .....</b>	<b>31</b>
<b>2.8.5 Repository interface `ProductRepository` : .....</b>	<b>33</b>
<b>2.8.6 Repository interface `CategoryRepository` : .....</b>	<b>33</b>
<b>2.8.7 Repository interface `OrderRepository` : .....</b>	<b>33</b>
<b>2.8.8 Repository interface `OrderDetailRepository` : .....</b>	<b>33</b>
<b>2.8.9 Cấu hình file application.properties.....</b>	<b>34</b>
<b>BÀI 3. XÂY DỰNG CHỨC NĂNG HIỂN THỊ/ THÊM/XÓA/SỬA.....</b>	<b>39</b>
<b>3.1 TẠO CÁC CONTROLLER CLASS CHO CÁC CHỨC NĂNG CRUD VÀ QUẢN LÝ NGƯỜI DÙNG .....</b>	<b>39</b>
<b>3.2 XÁC ĐỊNH CÁC ROUTE VÀ PHƯƠNG THỨC XỬ LÝ YÊU CẦU .....</b>	<b>40</b>
<b>3.3 GỌI PHƯƠNG THỨC TỪ SERVICE CLASS VÀ TRUYỀN DỮ LIỆU QUA MODEL .....</b>	<b>41</b>
<b>3.4 HƯỚNG DẪN CODE MẪU .....</b>	<b>41</b>

3.4.1	Xây dựng chức năng Hiển thị/Thêm Category .....	41
3.4.2	Chức năng xóa/ sửa cho Category.....	47
<b>3.5</b>	<b>YÊU CẦU BỔ SUNG.....</b>	<b>50</b>
<b>BÀI 4.</b>	<b>XÂY DỰNG CHỨC NĂNG GIỎ HÀNG VÀ ĐẶT HÀNG .....</b>	<b>59</b>
<b>4.1</b>	<b>XÂY DỰNG CHỨC NĂNG GIỎ HÀNG .....</b>	<b>59</b>
4.1.1	Thiết kế mô hình dữ liệu – tạo class 'CartItem' .....	59
4.1.2	Xây Dựng CartService.java .....	60
4.1.3	Xây Dựng CartController.java.....	61
4.1.4	Tạo View cho giỏ hàng .....	62
<b>4.2</b>	<b>XÂY DỰNG CHỨC NĂNG ĐẶT HÀNG .....</b>	<b>64</b>
4.2.1	Tạo OrderService.java.....	64
4.2.2	Tạo OrderController.java .....	65
4.2.3	Tạo View cho phần đặt hàng .....	66
4.2.4	Tiến hành build lại dự án và kiểm tra kết quả: .....	67
<b>4.3</b>	<b>YÊU CẦU BỔ SUNG.....</b>	<b>68</b>
<b>BÀI 5.</b>	<b>XÂY DỰNG CÁC CHỨC NĂNG XÁC THỰC VÀ PHÂN QUYỀN NGƯỜI DÙNG .....</b>	<b>69</b>
<b>5.1</b>	<b>THÊM DEPENDENCY VÀO `POM.XML` .....</b>	<b>69</b>
<b>5.2</b>	<b>BỔ SUNG LỚP `USER` VÀ `ROLE` .....</b>	<b>70</b>
5.2.1	Entity class User.java: .....	70
5.2.2	Entity class Role.java .....	72
<b>5.3</b>	<b>XÁC ĐỊNH VAI TRÒ NGƯỜI DÙNG SỬ DỤNG ENUM TRONG JAVA .....</b>	<b>73</b>
<b>5.4</b>	<b>CẤU HÌNH BẢO MẬT SỬ DỤNG SPRING SECURITY.....</b>	<b>74</b>
<b>5.5</b>	<b>XÂY DỰNG REPOSITORY .....</b>	<b>77</b>
5.5.1	UserRepository.java.....	77
5.5.2	IUserRepository.java.....	78
5.5.3	IRoleRepository.java.....	78
<b>5.6</b>	<b>XÂY DỰNG CLASS USERSERVICE.....</b>	<b>79</b>
<b>5.7</b>	<b>XÂY DỰNG USERCONTROLLER .....</b>	<b>80</b>
<b>5.8</b>	<b>PHÁT TRIỂN CÁC TRANG ĐĂNG NHẬP VÀ ĐĂNG KÝ .....</b>	<b>82</b>
5.8.1	File 'register.html' .....	82
5.8.2	File 'login.html' .....	83
5.8.3	Cập nhật 'layout.html'.....	84
5.8.4	Cập nhật file 'products-list.html' .....	87
<b>5.9</b>	<b>TIẾN HÀNH BUILD VÀ KIỂM TRA KẾT QUẢ .....</b>	<b>89</b>
5.9.1	Trang đăng ký.....	89
5.9.2	Trang đăng nhập .....	89
5.9.3	Kiểm tra Database.....	90
5.9.4	Tiến hành phân quyền cho các tài khoản: .....	91
5.9.5	Đăng nhập và kiểm tra phân quyền.....	92
<b>BÀI 6.</b>	<b>RESTFUL API.....</b>	<b>93</b>
<b>6.1</b>	<b>YÊU CẦU CÁC CHỨC NĂNG CẦN TRIỂN KHAI .....</b>	<b>93</b>

---

6.1.1 Xây dựng các API để quản lý sản phẩm.....	93
6.1.2 Giao diện người dùng .....	93
<b>6.2 HƯỚNG DẪN THỰC HIỆN .....</b>	<b>94</b>
<b>TÀI LIỆU THAM KHẢO .....</b>	<b>98</b>

# HƯỚNG DẪN

## MÔ TẢ MÔN HỌC

### Tóm tắt nội dung

Học phần cung cấp cho sinh viên kiến thức và kỹ năng cần thiết để phát triển các ứng dụng web hiện đại từ cơ bản tới nâng cao bằng cách sử dụng Spring Framework, một trong những framework phổ biến nhất của ngôn ngữ lập trình Java.

#### 1. Giới Thiệu về Spring Framework:

- Tổng quan về Spring Framework và lợi ích của việc sử dụng nó.
- Các module cốt lõi của Spring và cách chúng tương tác với nhau.

#### 2. Cấu Hình Ứng Dụng:

- Hiểu biết về cấu hình dựa trên XML và Java.
- Sử dụng Spring Boot để tự động cấu hình ứng dụng.

#### 3. Quản Lý Dependency và IoC Container:

- Cơ chế Dependency Injection (DI) và Inversion of Control (IoC) trong Spring.
- Tạo và quản lý các bean trong Spring ApplicationContext.

#### 4. Truy Cập Dữ Liệu:

- Sử dụng Spring Data JPA để tương tác với cơ sở dữ liệu.
- Cấu hình nguồn dữ liệu và tạo các repository.

#### 5. Xây Dựng ứng dụng theo mô hình MVC

- Nắm vững vai trò và chức năng của Controller, Model, View trong mô hình MVC.
- Xử lý các yêu cầu HTTP thông qua mô hình MVC

#### 6. Bảo Mật Ứng Dụng:

- Áp dụng Spring Security để xác thực và ủy quyền người dùng.

#### 7. Thực Hành và Dự Án Thực Tế:

- Áp dụng lý thuyết vào thực hành thông qua các bài lab và dự án cuối kỳ.

- Phát triển một ứng dụng web hoàn chỉnh

Môn học này đặc biệt phù hợp cho sinh viên muốn trở thành lập trình viên Java chuyên nghiệp, phát triển các ứng dụng web phức tạp, hoặc làm việc trong lĩnh vực microservices và đám mây \*.

## NỘI DUNG MÔN HỌC

- Bài 1. Chuẩn bị môi trường phát triển ứng dụng với Spring Boot và các thao tác cơ bản
- Bài 2. Tạo Cơ sở Dữ liệu cho website bán hàng
- Bài 3: Xây dựng chức năng Hiển thị/Thêm/Xóa/Sửa
- Bài 4: Xây dựng chức năng Giỏ hàng và Đặt hàng
- Bài 5: Xây dựng các chức năng Xác thực người dùng
- Bài 6: RESTful API

## KIẾN THỨC TIỀN ĐỀ

Trước khi bắt đầu môn học này, sinh viên cần có những kiến thức tiền đề sau để có thể theo kịp chương trình và hiểu sâu về các khái niệm được giảng dạy:

### 1. Ngôn ngữ lập trình Java:

- Hiểu biết vững chắc về Java Core, bao gồm OOP (Lập trình hướng đối tượng), cấu trúc dữ liệu, exception handling, và generics.

### 2. Cơ bản về mô hình MVC (Model-View-Controller):

- Hiểu rõ mô hình MVC và cách nó được áp dụng trong phát triển web.

### 3. Cơ sở dữ liệu và SQL:

- Kiến thức cơ bản về cơ sở dữ liệu quan hệ và ngôn ngữ truy vấn SQL.

### 4. Kiến thức cơ bản về HTML/CSS và JavaScript:

- Hiểu biết về cách xây dựng giao diện người dùng web với HTML/CSS.
- Cơ bản về JavaScript để có thể tương tác với front-end.

## 5. Kiến thức về RESTful APIs:

- Hiểu rõ về REST principles và cách thiết kế API cho các ứng dụng web.

# YÊU CẦU MÔN HỌC

Người học phải dự học đầy đủ các buổi lên lớp và làm bài tập đầy đủ ở nhà.

# CÁCH TIẾP NHẬN NỘI DUNG MÔN HỌC

Để học tốt môn này, người học cần 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, người học đọc trước mục tiêu và tóm tắt bài học, sau đó đọc nội dung bài học. Kết thúc mỗi ý của bài học, người đọc trả lời câu hỏi ôn tập và kết thúc toàn bộ bài học, người đọc làm các bài tập.

# PHƯƠNG PHÁP ĐÁNH GIÁ MÔN HỌC

Môn học được đánh giá gồm:

- Điểm quá trình (50%): Hình thức và cách đánh giá do giảng viên dạy lý thuyết quyết định được phê duyệt của bộ môn.
- Điểm thi cuối kỳ (50%): Hình thức làm đồ án môn học và chấm thi vẫn đáp dựa trên nội dung đồ án môn học kết hợp với lý thuyết trong các bài học. Danh sách đồ án do giảng viên quyết định được bộ môn kiểm duyệt và cung cấp vào đầu khóa học.

# BÀI 1. CHUẨN BỊ MÔI TRƯỜNG PHÁT TRIỂN ỨNG DỤNG VỚI SPRING BOOT VÀ CÁC THAO TÁC CƠ BẢN

Sau khi học xong bài này, học viên có thể:

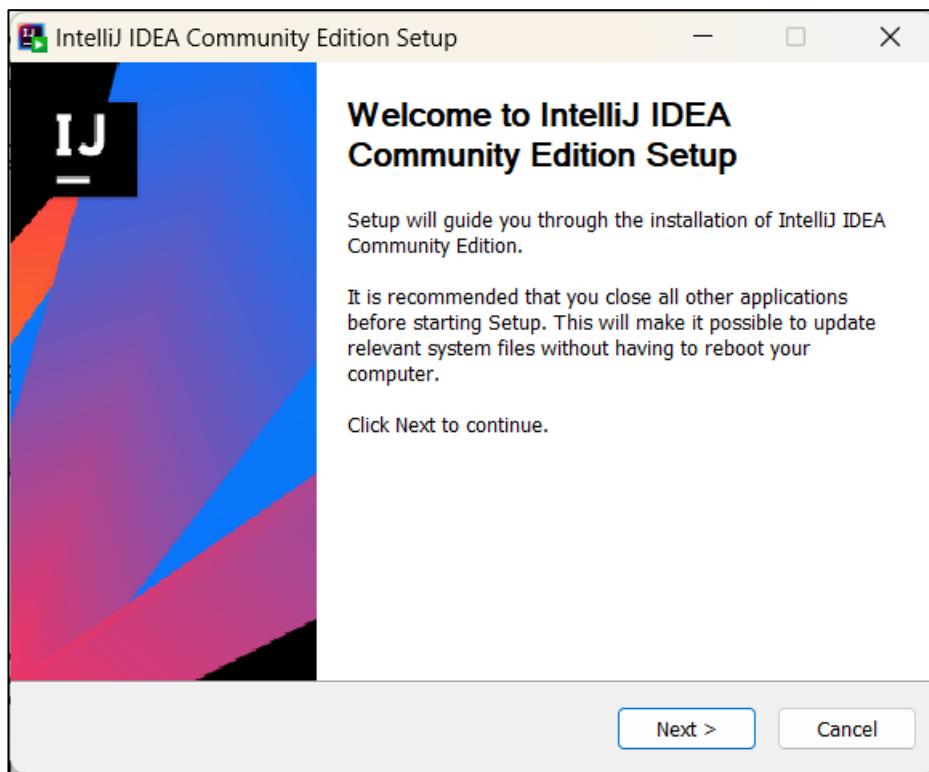
- Hiểu về IntelliJ IDEA: Bạn sẽ có kiến thức cơ bản về cách cài đặt và sử dụng IntelliJ IDEA làm môi trường phát triển tích hợp (IDE) cho việc phát triển ứng dụng Java.
- Cài đặt JDK 22: Bạn sẽ biết cách cài đặt Java Development Kit (JDK) phiên bản 22, bao gồm việc tải xuống và cấu hình biến môi trường để sử dụng JDK trong dự án của bạn.
- Tạo dự án Spring Boot: Bạn sẽ có khả năng tạo một dự án Spring Boot mới trong IntelliJ IDEA, chọn phiên bản Spring Boot và JDK phù hợp, và cấu hình các dependencies cần thiết cho dự án.
- Cấu hình cơ sở dữ liệu: Bạn sẽ hiểu cách tạo các entity class và repository interfaces để ánh xạ dữ liệu trong ứng dụng của bạn với cơ sở dữ liệu. Bạn sẽ được làm quen với các annotation như `@Entity`, `@Table`, `@Id`, `@GeneratedValue`, `@ManyToOne`, `@OneToMany` và biết cách sử dụng chúng để tương tác với cơ sở dữ liệu.
- Sử dụng IntelliJ IDEA và Java 22: Bạn sẽ có kỹ năng sử dụng IntelliJ IDEA như một IDE để phát triển ứng dụng Spring Boot và làm việc với phiên bản Java 22.

Với những kiến thức này, bạn đã chuẩn bị môi trường cho việc xây dựng website với Spring Boot và có nền tảng để tiếp tục các bước tiếp theo trong quá trình phát triển ứng dụng của bạn.

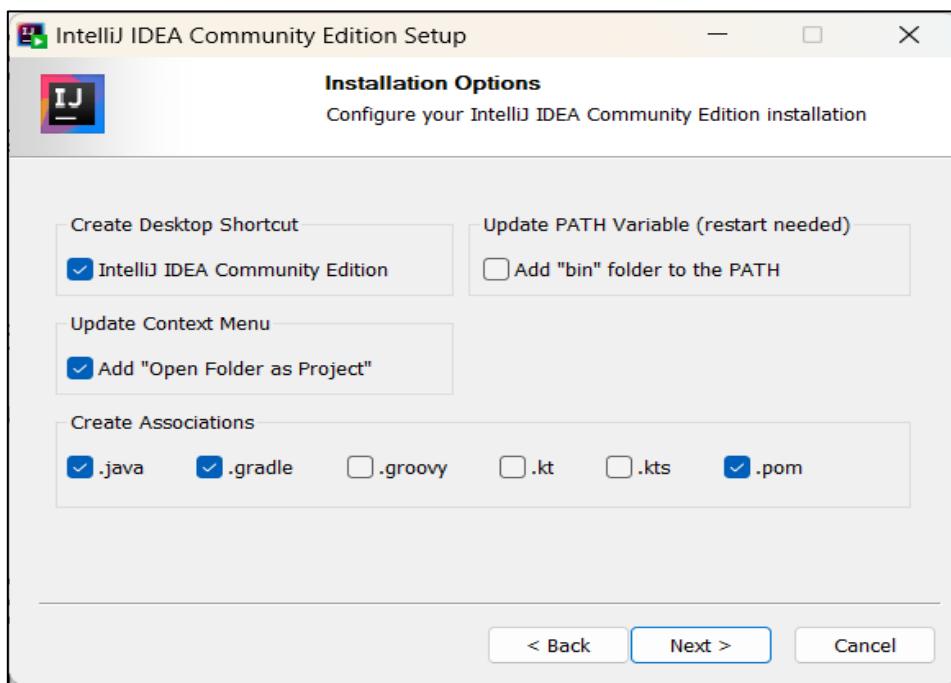
## 1.1 Cài đặt IntelliJ IDEA

---

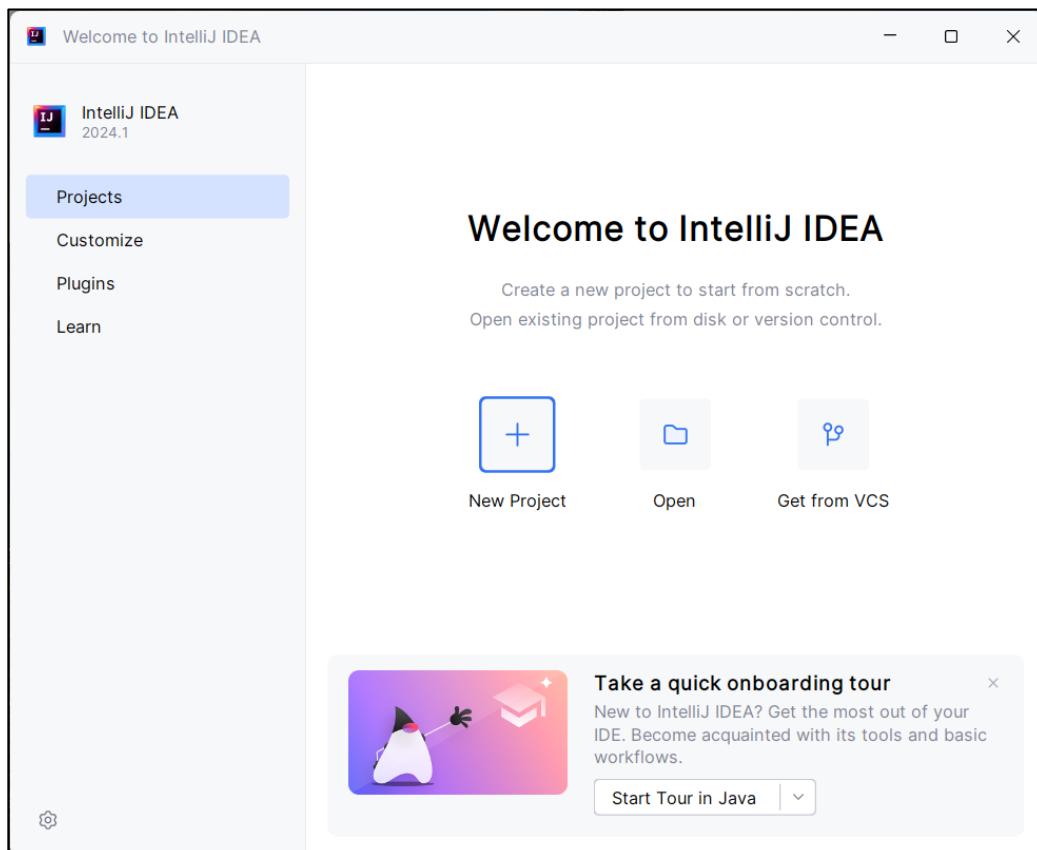
- Truy cập trang web chính thức của JetBrains và tải xuống phiên bản IntelliJ IDEA Community (Miễn phí).
- Sau khi tải xuống, cài đặt IntelliJ IDEA bằng cách chạy tệp cài đặt đã tải về.



- Chọn cài đặt như sau



- Hoàn thành quá trình cài đặt bằng cách làm theo các hướng dẫn trên màn hình.



## 1.2 Cài đặt Java 22

- Truy cập trang web chính thức của Oracle hoặc OpenJDK và tải xuống JDK 22.

Link: <https://www.oracle.com/java/technologies/downloads/#jdk22-windows>

- Chạy tệp cài đặt JDK đã tải xuống và làm theo hướng dẫn trên màn hình để cài đặt JDK.



- Sau khi cài đặt thành công, mở một cửa sổ **Terminal** và nhập lệnh `java -version` để kiểm tra phiên bản Java đã cài đặt.

A screenshot of a Windows Command Prompt window titled 'Command Prompt'. The window shows the following text:  
(c) Microsoft Corporation. All rights reserved.  
C:\Users\winda>java -version  
java version "22.0.1" 2024-04-16  
Java(TM) SE Runtime Environment (build 22.0.1+8-16)  
Java HotSpot(TM) 64-Bit Server VM (build 22.0.1+8-16, mixed mode, sharing)  
C:\Users\winda>

## 1.3 Tạo dự án Spring Boot

### 1.3.1 Download Template Spring Boot 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.

The screenshot shows the Spring.io website with the URL 'spring.io/projects/spring-boot' in the address bar. A green banner at the top says 'Call for papers is now open — [Submit your talk](#) by May 3!'. The main navigation menu includes 'Why Spring', 'Learn', 'Projects', and 'Academy'. On the left, there's a sidebar with links to 'Spring Boot' (which is highlighted in black), 'Spring Framework', 'Spring Data', and 'Spring Cloud'. The central content area features a large 'Spring Boot' title with a '3.2.5' badge. Below it are tabs for 'OVERVIEW' (which is selected and highlighted in grey), 'LEARN', 'SUPPORT', and 'SAMPLES'. To the right, a sidebar lists 'Overview', 'Spring Boot' (which is highlighted with a red box), 'Spring Framework', 'Spring Cloud', and 'Spring Cloud Data Flow'. The 'Spring Boot' item in the sidebar is also highlighted with a red box.

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

The screenshot shows the 'Spring Initializr' page with the URL 'spring.io/projects/spring-boot'. It features a list of developer tools and a link to join the community on Gitter. At the bottom, there's a call-to-action button with the text 'Quickstart Your Project' and the subtext 'Bootstrap your application with Spring Initializr.' The entire button is enclosed in a red box.

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

The screenshot shows the Spring Initializr interface for creating a new Spring Boot project. The configuration fields are outlined with red boxes:

- Project:** Maven (selected)
- Language:** Java (selected)
- Spring Boot:** 3.2.5 (selected)
- Project Metadata:**
  - Group: com.hutech
  - Artifact: demo
  - Name: demo
  - Description: Demo project for Spring Boot
  - Package name: com.hutech.demo
- Packaging:** Jar (selected)
- Java:** 22 (selected)

The **Dependencies** section on the right contains several pre-selected dependencies, each with a red box around it:

- Spring Web** (WEB): Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
- Spring Boot DevTools** (DEVELOPER TOOLS): Provides fast application restarts, LiveReload, and configurations for enhanced development experience.
- Thymeleaf** (TEMPLATE ENGINES): A modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes.

An "ADD DEPENDENCIES..." button is located at the top right of the dependencies area.

Ý nghĩa lựa chọn quan trọng trong **Spring Initializr**:

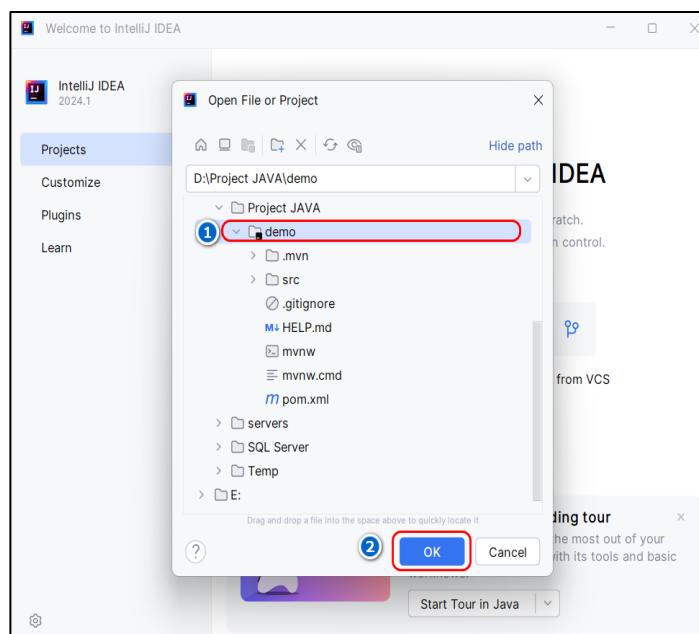
- **Project:** Lựa chọn loại dự án Spring Boot
- **Language:** Chọn ngôn ngữ lập trình cho dự án, bao gồm Java hoặc Kotlin.
- **Spring Boot Version:** Chọn phiên bản Spring Boot mà dự án sẽ sử dụng.
- **Group:** Nhập tên nhóm cho dự án, thường là domain name ngược lại.
- **Artifact:** Nhập tên đại diện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

Sau đó, nhấn **GENERATE** để tải về file **.zip** ứng với dự án.

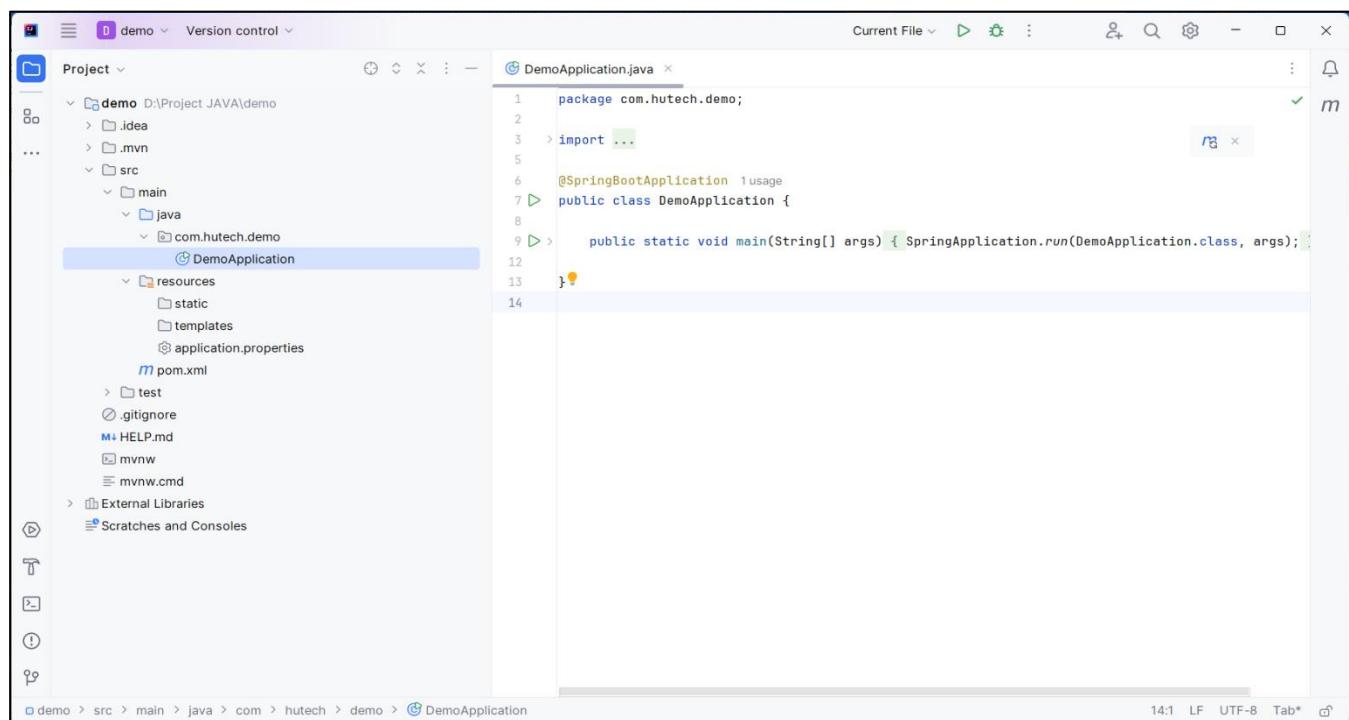
### 1.3.2 Cấu hình dự án

Tiến hành giải nén thư mục dự án **Spring boot** template có tên **demo.zip** đã được tải về trước đó. Sau khi giải nén thành công, ta có thể thấy thư mục **demo** xuất hiện.

Tiếp theo, mở **IntelliJ IDEA** và chọn **Open**. Duyệt đến thư mục dự án **demo**. IntelliJ IDEA sẽ tải và mở dự án **demo**.

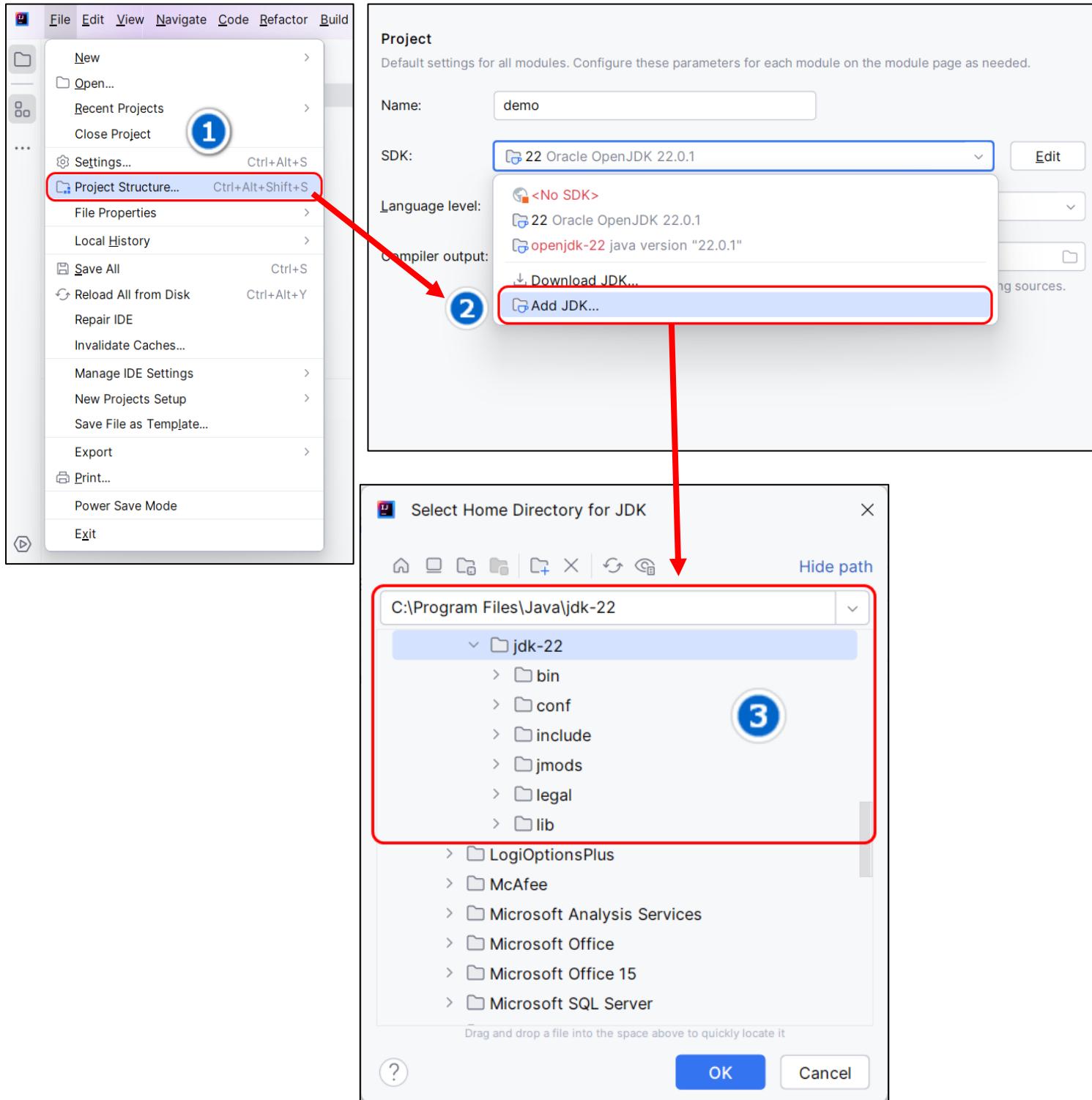


- Chọn "OK" và chọn "Trust Project" để khởi tạo dự án. Kết quả:



## Thiết lập đường dẫn tới thư mục JDK đã được cài đặt trước đó:

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



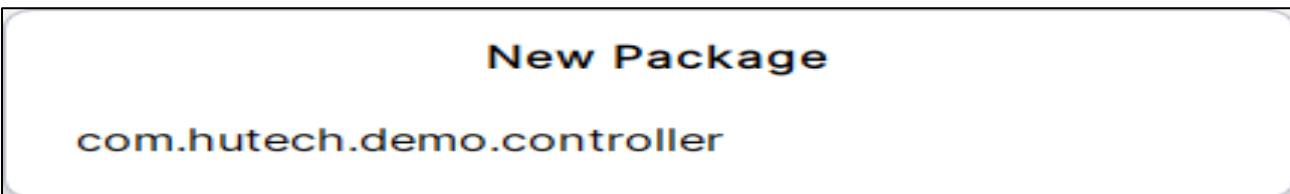
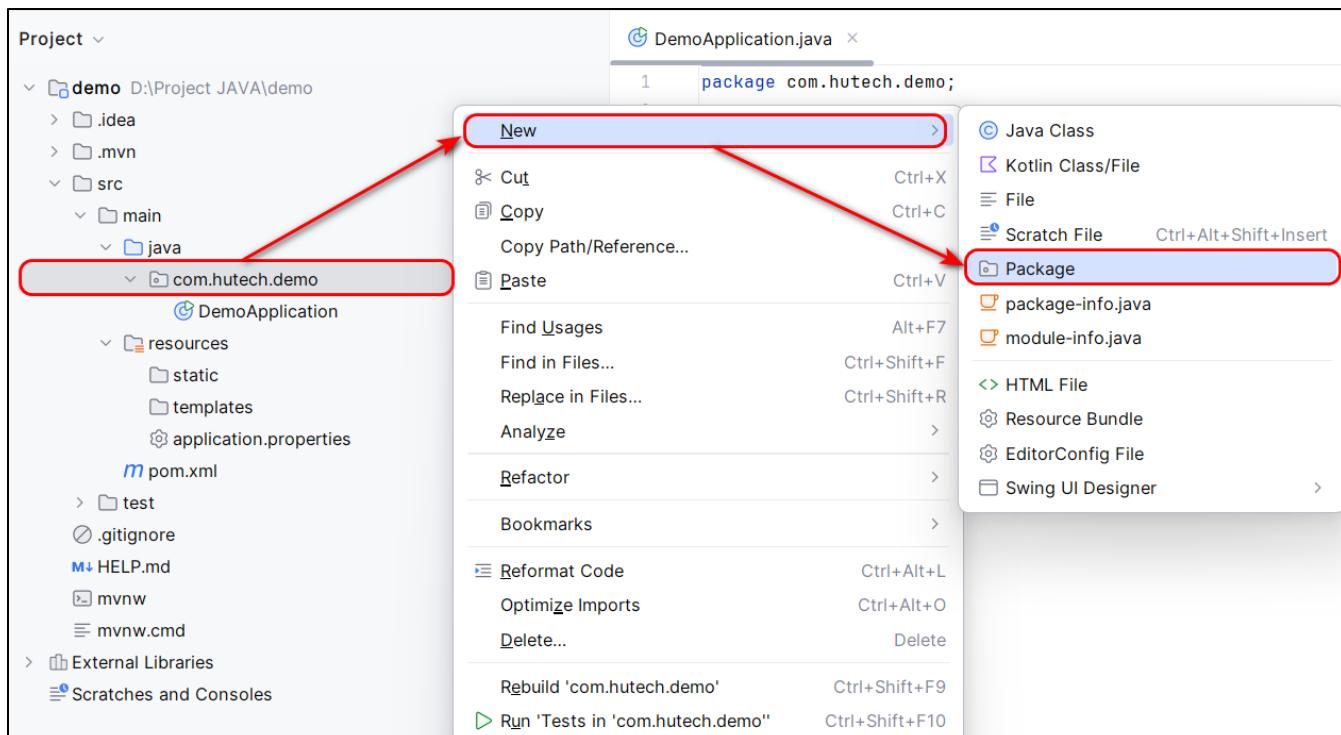
## 1.4 Các thao tác cơ bản trên Spring Boot

### 1.4.1 Khởi tạo trang Web đầu tiên

Trang Web in ra dòng chữ:

**"XIN CHÀO TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÀNH PHỐ HỒ CHÍ MINH!"**

Tạo package **controller** tại **src/main/java/com.hutech.demo**. Nhấn chuột phải vào **com.hutech.demo**, chọn **New → Package**



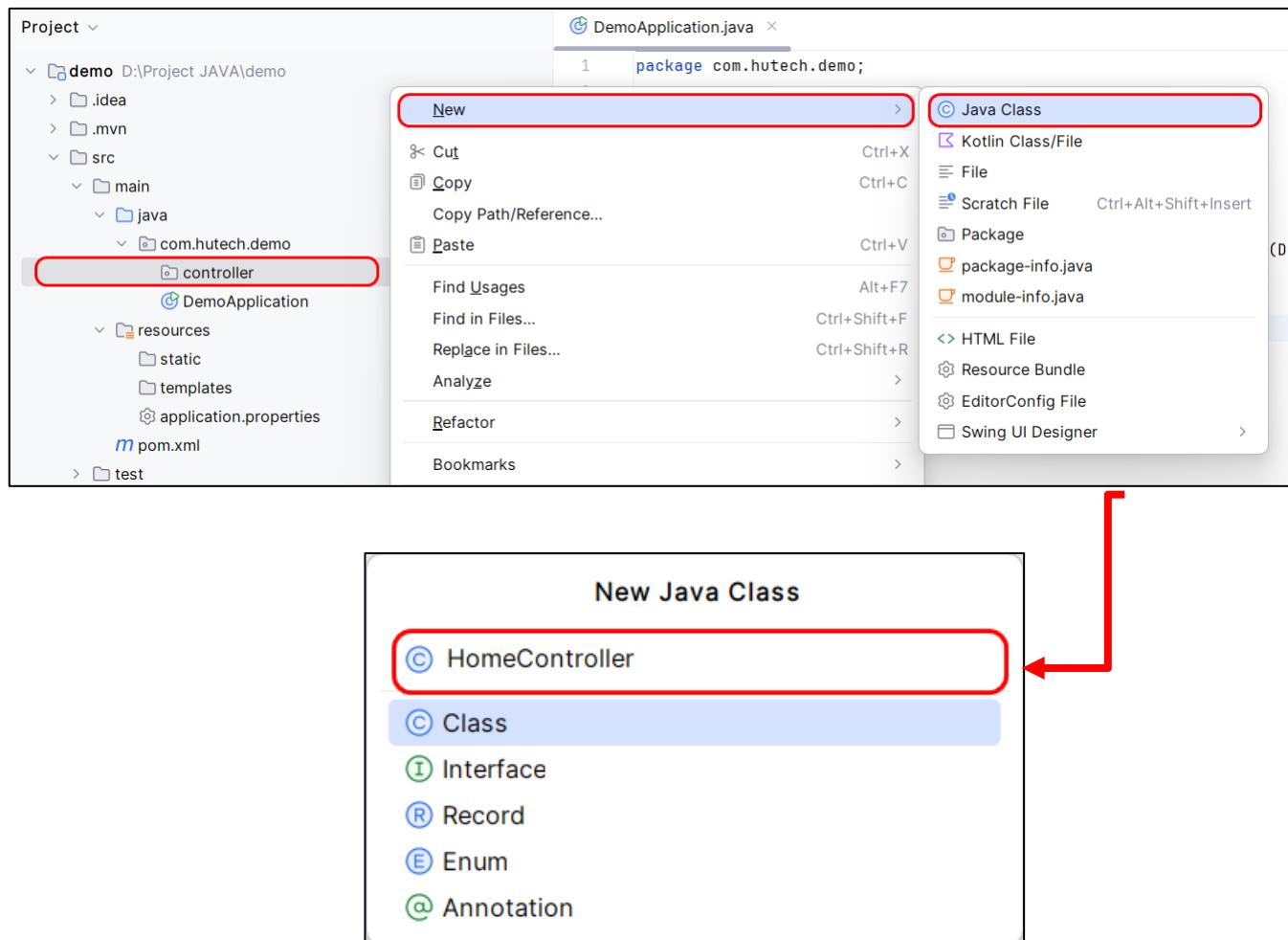
Đặt tên package là **controller** như trên

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

**src/main/java/com.hutech.demo/controller.**

Nhấn chuột phải vào **controller**. Sau đó chọn **New → Java Class → Class** →

Nhập tên class là **HomeController**. Cuối cùng nhấn **ENTER** để khởi tạo.



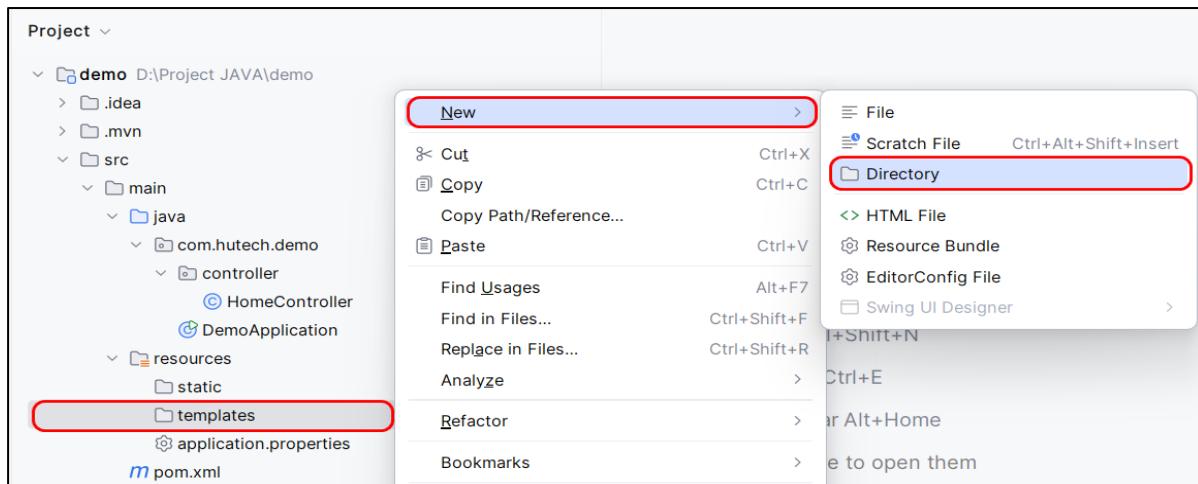
Chỉnh sửa nội dung **HomeController.java** như ảnh bên dưới:

```
package com.hutech.demo.controller;

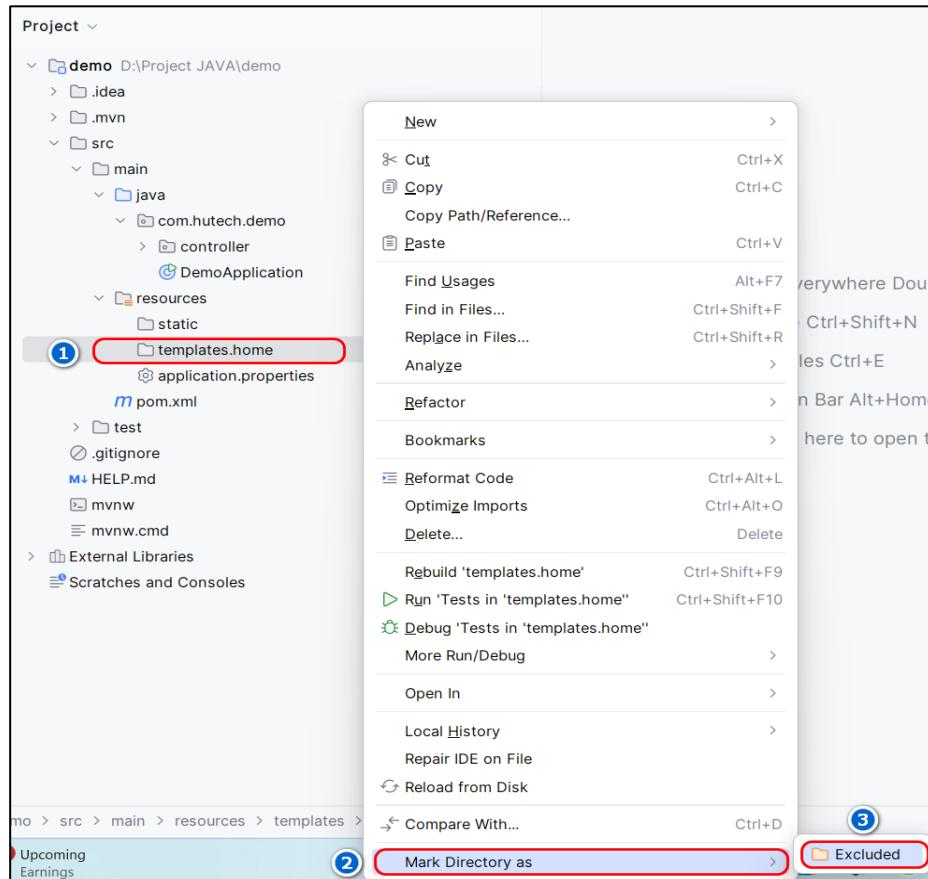
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

@Controller public class HomeController {
    @GetMapping("/")
    public String hello(Model model) {
        model.addAttribute("message", "XIN CHÀO TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÀNH PHỐ HỒ CHÍ MINH!");
        return "home/home";
    }
}
```

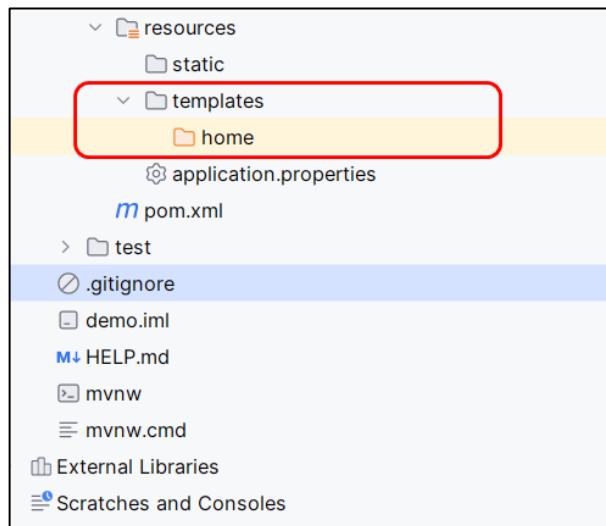
Tạo thư mục **home** đặt tại **src/main/resources/templates**. Chuột phải vào **templates**, sau đó chọn **New → Directory**.



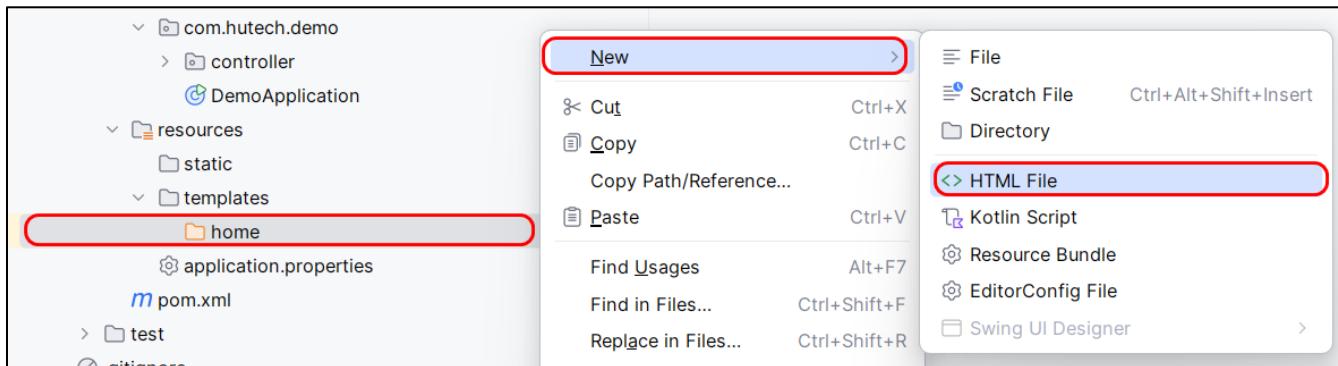
Để đảm bảo rằng thư mục **Home** hiển thị ngay dưới thư mục **Templates** như được thể hiện trong ảnh bên dưới, hãy thực hiện các bước sau. Nhấn chuột phải vào thư mục **templates.home**, sau đó chọn **Make Directory as → Exclude**.



Kết quả như sau:



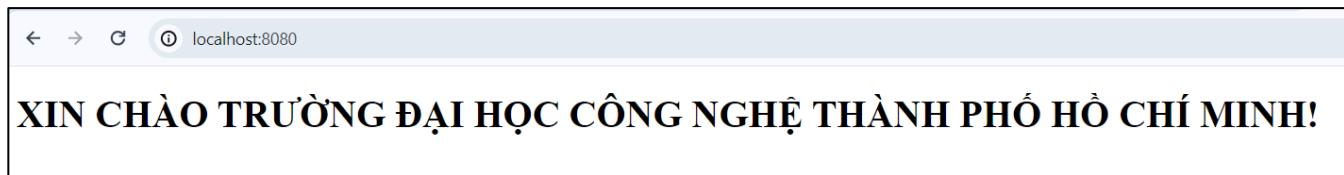
Tạo file '**home.html**' đặt tại **src/resources/templates/home**. Nhấn chuột phải vào thư mục home, sau đó chọn **New → HTML File** → đặt tên và ấn **ENTER**



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

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Hello</title>
</head>
<body>
<h1 th:text="${message}">XIN CHÀO HUTECH!</h1>
</body>
</html>
```

Tiến hành build và truy cập trang với địa chỉ <http://localhost:8080>:



### 1.4.2 Khởi tạo trang web hiển thị tên sinh viên

Xây dựng form nhập thông tin sinh viên, có chứa các kiểm tra ràng buộc sau:

- *Tên bắt buộc nhập.*
- *Tuổi phải là Số, có độ giá trị độ tuổi từ 18 đến 100.*
- *Khoa phải là Chữ*

Sau khi nhập thông tin sinh viên. Trang sẽ trả về kết quả thông tin sinh viên.

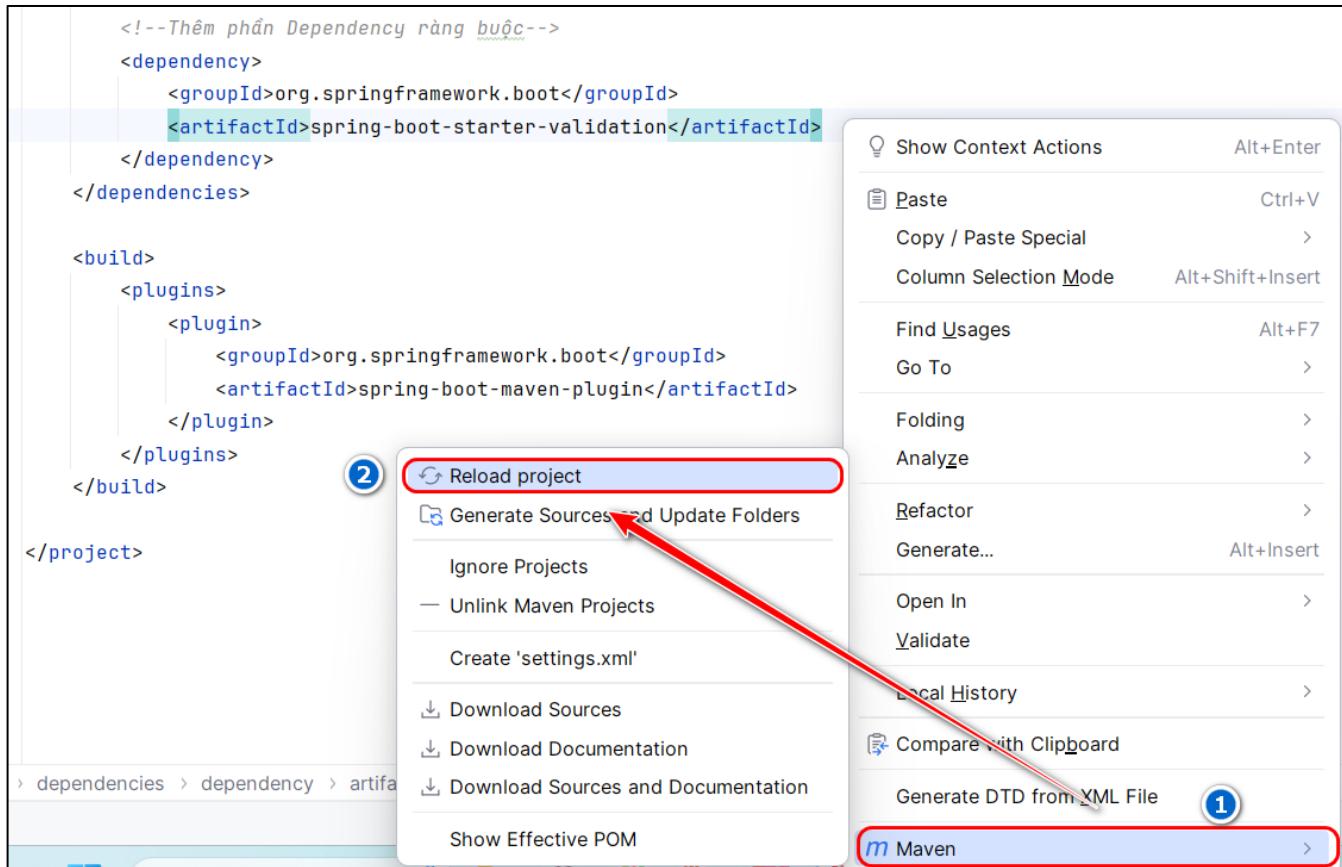
Để xây dựng một form nhập thông tin sinh viên với các ràng buộc kiểm tra trên ta có thể sử dụng Spring Boot, Thymeleaf cùng với Spring Validation. Dưới đây là các bước để thực hiện điều này:

#### Bước 1: Thêm Dependencies vào pom.xml

Thêm **Dependency Spring Boot Validation (Spring Boot Starter Validation)** và **Jakarta Validation** vào trong file **pom.xml** và trong thẻ **dependencies**.

```
<dependencies>
    <!-- Các Dependency khác-->
    <!-- Thêm phần Dependency ràng buộc-->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-validation</artifactId>
    </dependency>
    <dependency>
        <groupId>jakarta.validation</groupId>
        <artifactId>jakarta.validation-api</artifactId>
        <version>3.0.1</version>
    </dependency>
</dependencies>
```

Sau khi thêm **Dependency** ta tiến hành chuột phải chọn **Maven** → chọn **Reload project** để dự án tải và cập nhật **Spring Boot Validation** vào dự án.



## Bước 2: Tạo package model và xác định class SinhVien.java:

Tạo package **model** tại **src/main/java/com.hutech.demo**. Nhấn chuột phải vào **com.hutech.demo**, chọn **New** → **Package** → đặt tên là **model**. Trong package **model** class **SinhVien.java** với các Validation phù hợp:



## Code mẫu:

```
package com.hutech.demo.model;

import jakarta.validation.constraints.Max;
import jakarta.validation.constraints.Min;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.Pattern;

public class SinhVien {
    @NotBlank(message = "Tên là bắt buộc")
    private String ten;

    @Min(value = 18, message = "Tuổi phải lớn hơn hoặc bằng 18")
    @Max(value = 100, message = "Tuổi phải nhỏ hơn hoặc bằng 100")
    private int tuoi;

    @Pattern(regexp = "^[a-zA-Z\\s]+$", message = "Khoa phải là chữ")
    private String khoa;

    // Getters and Setters
    public String getTen() {
        return ten;
    }

    public void setTen(String ten) {
        this.ten = ten;
    }

    public int getTuoi() {
        return tuoi;
    }

    public void setTuoi(int tuoi) {
        this.tuoi = tuoi;
    }

    public String getKhoa() {
        return khoa;
    }

    public void setKhoa(String khoa) {
        this.khoa = khoa;
    }
}
```

## Bước 3: Thêm Controller

Tạo thêm **SinhVienController.java** để xử lý các yêu cầu GET và POST, bao gồm Validation.

## Code mẫu:

```

package com.hutech.demo.controller;

import com.hutech.demo.model.SinhVien;
import jakarta.validation.Valid;
import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.ui.Model;

@Controller
public class SinhVienController {

    @GetMapping("/sinhvien")
    public String showForm(Model model) {
        model.addAttribute("sinhVien", new SinhVien());
        return "form-sinhvien";
    }

    @PostMapping("/sinhvien")
    public String submitForm(@Valid SinhVien sinhVien, BindingResult bindingResult,
Model model) {
        if (bindingResult.hasErrors()) {
            return "form-sinhvien";
        }
        model.addAttribute("message", "Sinh viên đã được thêm thành công!");
        return "result-sinhvien";
    }
}

```

## Bước 4: Tạo Templates Thymeleaf

Trong package **templates** tạo thêm directory **sinhvien**. Trong directory **sinhvien** tạo thêm các trang hiển thị html: '**form-sinhvien.html**', '**result-sinhvien.html**'.

Code mẫu form nhập liệu '**form-sinhvien.html**':

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Form Sinh Viên</title>
</head>
<body>
<h1>Form Sinh Viên</h1>
<form th:action="@{/sinhvien}" th:object="${sinhVien}" method="post">
    <p>Tên: <input type="text" th:field="*{ten}" /></p>
    <p th:if="#fields.hasErrors('ten')" th:errors="*{ten}" style="color: red;"></p>
    <p>Tuổi: <input type="text" th:field="*{tuoi}" /></p>
    <p th:if="#fields.hasErrors('tuoi')" th:errors="*{tuoi}" style="color: red;"></p>
    <p>Khoa: <input type="text" th:field="*{khoa}" /></p>
    <p th:if="#fields.hasErrors('khoa')" th:errors="*{khoa}" style="color: red;"></p>

```

```
red;">></p>
<p><button type="submit">Submit</button></p>
</form>
</body>
</html>
```

Code mẫu trang hiển thị kết quả sinh viên 'result-sinhvien.html':

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Kết quả</title>
</head>
<body>
<h1 th:text="${message}"></h1>
<table border="1">
    <tr>
        <th>Tên</th>
        <th>Tuổi</th>
        <th>Khoa</th>
    </tr>
    <tr>
        <td th:text="${sinhVien.ten}">Tên</td>
        <td th:text="${sinhVien.tuoi}">Tuổi</td>
        <td th:text="${sinhVien.khoa}">Khoa</td>
    </tr>
</table>
</body>
</html>
```

## KẾT QUẢ THAM KHẢO:

### Trang nhập sinh viên

localhost:8080/sinhvien

**Form Sinh Viên**

Tên:

Tuổi:

Khoa:

### Trang hiển thị kết quả sinh viên

localhost:8080/sinhvien

**Sinh viên đã được thêm thành công!**

Tên	Tuổi	Khoa
Tran Van A	20	CNTT

## 1.5 Yêu cầu bổ sung

---

Tạo trang Layout dùng chung, sử dụng Bootstrap cho Layout và sửa lại tất cả các trang Thêm sinh viên, Hiển thị kết quả sinh viên.

### Hướng dẫn:

Để tạo một trang layout dùng chung cho các trang khác trong ứng dụng Spring Boot và Thymeleaf, có thể sử dụng cơ chế layout của Thymeleaf. Điều này sẽ duy trì nhất quán và dễ dàng quản lý các thay đổi trên toàn bộ ứng dụng.

#### Bước 1: Tạo Layout Chính

Tạo một file mới tên là **layout.html** trong đường dẫn

**src/main/resources/templates**

File này sẽ chứa cấu trúc **layout chính** cho các trang khác:

Code mẫu trang '**layout.html**':

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout">
<head>
    <meta charset="UTF-8">
    <title>Layout</title>
    <link rel="stylesheet"
          href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
</head>
<body>
<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <div class="container-fluid">
        <a class="navbar-brand" href="#">Sinh Viên</a>
        <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="/sinhvien">Nhập Sinh Viên</a>
                </li>
            </ul>
        </div>
    </div>
</nav>

<div class="container mt-4">
    <section layout:fragment="content">
        &lt;!&ndash; Content will be replaced by each specific page &ndash;&gt;
    </section>
</div>

<script
    src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js">
</script>
```

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

## Bước 2: Sử Dụng Layout Trong Các Trang Khác

Form Sinh Viên 'form-sinhvien.html':

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
      layout:decorate="~{layout}">
<head>
    <title>Form Sinh Viên</title>
</head>
<body>
<section layout:fragment="content">
    <h1>Form Sinh Viên</h1>
    <form th:action="@{/sinhvien}" th:object="${sinhVien}" method="post"
          class="needs-validation" novalidate>
        <p>Tên: <input type="text" th:field="*{ten}" /></p>
        <p th:if="#{fields.hasErrors('ten')}" th:errors="*{ten}" style="color: red;"></p>
        <p>Tuổi: <input type="text" th:field="*{tuoi}" /></p>
        <p th:if="#{fields.hasErrors('tuoi')}" th:errors="*{tuoi}" style="color: red;"></p>
        <p>Khoa: <input type="text" th:field="*{khoa}" /></p>
        <p th:if="#{fields.hasErrors('khoa')}" th:errors="*{khoa}" style="color: red;"></p>
        <p><button type="submit">Submit</button></p>
    </form>
</section>
</body>
</html>
```

Trang Kết Quả 'result-sinhvien.html':

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" xmlns:layout=""
      layout:decorate="~{layout}">
<head>
    <title th:text="${title}='Kết Quả'"></title>
</head>
<body>
<section layout:fragment="content">
    <h1 th:text="${message}"></h1>
    <table class="table table-striped">
        <thead>
            <tr>
                <th>Tên</th>
                <th>Tuổi</th>
                <th>Khoa</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td th:text="${sinhVien.ten}">Tên</td>
                <td th:text="${sinhVien.tuoi}">Tuổi</td>
            </tr>
        </tbody>
    </table>
</section>
</body>
</html>
```

```

        <td th:text="${sinhVien.khoa}">Khoa</td>
    </tr>
    </tbody>
</table>
</section>
</body>
</html>

```

### Bước 3: Cài đặt dependency Thymeleaf Layout Dialect vào pom.xml

**Lưu ý:** Đảm bảo rằng đã cài đặt **thymeleaf-layout-dialect** trong dự án của bạn để sử dụng tính năng **layout** của **Thymeleaf**:



```

demo.iml
HELP.md
mvnw
mvnw.cmd
pom.xml
External Libraries
Scratches and Consoles

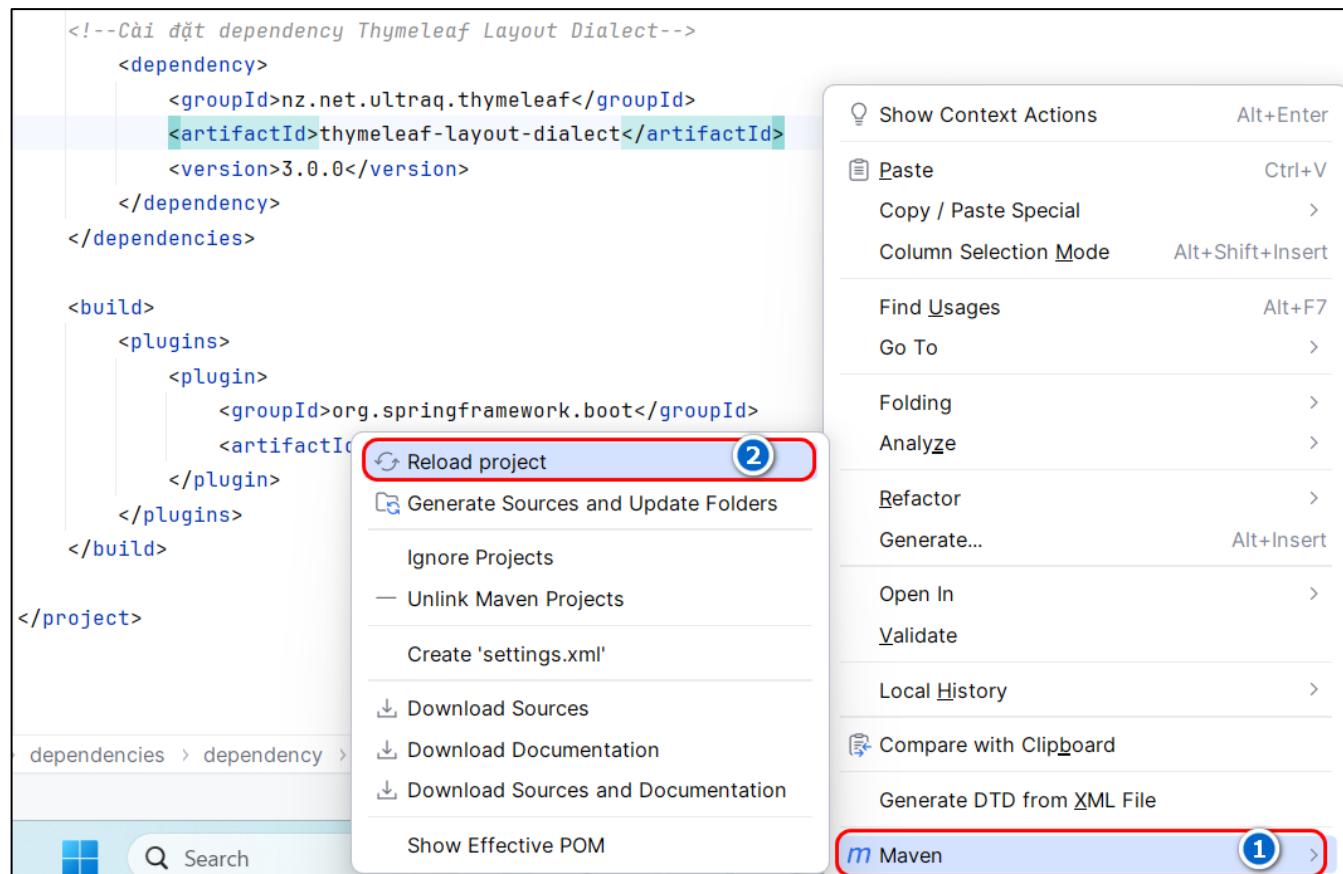
```

```

50      <!--Cài đặt dependency Thymeleaf Layout Dialect-->
51      <dependency>
52          <groupId>nz.net.ultraq.thymeleaf</groupId>
53          <artifactId>thymeleaf-layout-dialect</artifactId>
54          <version>3.0.0</version>
55      </dependency>
56  </dependencies>
57

```

Sau khi thêm **Dependency** ta tiến hành nhấp chuột phải → chọn **Maven** → chọn **Reload project** để dự án tải và cập nhật **Thymeleaf Layout Dialect** vào dự án.



## Kết quả tham khảo:

Trang nhập thông tin sinh viên:

The screenshot shows a web browser window with the URL `localhost:8080/sinhvien` in the address bar. The page has a blue header with the text "Quản lý Sinh Viên" and "Nhập Sinh Viên". The main content is titled "Form Sinh Viên". It contains three input fields: "Tên" with value "Tran Van A", "Tuổi" with value "25", and "Khoa" with value "CNTT". Below these fields is a blue "Submit" button.

Kết quả sau khi nhập xong:

The screenshot shows a web browser window with the URL `localhost:8080/sinhvien` in the address bar. The page has a blue header with the text "Quản lý Sinh Viên" and "Nhập Sinh Viên". The main content displays a success message "Sinh viên đã được thêm thành công!". Below it is a table showing the entered student data:

Tên	Tuổi	Khoa
Tran Van A	25	CNTT

# BÀI 2. TẠO CƠ SỞ DỮ LIỆU CHO WEBSITE BÁN HÀNG

Sau khi học xong bài này, sinh viên có thể:

- *Nắm vững các khái niệm cơ bản về cơ sở dữ liệu và mô hình hóa dữ liệu.*
- *Hiểu được cách thiết kế mô hình ER (Entity-Relationship) và biến đổi mô hình ER thành cấu trúc bảng trong cơ sở dữ liệu.*
- *Biết cách sử dụng hệ quản trị cơ sở dữ liệu (DBMS) để tạo và quản lý cơ sở dữ liệu.*
- *Có khả năng tạo bảng, thiết lập khóa chính và khóa ngoại, và định nghĩa các ràng buộc toàn vẹn dữ liệu.*
- *Phát triển kỹ năng phân tích yêu cầu của hệ thống và áp dụng vào thiết kế cơ sở dữ liệu.*

## 2.1 Cài đặt Laragon để quản lý cơ sở dữ liệu MySQL

---

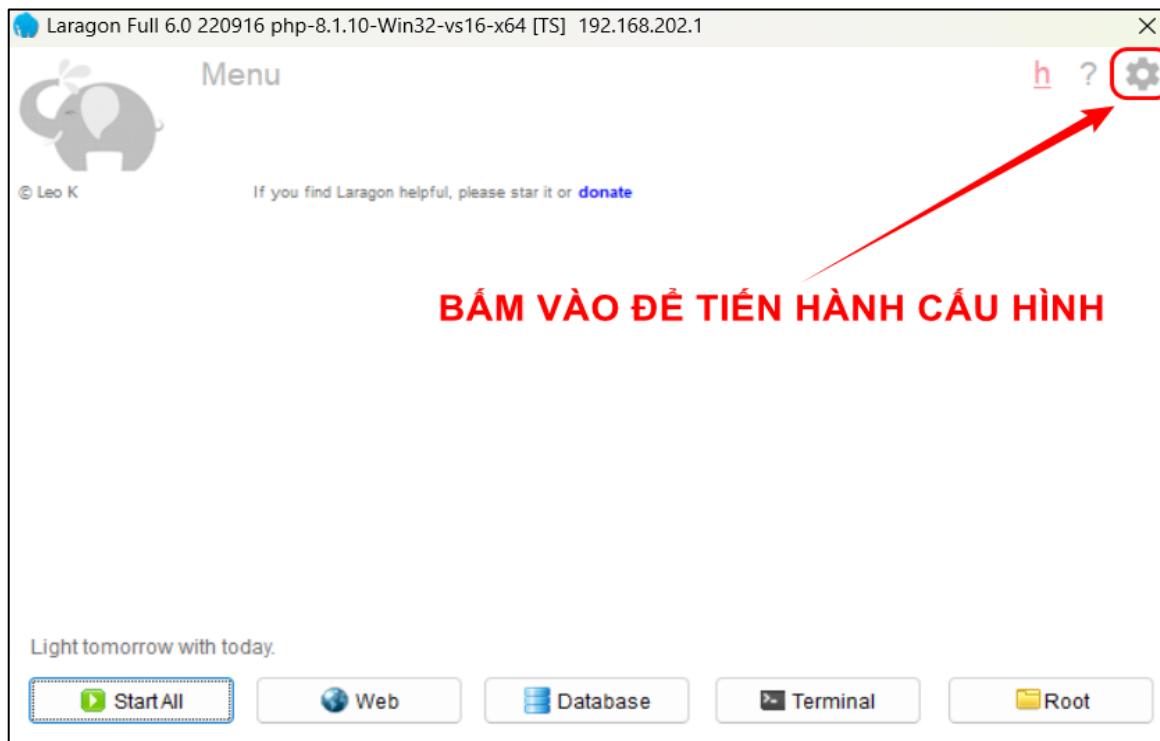
Laragon là một môi trường phát triển web dựa trên Windows, giúp người dùng dễ dàng cài đặt và quản lý các công cụ cần thiết như Apache, PHP và MySQL. Laragon cung cấp một giao diện đồ họa thân thiện và khả năng tùy chỉnh linh hoạt, cho phép người dùng dễ dàng tạo và quản lý các môi trường phát triển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

Truy cập trang website chính thức của Laragon để tải bản cài đặt về máy, sau đó tiến hành cài đặt.

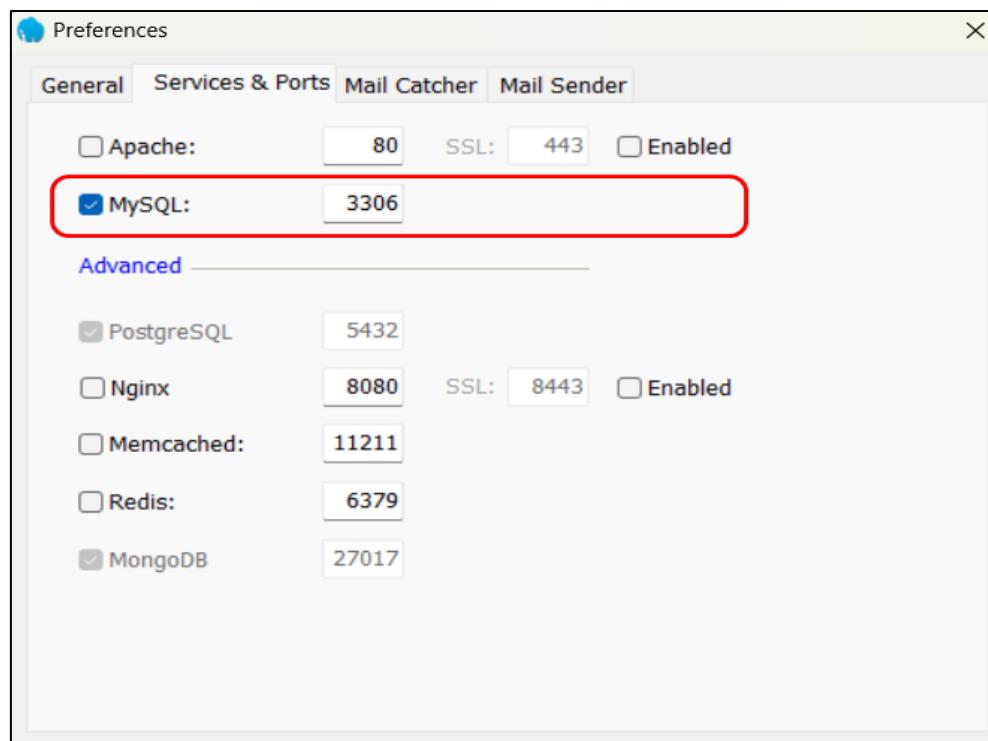
Link: <https://laragon.org/download/>

Tải xong bấm vào khởi động chương trình cài đặt phần mềm Laragon:

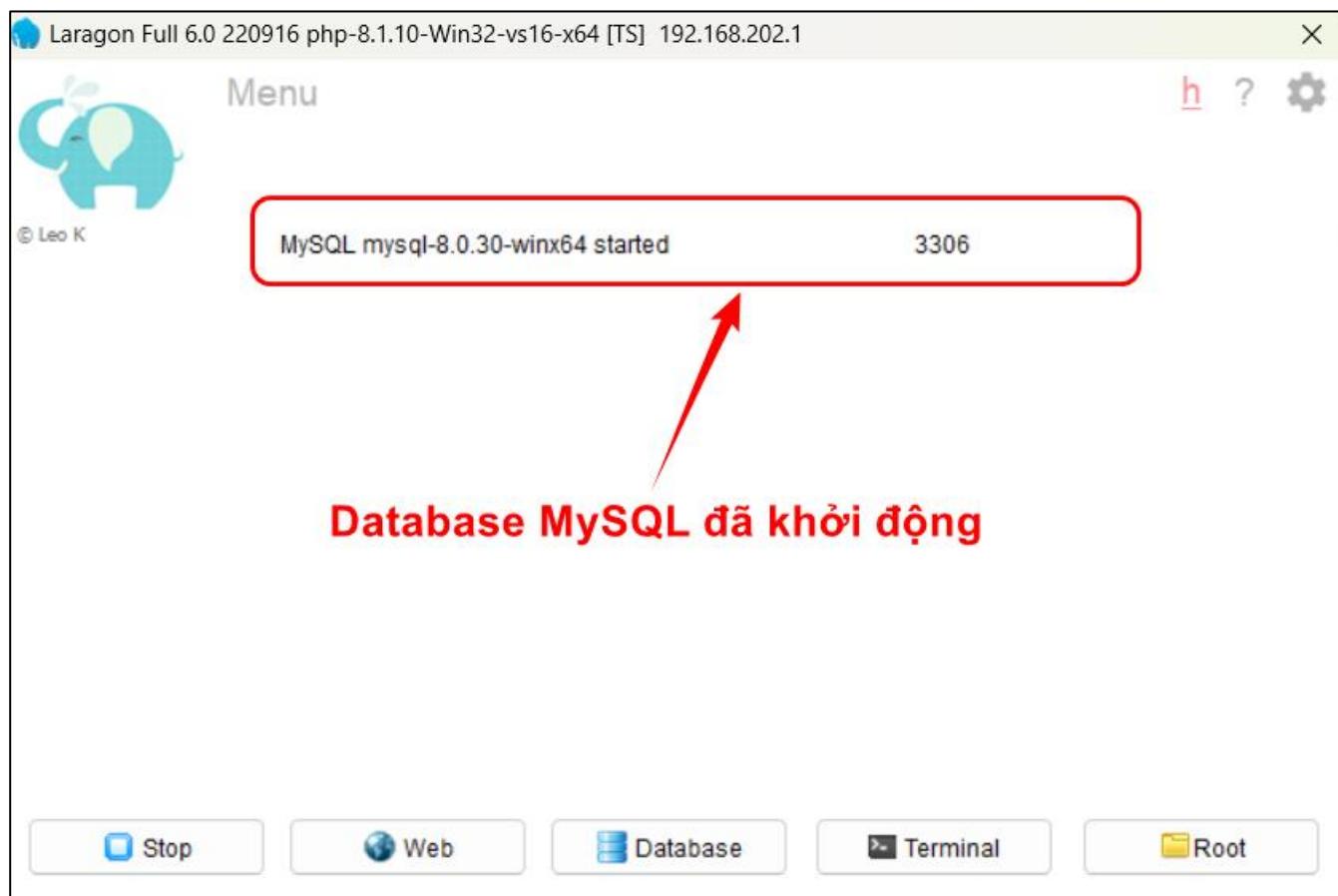
Tất cả lựa chọn cài đặt là mặc định. Sau khi cài đặt hoàn thành, tiến hành nhấp vào biểu tượng cài đặt để cấu hình:



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



Sau khi thiết lập xong ấn **Start All** để khởi động. Kết quả như hình bên dưới:



## 2.2 Thêm các Dependency

---

Để kết nối và làm việc với cơ sở dữ liệu MySQL trong một ứng dụng Spring Boot sử dụng Maven, bạn cần thêm các dependency liên quan đến Spring Data JPA và JDBC driver cho MySQL vào file pom.xml.

### 2.2.1 Dependency cho Spring Data JPA

**Spring Data JPA** giúp quản lý dữ liệu trong các ứng dụng Java qua JPA. Spring Boot cung cấp một starter kit giúp dễ dàng tích hợp và sử dụng JPA:

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

## 2.2.2 Dependency cho MySQL JDBC Driver

JDBC driver cho MySQL để Spring Boot có thể kết nối và thao tác với cơ sở dữ liệu MySQL. Sử dụng dependency sau:

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.19</version>
</dependency>
```

## 2.2.3 Dependency cho Lombok

**Lombok** là một thư viện Java giúp tự sinh ra các hàm setter/getter, hàm khởi tạo, `toString`... và tinh gọn chúng. Sử dụng dependency sau:

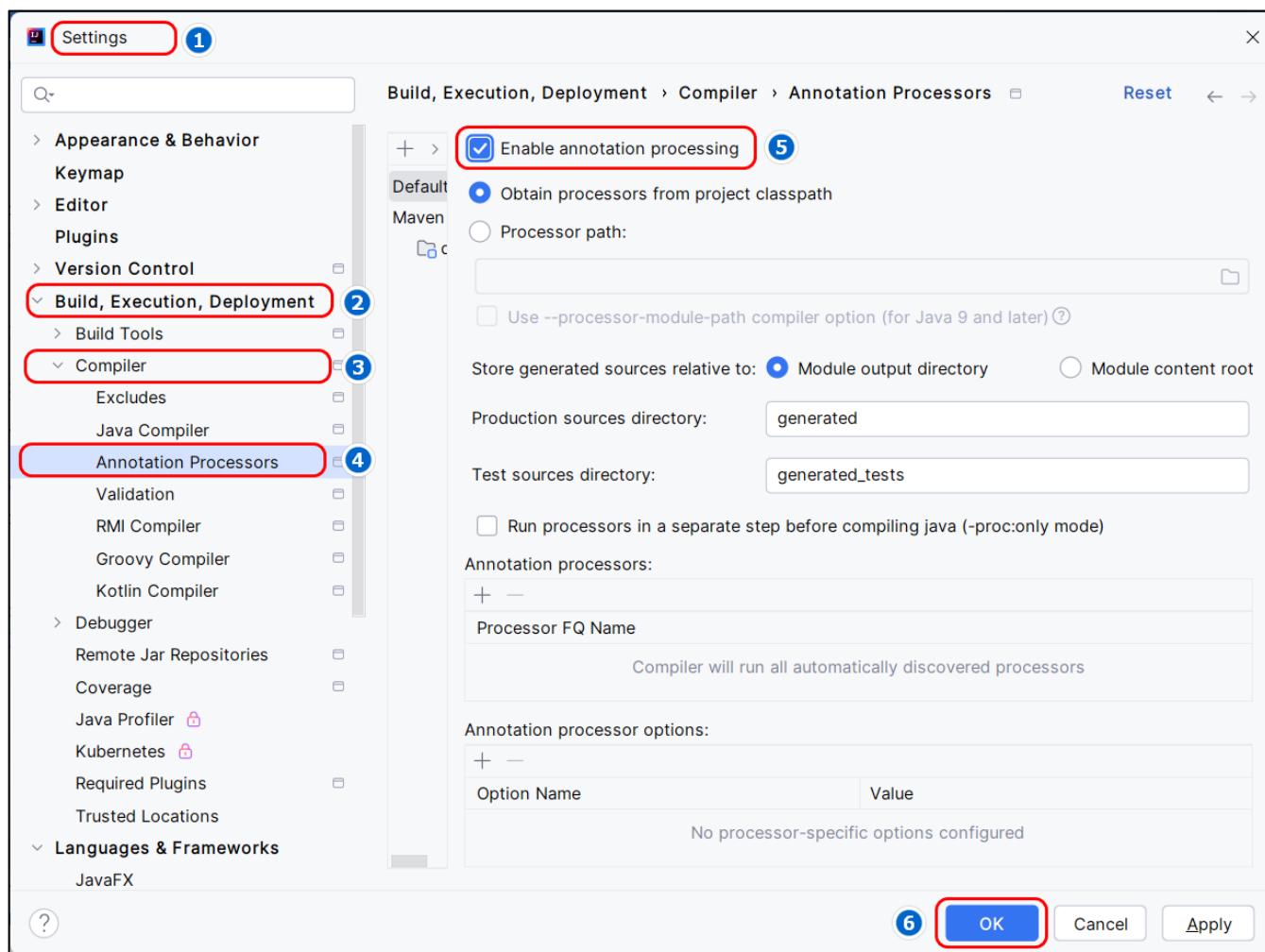
```
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>
```

### Cấu hình IDE:

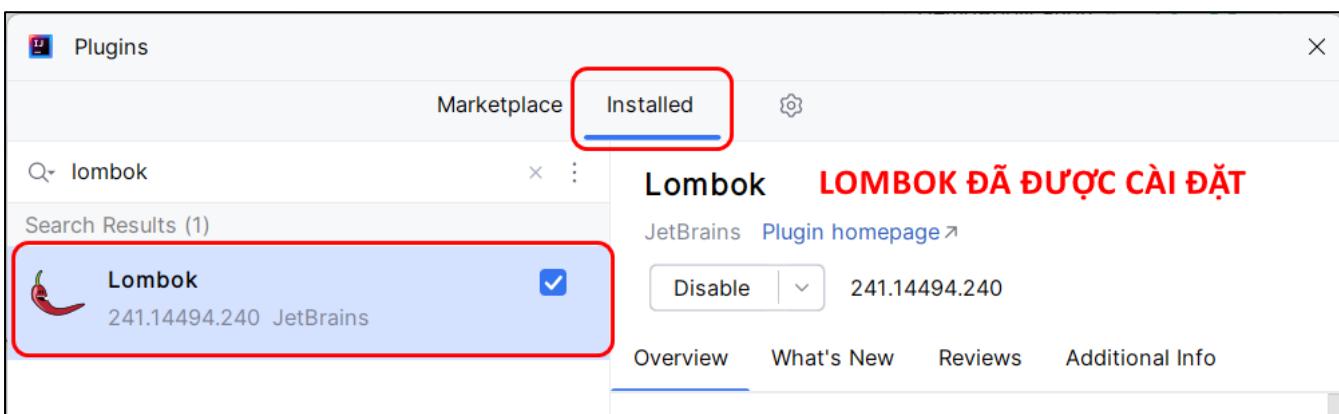
Khi sử dụng Lombok phải đảm bảo rằng đã kích hoạt xử lý annotation trong IDE.

Trong **IntelliJ IDEA**: **File → Settings → Build, Execution, Deployment → Compiler → Annotation Processors** → Tích vào "**Enable annotation processing**".

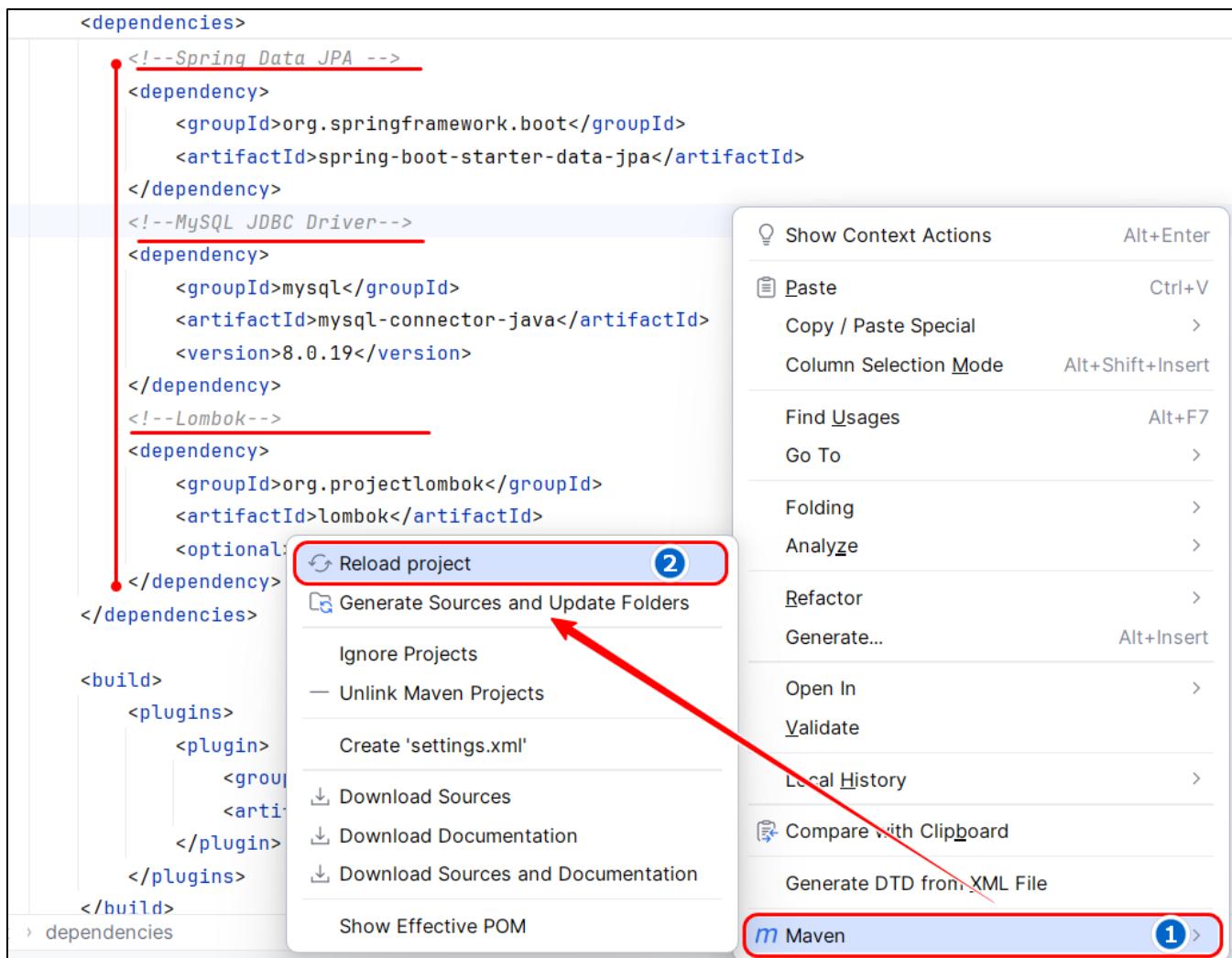
Kết quả như hình bên dưới:



Cuối cùng vào Setting → Plugin → Kiểm tra xem Plugin Lombok đã được cài đặt hay chưa, nếu chưa chọn và tiến hành cài đặt:



Kết quả sau khi đã thêm các Dependency cần thiết:



## 2.3 Tạo các entity class cho các bảng:

- Tạo một package mới trong dự án của bạn và đặt tên là `model` .
  - Trong package `model` , tạo các class `Product` , `Category` , `Order` , và `OrderDetail` .
  - Định nghĩa các thuộc tính và phương thức cho mỗi class theo yêu cầu của bảng tương ứng trong cơ sở dữ liệu.
- Ví dụ, class `Product` có thể có các thuộc tính như `id` , `name` , `price` , `description` , và `category` . Class `Category` có thể có thuộc tính như `id` và `name` .

## 2.4 Ánh xạ các entity với cơ sở dữ liệu

Trong Spring Boot để liên kết các class Java với các bảng trong cơ sở dữ liệu, ta sử dụng JPA (Java Persistence API). Mỗi class entity được ánh xạ tới một bảng thông qua các annotation sau:

**'@Entity'**: Khai báo class là một entity, làm đại diện cho một bảng trong cơ sở dữ liệu.

**'@Table'**: Xác định tên của bảng cơ sở dữ liệu mà entity này ánh xạ đến.

Ví dụ, trong class `Product`, có thể sử dụng annotation như sau:

```
@Entity
@Table(name = "products")
public class Product {

    // các thuộc tính và phương thức của class
}
```

## 2.5 Tạo các repository interfaces cho các entity

Repository là một pattern thiết kế cho phép bạn trừu tượng hóa logic truy cập dữ liệu. Trong Spring Boot, bạn sẽ tạo các '**repository**' interface cho mỗi entity:

- Tạo một package mới tên là '**repository**'.
- Trong package này, tạo các interface như '**ProductRepository**', '**CategoryRepository**', '**OrderRepository**', và '**OrderDetailRepository**'.
- Mỗi interface sẽ kế thừa từ '**JpaRepository**', cung cấp các phương thức **CRUD** sẵn có.

Ví dụ, interface `ProductRepository` có thể được định nghĩa như sau:

```
public interface ProductRepository extends JpaRepository<Product, Long> {
    // Các phương thức tùy biến...
}
```

## 2.6 Triển khai phương thức CRUD bằng JpaRepository

Kế thừa từ JpaRepository không chỉ giúp cung cấp các phương thức CRUD cơ bản mà còn giúp đảm bảo tính nhất quán và dễ dàng quản lý trong ứng dụng. Annotation @Repository chỉ ra rằng interface đó là một phần của lớp truy cập dữ liệu, cho phép Spring tự động phát hiện và cấu hình bean cho nó.

- Sử dụng annotation `@Repository` trước mỗi interface repository để đánh dấu rằng interface đó là một repository.
- Interface repository sẽ kế thừa từ `JpaRepository` và sử dụng generic type tương ứng với entity tương ứng.

Ví dụ, trong interface `ProductRepository` , bạn có thể sử dụng annotation như sau:

```
@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {
    // ...
}
```

## 2.7 Bổ sung các yêu cầu còn thiếu

Khi đã định nghĩa xong các entity và repositories cơ bản, bước tiếp theo là xác định và ánh xạ các quan hệ giữa các entity. Các quan hệ này giúp phản ánh cấu trúc và mối quan hệ của dữ liệu trong cơ sở dữ liệu.

### Xác Định Quan Hệ Giữa Các Entities

- **Many-to-One và One-to-Many:** Thường gặp trong các mô hình dữ liệu nơi một bản ghi có thể liên kết với nhiều bản ghi khác. Ví dụ, một **Order** có thể chứa nhiều **Product** và mỗi **Product** có thể thuộc về một **Category**.
  - Sử dụng **@ManyToOne** để chỉ định quan hệ nhiều đến một. Áp dụng annotation này trên thuộc tính trong class con.
  - Sử dụng **@OneToMany** để chỉ định quan hệ một đến nhiều. Áp dụng annotation này trên thuộc tính trong class chính.

## Tùy Chỉnh Repository Interfaces

- **Phương thức tùy chỉnh:** Đôi khi, các phương thức CRUD cơ bản không đủ để xử lý các yêu cầu phức tạp hoặc logic kinh doanh đặc thù của ứng dụng.
  - Có thể thêm các phương thức tùy chỉnh vào repository interfaces để thực hiện truy vấn đặc biệt hoặc xử lý các nghiệp vụ cụ thể.
  - Sử dụng **@Query** để định nghĩa các truy vấn SQL hoặc JPQL trực tiếp trong interface.

Với các bước trên đã thiết lập cơ bản và tùy biến cho các entity class và repository interfaces của mình, đảm bảo chúng được ánh xạ chính xác với cơ sở dữ liệu trong dự án Spring Boot.

## 2.8 Hướng dẫn Code mẫu

---

Code mẫu trong package model với các class: Product, Category, Order, OrderDetail

### 2.8.1 Entity class `Product`:

```
package com.hutech.demo.model;

import jakarta.persistence.*;
import lombok.*;

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

    private String name;
    private double price;
    private String description;

    @ManyToOne
    @JoinColumn(name = "category_id")
    private Category category;
}
```

## 2.8.2 Entity class `Category`:

```
package com.hutech.demo.model;

import jakarta.persistence.*;
import jakarta.validation.constraints.NotBlank;
import lombok.*;

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

    @NotBlank(message = "Tên là bắt buộc")
    private String name;
}
```

## 2.8.3 Entity class `Order`:

```
package com.hutech.demo.model;

import jakarta.persistence.*;
import lombok.*;
import java.util.List;

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

    private String customerName;

    @OneToMany(mappedBy = "order")
    private List<OrderDetail> orderDetails;
}
```

## 2.8.4 Entity class `OrderDetail`:

```
package com.hutech.demo.model;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.RequiredArgsConstructor;
import lombok.Setter;
```

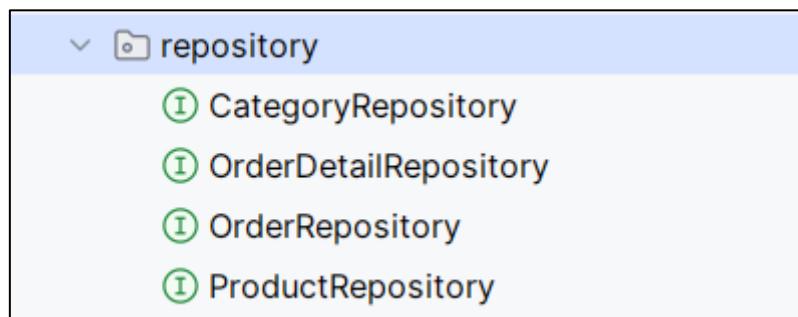
```
@Setter  
@Getter  
@RequiredArgsConstructor  
@AllArgsConstructor  
@Entity  
@Table(name = "order_details")  
public class OrderDetail {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
  
    private int quantity;  
  
    @ManyToOne  
    @JoinColumn(name = "product_id")  
    private Product product;  
  
    @ManyToOne  
    @JoinColumn(name = "order_id")  
    private Order order;  
}
```

Tạo package **repository** tại **src/main/java/com.hutech.demo**. Nhấn chuột phải vào **com.hutech.demo**, chọn **New → Package** → đặt tên là **repository**.

Tiếp theo tạo thêm các interface như sau:

`**ProductRepository**` , ` **CategoryRepository**` ,  
` **OrderRepository**` , ` **OrderDetailRepository**`

Package sau khi hoàn thành:



Sau đây là code mẫu:

### 2.8.5 Repository interface `ProductRepository` :

```
package com.hutech.demo.repository;

import com.hutech.demo.model.Product;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface ProductRepository extends JpaRepository<Product, Long> { }
```

### 2.8.6 Repository interface `CategoryRepository` :

```
package com.hutech.demo.repository;

import com.hutech.demo.model.Category;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

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

### 2.8.7 Repository interface `OrderRepository` :

```
package com.hutech.demo.repository;

import com.hutech.demo.model.Order;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface OrderRepository extends JpaRepository<Order, Long> { }
```

### 2.8.8 Repository interface `OrderDetailRepository` :

```
package com.hutech.demo.repository;

import com.hutech.demo.model.OrderDetail;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface OrderDetailRepository extends JpaRepository<OrderDetail, Long> { }
```

## 2.8.9 Cấu hình file application.properties

File **application.properties** nằm trong thư mục **resources**:



Cấu hình trong **application.properties** để kết nối database **MySQL** trong **Laragon**:

```

1  spring.application.name=demo
2
3  # Database connection properties
4  spring.datasource.url=jdbc:mysql://localhost:3306/WebBanHang
5  spring.datasource.username=root
6  spring.datasource.password=
7  spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
8
9  # Hibernate properties
10 spring.jpa.hibernate.ddl-auto=update
11 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
12
  
```

*Port ứng với MySQL trong Laragon*

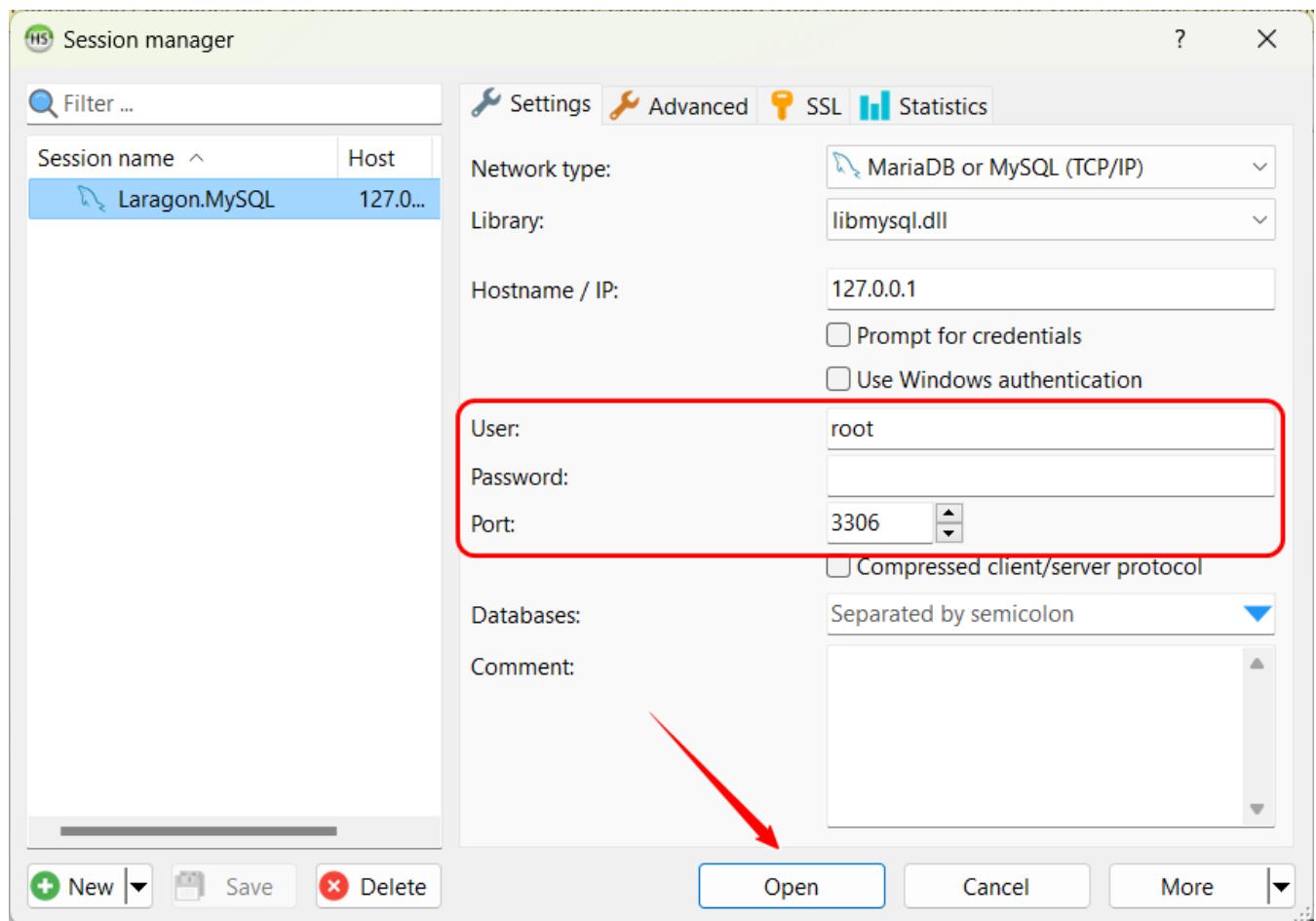
*Tên database của dự án*

**Tiến hành khởi tạo database WebBanHang trong MySQL:**

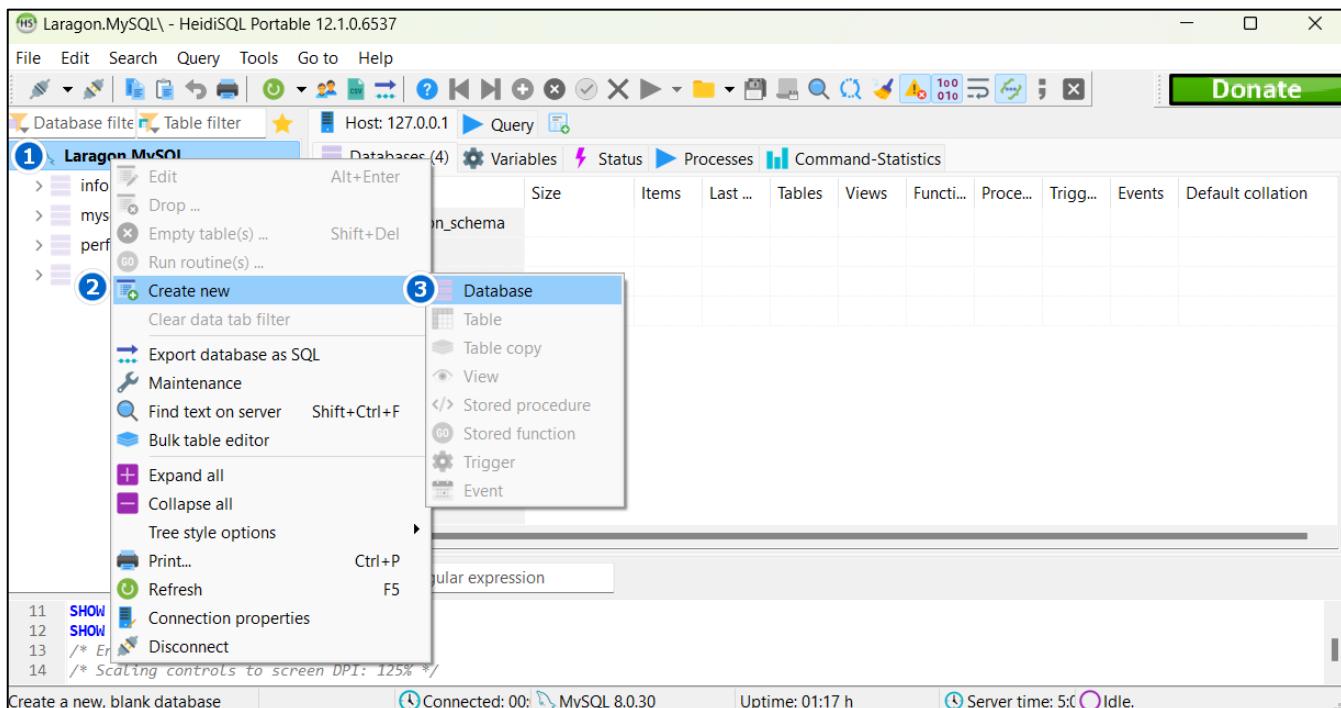
**Bước 1:** Mở MySQL trên Laragon



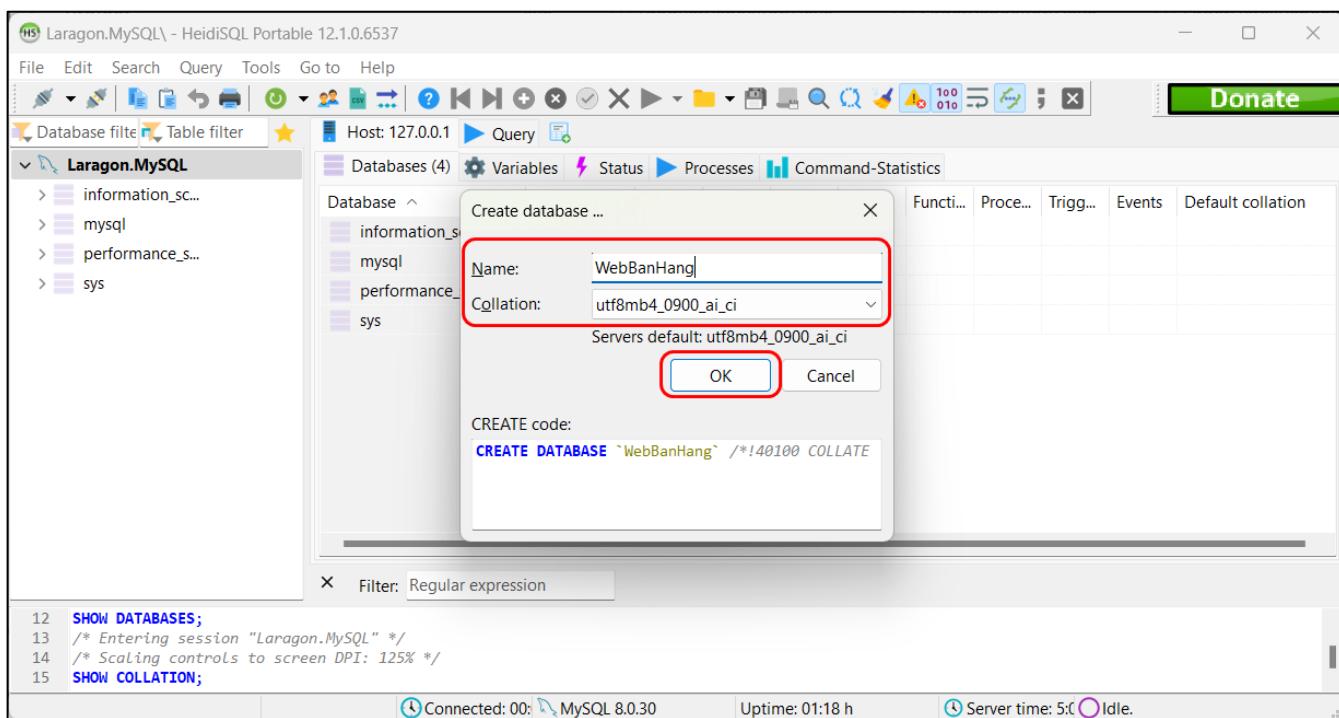
**Bước 2:** Cấu hình và khởi động MySQL. Cấu hình như sau:



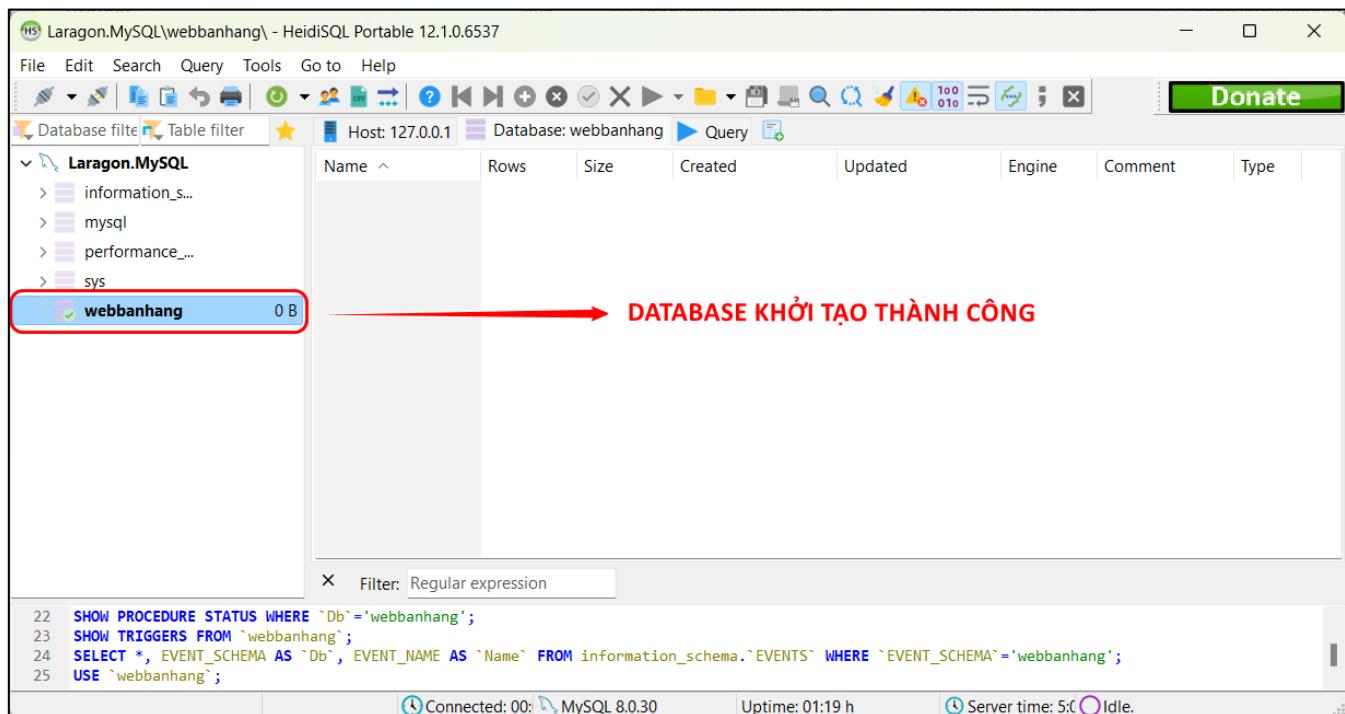
**Bước 3:** Tiến hành tạo database. Chuột phải vào **Laragon MySQL** → Chọn **Create new** → **Database**:



Ví dụ trường hợp này là **Database** tên **WebBanHang**:



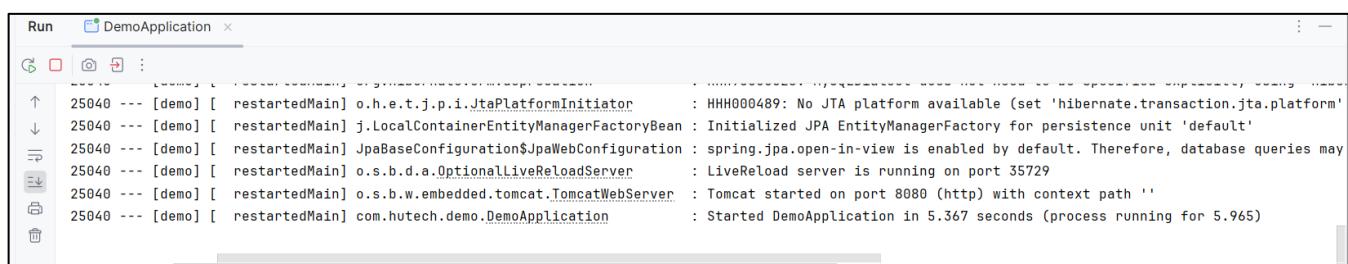
## Kết quả khởi tạo thành công database 'WebBanHang'



**Bước 4:** Tiến hành khởi tạo dự án trong IntelliJ IDEA và kiểm tra kết nối database:



Dự án Build thành công:



Kiểm tra kết nối Database MySQL trong Laragon:

The screenshot shows the HeidiSQL Portable interface connected to the 'Laragon.MySQL\webbanhang\products' database. The 'Basic' tab is selected for the 'products' table. A red box highlights the table structure in the center pane, which includes columns: id (BIGINT, Primary Key, AUTO\_INCREMENT), description (VARCHAR, 255), name (VARCHAR, 255), price (DOUBLE), and category\_id (BIGINT). The 'Indexes (2)' tab is also visible. The status bar at the bottom shows the connection details: Connected: 00 MySQL 8.0.30, Uptime: 01:36 h, Server time: 5: Idle.

#	Name	Datatype	Length/Set	Unsigned	Allow N...	Zerofill	Default
1	id	BIGINT	19	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	AUTO_INCREMENT
2	description	VARCHAR	255	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL
3	name	VARCHAR	255	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL
4	price	DOUBLE		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	No default
5	category_id	BIGINT	19	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL

# BÀI 3. XÂY DỰNG CHỨC NĂNG HIỂN THỊ/ THÊM/XÓA/SỬA

Sau khi học xong bài này, học viên có thể nắm được:

- *Hiểu rõ nguyên lý hoạt động của mô hình MVC và vai trò của Controllers trong kiến trúc này.*
- *Có kỹ năng thiết kế và cài đặt Controllers để xử lý các yêu cầu từ người dùng (HTTP requests).*
- *Học cách sử dụng annotations để định nghĩa các routes và actions trong Controller.*
- *Hiểu cách truyền dữ liệu từ Controller đến View sử dụng Model hoặc ModelAndView.*
- *Biết cách nhận và xử lý dữ liệu đầu vào từ người dùng thông qua form submissions.*
- *Phát triển kỹ năng xử lý các tình huống lỗi trong ứng dụng web.*
- *Biết cách gửi phản hồi thích hợp đến người dùng thông qua HTTP status codes và messages.*
- *Học cách tích hợp Controllers với các Services để xử lý nghiệp vụ như xác thực người dùng, quản lý giỏ hàng, và thực hiện thanh toán.*
- *Hiểu được cách sử dụng Dependency Injection để quản lý các phụ thuộc trong ứng dụng.*

## 3.1 Tạo các controller class cho các chức năng CRUD và quản lý người dùng

Để xử lý các yêu cầu đến từ người dùng, bạn cần tạo các controller class trong Spring Boot:

- **Khởi tạo Package:** Tạo một package mới trong dự án của bạn và đặt tên là controller.
- **Tạo Controller Classes:** Trong package controller, tạo các class như '**ProductController**', '**CategoryController**', và '**OrderController**'.

- Xác Định Các Phương Thức Xử Lý:** Định nghĩa các phương thức trong mỗi class để xử lý các chức năng tương ứng.

Ví dụ, 'ProductController' có thể bao gồm các phương thức như 'listProducts()', 'showProductForm()', 'saveProduct()', 'showUpdateForm()', và 'deleteProduct()' để quản lý sản phẩm.

## 3.2 Xác định các route và phương thức xử lý yêu cầu

Mỗi controller class cần các annotation để xác định cách nó xử lý các yêu cầu:

- Sử Dụng @Controller:** Áp dụng annotation này trên mỗi class để đánh dấu nó như một phần của bộ điều khiển giao diện người dùng.
- Định Nghĩa Route Với @RequestMapping:** Sử dụng annotation này trên mỗi phương thức để chỉ định đường dẫn mà phương thức đó xử lý. Nếu không chỉ định, route mặc định sẽ là /.
- Xác Định Phương Thức HTTP:** Sử dụng các annotation như @GetMapping cho các yêu cầu GET, @PostMapping cho POST, và các annotation như @ModelAttribute, @PathVariable để xử lý dữ liệu đầu vào.

Ví dụ, trong class 'ProductController', bạn có thể sử dụng annotation như sau:

```

@Controller
@RequestMapping("/products")
public class ProductController {
    private final ProductService productService;

    public ProductController(ProductService productService) {
        this.productService = productService;
    }

    @GetMapping
    public String listProducts(Model model) {
        List<Product> products = productService.getAllProducts();
        model.addAttribute("products", products);
        return "product-list";
    }

    @GetMapping("/new")
    public String showProductForm(Model model) {
        Product product = new Product();
        model.addAttribute("product", product);
        return "product-form";
    }

    @PostMapping
    public String saveProduct(@ModelAttribute("product") Product product) {
        productService.saveProduct(product);
        return "redirect:/products";
    }
}

```

```

    }
    // ... and so on
}

```

### 3.3 Gọi phương thức từ service class và truyền dữ liệu qua model

Trong mỗi phương thức xử lý yêu cầu của controller, việc tương tác với các service class là cần thiết để xử lý logic nghiệp vụ và truy xuất dữ liệu. Sau đó, dữ liệu được truyền đến view thông qua model để hiển thị cho người dùng.

#### Ví dụ về Phương Thức Xử Lý:

Trong **ProductController**, phương thức **listProducts()** là một ví dụ điển hình:

- Gọi Service Class:** Phương thức này sẽ gọi **getAllProducts()** từ class service tương ứng để lấy danh sách tất cả sản phẩm.
- Truyền Dữ Liệu Qua Model:** Danh sách sản phẩm thu được sẽ được thêm vào model, để có thể truyền dữ liệu này vào view.

### 3.4 Hướng dẫn code mẫu

#### 3.4.1 Xây dựng chức năng Hiển thị/Thêm Category

##### Bước 1: Tạo class CategoryService.java

Tạo một package **service** trong thư mục **src/main/java/com.hutech.demo**

Tiếp theo, tạo một file mới có tên **CategoryService.java** trong package **src/main/java/com.hutech.demo/service**

Trong lớp **CategoryService.java**, 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 com.hutech.demo.service;
```

```
import com.hutech.demo.model.Category;
import com.hutech.demo.repository.CategoryRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import jakarta.validation.constraints.NotNull;
import java.util.List;
import java.util.Optional;

/**
 * Service class for managing categories.
 */
@Service
@RequiredArgsConstructor
@Transactional
public class CategoryService {

    private final CategoryRepository categoryRepository;

    /**
     * Retrieve all categories from the database.
     * @return a list of categories
     */
    public List<Category> getAllCategories() {
        return categoryRepository.findAll();
    }

    /**
     * Retrieve a category by its id.
     * @param id the id of the category to retrieve
     * @return an Optional containing the found category or empty if not found
     */
    public Optional<Category> getCategoryById(Long id) {
        return categoryRepository.findById(id);
    }

    /**
     * Add a new category to the database.
     * @param category the category to add
     */
    public void addCategory(Category category) {
        categoryRepository.save(category);
    }

    /**
     * Update an existing category.
     * @param category the category with updated information
     */
    public void updateCategory(@NotNull Category category) {
        Category existingCategory = categoryRepository.findById(category.getId())
            .orElseThrow(() -> new IllegalStateException("Category with ID " +
category.getId() + " does not exist."));
        existingCategory.setName(category.getName());
        categoryRepository.save(existingCategory);
    }

    /**
     * Delete a category by its id.
     * @param id the id of the category to delete
     */
```

```

    */
    public void deleteCategoryById(Long id) {
        if (!categoryRepository.existsById(id)) {
            throw new IllegalStateException("Category with ID " + id + " does not
exist.");
        }
        categoryRepository.deleteById(id);
    }
}

```

## Bước 2: Tạo class CategoryController.java

Trong đường dẫn **src/main/java/com.hutech.demo/controller** tạo một class '**CategoryController.java**'.

```

package com.hutech.demo.controller;

import com.hutech.demo.model.Category;
import com.hutech.demo.service.CategoryService;
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;
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.PostMapping;

import java.util.List;

@Controller
@RequiredArgsConstructor
public class CategoryController {

    @Autowired
    private final CategoryService categoryService;

    @GetMapping("/categories/add")
    public String showAddForm(Model model) {
        model.addAttribute("category", new Category());
        return "/categories/add-category";
    }

    @PostMapping("/categories/add")
    public String addCategory(@Valid Category category, BindingResult result) {
        if (result.hasErrors()) {
            return "/categories/add-category";
        }
        categoryService.addCategory(category);
        return "redirect:/categories";
    }
    // Hiển thị danh sách danh mục
    @GetMapping("/categories")
    public String listCategories(Model model) {
        List<Category> categories = categoryService.getAllCategories();
        model.addAttribute("categories", categories);
        return "/categories/categories-list";
    }
}

```

```
}
```

## Bước 4: Tạo view cho category

Trong đường dẫn **src/main/resources/templates** tạo thêm directory **categories**.

Sau đó tạo thêm 2 file html: '**add-category.html**' và '**categories-list.html**'

### File '**add-category.html**:

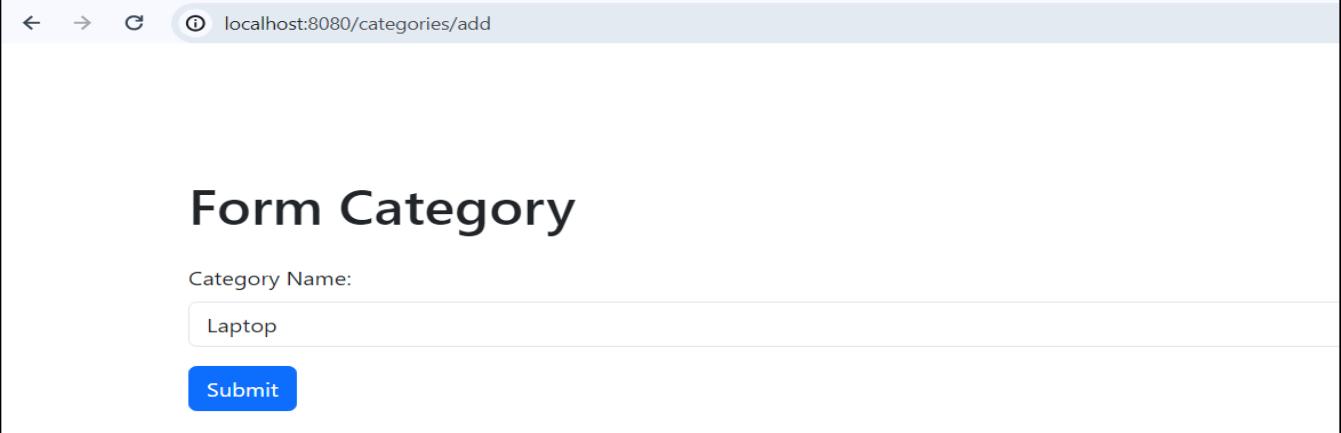
```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
layout:decorate="~{layout}">
<head>
    <title>Form Thêm Category</title>
    <link rel="stylesheet"
    href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
</head>
<body>
<section layout:fragment="content" class="container mt-5">
    <h1 class="mb-4">Form Category</h1>
    <form th:action="@{/categories/add}" th:object="${category}" method="post">
        <div class="mb-3">
            <label for="name" class="form-label">Category Name:</label>
            <input type="text" id="name" th:field="*{name}" class="form-control"/>
            <div class="text-danger" th:if="#{!#fields.hasErrors('name')}" th:errors="*{name}"></div>
        </div>
        <button type="submit" class="btn btn-primary">Submit</button>
    </form>
</section>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js">
</script>
</body>
</html>
```

### File '**categories-list.html**:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
layout:decorate="~{layout}">
<head>
    <title th:text="${title} ?: 'Kết Quả'">Kết Quả</title>
    <link rel="stylesheet"
    href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
</head>
<body>
<section layout:fragment="content">
    <h1>Category List</h1>
    <table class="table table-bordered table-hover">
```

```
<thead class="table-dark">
<tr>
    <th>ID</th>
    <th>Name</th>
</tr>
</thead>
<tbody>
<tr th:each="category : ${categories}">
    <td th:text="${category.id}"></td>
    <td th:text="${category.name}"></td>
</tr>
</tbody>
</table>
</section>
</body>
</html>
```

### Bước 5: Tiến hành build dự án và kiểm tra kết quả:



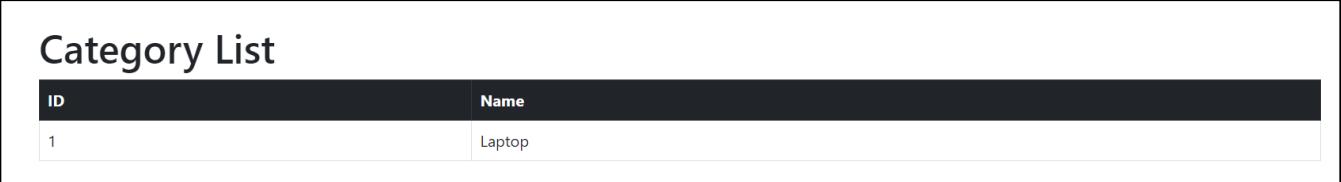
localhost:8080/categories/add

## Form Category

Category Name:

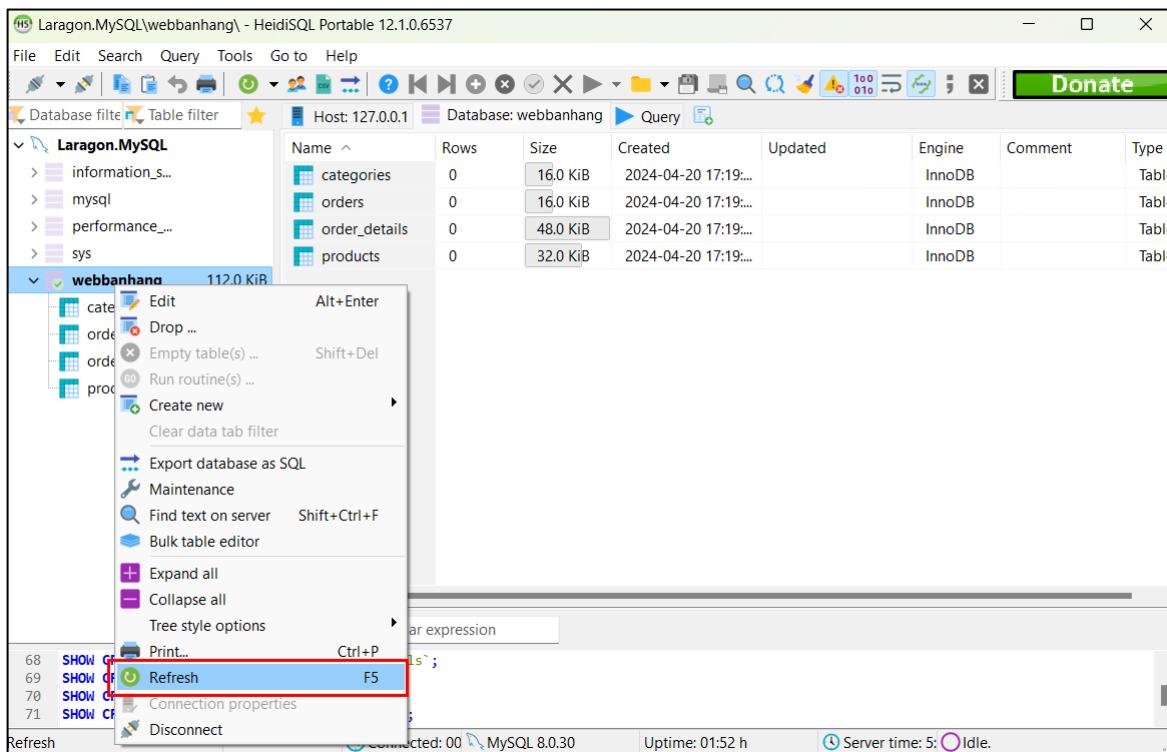
**Submit**

Thêm thành công và hiển thị danh sách category:

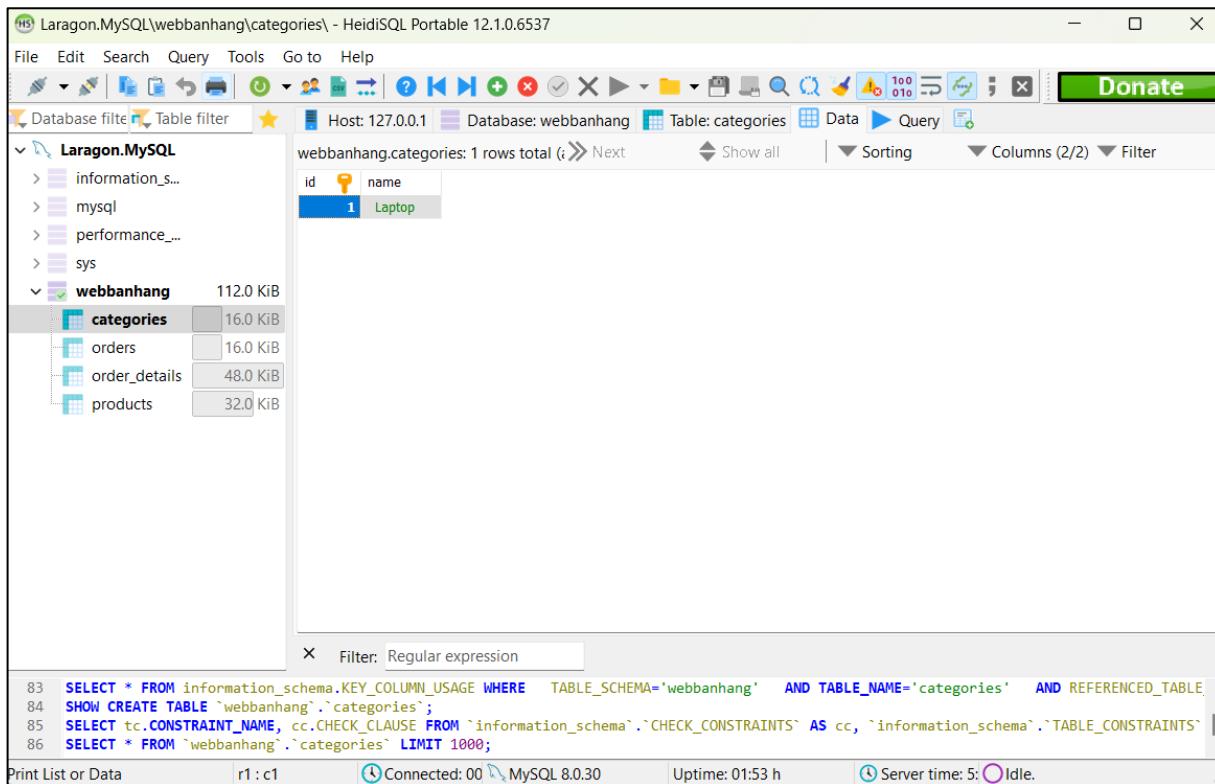


ID	Name
1	Laptop

Kiểm tra Database:



Thêm thành công category “Laptop” vào database **webbanhang**:



### 3.4.2 Chức năng xóa/ sửa cho Category

Xây dựng hoàn chỉnh chức năng SỬA/XÓA Category, thêm các view tương ứng sử dụng Bootstrap.

Code bổ sung cho **CategoryController.java**:

```
// GET request to show category edit form
@GetMapping("/categories/edit/{id}")
public String showUpdateForm(@PathVariable("id") Long id, Model model) {
    Category category = categoryService.getCategoryById(id)
        .orElseThrow(() -> new IllegalArgumentException("Invalid category Id:" +
+ id));
    model.addAttribute("category", category);
    return "/categories/update-category";
}

// POST request to update category
@PostMapping("/categories/update/{id}")
public String updateCategory(@PathVariable("id") Long id, @Valid Category category,
BindingResult result, Model model) {
    if (result.hasErrors()) {
        category.setId(id);
        return "/categories/update-category";
    }

    categoryService.updateCategory(category);
    model.addAttribute("categories", categoryService.getAllCategories());
    return "redirect:/categories";
}

// GET request for deleting category
@GetMapping("/categories/delete/{id}")
public String deleteCategory(@PathVariable("id") Long id, Model model) {
    Category category = categoryService.getCategoryById(id)
        .orElseThrow(() -> new IllegalArgumentException("Invalid category Id:" +
+ id));

    categoryService.deleteCategoryById(id);
    model.addAttribute("categories", categoryService.getAllCategories());
    return "redirect:/categories";
}
```

Cập nhật view **category-list**:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
layout:decorate="~{layout}">
<head>
    <title th:text="${title} ?: 'Categories List'"> Categories List </title>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
</head>
<body>
<section layout:fragment="content">
    <h1>Categories List</h1>
```

```

<div>
    <a th:href="@{/categories/add}" class="btn btn-primary mb-3">Add New
Category</a>
</div>
<table class="table table-bordered table-hover">
    <thead class="table-dark">
        <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Actions</th>
        </tr>
    </thead>
    <tbody>
        <tr th:each="category : ${categories}">
            <td th:text="${category.id}"></td>
            <td th:text="${category.name}"></td>
            <td>
                <a th:href="@{/categories/edit/{id} (id=${category.id})}" class="btn
btn-success btn-sm">Edit</a>
                <a th:href="@{/categories/delete/{id} (id=${category.id})}" class="btn
btn-danger btn-sm" onclick="return confirm('Are you sure?')">Delete</a>
            </td>
        </tr>
    </tbody>
</table>
</section>
</body>
</html>

```

### Code View 'update-category':

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
layout:decorate="~{layout}">
<head>
    <title>Update Category</title>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
</head>
<body>
<section layout:fragment="content" class="container mt-5">
    <h1 class="mb-4">Update Category</h1>
    <form th:action="@{/categories/update/{id} (id=${category.id})}"
th:object="${category}" method="post" class="needs-validation" novalidate>
        <div class="mb-3">
            <label for="name" class="form-label">Category Name:</label>
            <input type="text" th:field="*{name}" class="form-control" id="name"
required>
            <div class="invalid-feedback" th:if="#{fields.hasErrors('name')}">
                th:errors="*{name}">Valid name is required.</div>
        </div>
        <button type="submit" class="btn btn-primary">Update</button>
        <a th:href="@{/categories}" class="btn btn-link">Cancel</a>
    </form>
</section>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js">
</script>

```

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

**Tiến hành build dự án và kiểm tra kết quả:**

**Trang hiển thị danh sách category:**

ID	Name	Actions
1	Laptop	<button>Edit</button> <button>Delete</button>
2	PC	<button>Edit</button> <button>Delete</button>

**Chức năng Edit:**

Nhấn nút Edit và thay category PC thành Computer:

Update Category

Category Name:

Update [Cancel](#)

Trang hiển thị danh sách category sau khi đã sửa:

## Categories List

Add New Category

ID	Name	Actions
1	Laptop	<button>Edit</button> <button>Delete</button>
2	Computer	<button>Edit</button> <button>Delete</button>

PC đã sửa thành Computer

Kiểm tra Database cho thấy database đã cập nhật thành công

Database filter Table filter Host: 127.0.0.1

Laragon.MySQL

- information\_s...
- mysql
- performance\_...
- sys
- webbanhang 112.0 KiB
  - categories 16.0 KiB
  - orders 16.0 KiB
  - order\_details 48.0 KiB
  - products 32.0 KiB

webbanhang.categories:

id	name
1	Laptop
2	Computer

## 3.5 Yêu cầu bổ sung

Xây dựng chức năng **Hiển thị/Thêm/Xóa/Sửa** cho **Product**

**Bước 1: Tạo ProductService.java**

Trong đường dẫn **src/main/java/com.hutech.demo/service** xây dựng

**ProductService.java:**

```
package com.hutech.demo.service;

import com.hutech.demo.model.Product;
import com.hutech.demo.repository.ProductRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
```

```

import org.springframework.transaction.annotation.Transactional;
import jakarta.validation.constraints.NotNull;
import java.util.List;
import java.util.Optional;

@Service
@RequiredArgsConstructor
@Transactional
public class ProductService {
    private final ProductRepository productRepository;
    // Retrieve all products from the database
    public List<Product> getAllProducts() {
        return productRepository.findAll();
    }

    // Retrieve a product by its id
    public Optional<Product> getProductById(Long id) {
        return productRepository.findById(id);
    }

    // Add a new product to the database
    public Product addProduct(Product product) {
        return productRepository.save(product);
    }

    // Update an existing product
    public Product updateProduct(@NotNull Product product) {
        Product existingProduct = productRepository.findById(product.getId())
            .orElseThrow(() -> new IllegalStateException("Product with ID " +
product.getId() + " does not exist."));
        existingProduct.setName(product.getName());
        existingProduct.setPrice(product.getPrice());
        existingProduct.setDescription(product.getDescription());
        existingProduct.setCategory(product.getCategory());
        return productRepository.save(existingProduct);
    }

    // Delete a product by its id
    public void deleteProductById(Long id) {
        if (!productRepository.existsById(id)) {
            throw new IllegalStateException("Product with ID " + id + " does not
exist.");
        }
        productRepository.deleteById(id);
    }
}

```

## Bước 2: Tạo ProductController.java

Trong đường dẫn **src/main/java/com.hutech.demo/controller** tạo một class **ProductController.java**

```

package com.hutech.demo.controller;

import com.hutech.demo.model.Product;
import com.hutech.demo.service.CategoryService;
import com.hutech.demo.service.ProductService;

```

```

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.*;

import jakarta.validation.Valid;

@Controller
@RequestMapping("/products")
public class ProductController {

    @Autowired
    private ProductService productService;
    @Autowired
    private CategoryService categoryService; // Đảm bảo bạn đã inject
CategoryService
    // Display a list of all products
    @GetMapping
    public String showProductList(Model model) {
        model.addAttribute("products", productService.getAllProducts());
        return "/products/products-list";
    }

    // For adding a new product
    @GetMapping("/add")
    public String showAddForm(Model model) {
        model.addAttribute("product", new Product());
        model.addAttribute("categories", categoryService.getAllCategories()); // Load categories
        return "/products/add-product";
    }

    // Process the form for adding a new product
    @PostMapping("/add")
    public String addProduct(@Valid Product product, BindingResult result) {
        if (result.hasErrors()) {
            return "/products/add-product";
        }
        productService.addProduct(product);
        return "redirect:/products";
    }

    // For editing a product
    @GetMapping("/edit/{id}")
    public String showEditForm(@PathVariable Long id, Model model) {
        Product product = productService.getProductById(id)
            .orElseThrow(() -> new IllegalArgumentException("Invalid product Id:" + id));
        model.addAttribute("product", product);
        model.addAttribute("categories", categoryService.getAllCategories()); // Load categories
        return "/products/update-product";
    }

    // Process the form for updating a product
    @PostMapping("/update/{id}")
    public String updateProduct(@PathVariable Long id, @Valid Product product,
BindingResult result) {
        if (result.hasErrors()) {
            product.setId(id); // set id to keep it in the form in case of errors
        }
    }
}

```

```

        return "/products/update-product";
    }
    productService.updateProduct(product);
    return "redirect:/products";
}
// Handle request to delete a product
@GetMapping("/delete/{id}")
public String deleteProduct(@PathVariable Long id) {
    productService.deleteProductById(id);
    return "redirect:/products";
}
}

```

### Bước 3: Tạo view cho product

Trong đường dẫn **src/main/resources/templates** tạo thêm directory **products**. Sau đó tạo thêm 2 file html: '**add-product.html**', '**product-list.html**', '**update-product**'

File '**add-product.html**:

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
layout:decorate="~{layout}">
<head>
    <title>Add Product</title>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
</head>
<body>
<section layout:fragment="content" class="container mt-5">
    <h1>Add New Product</h1>
    <form th:action="@{/products/add}" th:object="${product}" method="post"
class="needs-validation" novalidate>
        <div class="mb-3">
            <label for="name" class="form-label">Name:</label>
            <input type="text" th:field="*{name}" class="form-control" id="name"
required>
            <div class="text-danger" th:if="#{#fields.hasErrors('name')}"
th:errors="*{name}"></div>
        </div>
        <div class="mb-3">
            <label for="price" class="form-label">Price:</label>
            <input type="text" th:field="*{price}" class="form-control" id="price"
required>
        </div>
        <div class="mb-3">
            <label for="category" class="form-label">Category:</label>
            <select th:field="*{category}" class="form-control" id="category">
                <option th:each="category : ${categories}"
th:value="${category.id}" th:text="${category.name}"></option>
            </select>
        </div>
        <div class="mb-3">
            <label for="description" class="form-label">Description:</label>
            <textarea th:field="*{description}" class="form-control"
id="description" required></textarea>
        </div>
    </form>
</section>

```

```

        </div>
        <button type="submit" class="btn btn-primary">Add Product</button>
        <a th:href="@{/products}" class="btn btn-link">Cancel</a>
    </form>
</section>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js">
</script>
</body>
</html>

```

### File 'product-list.html':

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
layout:decorate="~{layout}">
<head>
    <title th:text="${title} ?: 'Product'">Product</title>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
</head>
<body>
<section layout:fragment="content">
    <h1>Products List</h1>
    <div>
        <a th:href="@{/products/add}" class="btn btn-primary mb-3">Add New
Product</a>
    </div>
    <table class="table table-bordered table-hover">
        <thead class="table-dark">
            <tr>
                <th>ID</th>
                <th>Name</th>
                <th>Price</th>
                <th>Description</th>
                <th>Category Name</th>
                <th>Actions</th>
                <th>Add To Cart</th>
            </tr>
        </thead>
        <tbody>
            <tr th:each="product : ${products}">
                <td th:text="${product.id}"></td>
                <td th:text="${product.name}"></td>
                <td th:text="${product.price}"></td>
                <td th:text="${product.description}"></td>
                <td th:text="${product.category.name}"></td>
                <td>
                    <a th:href="@{/products/edit/{id}(id=${product.id})}" class="btn
btn-success btn-sm">Edit</a>
                    <a th:href="@{/products/delete/{id}(id=${product.id})}" class="btn
btn-danger btn-sm" onclick="return confirm('Are you sure?')">Delete</a>
                </td>
                <td>
                    <form th:action="@{/cart/add}" method="post">
                        <input type="number" name="quantity" min="1" value="1"
class="form-control d-inline-block" style="width: 70px;">
                        <input type="hidden" th:value="${product.id}"
name="productId"/>
                    </form>
                </td>
            </tr>
        </tbody>
    </table>
</section>

```

```

        <button type="submit" class="btn btn-warning btn-sm">Add to
Cart</button>
    </form>
</td>
</tr>
</tbody>
</table>
</section>
</body>
</html>

```

### File 'update-product.html':

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
layout:decorate="~{layout}">
<head>
    <title>Update Product</title>
    <link rel="stylesheet"
    href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
</head>
<body>
<section layout:fragment="content" class="container mt-5">
    <h1>Edit Product</h1>
    <form th:action="@{/products/update/{id} (id=${product.id}) }"
th:object="${product}" method="post" class="needs-validation" novalidate>
        <div class="mb-3">
            <label for="name" class="form-label">Name:</label>
            <input type="text" th:field="*{name}" class="form-control" id="name"
required>
            <div class="text-danger" th:if="#fields.hasErrors('name') "
th:errors="*{name}"></div>
        </div>
        <div class="mb-3">
            <label for="price" class="form-label">Price:</label>
            <input type="text" th:field="*{price}" class="form-control" id="price"
required>
        </div>
        <div class="mb-3">
            <label for="category" class="form-label">Category:</label>
            <select th:field="*{category}" class="form-control" id="category">
                <option th:each="category : ${categories}"
th:value="${category.id}" th:text="${category.name}" th:selected="${category.id ==
product.category.id}"></option>
            </select>
        </div>
        <div class="mb-3">
            <label for="description" class="form-label">Description:</label>
            <textarea th:field="*{description}" class="form-control"
id="description" required></textarea>
        </div>
        <button type="submit" class="btn btn-primary">Save Changes</button>
        <a th:href="@{/products}" class="btn btn-link">Cancel</a>
    </form>
</section>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js">
</script>

```

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

#### Bước 4: Tiến hành build dự án và kiểm tra kết quả:

Truy cập đường dẫn <http://localhost:8080/products/add> và tiến hành thêm một sản phẩm:

Add New Product

Name: Laptop Hutech 1

Price: 2000

Category: Laptop

Description: Laptop Hutech 1

Add Product Cancel

Trang hiển thị danh sách sản phẩm sau khi đã thêm thành công:

ID	Name	Price	Description	Category Name	Actions
1	Laptop Hutech 1	2000.0	Laptop Hutech 1	Laptop	<a href="#">Edit</a> <a href="#">Delete</a>

Chức năng **Edit:**

## Edit Product

Name:

Price:

Category:

Description:

Save Changes[Cancel](#)

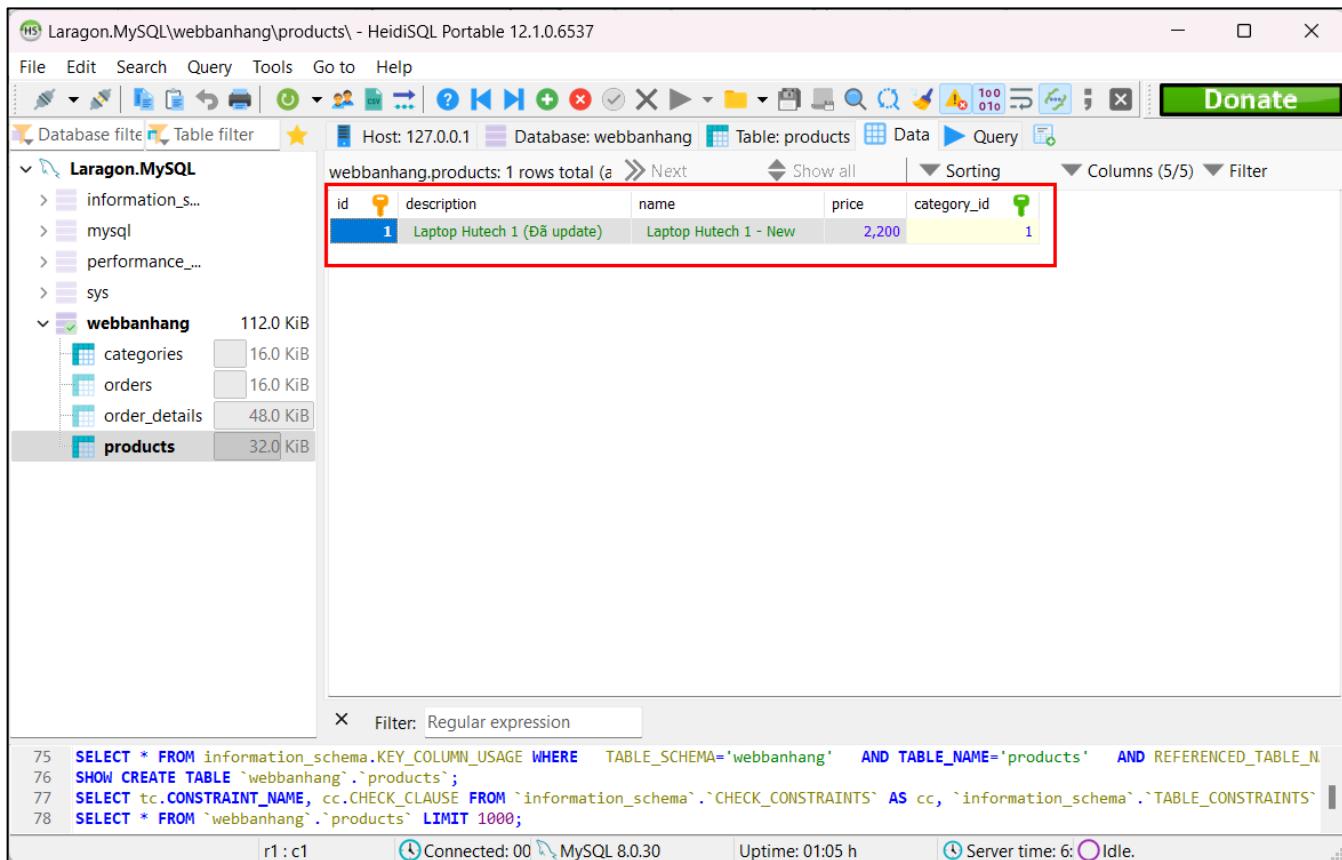
Hiển thị danh sách sản phẩm sau khi đã sửa:

## Products List

[Add New Product](#)

ID	Name	Price	Description	Category Name	Actions
1	Laptop Hutech 1 - New	2200.0	Laptop Hutech 1 (Đã update)	Laptop	<a href="#">Edit</a> <a href="#">Delete</a>

Kiểm tra Database cho thấy kết quả sản phẩm đã tự động lưu, cập nhật vào trong Database:



The screenshot shows the HeidiSQL Portable interface connected to the Laragon MySQL database. The left sidebar shows the database structure with the 'webbanhang' database selected. The main pane displays the 'products' table, which contains one row. The row is highlighted with a red box. The columns are labeled: id, description, name, price, and category\_id. The data for the highlighted row is: id=1, description='Laptop Hutech 1 (Đã update)', name='Laptop Hutech 1 - New', price=2,200, category\_id=1. Below the table, the SQL query used to select the data is shown in the status bar.

id	description	name	price	category_id
1	Laptop Hutech 1 (Đã update)	Laptop Hutech 1 - New	2,200	1

```
75 SELECT * FROM information_schema.KEY_COLUMN_USAGE WHERE TABLE_SCHEMA='webbanhang' AND TABLE_NAME='products' AND REFERENCED_TABLE_N
76 SHOW CREATE TABLE `webbanhang`.`products`;
77 SELECT tc.CONSTRAINT_NAME, cc.CHECK_CLAUSE FROM `information_schema`.`CHECK_CONSTRAINTS` AS cc, `information_schema`.`TABLE_CONSTRAINTS` tc
78 SELECT * FROM `webbanhang`.`products` LIMIT 1000;
```

# BÀI 4. XÂY DỰNG CHỨC NĂNG GIỎ HÀNG VÀ ĐẶT HÀNG

Xây dựng chức năng giỏ hàng và thanh toán bằng cách sử dụng Session.

Để xây dựng chức năng giỏ hàng sử dụng session trong một ứng dụng web Spring Boot, ta lưu trữ thông tin giỏ hàng trong session người dùng. Điều này giúp dữ liệu giỏ hàng được duy trì suốt phiên làm việc của người dùng mà không cần lưu vào cơ sở dữ liệu ngay lập tức. Chức năng giỏ hàng được xây dựng dựa trên session cho phép giữ dữ liệu giỏ hàng qua nhiều yêu cầu mà không cần lưu trữ ngay lập tức vào cơ sở dữ liệu.

Dưới đây là một hướng dẫn từng bước để xây dựng chức năng giỏ hàng sử dụng session.

## 4.1 Xây dựng chức năng Giỏ hàng

### 4.1.1 Thiết kế mô hình dữ liệu – tạo class 'CartItem'

Dự án không cần tạo mô hình dữ liệu mới trong cơ sở dữ liệu vì giỏ hàng sẽ được lưu trữ trong session. Tuy nhiên, chúng ta cần một class '**CartItem**' để đại diện cho các mặt hàng trong giỏ hàng.

- Mục đích:** Thay vì tạo mô hình dữ liệu mới trong cơ sở dữ liệu, chúng ta sẽ lưu trữ giỏ hàng tạm thời trong session. Để quản lý các mặt hàng trong giỏ hàng, cần có một class **CartItem**.
- Thực hiện:** Trong package **model** tại đường dẫn **src/main/java/com.hutech.demo/model**, tạo lớp **CartItem.java** với hai thuộc tính là sản phẩm (**Product**) và số lượng (**quantity**). Lớp này cũng cần các phương thức getter và setter để truy cập và cập nhật thuộc tính.

```
package com.hutech.demo.model;

public class CartItem {
    private Product product;
    private int quantity;

    // Constructors
    public CartItem(Product product, int quantity) {
        this.product = product;
        this.quantity = quantity;
    }
}
```

```
// Getters and Setters
public Product getProduct() {
    return product;
}

public void setProduct(Product product) {
    this.product = product;
}

public int getQuantity() {
    return quantity;
}

public void setQuantity(int quantity) {
    this.quantity = quantity;
}
```

## 4.1.2 Xây Dựng CartService.java

Để quản lý các hoạt động liên quan đến giỏ hàng như thêm sản phẩm, xóa sản phẩm khỏi giỏ hàng, và xóa sạch giỏ hàng.

Tạo **CartService.java** trong package **service** trong đường dẫn **src/main/java/com.hutech.demo/service** với các phương thức để thêm sản phẩm vào giỏ hàng, lấy tất cả sản phẩm trong giỏ, xóa sản phẩm khỏi giỏ và xóa sạch giỏ hàng. Dịch vụ này được định nghĩa với phạm vi session để duy trì thông tin giỏ hàng trong suốt phiên làm việc của người dùng.

```
package com.hutech.demo.service;

import com.hutech.demo.model.CartItem;
import com.hutech.demo.model.Product;
import com.hutech.demo.repository.ProductRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.web.context.annotation.SessionScope;

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

@Service
@SessionScope
public class CartService {
    private List<CartItem> cartItems = new ArrayList<>();

    @Autowired
    private ProductRepository productRepository;

    public void addToCart(Long productId, int quantity) {
        Product product = productRepository.findById(productId)
```

```

        .orElseThrow(() -> new IllegalArgumentException("Product not found:
" + productId));
        cartItems.add(new CartItem(product, quantity));
    }

    public List<CartItem> getCartItems() {
        return cartItems;
    }

    public void removeFromCart(Long productId) {
        cartItems.removeIf(item -> item.getProduct().getId().equals(productId));
    }

    public void clearCart() {
        cartItems.clear();
    }
}

```

### 4.1.3 Xây Dựng CartController.java

- Mục đích:** Xử lý các yêu cầu liên quan đến giỏ hàng từ người dùng.
- Thực hiện:** Trong **CartController.java**, cung cấp các đường dẫn để xem giỏ hàng, thêm sản phẩm vào giỏ hàng, loại bỏ sản phẩm từ giỏ hàng, và xóa sạch giỏ hàng. Mỗi hành động này sẽ tương ứng với một phương thức trong controller.

Tạo một file mới có tên '**CartController.java**' trong đường dẫn

**src/main/java/com.hutech.demo/controller**

```

package com.hutech.demo.controller;

import com.hutech.demo.service.CartService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

@Controller
@RequestMapping("/cart")
public class CartController {

    @Autowired
    private CartService cartService;

    @GetMapping
    public String showCart(Model model) {
        model.addAttribute("cartItems", cartService.getCartItems());
        return "/cart/cart";
    }

    @PostMapping("/add")
    public String addToCart(@RequestParam Long productId, @RequestParam int
quantity) {
        cartService.addCart(productId, quantity);
    }
}

```

```

        return "redirect:/cart";
    }

    @GetMapping("/remove/{productId}")
    public String removeFromCart(@PathVariable Long productId) {
        cartService.removeFromCart(productId);
        return "redirect:/cart";
    }

    @GetMapping("/clear")
    public String clearCart() {
        cartService.clearCart();
        return "redirect:/cart";
    }
}

```

#### 4.1.4 Tạo View cho giỏ hàng

Hiển thị giỏ hàng cho người dùng với các tùy chọn thay đổi số lượng sản phẩm hoặc xóa sản phẩm khỏi giỏ hàng.

Trong đường dẫn **src/main/resources/templates** tạo thêm directory **cart**. Sau đó tạo thêm file html: '**cart.html**' để hiển thị các sản phẩm trong giỏ hàng. Trang này bao gồm một bảng với các sản phẩm được liệt kê, mỗi sản phẩm có tùy chọn để thay đổi số lượng hoặc xóa khỏi giỏ.

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
      layout:decorate="~{layout}">
<head>
    <title th:text="${title} ?: 'Your Cart'">Your Cart</title>
    <link rel="stylesheet"
          href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
</head>
<body>
<section layout:fragment="content" class="container mt-3">
    <h1>Your Cart</h1>
    <div th:if="${cartItems.isEmpty()}" class="alert alert-info">Your cart is empty.</div>
    <table class="table" th:unless="${cartItems.isEmpty()}">
        <thead class="table-light">
            <tr>
                <th>Product Name</th>
                <th>Quantity</th>
                <th>Action</th>
            </tr>
        </thead>
        <tbody>
            <tr th:each="item : ${cartItems}">
                <td th:text="${item.product.name}"></td>
                <td th:text="${item.quantity}"></td>
                <td>

```

```

        <a
th:href="@{/cart/remove/{productId} (productId=${item.product.id})}" class="btn btn-
danger btn-sm">Remove</a>
    </td>
</tr>
</tbody>
</table>
<div class="mt-3">
    <a th:href="@{/cart/clear}" class="btn btn-secondary">Clear Cart</a>
    |
    <a th:href="@{/order/checkout}" class="btn btn-primary mb-3">Check Out</a>
</div>
</section>
</body>
</html>

```

Cập nhật 'products-list.html' trong directory **products** để tạo nút thêm giỏ hàng

### "Add to Cart":

```

<h1>Products List</h1>
<div>
    <a th:href="@{/products/add}" class="btn btn-primary mb-3">Add New Product</a>
</div>
<table class="table table-bordered table-hover">
    <thead class="table-dark">
        <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Price</th>
            <th>Description</th>
            <th>Category Name</th>
            <th>Actions</th>
            <th>Add To Cart</th>
        </tr>
    </thead>
    <tbody>
        <tr th:each="product : ${products}">
            <td th:text="${product.id}"></td>
            <td th:text="${product.name}"></td>
            <td th:text="${product.price}"></td>
            <td th:text="${product.description}"></td>
            <td th:text="${product.category.name}"></td>
            <td>
                <a th:href="@{/products/edit/{id} (id=${product.id})}" class="btn btn-
success btn-sm">Edit</a>
                <a th:href="@{/products/delete/{id} (id=${product.id})}" class="btn btn-
danger btn-sm" onclick="return confirm('Are you sure?')">Delete</a>
            </td>
            <td>
                <form th:action="@{/cart/add}" method="post">
                    <input type="number" name="quantity" min="1" value="1" class="form-
control d-inline-block" style="width: 70px;">
                    <input type="hidden" th:value="${product.id}" name="productId"/>
                    <button type="submit" class="btn btn-warning btn-sm">Add to
Cart</button>
                </form>
            </td>
        </tr>
    </tbody>
</table>

```

```
</tbody>
</table>
```

## 4.2 Xây dựng chức năng Đặt hàng

Để xây dựng chức năng đặt hàng trong một ứng dụng Spring Boot, ta cần lưu trữ thông tin về đơn hàng (Order) và các chi tiết đơn hàng (OrderDetail) dựa vào thông tin trong giỏ hàng. Dưới đây là hướng dẫn từng bước để triển khai chức năng này, bao gồm xử lý back-end và cập nhật mô hình dữ liệu:

### 4.2.1 Tạo OrderService.java

Dịch vụ này sẽ xử lý logic để tạo mới đơn hàng từ giỏ hàng. Trong package **service** tạo '**OrderService.java**'

```
package com.hutech.demo.service;

import com.hutech.demo.model.CartItem;
import com.hutech.demo.model.Order;
import com.hutech.demo.model.OrderDetail;
import com.hutech.demo.repository.OrderDetailRepository;
import com.hutech.demo.repository.OrderRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;

@Service
@RequiredArgsConstructor
@Transactional
public class OrderService {
    @Autowired
    private OrderRepository orderRepository;
    @Autowired
    private OrderDetailRepository orderDetailRepository;
    @Autowired
    private CartService cartService; // Assuming you have a CartService

    @Transactional
    public Order createOrder(String customerName, List<CartItem> cartItems) {
        Order order = new Order();
        order.setCustomerName(customerName);
        order = orderRepository.save(order);

        for (CartItem item : cartItems) {
            OrderDetail detail = new OrderDetail();
            detail.setOrder(order);
            detail.setProduct(item.getProduct());
            detail.setQuantity(item.getQuantity());
            orderDetailRepository.save(detail);
        }
    }
}
```

```
// Optionally clear the cart after order placement
cartService.clearCart();

return order;
}
```

## 4.2.2 Tạo OrderController.java

Controller này sẽ xử lý các yêu cầu HTTP để đặt hàng.

```
package com.hutech.demo.controller;

import com.hutech.demo.model.CartItem;
import com.hutech.demo.model.Order;
import com.hutech.demo.model.Product;
import com.hutech.demo.service.CartService;
import com.hutech.demo.service.OrderService;
import org.springframework.ui.Model;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;

import java.util.List;

@Controller
@RequestMapping("/order")
public class OrderController {
    @Autowired
    private OrderService orderService;
    @Autowired
    private CartService cartService;

    @GetMapping("/checkout")
    public String checkout() {
        return "/cart/checkout";
    }
    @PostMapping("/submit")
    public String submitOrder(String customerName) {
        List<CartItem> cartItems = cartService.getCartItems();
        if (cartItems.isEmpty()) {
            return "redirect:/cart"; // Redirect if cart is empty
        }
        orderService.createOrder(customerName, cartItems);
        return "redirect:/order/confirmation";
    }

    @GetMapping("/confirmation")
    public String orderConfirmation(Model model) {
        model.addAttribute("message", "Your order has been successfully placed.");
        return "cart/order-confirmation";
    }
}
```

### 4.2.3 Tạo View cho phần đặt hàng

Trong đường dẫn **src/main/resources/templates/cart** tạo thêm 2 file html: '**checkout.html**', '**order-confirmation.html**'.

Form để người dùng nhập tên và gửi thông tin đơn hàng. File '**checkout.html**':

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
      layout:decorate="~{layout}">
<head>
    <title th:text="${title} ?: 'Your Cart'">Place Order</title>
    <link rel="stylesheet"
          href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
</head>
<body>
<section layout:fragment="content" class="container mt-3">
    <h1>Place Your Order</h1>
    <form th:action="@{/order/submit}" method="post">
        <div class="mb-3">
            <label for="customerName" class="form-label">Your Name:</label>
            <input type="text" id="customerName" name="customerName" class="form-control" required>
        </div>
        <button type="submit" class="btn btn-primary">Submit Order</button>
    </form>
</section>
</body>
</html>
```

Phần xác nhận hoàn tất đặt hàng, file '**order-confirmation.html**':

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
      layout:decorate="~{layout}">
<head>
    <title th:text="${title} ?: 'Your Cart'">Order Confirmation</title>
    <link rel="stylesheet"
          href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
</head>
<body>
<section layout:fragment="content" class="container mt-3">
    <h1>Order Confirmation</h1>
    <p th:text="${message}"></p>
</section>
</body>
</html>
```

#### 4.2.4 Tiến hành build lại dự án và kiểm tra kết quả:

Trang danh sách sản phẩm đã thêm nút **Add to Cart**:

Products List												
ID	Name	Price	Description	Category Name	Actions	Add To Cart						
1	Laptop Hutech 1 - New	2200.0	Laptop Hutech 1 (Đã update)	Laptop	<button>Edit</button> <button>Delete</button>	<table border="1"> <tr> <td>1</td> <td><button>Add to Cart</button></td> </tr> <tr> <td>2</td> <td><button>Add to Cart</button></td> </tr> <tr> <td>3</td> <td><button>Add to Cart</button></td> </tr> </table>	1	<button>Add to Cart</button>	2	<button>Add to Cart</button>	3	<button>Add to Cart</button>
1	<button>Add to Cart</button>											
2	<button>Add to Cart</button>											
3	<button>Add to Cart</button>											
3	PC Hutech 1	1500.0	PC Hutech 1	Computer	<button>Edit</button> <button>Delete</button>							
4	Hphone 24	3000.0	Điện thoại Hphone 24	Phone	<button>Edit</button> <button>Delete</button>							

Chức năng Giỏ hàng: <http://localhost:8080/cart>

Your Cart		
Product Name	Quantity	Action
Laptop Hutech 1 - New	1	<button>Remove</button>
PC Hutech 1	2	<button>Remove</button>
Hphone 24	3	<button>Remove</button>

Clear Cart

Check Out

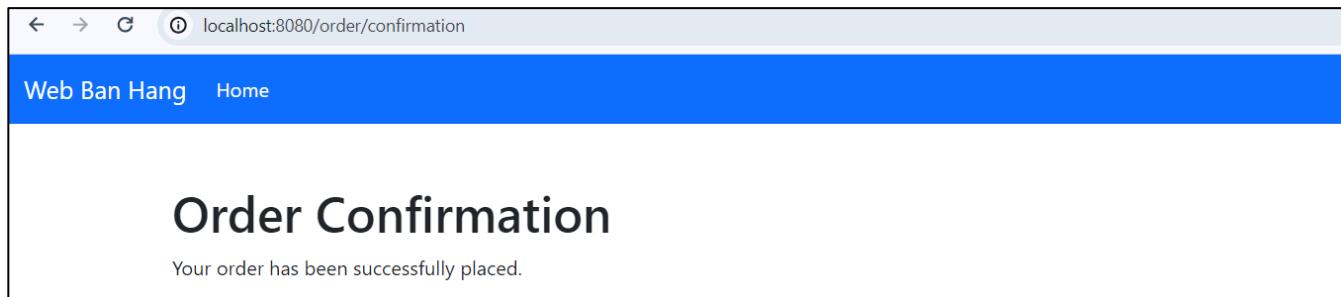
Chức năng Check out: <http://localhost:8080/order/checkout>

### Place Your Order

Your Name:

Submit Order

Thông báo đặt hàng thành công:



**Kiểm tra thông tin tại database:**

**Thông tin tại bảng order\_detail**

webbanhang.order_details: 5 rows tot >> Next				
id	quantity	order_id	product_id	
1	3	1	1	
2	3	1	3	
3	1	2	1	
4	2	2	3	
5	3	2	4	

**Thông tin tại bảng order**

webbanhang.orders: 2 rows total (appr)	
id	customer_name
1	Nguyen Van A
2	Tran Van A

→ Hoàn tất chức năng giỏ hàng và đặt hàng

### 4.3 Yêu cầu bổ sung

Thêm các thông tin bổ sung cho đơn đặt hàng như địa chỉ giao hàng, số điện thoại, email, ghi chú, phương thức thanh toán.

# BÀI 5. XÂY DỰNG CÁC CHỨC NĂNG XÁC THỰC VÀ PHÂN QUYỀN NGƯỜI DÙNG

Sau khi học xong bài này, sinh viên có thể nắm được:

- *Nắm vững các khái niệm cơ bản về xác thực người dùng (authentication) và phân quyền (authorization).*
- *Hiểu rõ sự khác biệt giữa xác thực người dùng và phân quyền người dùng, cũng như tầm quan trọng của việc bảo mật thông tin người dùng.*
- *Biết cách thiết kế và triển khai hệ thống xác thực người dùng, từ việc đăng nhập, đăng ký đến quên mật khẩu.*
- *Có kỹ năng lập trình để xác thực thông tin người dùng thông qua form và cơ sở dữ liệu.*
- *Hiểu cách sử dụng sessions và cookies để duy trì trạng thái đăng nhập của người dùng.*
- *Nắm vững các phương pháp bảo mật khi làm việc với sessions và cookies,...*

## 5.1 Thêm Dependency vào `pom.xml`

Mở thư mục `pom.xml` trong dự án và tiến hành thêm dependency của **Spring Boot Security** và **Thymeleaf Extras Spring Security** cho Thymeleaf.

```
<!-- Spring Boot Starter Security -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<!-- Thymeleaf Extras Spring Security for integrating with Thymeleaf -->
<dependency>
    <groupId>org.thymeleaf.extras</groupId>
    <artifactId>thymeleaf-extras-springsecurity6</artifactId>
</dependency>
```

**Lưu ý:** Sau khi thêm dependency đảm bảo rằng phải Reload Project

## 5.2 Bổ sung lớp `User` và `Role`

Tạo các entity `User` và `Role` để đại diện cho người dùng và quyền hạn của họ trong cơ sở dữ liệu.

Tại package model trong đường dẫn **src/main/java/com/hutech/demo/model** tạo thêm các class **Role, User**

### 5.2.1 Entity class User.java:

```
package com.hutech.demo.model;

import jakarta.persistence.*;
import jakarta.validation.constraints.*;
import lombok.*;
import org.hibernate.Hibernate;
import org.hibernate.validator.constraints.Length;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

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

@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")
    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;

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

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

@Override
public Collection<? extends GrantedAuthority> getAuthorities() {
    Set<Role> userRoles = this.getRoles();
    return userRoles.stream()
        .map(role -> new SimpleGrantedAuthority(role.getName()))
        .toList();
}

@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();
    }
}

```

## 5.2.2 Entity class Role.java

```

package com.hutech.demo.model;

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();
    }
}

```

## 5.3 Xác định vai trò người dùng sử dụng enum trong java

Trong môi trường phát triển phần mềm, việc quản lý các vai trò người dùng một cách hiệu quả là rất quan trọng để đảm bảo các quyền truy cập phù hợp đến các tài nguyên và chức năng của ứng dụng. Trường hợp này sử dụng một enum để quản lý các vai trò người dùng trong ứng dụng Java:

Trong đường dẫn **src/main/java/com.hutech.demo** tạo file **Role.enum**

```

package com.hutech.demo;

import lombok.AllArgsConstructorConstructor;

@AllArgsConstructor
public enum Role {
    ADMIN(1), // Vai trò quản trị viên, có quyền cao nhất trong hệ thống.
    USER(2); // Vai trò người dùng bình thường, có quyền hạn giới hạn.
    public final long value; / Biến này lưu giá trị số tương ứng với mỗi vai trò.
}

```

Ở đây, '**Role.enum**' định nghĩa hai vai trò: **ADMIN** và **USER**, mỗi vai trò được gán một giá trị số duy nhất. **ADMIN** được gán giá trị **1** và **USER** được gán giá trị **2**. Mỗi vai trò này sau đó có thể được sử dụng để kiểm soát quyền truy cập trong các tình huống khác nhau trong ứng dụng.

## 5.4 Cấu hình bảo mật sử dụng Spring Security

Lớp cấu hình này bao gồm các điều chỉnh cho phép kiểm soát truy cập, xác thực và quản lý phiên người dùng trong ứng dụng web. Bao gồm cấu hình cho trang đăng nhập, đăng xuất, quản lý phiên, và trang xử lý các lỗi truy cập bị từ chối.

Tạo một class cấu hình '**SecurityConfig.java**' tại đường dẫn

**src/main/java/com.hutech.demo:**

```
package com.hutech.demo;

import com.hutech.demo.service.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.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 // Đánh dấu lớp này là một lớp cấu hình cho Spring Context.
@EnableWebSecurity // Kích hoạt tính năng bảo mật web của Spring Security.
@RequiredArgsConstructor // Lombok tự động tạo constructor có tham số cho tất cả các trường final.
public class SecurityConfig {

    private final UserService userService; // Tiêm UserService vào lớp cấu hình này.

    @Bean // Đánh dấu phương thức trả về một bean được quản lý bởi Spring Context.
    public UserDetailsService userDetailsService() {
        return new UserService(); // Cung cấp dịch vụ xử lý chi tiết người dùng.
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder(); // Bean mã hóa mật khẩu sử dụng BCrypt.
    }

    @Bean
    public DaoAuthenticationProvider authenticationProvider() {
        var auth = new DaoAuthenticationProvider(); // Tạo nhà cung cấp xác thực.
        auth.setUserDetailsService(userDetailsService()); // Thiết lập dịch vụ chi tiết người dùng.
        auth.setPasswordEncoder(passwordEncoder()); // Thiết lập cơ chế mã hóa mật khẩu.
        return auth; // Trả về nhà cung cấp xác thực.
    }

    @Bean
    public SecurityFilterChain securityFilterChain(@NotNull HttpSecurity http) throws Exception {
        return http
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/css/**", "/js/**", "/", "/oauth/**", "/register", "/error",
"/products", "/cart", "/cart/**")
                .permitAll() // Cho phép truy cập không cần xác thực.
                .requestMatchers("/products/edit/**", "/products/add", "/products/delete")
                .hasAnyAuthority("ADMIN") // Chỉ cho phép ADMIN truy cập.
                .requestMatchers("/api/**"))
    }
}
```

```

        .permitAll() // API mở cho mọi người dùng.
        .anyRequest().authenticated() // Bất kỳ yêu cầu nào khác cần xác thực.
    )
    .logout(logout -> logout
        .logoutUrl("/logout")
        .logoutSuccessUrl("/login") // Trang chuyển hướng sau khi đăng xuất.
        .deleteCookies("JSESSIONID") // Xóa cookie.
        .invalidateHttpSession(true) // Hủy phiên làm việc.
        .clearAuthentication(true) // Xóa xác thực.
        .permitAll()
    )
    .formLogin(formLogin -> formLogin
        .loginPage("/login") // Trang đăng nhập.
        .loginProcessingUrl("/login") // URL xử lý đăng nhập.
        .defaultSuccessUrl("/") // Trang sau đăng nhập thành công.
        .failureUrl("/login?error") // Trang đăng nhập thất bại.
        .permitAll()
    )
    .rememberMe(rememberMe -> rememberMe
        .key("hutech")
        .rememberMeCookieName("hutech")
        .tokenValiditySeconds(24 * 60 * 60) // Thời gian nhớ đăng nhập.
        .userDetailsService(userDetailsService())
    )
    .exceptionHandling(exceptionHandling -> exceptionHandling
        .accessDeniedPage("/403") // Trang báo lỗi khi truy cập không được phép.
    )
    .sessionManagement(sessionManagement -> sessionManagement
        .maximumSessions(1) // Giới hạn số phiên đăng nhập.
        .expiredUrl("/login") // Trang khi phiên hết hạn.
    )
    .httpBasic(httpBasic -> httpBasic
        .realmName("hutech") // Tên miền cho xác thực cơ bản.
    )
    .build(); // Xây dựng và trả về chuỗi lọc bảo mật.
}
}

```

Phần phân quyền trong code trên được xử lý chủ yếu thông qua cấu hình **HttpSecurity** trong phương thức **securityFilterChain**. Đây là một cách tiếp cận tập trung cho phép bạn định nghĩa các quy tắc truy cập đến các tài nguyên khác nhau trong ứng dụng của bạn dựa trên vai trò của người dùng. Dưới đây là giải thích chi tiết về từng phần trong cấu hình này:

### Xác thực người dùng và phân quyền:

```
.authorizeHttpRequests(auth -> auth...)
```

Phương thức này bắt đầu một chuỗi các cấu hình xác thực và phân quyền cho các yêu cầu HTTP.

```
.requestMatchers("/css/**", "/js/**", "/", "/oauth/**", "/register", "/error")
    .permitAll()
```

Cấu hình này cho phép tất cả người dùng truy cập vào các tài nguyên cơ bản như CSS, JS, trang chủ, và các trang đăng ký hay báo lỗi mà không cần xác thực.

```
.requestMatchers("/product/edit/**", "/product/add", "/product/delete")
    .hasAnyAuthority("ADMIN")
```

Chỉ người dùng có vai trò "ADMIN" mới có quyền truy cập vào các đường dẫn liên quan đến chỉnh sửa, thêm, hoặc xóa sản phẩm.

```
.requestMatchers("/product", "/cart", "/cart/**")
    .hasAnyAuthority("ADMIN", "USER")
```

Người dùng có vai trò "ADMIN" hoặc "USER" đều có thể truy cập vào các đường dẫn liên quan đến sản phẩm và giỏ hàng.

```
.requestMatchers("/api/**")
    .permitAll()
```

Tất cả người dùng đều có thể truy cập vào các đường dẫn API mà không cần xác thực.

### Cấu hình đăng nhập và đăng xuất:

```
.formLogin(...)
```

Định nghĩa cách thức đăng nhập bằng form, bao gồm trang đăng nhập, URL xử lý đăng nhập, và trang chuyển hướng sau khi đăng nhập thành công hoặc thất bại.

```
.logout(...)
```

Định nghĩa cách thức đăng xuất, bao gồm URL xử lý đăng xuất, trang chuyển hướng sau khi đăng xuất, và các cài đặt về cookie và phiên làm việc.

### Quản lý phiên và nhớ đăng nhập:

```
.sessionManagement(...)
```

Cấu hình quản lý phiên, chẳng hạn như số lượng phiên tối đa cho phép từ một tài khoản.

```
.rememberMe(...)
```

Cài đặt cho tính năng nhớ đăng nhập, giúp người dùng không cần đăng nhập lại trong một khoảng thời gian nhất định.

### Xử lý ngoại lệ:

```
.exceptionHandling(...)
```

Cấu hình trang xử lý khi truy cập bị từ chối (khi người dùng không có quyền truy cập vào tài nguyên).

Mỗi phần của cấu hình này đóng vai trò quan trọng trong việc bảo vệ ứng dụng và đảm bảo rằng mỗi người dùng chỉ có thể truy cập vào các tài nguyên phù hợp với vai trò của họ. Cấu hình phân quyền như vậy giúp tăng cường an toàn và bảo mật cho ứng dụng của bạn.

## 5.5 Xây dựng Repository

Repository trong Spring Boot là các interface giúp tương tác với cơ sở dữ liệu một cách dễ dàng, sử dụng Spring Data JPA để tự động hóa các thao tác CRUD và truy vấn. Dưới đây là hướng dẫn chi tiết về cách tạo các repository cần thiết:

Trong thư mục theo đường dẫn `src/main/java/fit.hutech.spring/repository` tạo các file sau: ' **UserRepository.java**', '**IUserRepository.java**', '**IRoleRepository.java**'

### 5.5.1 UserRepository.java

File này chứa interface **UserRepository** để quản lý các thao tác liên quan đến đối tượng **User**. **UserRepository** mở rộng **JpaRepository**, cho phép thực hiện các thao tác cơ bản và cả một số truy vấn tùy chỉnh:

```
package com.hutech.demo.repository;

import com.hutech.demo.model.User;
import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, Long> {
    User findByUsername(String username);
}
```

Phương thức '`findByUsername`' sẽ truy vấn cơ sở dữ liệu để tìm một người dùng dựa trên tên đăng nhập của họ.

## 5.5.2 I UserRepository.java

Để minh họa việc quản lý các phiên bản khác nhau của repository cho cùng một mô hình, chúng ta có **I UserRepository**. Đây là một phiên bản khác của repository cho **User**, có thể được sử dụng trong các trường hợp khác nhau:

```
package com.hutech.demo.repository;

import com.hutech.demo.model.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.Optional;

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

Phương thức '`findByUsername`' ở đây trả về '`Optional<User>`', giúp xử lý trường hợp không tìm thấy người dùng một cách an toàn hơn.

## 5.5.3 I RoleRepository.java

Interface **I RoleRepository** quản lý các thao tác với đối tượng Role. Nó cũng mở rộng từ JpaRepository và bao gồm các phương thức cụ thể cho việc quản lý vai trò:

```
package com.hutech.demo.repository;

import com.hutech.demo.model.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);
}
```

Phương thức `findRoleById` trả về một **Role** dựa trên **ID**, hỗ trợ việc truy xuất thông tin chi tiết của vai trò.

## 5.6 Xây dựng class UserService

Class **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. Điều này đảm bảo tính nhất quán và hiệu quả trong việc quản lý người dùng và vai trò của họ.

Tiếp theo, tạo class **UserService.java** được đặt tại đường dẫn sau

**src/main/java/com.hutech.demo/service.**

Mã nguồn mẫu '**UserService.java**':

```
package com.hutech.demo.service;

import com.hutech.demo.Role;
import com.hutech.demo.model.User;
import com.hutech.demo.repository.IRoleRepository;
import com.hutech.demo.repository.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.Transactional;

import java.util.Optional;

@Service
@Slf4j
@Transactional
public class UserService implements UserDetailsService {

    @Autowired
    private IUserRepository userRepository;
    @Autowired
    private IRoleRepository roleRepository;

    // Lưu người dùng mới vào cơ sở dữ liệu sau khi mã hóa mật khẩu.
    public void save(@NotNull User user) {
        user.setPassword(new BCryptPasswordEncoder().encode(user.getPassword()));
        userRepository.save(user);
    }

    // Gán vai trò mặc định cho người dùng dựa trên tên người dùng.
    public void setDefaultRole(String username) {
        userRepository.findByUsername(username).ifPresentOrElse(
            user -> {

                user.getRoles().add(roleRepository.findRoleById(Role.USER.value));
                userRepository.save(user);
            },
            () -> { throw new UsernameNotFoundException("User not found"); }
        );
    }
}
```

```

}

// Tải thông tin chi tiết người dùng để xác thực.
@Override
public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
    var user = userRepository.findByUsername(username)
        .orElseThrow(() -> new UsernameNotFoundException("User not
found"));
    return org.springframework.security.core.userdetails.User
        .withUsername(user.getUsername())
        .password(user.getPassword())
        .authorities(user.getAuthorities())
        .accountExpired(!user.isAccountNonExpired())
        .accountLocked(!user.isAccountNonLocked())
        .credentialsExpired(!user.isCredentialsNonExpired())
        .disabled(!user.isEnabled())
        .build();
}

// Tìm kiếm người dùng dựa trên tên đăng nhập.
public Optional<User> findByUsername(String username) throws
UsernameNotFoundException {
    return userRepository.findByUsername(username);
}
}

```

### Giải thích mã nguồn:

- Autowired:** Tự động tiêm **IUserRepository** và **IRoleRepository** để tương tác với cơ sở dữ liệu.
- Mã hóa mật khẩu:** Sử dụng **BCryptPasswordEncoder** để mã hóa mật khẩu trước khi lưu vào cơ sở dữ liệu.
- Quản lý vai trò:** Phương thức **setDefaultRole** gán vai trò mặc định cho người dùng khi đăng ký.
- Xác thực người dùng:** Phương thức **loadUserByUsername** cung cấp cơ chế xác thực người dùng thông qua Spring Security.

## 5.7 Xây dựng UserController

**UserController** xử lý các yêu cầu từ người dùng cuối liên quan đến hoạt động của người dùng, chẳng hạn như đăng nhập và đăng ký. **Controller** này sử dụng **UserService** để thực hiện các nghiệp vụ liên quan đến người dùng và vai trò của họ trong hệ thống.

Trong đường dẫn **src/main/java/com.hutech.demo/controller** tạo file '**UserController.java**':

```
package com.hutech.demo.controller;

import com.hutech.demo.model.User;
import com.hutech.demo.service.UserService;
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 // Đánh dấu lớp này là một Controller trong Spring MVC.
@RequestMapping("/")
@RequiredArgsConstructor
public class UserController {
    private final UserService userService;

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

    @GetMapping("/register")
    public String register(@NotNull Model model) {
        model.addAttribute("user", new User()); // Thêm một đối tượng User mới vào model
        return "users/register";
    }

    @PostMapping("/register")
    public String register(@Valid @ModelAttribute("user") User user, // Validate đối tượng User
                           @NotNull BindingResult bindingResult, // Kết quả của quá trình validate
                           Model model) {
        if (bindingResult.hasErrors()) { // Kiểm tra nếu có lỗi validate
            var errors = bindingResult.getAllErrors()
                .stream()
                .map(DefaultMessageSourceResolvable::getDefaultMessage)
                .toArray(String[]::new);
            model.addAttribute("errors", errors);
            return "users/register"; // Trả về lại view "register" nếu có lỗi
        }
        userService.save(user); // Lưu người dùng vào cơ sở dữ liệu
        userService.setDefaultRole(user.getUsername()); // Gán vai trò mặc định cho người dùng
        return "redirect:/login"; // Chuyển hướng người dùng tới trang "login"
    }
}
```

## 5.8 Phát triển các trang đăng nhập và đăng ký

Xây dựng các trang HTML/Thymeleaf cho việc đăng nhập và đăng ký.

Đầu tiên tạo thư mục **users** tại đường dẫn **src/main/resources/templates**.

Tạo 2 view '**login.html**' và '**register.html**' đặt tại thư mục **src/main/resources/templates/users**

### 5.8.1 File 'register.html'

Trang đăng ký cho phép người dùng mới nhập thông tin cá nhân và tạo một tài khoản trong hệ thống. Trang này sử dụng Bootstrap để đảm bảo tính thẩm mỹ và phù hợp với các thiết bị khác nhau, đồng thời tích hợp Thymeleaf để xử lý các tương tác phía server như xác thực dữ liệu và hiển thị thông báo lỗi.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
      layout:decorate="~{layout}">
<head>
    <title>Register</title>
    <link rel="stylesheet"
          href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
</head>
<body>
<section layout:fragment="content" class="container mt-5">
    <h1 class="mb-4">Login</h1>
    <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" 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" 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" id="password" name="password" placeholder="Enter your password">
        </div>
        <div class="form-group mb-4">
            <label for="phone"></label>
        </div>
    </form>
</section>
```

```

        <input type="tel" class="form-control" 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>
</section>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js">
</script>
</body>
</html>

```

## 5.8.2 File 'login.html'

Trang đăng nhập cho phép người dùng truy cập bằng tên người dùng và mật khẩu đã đăng ký. Trang này cũng hỗ trợ hiển thị các thông báo liên quan đến quá trình đăng nhập, như thông báo lỗi khi nhập sai thông tin hoặc xác nhận đã đăng xuất thành công.

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
layout:decorate=~{layout}>
<head>
    <title>Login</title>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
</head>
<body>
<section layout:fragment="content" class="container mt-5">
    <h1 class="mb-4">Login</h1>
    <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>
    </form>
</section>

```

```

<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>
</section>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js">
</script>
</body>
</html>

```

### 5.8.3 Cập nhật 'layout.html'

Để hỗ trợ việc đăng nhập và đăng xuất, **layout.html** được cập nhật với các nút **Login** và **Logout** trên thanh điều hướng, cùng với các thông báo khi đăng nhập thành công. Sử dụng **Spring Security**, trang này sẽ thích ứng hiển thị dựa trên trạng thái xác thực của người dùng, tăng cường trải nghiệm người dùng và bảo mật cho ứng dụng.

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/extras/spring-security5"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout">
<head>
    <meta charset="UTF-8">
    <title>Layout</title>
    <link rel="stylesheet"
          href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
</head>
<body>
<nav class="navbar navbar-expand-lg navbar-light bg-primary">
    <div class="container-fluid">
        <a class="navbar-brand text-white" href="/products">Web Bán Hàng</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 text-white active" aria-current="page"
                       href="/products">Product</a>
                </li>
                <li class="nav-item" sec:authorize="hasAnyAuthority('ADMIN')">
                    <a th:href="@{/products/add}" class="nav-link text-white">Add
New Product</a>
                </li>
                <li class="nav-item" sec:authorize="hasAnyAuthority('ADMIN')">
                    <a th:href="@{/categories/add}" class="nav-link text-white">Add
New Category</a>
                </li>
            </ul>
        </div>
    </div>
</nav>

```

```

        <ul class="navbar-nav ms-auto mb-2 mb-lg-0 align-items-center">
            <li class="nav-item d-flex align-items-center"
sec:authorize="isAuthenticated()">
                <span class="navbar-text text-white">
                    Xin chào, <span sec:authentication="name" style="margin-
right: 20px;"></span>
                </span>
            </li>
            <li class="nav-item" sec:authorize="isAuthenticated()">
                <form th:action="@{/logout}" method="post">
                    <button class="btn btn-outline-light"
type="submit">Logout</button>
                </form>
            </li>
            <li class="nav-item" sec:authorize="!isAuthenticated()">
                <a class="btn btn-outline-light" href="/login">Login</a>
            </li>
        </ul>
    </div>
</div>

<div class="container mt-5">
    <section layout:fragment="content">
        <!-- Nội dung cụ thể của từng trang sẽ được đặt tại đây -->
    </section>
</div>

<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js">
</script>
</body>
</html>

```

Đoạn mã trên sử dụng Thymeleaf và Spring Security, tích hợp một hệ thống phân quyền để quản lý hiển thị các phần của giao diện người dùng dựa trên các quyền của người dùng. Đây là một cách tiếp cận hiệu quả để bảo vệ và cá nhân hóa nội dung của ứng dụng web.

### **Giải thích mã nguồn:**

#### **a) Phân Quyền Quản Trị (Admin)**

```

</li>
<li class="nav-item" sec:authorize="hasAnyAuthority('ADMIN')">
    <a th:href="@{/products/add}" class="nav-link text-white">Add New Product</a>
</li>
<li class="nav-item" sec:authorize="hasAnyAuthority('ADMIN')">
    <a th:href="@{/categories/add}" class="nav-link text-white">Add New
Category</a>
</li>

```

Sử dụng thuộc tính `sec:authorize="hasAnyAuthority('ADMIN')"` để kiểm tra nếu người dùng hiện tại có quyền '**ADMIN**'. Chỉ những người dùng có quyền này mới có thể

nhìn thấy và truy cập vào các liên kết để thêm sản phẩm mới và thêm danh mục mới trên thanh điều hướng. Điều này đảm bảo rằng chỉ những người có thẩm quyền quản lý mới có thể thực hiện các thao tác này, giúp bảo vệ tính toàn vẹn của ứng dụng.

### b) Xác Thực Người Dùng

```
<li class="nav-item d-flex align-items-center"
sec:authorize="isAuthenticated()">
    <span class="navbar-text text-white">
        Xin chào, <span sec:authentication="name" style="margin-right:
20px;"></span>
    </span>
</li>
<li class="nav-item" sec:authorize="isAuthenticated() ">
    <form th:action="@{/logout}" method="post">
        <button class="btn btn-outline-light" type="submit">Logout</button>
    </form>
</li>
```

- `sec:authorize="isAuthenticated()"` được sử dụng để kiểm tra nếu người dùng đã đăng nhập. Khi người dùng đã xác thực, họ sẽ thấy tên của mình trên thanh điều hướng cùng với một nút đăng xuất. Điều này cung cấp trải nghiệm người dùng liền mạch và cá nhân hóa, tăng cường tính năng bảo mật và sự thuận tiện.

```
<li class="nav-item" sec:authorize="!isAuthenticated()">
    <a class="btn btn-outline-light" href="/login">Login</a>
</li>
```

- `sec:authorize="!isAuthenticated()"` cho phép hiển thị nút đăng nhập cho những người dùng chưa đăng nhập. Điều này khuyến khích người dùng tham gia và tương tác với các tính năng của trang web, đồng thời duy trì tính bảo mật bằng cách không hiển thị các tùy chọn nhạy cảm.

### Ý Nghĩa và Lợi Ích

Sử dụng Thymeleaf và Spring Security để quản lý quyền và xác thực người dùng không chỉ giúp tăng cường bảo mật cho ứng dụng web bằng cách hạn chế truy cập vào các chức năng nhạy cảm mà còn cải thiện trải nghiệm người dùng bằng cách cung cấp giao diện đáp ứng tốt với trạng thái xác thực của họ. Điều này làm cho trang web trở nên thân thiện và dễ sử dụng hơn, đồng thời giữ cho thông tin người dùng an toàn và bảo mật.

## 5.8.4 Cập nhật file 'products-list.html'

Cập nhật file **products-list.html** để thêm chức năng quản lý sản phẩm cho người dùng có quyền quản trị (**ADMIN**) và tính năng thêm sản phẩm vào giỏ hàng cho tất cả người dùng đã xác thực. Đây là bước cần thiết để đảm bảo tính bảo mật và hiệu quả của ứng dụng thương mại điện tử.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/extras/spring-security5"
      >!-- Namespace của Spring Security, cho phép kiểm soát quyền truy cập -->
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
      layout:decorate="~{layout}">
<head>
    <title th:text="${title} ?: 'Products List'">Products List</title>
    <link rel="stylesheet"
          href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
</head>
<body>
<section layout:fragment="content">
    <h1>Products List</h1>
    <table class="table table-bordered table-hover">
        <thead class="table-dark">
            <tr>
                <th>ID</th>
                <th>Name</th>
                <th>Price</th>
                <th>Description</th>
                <th>Category Name</th>
                <th>Actions</th>
                <th>Add To Cart</th>
            </tr>
        </thead>
        <tbody>
            <tr th:each="product : ${products}">
                <td th:text="${product.id}"></td>
                <td th:text="${product.name}"></td>
                <td th:text="${product.price}"></td>
                <td th:text="${product.description}"></td>
                <td th:text="${product.category.name}"></td>
                <td>
                    <!-- Hiển thị nút sửa và xóa chỉ dành cho người dùng ADMIN -->
                    <div sec:authorize="hasAuthority('ADMIN')">
                        <a th:href="@{/products/edit/{id}(id=${product.id})}">
                            <button class="btn btn-success btn-sm" type="button">Sửa</button>
                        <a th:href="@{/products/delete/{id}(id=${product.id})}">
                            <button class="btn btn-danger btn-sm" type="button" onclick="return confirm('Bạn có chắc không?')">Xóa</button>
                        </div>
                    </td>
                <td>
                    <!-- Nút thêm vào giỏ hàng, hiển thị cho tất cả người dùng đã xác thực -->
                    <form th:action="@{/cart/add}" method="post">
                        <sec:authorize="isAuthenticated()">
                            <input type="number" name="quantity" min="1" value="1" />
                        </sec:authorize>
                    </form>
                </td>
            </tr>
        </tbody>
    </table>
</section>
```

```
class="form-control d-inline-block" style="width: 70px;">>
    <input type="hidden" th:value="${product.id}" name="productId"/>
    <button type="submit" class="btn btn-warning btn-sm">Thêm Vào
Giỏ</button>
</td>
</tr>
</tbody>
</table>
</section>
</body>
</html>
```

### Giải thích Các Thay Đổi:

**Quyền Quản Trị (ADMIN):** Các nút "**Edit**" và "**Delete**" chỉ được hiển thị cho người dùng có quyền ADMIN. Điều này giúp đảm bảo rằng chỉ người dùng được phép mới có thể chỉnh sửa hoặc xóa sản phẩm, bảo vệ thông tin sản phẩm khỏi các thay đổi không mong muốn hoặc truy cập trái phép.

**Thêm vào Giỏ Hàng:** Chức năng này được hiển thị cho mọi người dùng đã xác thực, cho phép họ thêm sản phẩm vào giỏ hàng của mình, hỗ trợ các hoạt động mua sắm trên trang web.

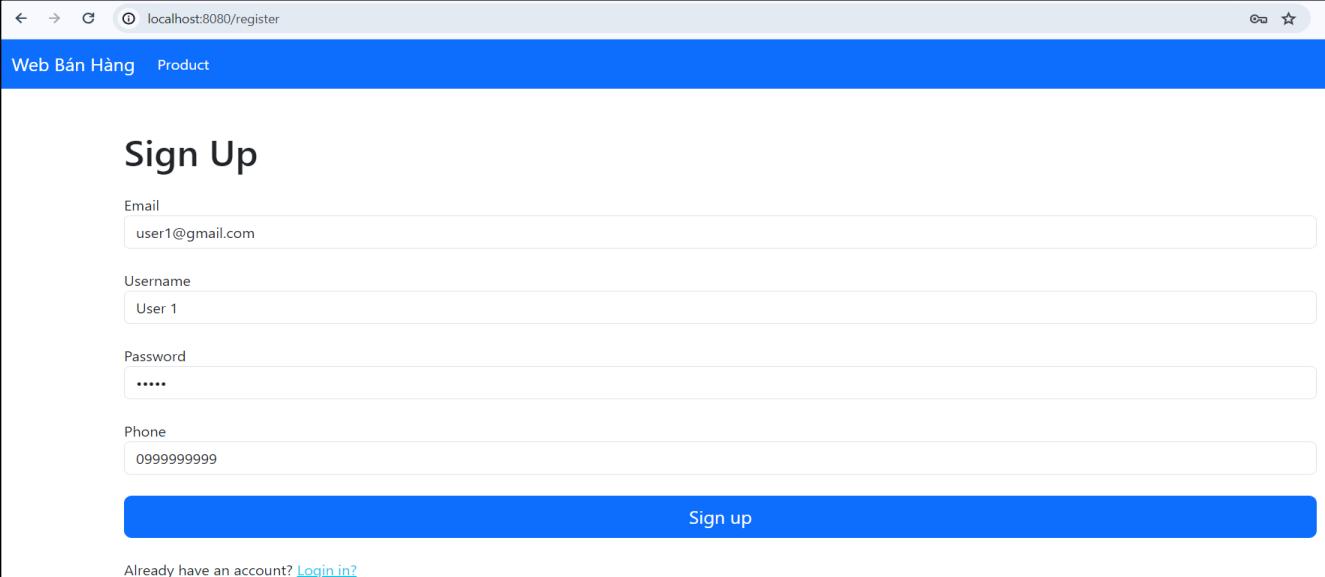
Qua cách tiếp cận này, bạn không chỉ cải thiện tính năng bảo mật của trang web mà còn tăng cường khả năng tương tác người dùng, tạo trải nghiệm mua sắm dễ dàng và an toàn hơn.

## 5.9 Tiến hành build và kiểm tra kết quả

Tiến hành xây dựng, kiểm tra và đánh giá các chức năng quan trọng của ứng dụng web, bao gồm việc đăng ký, đăng nhập, và phân quyền cho người dùng. Dưới đây là một hướng dẫn chi tiết về cách thực hiện từng bước:

### 5.9.1 Trang đăng ký

Người dùng cần truy cập vào URL `http://localhost:8080/register` để mở trang đăng ký. Tại đây, người dùng có thể điền thông tin cần thiết để tạo một tài khoản mới. Sau khi điền đầy đủ thông tin và gửi biểu mẫu, tài khoản người dùng mới sẽ được tạo trong hệ thống.



The screenshot shows a web browser window with the URL `localhost:8080/register` in the address bar. The page title is "Web Bán Hàng". The main content is a "Sign Up" form with the following fields:

- Email: user1@gmail.com
- Username: User 1
- Password: \*\*\*\*\*
- Phone: 0999999999

A large blue "Sign up" button is at the bottom of the form. Below the form, there is a link "Already have an account? [Login in?](#)".

### 5.9.2 Trang đăng nhập

Sau khi tài khoản đã được tạo, người dùng có thể truy cập vào trang đăng nhập `http://localhost:8080/login` để kiểm tra khả năng đăng nhập sử dụng tài khoản vừa đăng ký.

localhost:8080/login

Web Bán Hàng Product

## Login

user 1

.....

Remember me

Login

Don't have an account? [Sign up?](#)

### 5.9.3 Kiểm tra Database

Sau khi người dùng đăng ký, kiểm tra trong cơ sở dữ liệu, đặc biệt là bảng “user”, để đảm bảo rằng thông tin người dùng đã được lưu trữ chính xác. Điều này quan trọng để xác nhận rằng hệ thống đang hoạt động chính xác và không có lỗi xảy ra trong quá trình đăng ký.

Laragon.MySQL\webbanhang\user - HeidiSQL Portable 12.1.0.6537

File Edit Search Query Tools Go to Help

Database filter Table filter Host: 127.0.0.1 Database: webbanhang Table: user Data Query

Laragon.MySQL

- information\_s...
- mysql
- performance\_...
- sys
- webbanhang
  - categories
  - orders
  - order\_details
  - products
  - role
  - user**
  - user\_role

webbanhang.user: 3 rows total (apprc) Next Show all Sorting Columns (6/6) Filter

ID	Email	Password	Phone	Provider	Username
2	user1@gmail.com	\$2a\$10\$.5JFoCxCGUj/5xR5k7Lzf.c43Dlzd0G1...	0999999999	(NULL)	User 1
3	user2@gmail.com	\$2a\$10\$x5K7KvjdYRjHjaUTLS5uAnVXhhOTC...	0888888888	(NULL)	User 2
4	user3@gmail.com	\$2a\$10\$HkVxfjGx46ySABcFXvKuceeqJ7kbhEq/f...	0777777777	(NULL)	User 3

Filter: Regular expression

```

119 SELECT * FROM `webbanhang`.`user_role` LIMIT 1000;
120 SHOW CREATE TABLE `webbanhang`.`user`;
121 SELECT tc.CONSTRAINT_NAME, cc.CHECK_CLAUSE FROM `information_schema`.`CHECK_CONSTRAINTS` AS cc, `information_schema`.`TABLE_CONSTRAINTS` AS tc WHERE tc.CONSTRAINT_NAME = cc.CONSTRAINT_NAME;
122 SELECT * FROM `webbanhang`.`user` LIMIT 1000;
  
```

Connected: 01 MySQL 8.0.30 Uptime: 02:19 h Server time: 11:00:00 Idle.

## 5.9.4 Tiến hành phân quyền cho các tài khoản:

Truy cập vào database và đi tới bảng ‘**user\_role**’. Tại đây, thêm hoặc sửa đổi các mục để phân quyền cho người dùng. Chẳng hạn, bạn có thể phân quyền ‘ADMIN’ cho ‘user 1’:

The screenshot shows the HeidiSQL interface with the 'webbanhang' database selected. The 'user\_role' table is open, showing 0 rows total. A context menu is open over the table, with the 'Insert row' option highlighted by a red box. The table structure includes columns 'user\_id' and 'role\_id'. The query history at the bottom shows some SQL queries related to users and roles.

Phân quyền cho ‘user 1’ trở thành quyền ‘ADMIN’:

The screenshot shows the HeidiSQL interface with the 'webbanhang' database selected. The 'user\_role' table now contains 1 row. The inserted row has 'user\_id' value 2 and 'role\_id' value 1, both highlighted with red boxes. Two callout boxes point to these values: 'user\_id: 2 tương ứng với 'user 1'' and 'role\_id: 1 tương ứng với quyền ADMIN'. The table structure includes columns 'user\_id' and 'role\_id'. The query history at the bottom shows the SQL command used to insert the row.

Các tài khoản người dùng khác mặc định quyền 'USER'

### 5.9.5 Đăng nhập và kiểm tra phân quyền

Sau khi phân quyền, đăng nhập vào ứng dụng bằng tài khoản có quyền 'ADMIN' và kiểm tra các chức năng đặc quyền như quản lý sản phẩm hoặc người dùng. Đồng thời, đăng nhập bằng tài khoản người dùng bình thường để đảm bảo rằng họ không truy cập được các chức năng đặc quyền.

Trang đăng nhập tài khoản quyền '**ADMIN**':

ID	Name	Price	Description	Category Name	Actions	Add To Cart
1	Laptop Hutech 1 - New	2200.0	Laptop Hutech 1 (Đã update)	Laptop	<button>Edit</button> <button>Delete</button>	<input type="checkbox"/> <button>Add to Cart</button>
3	PC Hutech 1	1500.0	PC Hutech 1	Computer	<button>Edit</button> <button>Delete</button>	<input type="checkbox"/> <button>Add to Cart</button>
4	Hphone 24	3000.0	Điện thoại Hphone 24	Phone	<button>Edit</button> <button>Delete</button>	<input type="checkbox"/> <button>Add to Cart</button>
5	PC Hutech 2	2000.0	PC Hutech 2	Laptop	<button>Edit</button> <button>Delete</button>	<input type="checkbox"/> <button>Add to Cart</button>

Trang đăng nhập của người dùng bình thường:

ID	Name	Price	Description	Category Name	Actions	Add To Cart
1	Laptop Hutech 1 - New	2200.0	Laptop Hutech 1 (Đã update)	Laptop		<input type="checkbox"/> <button>Thêm Vào Giỏ</button>
3	PC Hutech 1	1500.0	PC Hutech 1	Computer		<input type="checkbox"/> <button>Thêm Vào Giỏ</button>
4	Hphone 24	3000.0	Điện thoại Hphone 24	Phone		<input type="checkbox"/> <button>Thêm Vào Giỏ</button>
5	PC Hutech 2	2000.0	PC Hutech 2	Laptop		<input type="checkbox"/> <button>Thêm Vào Giỏ</button>

Thực hiện các bước này sẽ giúp đảm bảo rằng ứng dụng của bạn hoạt động chính xác theo yêu cầu và quyền của người dùng được xử lý một cách an toàn và hiệu quả.

# BÀI 6. RESTful API

Sau khi học xong bài này, Sinh viên có thể nắm được:

- *Nắm vững các bước cơ bản để xây dựng một RESTful API trong Spring Boot*
- *Biết cách sử dụng Spring Data JPA để tương tác với cơ sở dữ liệu trong các API*
- *Hiểu được vai trò của các endpoint API để xử lý các yêu cầu HTTP.*
- *Biết cách tạo giao diện người dùng đơn giản với HTML và AJAX để tương tác với API một cách hiệu quả, không cần tải lại trang.*

## 6.1 Yêu cầu các chức năng cần triển khai

### 6.1.1 Xây dựng các API để quản lý sản phẩm

- *Thêm sản phẩm mới.*
- *Cập nhật thông tin sản phẩm.*
- *Xóa sản phẩm.*
- *Hiển thị danh sách tất cả sản phẩm.*

### 6.1.2 Giao diện người dùng

- *Trang web sử dụng AJAX để:*
- *Hiển thị danh sách sản phẩm.*
- *Gửi yêu cầu thêm sản phẩm mới tới server mà không cần tải lại trang.*
- *Cập nhật thông tin một sản phẩm.*
- *Xóa một sản phẩm.*

## 6.2 Hướng dẫn thực hiện

Tạo RESTful Controller 'ProductApiController'

```
package com.hutech.demo.controller;

import com.hutech.demo.model.Product;
import com.hutech.demo.repository.ProductRepository;
import com.hutech.demo.service.ProductService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@CrossOrigin
@RestController
@RequestMapping("/api/products")
public class ProductApiController {
    @Autowired
    private ProductService productService;

    @GetMapping
    public List<Product> getAllProducts() {
        return productService.getAllProducts();
    }

    @PostMapping
    public Product createProduct(@RequestBody Product product) {
        return productService.addProduct(product);
    }

    @GetMapping("/{id}")
    public ResponseEntity<Product> getProductById(@PathVariable Long id) {
        Product product = productService.getProductById(id)
            .orElseThrow(() -> new RuntimeException("Product not found on :: " +
+ id));
        return ResponseEntity.ok().body(product);
    }

    @PutMapping("/{id}")
    public ResponseEntity<Product> updateProduct(@PathVariable Long id,
@RequestBody Product productDetails) {
        Product product = productService.getProductById(id)
            .orElseThrow(() -> new RuntimeException("Product not found on :: " +
+ id));

        product.setName(productDetails.getName());
        product.setPrice(productDetails.getPrice());
        product.setDescription(productDetails.getDescription());

        final Product updatedProduct = productService.addProduct(product);
        return ResponseEntity.ok(updatedProduct);
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteProduct(@PathVariable Long id) {
        Product product = productService.getProductById(id)
            .orElseThrow(() -> new RuntimeException("Product not found on :: " +
+ id));
    }
}
```

```
+ id));  
  
        productService.deleteProductById(id);  
        return ResponseEntity.ok().build();  
    }  
}
```

## Tạo dự án front-end để tương tác với API

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <meta charset="UTF-8">  
    <title>Product Management</title>  
    <link rel="stylesheet"  
        href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">  
        <script  
            src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>  
</head>  
<body>  
    <div class="container mt-5">  
        <h1>Product Management</h1>  
        <button onclick="loadProducts()" class="btn btn-primary mb-3">Refresh  
Products</button>  
        <table class="table table-bordered">  
            <thead>  
                <tr>  
                    <th>ID</th>  
                    <th>Name</th>  
                    <th>Price</th>  
                    <th>Description</th>  
                    <th>Actions</th>  
                </tr>  
            </thead>  
            <tbody id="productList"></tbody>  
        </table>  
        <!-- Form to add/update a product -->  
        <form id="productForm">  
            <input type="hidden" id="productId">  
            <div class="mb-3">  
                <label for="name" class="form-label">Name:</label>  
                <input type="text" class="form-control" id="name" required>  
            </div>  
            <div class="mb-3">  
                <label for="price" class="form-label">Price:</label>  
                <input type="number" class="form-control" id="price" required>  
            </div>  
            <div class="mb-3">  
                <label for="description" class="form-label">Description:</label>  
                <input type="text" class="form-control" id="description">  
            </div>  
            <button type="submit" class="btn btn-success">Save Product</button>  
        </form>  
    </div>  
  
<script>  
    $(document).ready(function() {  
        loadProducts();  
        $("#productForm").on('submit', function(e) {  
            e.preventDefault();
```

```
        saveProduct();
    });
});

function loadProducts() {
    $.ajax({
        url: '/api/products',
        type: 'GET',
        success: function(products) {
            let productList = '';
            $.each(products, function(index, product) {
                productList += `<tr>
                    <td>${product.id}</td>
                    <td>${product.name}</td>
                    <td>${product.price}</td>
                    <td>${product.description}</td>
                    <td>
                        <button
                            onclick="editProduct(${product.id})" class="btn btn-warning">Edit</button>
                        <button
                            onclick="deleteProduct(${product.id})" class="btn btn-danger">Delete</button>
                    </td>
                </tr>`;
            });
            $('#productList').html(productList);
        }
    });
}

function saveProduct() {
    const productData = {
        id: $('#productId').val(),
        name: $('#name').val(),
        price: $('#price').val(),
        description: $('#description').val()
    };
    const apiUrl = productData.id ? `/api/products/${productData.id}` :
    '/api/products';
    const apiType = productData.id ? 'PUT' : 'POST';

    $.ajax({
        url: apiUrl,
        type: apiType,
        contentType: 'application/json',
        data: JSON.stringify(productData),
        success: function() {
            resetForm();
            loadProducts();
        }
    });
}

function editProduct(id) {
    $.ajax({
        url: `/api/products/${id}`,
        type: 'GET',
        success: function(product) {
            $('#productId').val(product.id);
            $('#name').val(product.name);
            $('#price').val(product.price);
        }
    });
}
```

```
        $('#description').val(product.description);
    }
})
}

function deleteProduct(id) {
    if (confirm('Are you sure you want to delete this product?')) {
        $.ajax({
            url: `/api/products/${id}`,
            type: 'DELETE',
            success: function() {
                loadProducts();
            }
        });
    }
}

function resetForm() {
    $('#productId').val('');
    $('#name').val('');
    $('#price').val('');
    $('#description').val('');
}
</script>
</body>
</html>
```

# TÀI LIỆU THAM KHẢO

1. Clean Code: A Handbook of Agile Software Craftsmanship, Robert C. Martin, 2008
2. Eloquent JavaScript: A Modern Introduction to Programming, Marijn Haverbeke, 2018 (3rd edition)
3. You Don't Know JS (series), Kyle Simpson, 2015
4. MDN Web Docs (<https://developer.mozilla.org>)
5. Stack Overflow (<https://stackoverflow.com>)
6. GitHub (<https://github.com>)
7. W3Schools (<https://www.w3schools.com>)
8. Smashing Magazine (<https://www.smashingmagazine.com>)