BỘ GIÁO DỤC VÀ ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC THĂNG LONG**

A red and blue logo

Description automatically generated

**BÁO CÁO BÀI TẬP LỚN**

**DỰ ÁN CÔNG NGHỆ THÔNG TIN**

**ĐỀ TÀI: TẠO WEBSIDE BÁN LAPTOP**

**GIẢNG VIÊN HƯỚNG DẪN: Nguyễn Hùng Cường**

**SINH VIÊN THỰC HIỆN: NHÓM 3**

**A44254 - Nguyễn Quang Huy**

**A47779 - Dương Kim Tài**

**A48687 – Lê Huy Đạt**

**A47113-Lưu Huy Hoàng**

**MÔN HỌC: IT380 – Dự án CNTT**

**HÀ NỘI - 2025**

**MỤC LỤC**

[CHƯƠNG 1. CƠ SỞ LÝ THUYẾT 2](#_Toc217987553)

[1.1. Kiến trúc Hệ thống (Fullstack Architecture) 2](#_Toc217987554)

[1.2. Các Công nghệ Sử dụng 2](#_Toc217987555)

[1.3. Các Design Pattern Áp dụng 3](#_Toc217987556)

[CHƯƠNG 2. THỰC HÀNH THỰC NGHIỆM 6](#_Toc217987557)

[2.1. FrontEnd 6](#_Toc217987558)

[2.1.1. Khởi tạo ứng dụng 6](#_Toc217987559)

[2.1.2. Cấu trúc dự án 8](#_Toc217987560)

[2.1.3. Trang indext 9](#_Toc217987561)

[2.1.4. App 11](#_Toc217987562)

[2.1.5. Trang chủ 13](#_Toc217987563)

[2.1.6. Product 15](#_Toc217987564)

[2.1.7. Detail 16](#_Toc217987565)

[2.1.8. Thanh toán 18](#_Toc217987566)

[2.2. BackEnd 18](#_Toc217987567)

[2.2.1. Kết nối với mysql 18](#_Toc217987568)

[2.2.2. Đăng nhập đăng ký (AuthController.java) 19](#_Toc217987569)

[2.2.3. Giỏ hàng (CartController.java) 20](#_Toc217987570)

[2.2.4. Thanh toán 21](#_Toc217987571)

[2.2.5. Đơn hàng đã mua 22](#_Toc217987572)

[2.2.6. Truyền/nhận dữ liệu với frontend 22](#_Toc217987573)

[2.2.7. Lấy dữ liệu từ bảng trong cơ sở dữ liệu 24](#_Toc217987574)

[2.3. Các sơ đồ 29](#_Toc217987575)

[2.3.1. Sơ đồ Ca sử dụng (Use Case Diagram) 29](#_Toc217987576)

[2.3.2. Sơ đồ Lớp (Class Diagram) 30](#_Toc217987577)

[2.3.3. Sơ đồ Tuần tự (Sequence Diagram) 31](#_Toc217987578)

[2.3.4. Builder 33](#_Toc217987579)

[2.3.5. Singleton 33](#_Toc217987580)

[2.3.6. Facade Pattern 35](#_Toc217987581)

[2.3.7. Factory Method Pattern (Mẫu Nhà Máy) 36](#_Toc217987582)

[2.3.8. Repository Pattern 37](#_Toc217987583)

[2.4. Cách chạy chương trình 38](#_Toc217987584)

[2.4.1. Tạo cơ sở dữ liệu 38](#_Toc217987585)

[2.4.2. Chạy chương trình 39](#_Toc217987586)

[CHƯƠNG 3. tìm hiểu thêm 42](#_Toc217987587)

[3.1. Template Method 42](#_Toc217987588)

[3.1.1. Giới thiệu chung 42](#_Toc217987589)

[3.1.2. Ý nghĩa và Động cơ sử dụng 42](#_Toc217987590)

[3.1.3. Kiến trúc và Thành phần 42](#_Toc217987591)

[3.1.4. Đặc điểm của Template Method Pattern 43](#_Toc217987592)

[3.1.5. Ưu điểm và Nhược điểm 43](#_Toc217987593)

[3.1.6. Khi nào nên sử dụng Template Method Pattern 44](#_Toc217987594)

[3.1.7. Ví dụ về quy trình pha chế đồ uống (Trà, Cà phê) bằng python. 44](#_Toc217987595)

[3.1.8. Ví dụ về quy trình pha chế đồ uống (Trà, Cà phê) bằng Java. 46](#_Toc217987596)

**DANH MỤC HÌNH**

[Ảnh 1.1 Minh họa Singleton pattern 3](#_Toc217987409)

[Ảnh 1.2 Minh họa Builder pattern 4](#_Toc217987410)

[Ảnh 1.3 Minh họa Factory pattern 5](#_Toc217987411)

[Ảnh 1.4 Minh họa Facade Pattern 5](#_Toc217987412)

[Ảnh 2.1 file pom.xml 6](#_Toc217987413)

[Ảnh 2.2 Đồ bộ hóa dự án 7](#_Toc217987414)

[Ảnh 2.3 Cấu trúc dự án 8](#_Toc217987415)

[Ảnh 2.4 import dữ liệu 9](#_Toc217987416)

[Ảnh 2.5 xây dự bố cục của trang web 9](#_Toc217987417)

[Ảnh 2.6 khởi tạo trang web và import dữ liệu 10](#_Toc217987418)

[Ảnh 2.7 cấu trúc js 11](#_Toc217987419)

[Ảnh 2.8 Ẩn menu 11](#_Toc217987420)

[Ảnh 2.9 Gọi trang trang khác 12](#_Toc217987421)

[Ảnh 2.10 Trang chủ 13](#_Toc217987422)

[Ảnh 2.11 khởi tạo banner 13](#_Toc217987423)

[Ảnh 2.12 danh mục sản phẩm nổi bật 14](#_Toc217987424)

[Ảnh 2.13 Trước khi đăng nhập 14](#_Toc217987425)

[Ảnh 2.14 Sau khi đăng nhập 14](#_Toc217987426)

[Ảnh 2.15 Trang products 15](#_Toc217987427)

[Ảnh 2.16 Chức năng tìm kiếm 15](#_Toc217987428)

[Ảnh 2.17 Hàm sắp xếp 16](#_Toc217987429)

[Ảnh 2.18 Thông tin sản phẩm 16](#_Toc217987430)

[Ảnh 2.19 Hàm mua ngay 17](#_Toc217987431)

[Ảnh 2.20 Giỏ hàng 17](#_Toc217987432)

[Ảnh 2.21 Trang giỏ hàng 18](#_Toc217987433)

[Ảnh 2.22 Trang thanh toán 18](#_Toc217987434)

[Ảnh 2.23 Đăng nhập 19](#_Toc217987435)

[Ảnh 2.24 Hàm đăng ký 20](#_Toc217987436)

[Ảnh 2.25 Hàm thêm sản phẩm 20](#_Toc217987437)

[Ảnh 2.26 Hàm xóa sản phẩm 21](#_Toc217987438)

[Ảnh 2.27 Class thanh toán 21](#_Toc217987439)

[Ảnh 2.28 Class đơn hàng 22](#_Toc217987440)

[Ảnh 2.29 Truyền, nhân dữ liệu hóa đơn 22](#_Toc217987441)

[Ảnh 2.30 Đăng nhập 23](#_Toc217987442)

[Ảnh 2.31 Đăng ký 23](#_Toc217987443)

[Ảnh 2.32 Bảng giỏ hàng 24](#_Toc217987444)

[Ảnh 2.33 Hóa đơn 25](#_Toc217987445)

[Ảnh 2.34 Hóa đơn chi tiết 26](#_Toc217987446)

[Ảnh 2.35 Sản phẩm 27](#_Toc217987447)

[Ảnh 2.36 khách hàng 28](#_Toc217987448)

[Ảnh 2.37. Sơ đồ tác nhân 29](#_Toc217987449)

[Ảnh 2.38 Class Diagram 31](#_Toc217987450)

[Ảnh 2.39 Sequence Diagram 32](#_Toc217987451)

[Ảnh 2.40 Minh Họa Order Builder 33](#_Toc217987452)

[Ảnh 2.41 Singleton CartService 34](#_Toc217987453)

[Ảnh 2.42 Kiểm tra khách hàng và giỏ hàng 35](#_Toc217987454)

[Ảnh 2.43 Thanh toán, kiểm tra và cập nhật kho hàng 36](#_Toc217987455)

[Ảnh 2.44 Xây dựng phương thức thanh toán 37](#_Toc217987456)

[Ảnh 2.45 Cơ sở dữ liệu 38](#_Toc217987457)

[Ảnh 2.46 Thư mục chạy dự án 39](#_Toc217987458)

[Ảnh 2.47 Chạy dự án 39](#_Toc217987459)

[Ảnh 2.48 Sau khi chạy thành công 39](#_Toc217987460)

[Ảnh 3.1 Kết quả 46](#_Toc217987461)

[Ảnh 3.2 kết quả pha chế 48](#_Toc217987462)

**DANH MỤC BẢNG**

[Table 1 Cấu trúc dự án 9](#_Toc217619530)

[Table 2 Đặc tả use case 28](#_Toc217619531)

**Lời mở Đầu**

Trong bối cảnh cuộc Cách mạng công nghiệp 4.0 đang diễn ra mạnh mẽ, thương mại điện tử đã và đang trở thành một xu hướng tất yếu, đóng vai trò quan trọng trong hoạt động kinh doanh của các doanh nghiệp. Việc xây dựng các website hoặc ứng dụng bán hàng không chỉ giúp doanh nghiệp mở rộng thị trường, tiếp cận khách hàng nhanh chóng mà còn góp phần tối ưu hóa quy trình quản lý, nâng cao hiệu quả kinh doanh và chất lượng dịch vụ khách hàng.

Xuất phát từ yêu cầu thực tiễn đó, nhóm/chúng em thực hiện bài tập lớn với đề tài: “Xây dựng website/ứng dụng bán hàng”. Mục tiêu của bài tập là vận dụng các kiến thức đã học về phân tích – thiết kế hệ thống, lập trình hướng đối tượng và Design Pattern để xây dựng một ứng dụng bán hàng có cấu trúc rõ ràng, dễ mở rộng và dễ bảo trì.

Ứng dụng được thiết kế và cài đặt theo tư duy lập trình hướng đối tượng kết hợp với Design Pattern nhằm nâng cao tính linh hoạt và khả năng tái sử dụng của mã nguồn. Trong đó, bài tập áp dụng ít nhất hai Design Pattern, bao gồm:

* Singleton Pattern: đảm bảo chỉ tồn tại một thể hiện duy nhất của một số lớp quản lý quan trọng (ví dụ: quản lý kết nối cơ sở dữ liệu, cấu hình hệ thống).
* Builder Pattern: hỗ trợ việc xây dựng các đối tượng phức tạp (ví dụ: đơn hàng, sản phẩm) theo từng bước, giúp mã nguồn rõ ràng và dễ mở rộng.

Thông qua bài tập lớn này, nhóm/chúng em không chỉ củng cố kiến thức lý thuyết mà còn rèn luyện kỹ năng phân tích, thiết kế và xây dựng một hệ thống bán hàng hoàn chỉnh, tạo nền tảng cho việc phát triển các ứng dụng thực tế trong   
tương lai.

# CƠ SỞ LÝ THUYẾT

## Kiến trúc Hệ thống (Fullstack Architecture)

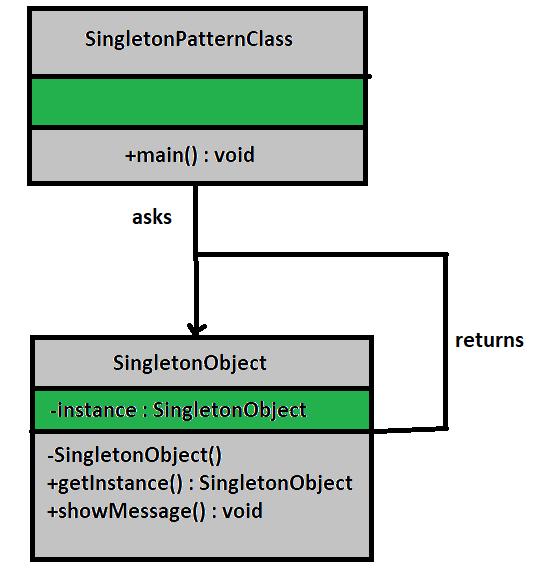
* Mô hình Client-Server: Ứng dụng được xây dựng theo mô hình tách biệt giữa Frontend (React) và Backend (Spring Boot). Hai phần giao tiếp với nhau thông qua giao thức HTTP và định dạng dữ liệu JSON.
* Kiến trúc Phân tầng (Layered Architecture): Backend được tổ chức thành các tầng: Controller, Service, Repository và Entity giúp tách biệt mã nguồn và dễ dàng bảo trì.

## Các Công nghệ Sử dụng

* Spring Boot: Framework chính phía Backend, hỗ trợ quản lý cấu hình và khởi tạo ứng dụng nhanh chóng thông qua file Run.java và @SpringBootApplication.
* ReactJS: Thư viện phía Frontend dùng để xây dựng giao diện người dùng dựa trên Component, giúp tăng tốc độ phản hồi của trang web.
* MySQL: Hệ quản trị cơ sở dữ liệu quan hệ dùng để lưu trữ dữ liệu sản phẩm, người dùng và đơn hàng.

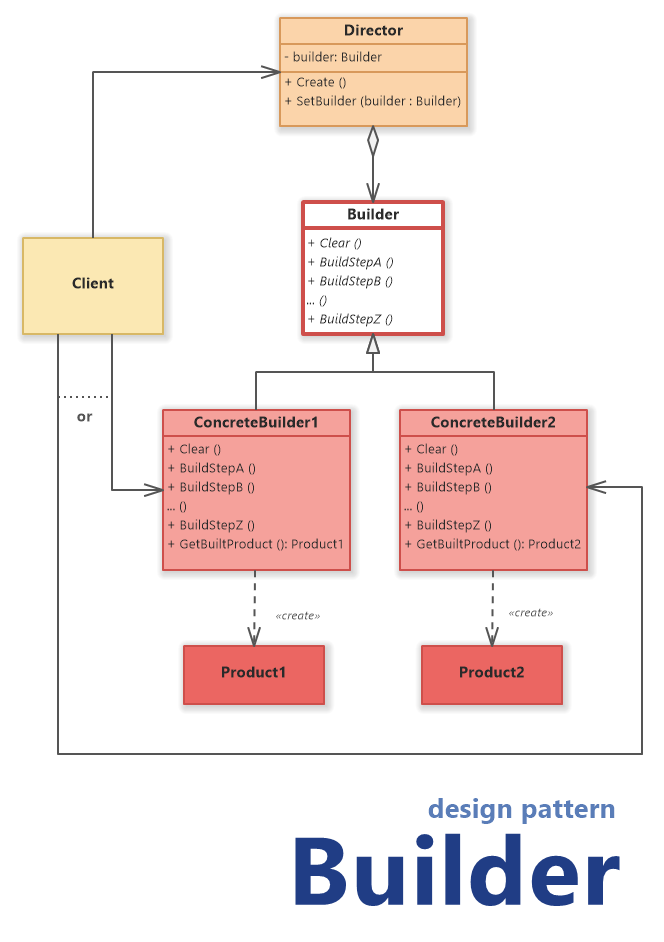
## Các Design Pattern Áp dụng

* Singleton Pattern (Bắt buộc): Được áp dụng mặc định bởi Spring Framework để quản lý các Bean như CartService, ProductService. Mỗi Service chỉ có một thực thể duy nhất hoạt động trong hệ thống.



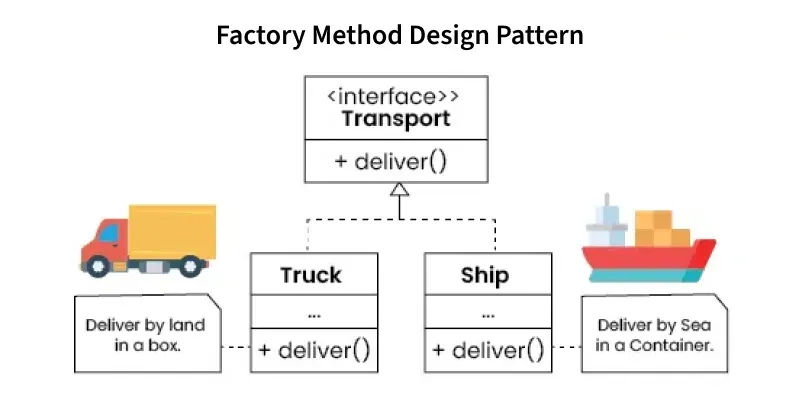
Ảnh . Minh họa Singleton pattern

* Builder Pattern (Bắt buộc): Được sử dụng để khởi tạo các đối tượng phức tạp như Order hoặc OrderDetail, giúp mã nguồn sạch hơn và tránh việc sử dụng quá nhiều Constructor.



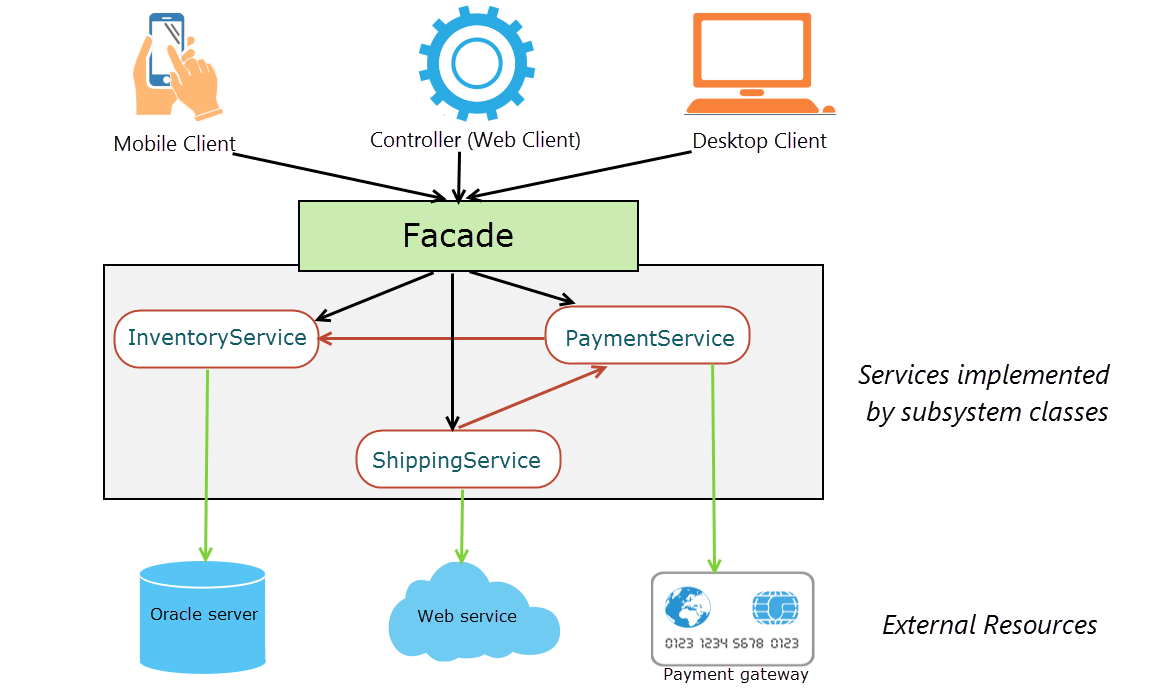
Ảnh . Minh họa Builder pattern

* Factory Pattern: Triển khai tại PaymentFactory, giúp hệ thống tự động quyết định sử dụng CodPaymentService hay BankingPaymentService dựa trên yêu cầu của khách hàng.



Ảnh . Minh họa Factory pattern

* Facade Pattern: Triển khai tại CheckoutFacade, giúp đơn giản hóa quy trình thanh toán phức tạp (gồm trừ kho, tạo đơn, xóa giỏ) thành một phương thức duy nhất cho Controller gọi đến.



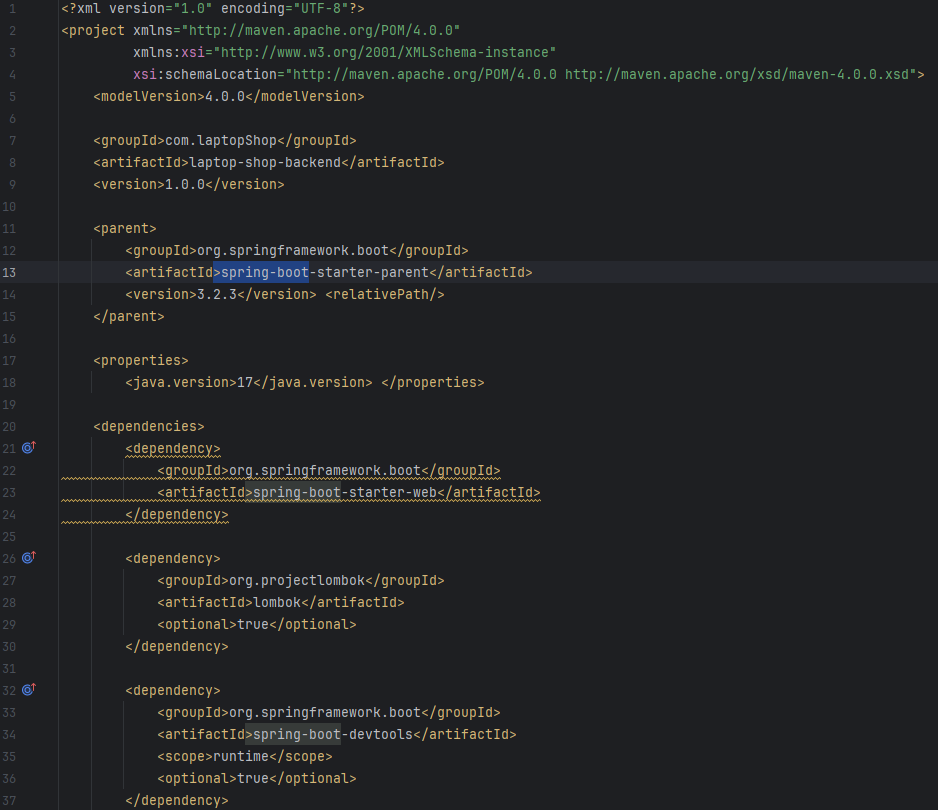
Ảnh . Minh họa Facade Pattern

# THỰC HÀNH THỰC NGHIỆM

## FrontEnd

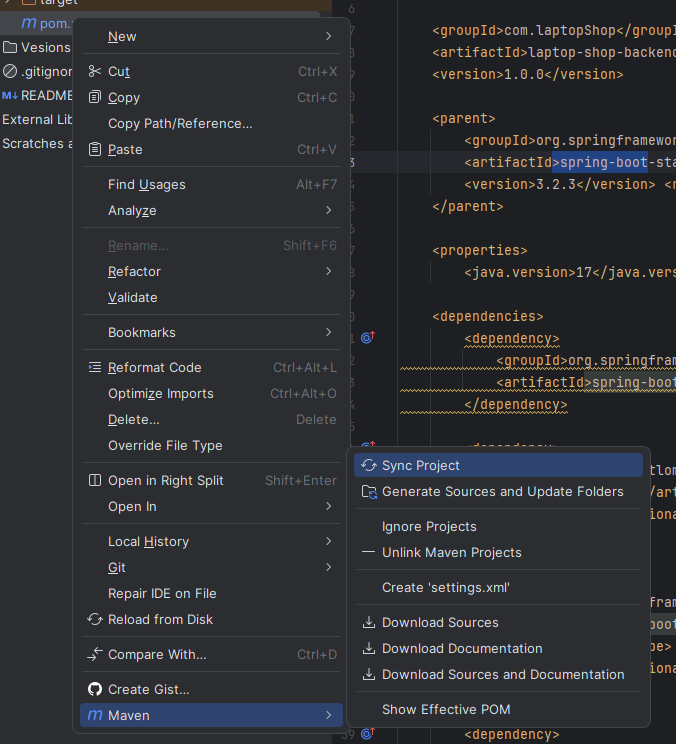
### Khởi tạo ứng dụng

Mở IntelliJ idea, tạo một dự án mới sau đó tạo một file tên là pom.xml để cài sử dụng spring boot.



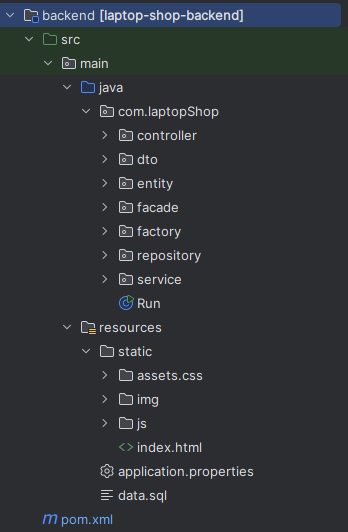
Ảnh . file pom.xml

Đồng bộ hóa dự án để có thể sự sụng Maven.



Ảnh . Đồ bộ hóa dự án

### Cấu trúc dự án

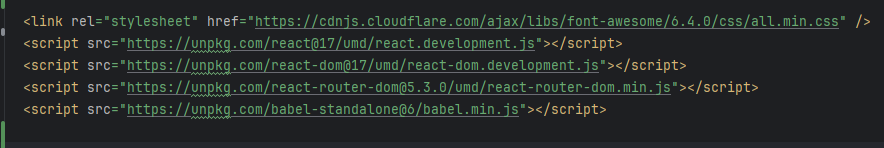


Ảnh . Cấu trúc dự án

| Tên thư mục/tiệp | Chức năng |
| --- | --- |
| controller | Nơi tiếp nhận yêu cầu từ người dùng |
| dto | Nơi truyền dữ liệu ra frontend |
| entity | Liên kết trực tiếp với database |
| facade | Đơn giản hóa hệ thống |
| factory | Khởi tạo đối tượng |
| repository | Nơi chứa các interface giao tiếp với database |
| service | Chứa các logic nghiệp vụ |
| Run | Chạy backend |
| static | Nơi khởi tạo frontend |

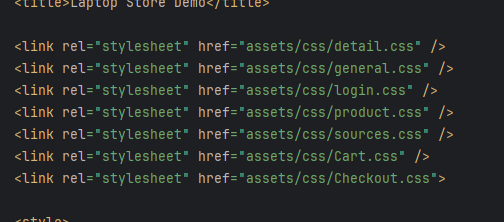
Table Cấu trúc dự án

### Trang indext



Ảnh . import dữ liệu

* <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.0/css/all.min.css" />: Cung cấp icon.
* <script src="https://unpkg.com/react@17/umd/react.development.js"></script>: thư viện react
* <script src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>: thư viện react dom
* <script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>: thư viện Babel



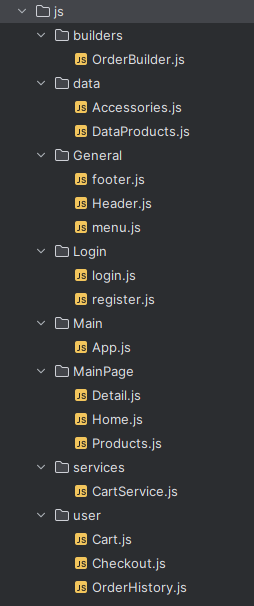
Ảnh . xây dự bố cục của trang web



Ảnh . khởi tạo trang web và import dữ liệu

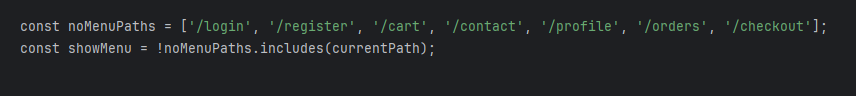
### App

* Cấu trúc js



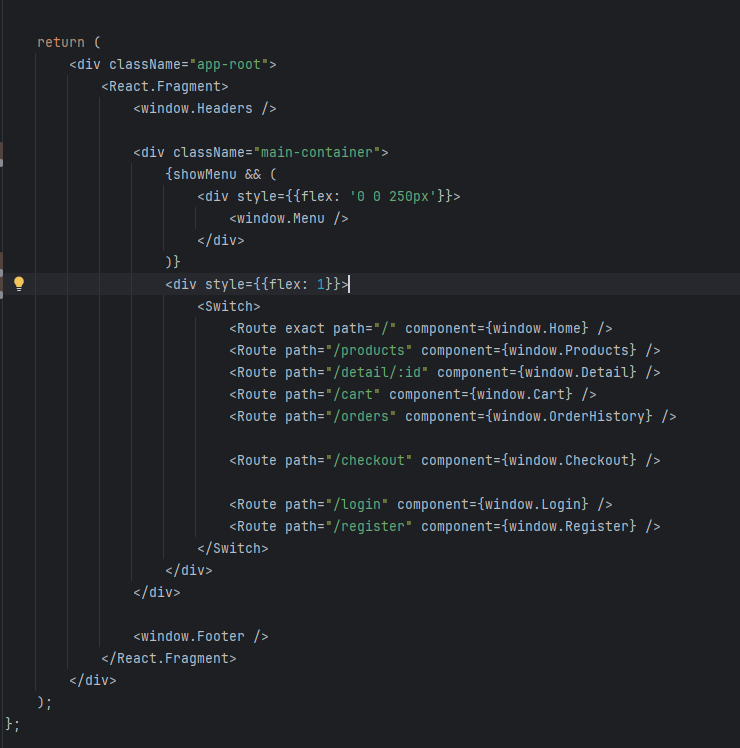
Ảnh . cấu trúc js

Ẩn danh mục sản phẩm ở một vài trang



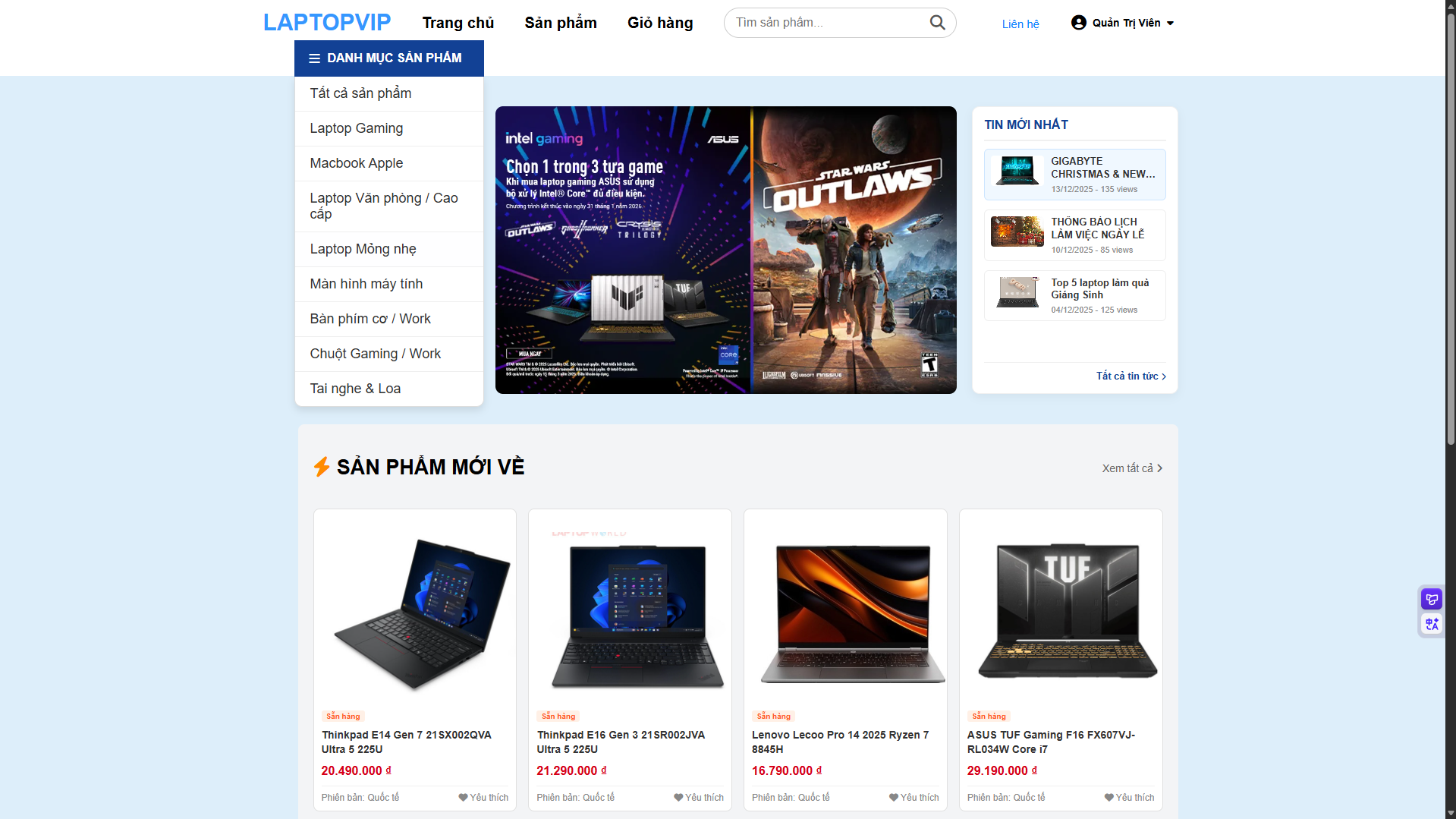
Ảnh . Ẩn menu

* Gọi trang trang khác



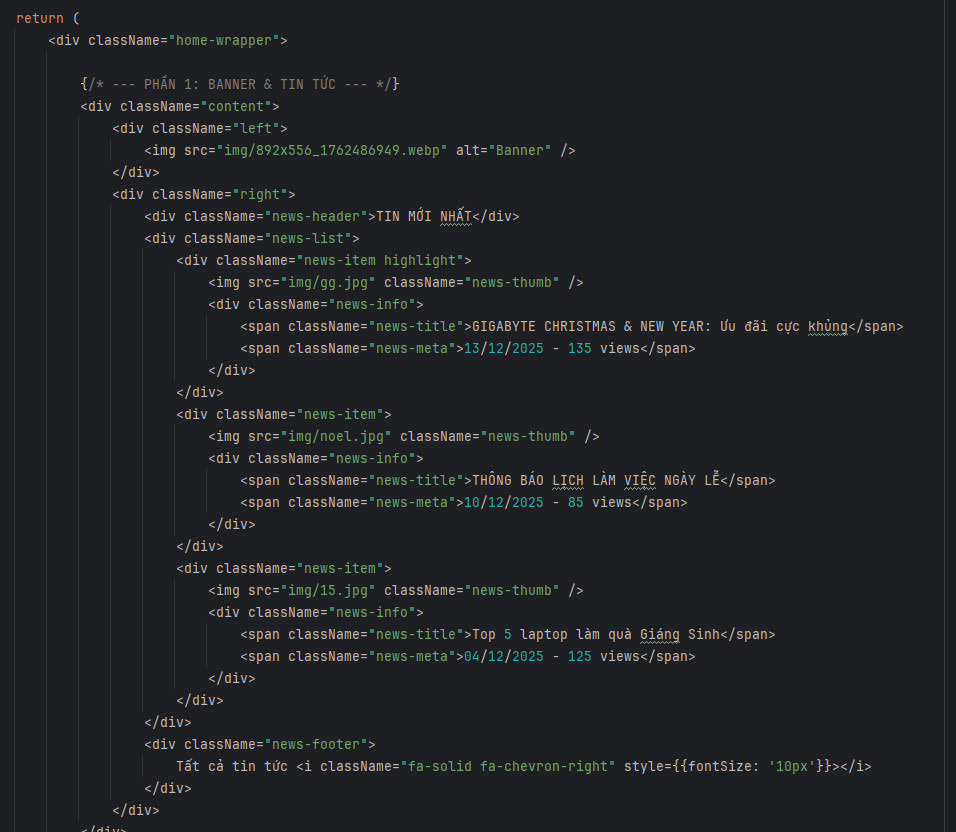
Ảnh . Gọi trang trang khác

### Trang chủ

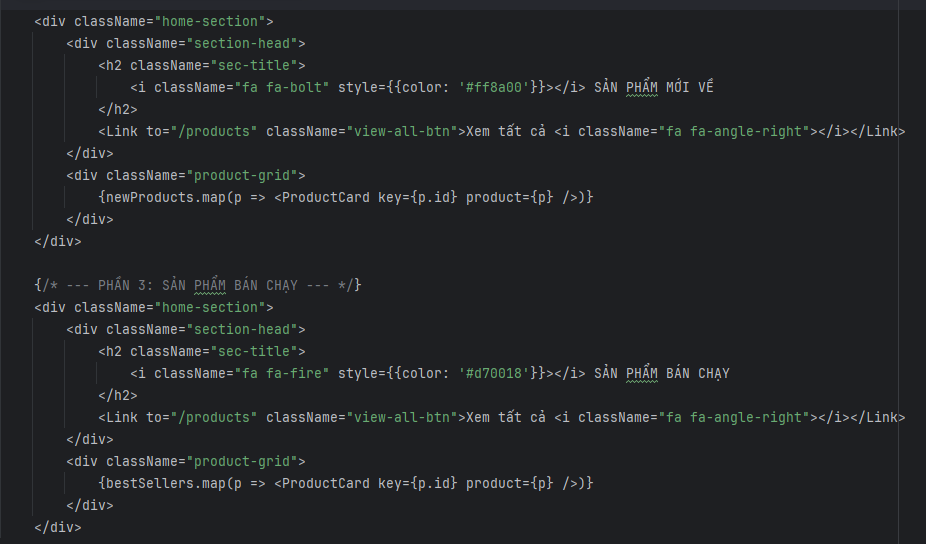


Ảnh . Trang chủ

* Tạo banner và tin tức

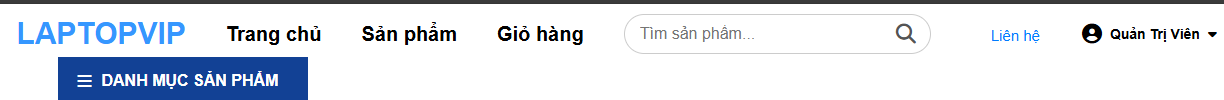


Ảnh . khởi tạo banner



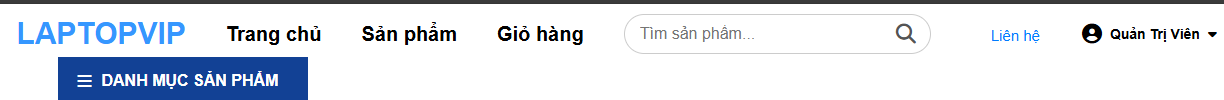
Ảnh . danh mục sản phẩm nổi bật

* Trước khi đăng nhâp



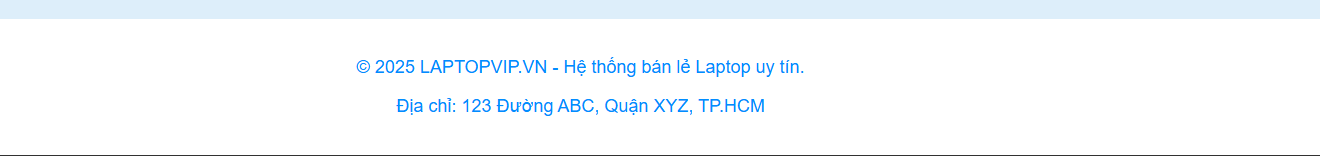
Ảnh . Trước khi đăng nhập

* Sau khi đăng nhập



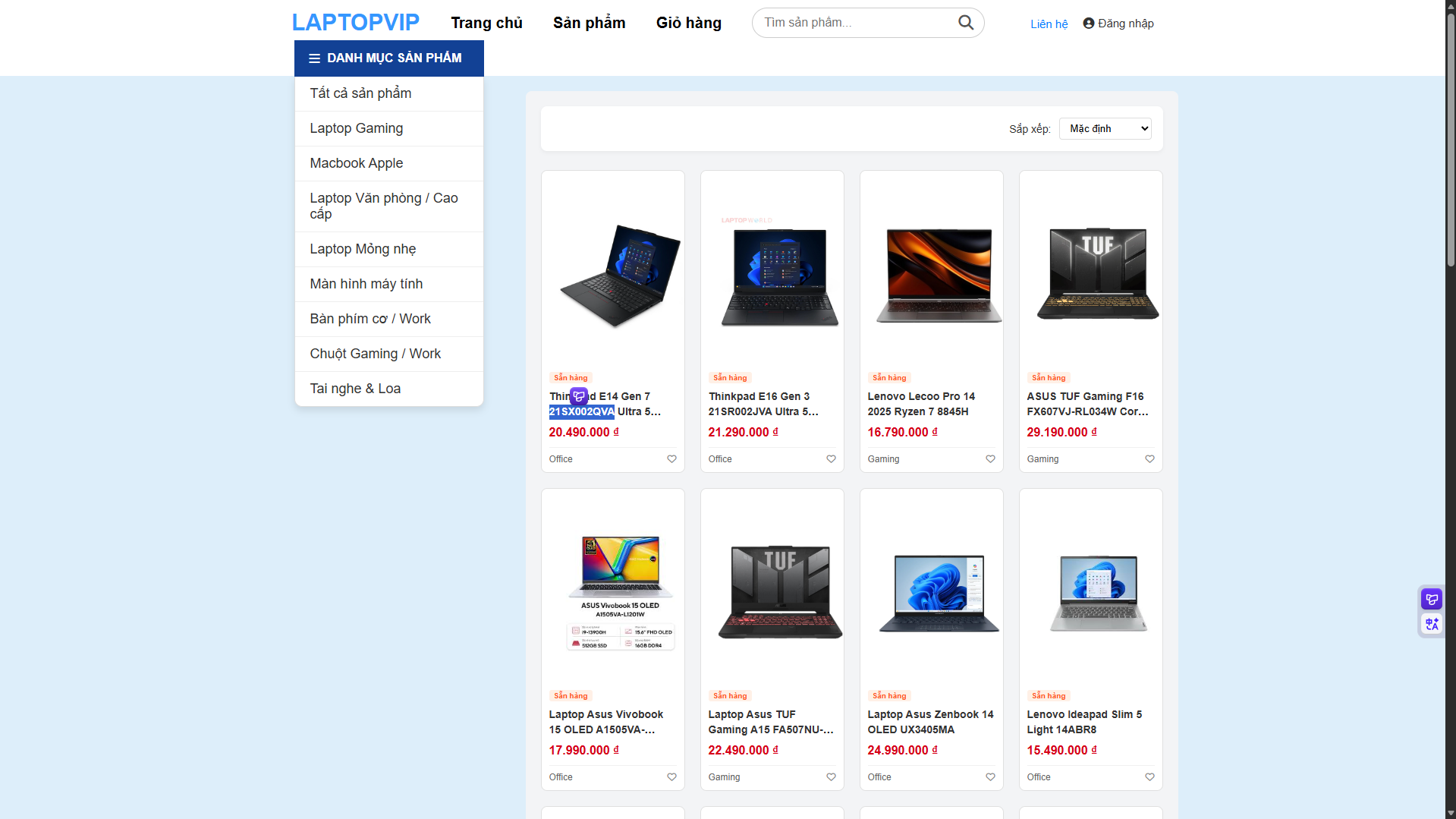
Ảnh . Sau khi đăng nhập

* Footer



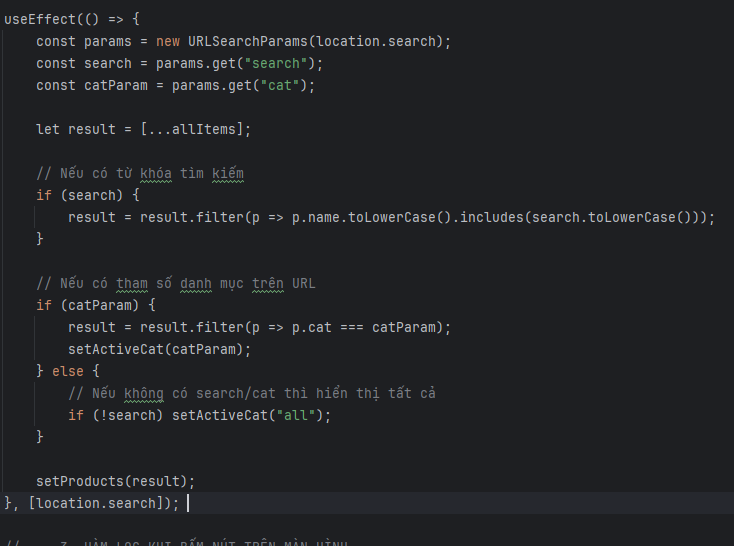
### Product

* Hình ảnh



Ảnh . Trang products

* Hàm sử lý tìm kiếm



Ảnh . Chức năng tìm kiếm

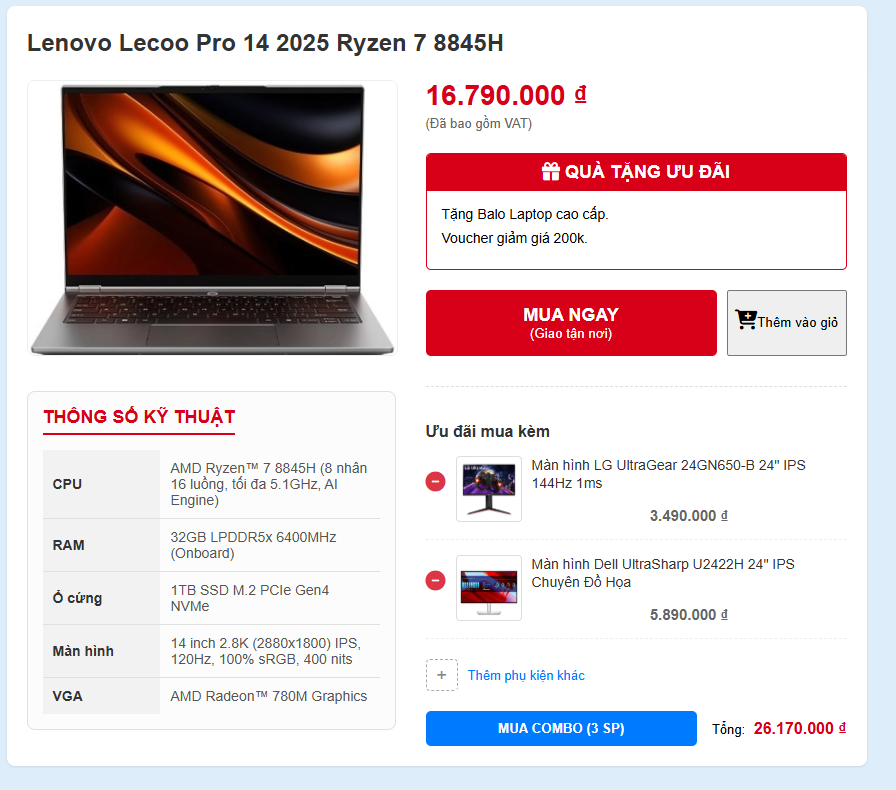
* Hàm Sắp xếp sản phẩm



Ảnh . Hàm sắp xếp

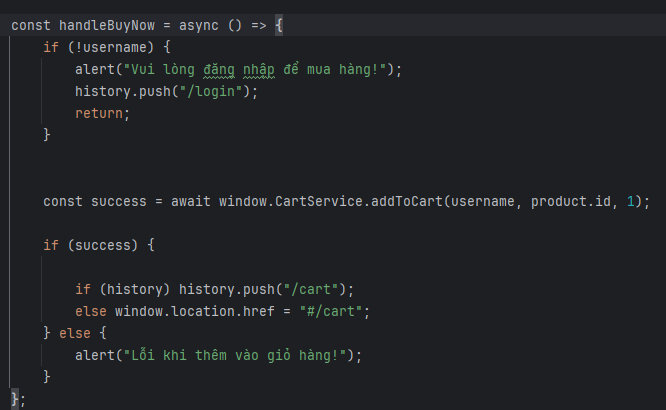
### Detail

* Trang thông tin sản phẩm



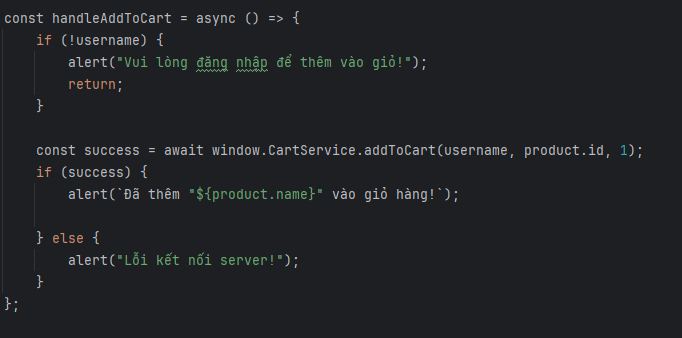
Ảnh . Thông tin sản phẩm

* Mua ngay



Ảnh . Hàm mua ngay

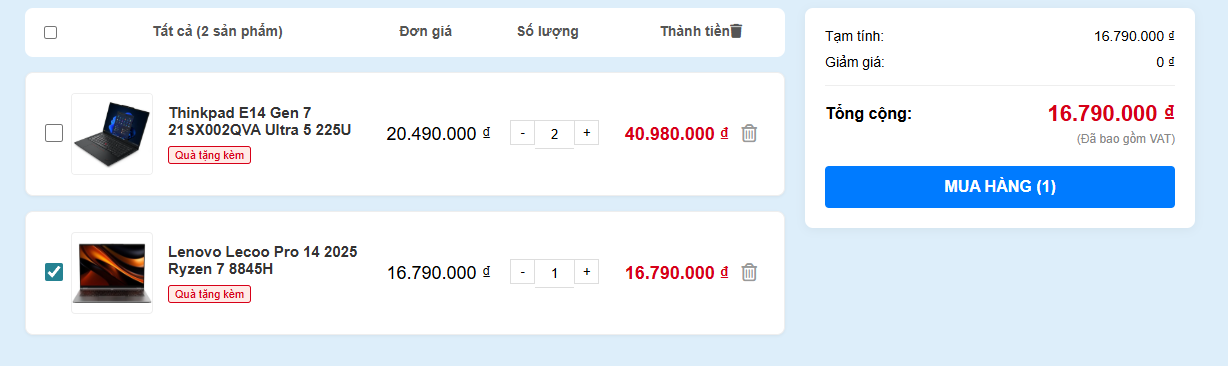
* Thêm vào giỏ hàng



Ảnh . Giỏ hàng

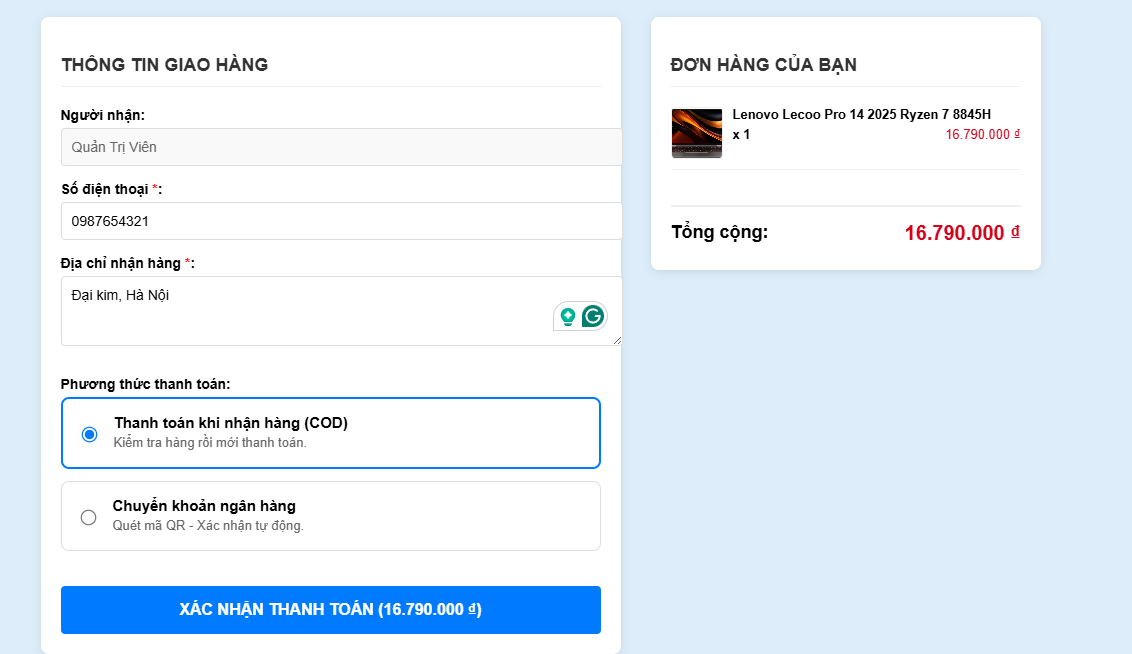
### Thanh toán

* Trang giỏ hàng



Ảnh . Trang giỏ hàng

* Trang Thanh toán



Ảnh . Trang thanh toán

## BackEnd

### Kết nối với mysql

Tạo file application.properties với nội dung:

spring.datasource.url=jdbc:mysql://localhost:3306/laptop\_shop

spring.datasource.username=tên user của bạn

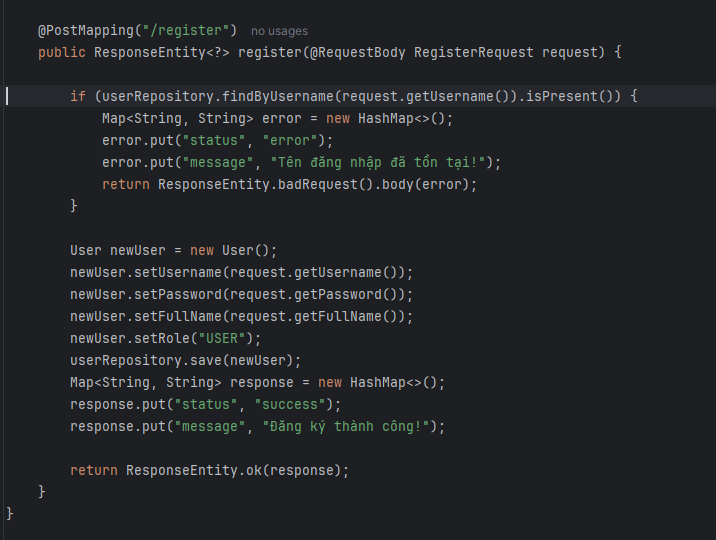
spring.datasource.password=Mật khẩu

### Đăng nhập đăng ký (AuthController.java)



Ảnh . Đăng nhập

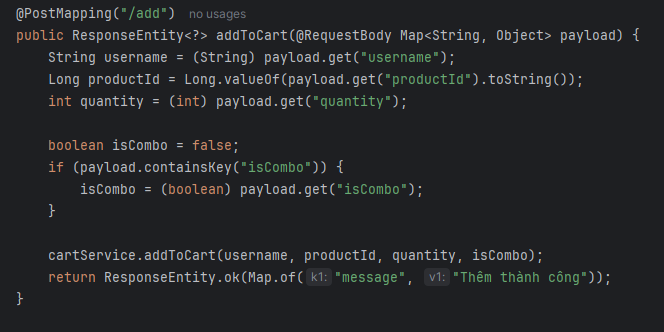
* Đăng ký



Ảnh . Hàm đăng ký

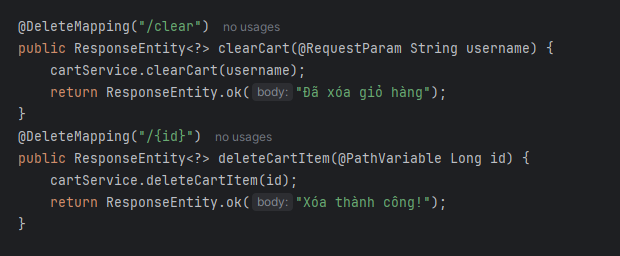
### Giỏ hàng (CartController.java)

* Thêm sản phẩm



Ảnh . Hàm thêm sản phẩm

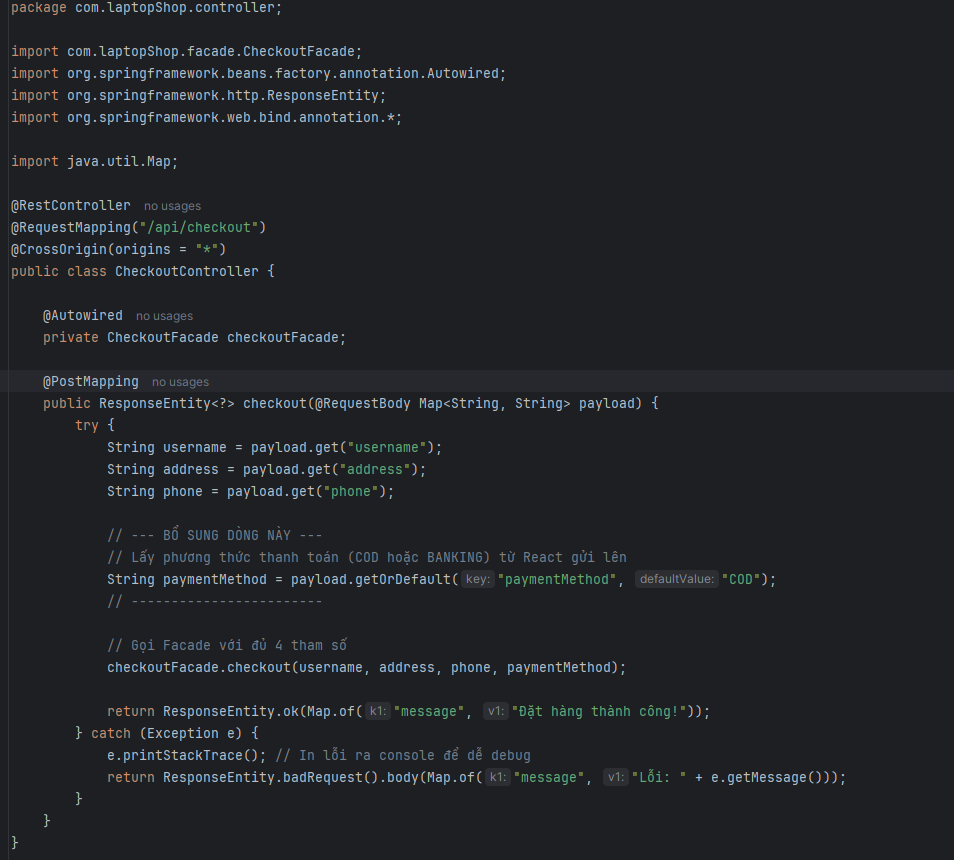
* Xóa sản phẩm khỏi giỏ hàng



Ảnh . Hàm xóa sản phẩm

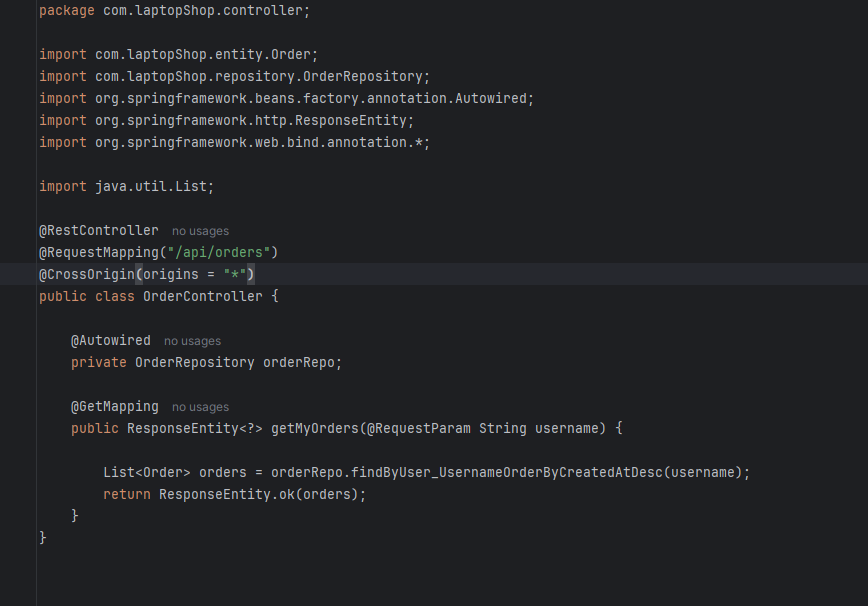
### Thanh toán

Class thanh toán



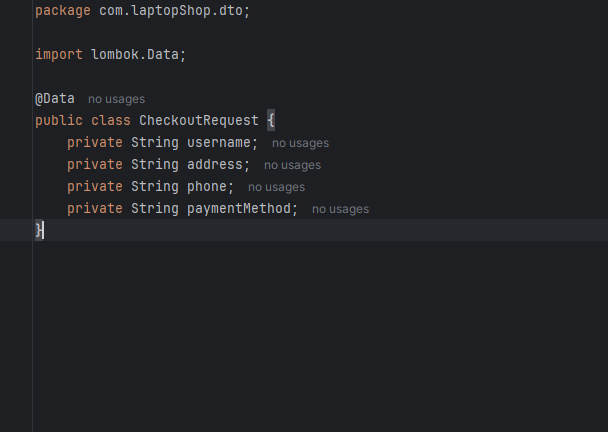
Ảnh . Class thanh toán

### Đơn hàng đã mua

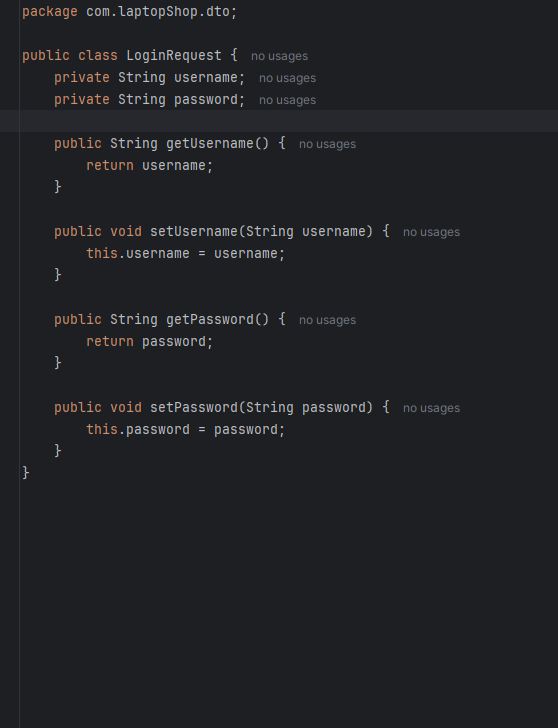


Ảnh . Class đơn hàng

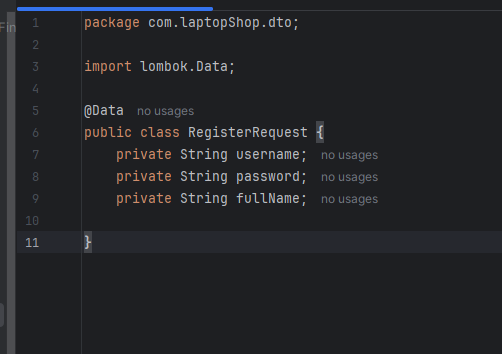
### Truyền/nhận dữ liệu với frontend



Ảnh . Truyền, nhân dữ liệu hóa đơn

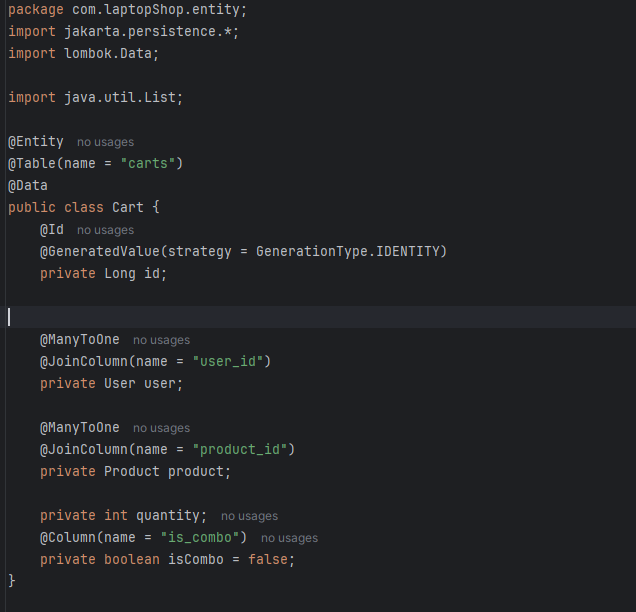


Ảnh . Đăng nhập

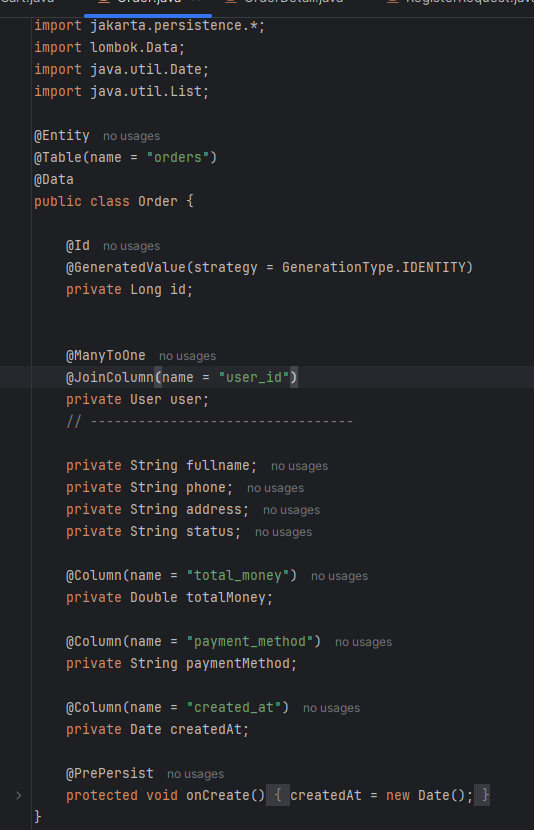


Ảnh . Đăng ký

### Lấy dữ liệu từ bảng trong cơ sở dữ liệu



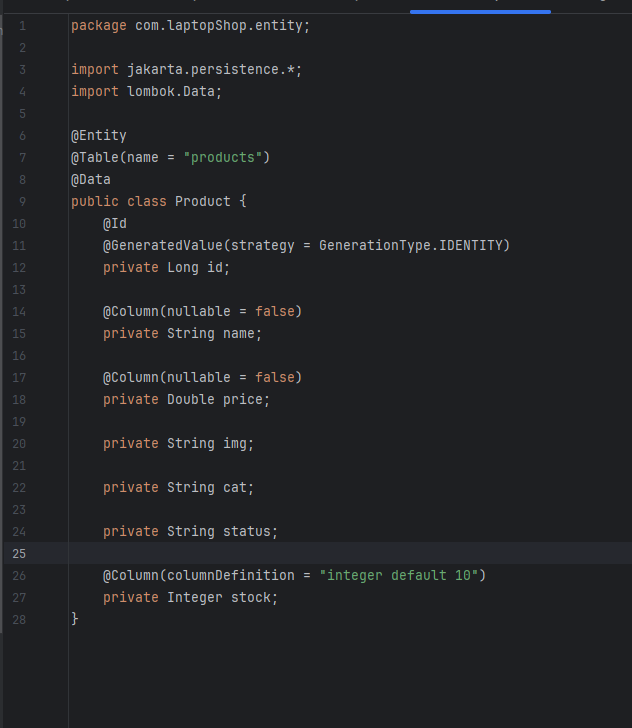
Ảnh . Bảng giỏ hàng



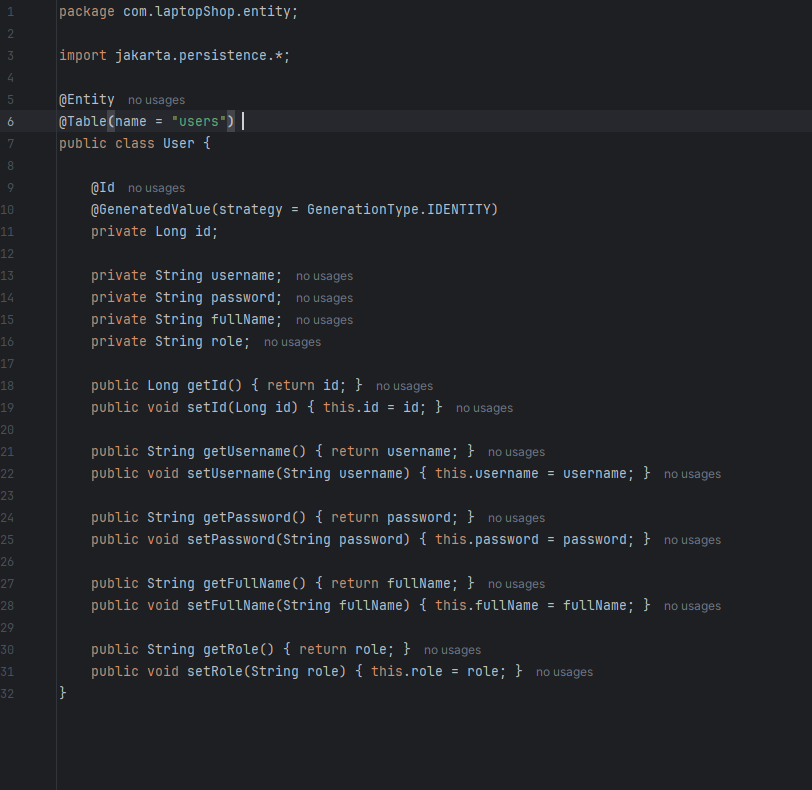
Ảnh . Hóa đơn



Ảnh . Hóa đơn chi tiết



Ảnh . Sản phẩm



Ảnh . khách hàng

## Các sơ đồ

### Sơ đồ Ca sử dụng (Use Case Diagram)

#### Các tác nhân (Actors)



Ảnh .. Sơ đồ tác nhân

|  |  |
| --- | --- |
| **TÁC NHÂN** | **MÔ TẢ** |
| **Guest** (Khách vãng lai) | Người dùng chưa đăng nhập, chỉ có thể xem thông tin cơ bản. |
| **Customer** (Khách hàng) | Người dùng đã đăng nhập, có quyền quản lý giỏ hàng và thực hiện thanh toán. |

Table Đặc tả use case

#### Danh sách Use Case

* **Nhóm Guest:**
  + **Đăng ký** (Register): Tạo tài khoản mới để tham gia hệ thống.
  + **Đăng nhập** (Login): Xác thực danh tính để cấp quyền truy cập các tính năng nâng cao.
  + **Xem sản phẩm** (View Products): Xem danh sách laptop và thông số kỹ thuật chi tiết.
* **Nhóm Customer** (Bao gồm các quyền của Guest):
  + **Quản lý giỏ hàng** (Manage Cart): Thêm/Xóa sản phẩm, cập nhật số lượng và chọn combo quà tặng.
  + **Đặt hàng** (Checkout): Cung cấp thông tin giao nhận và xác nhận thanh toán.
  + **Đăng xuất** (Logout): Hủy phiên làm việc hiện tại.

**1.1.3. Mô tả quan hệ**

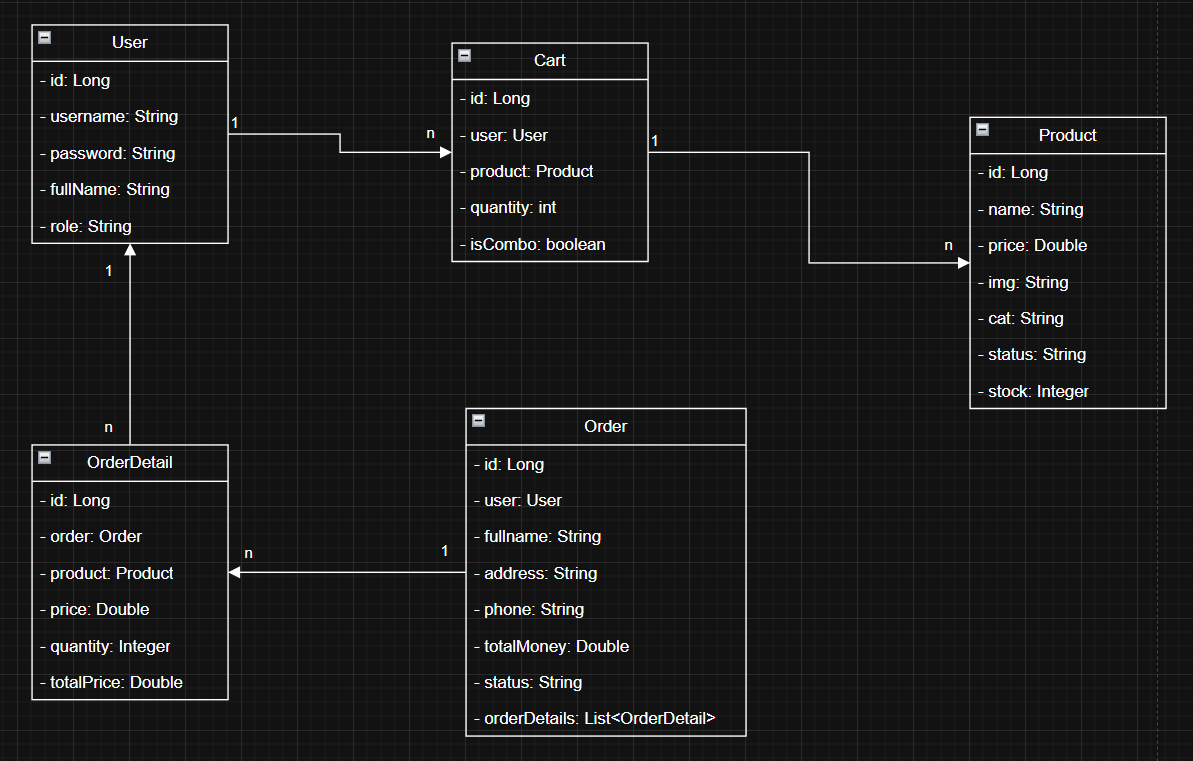
* **Guest** ---> (Register), (Login), (View Products).
* **Customer** ---> (Manage Cart), (Checkout), (Logout).
* **(Checkout)** ---> <<include>> ---> **(Login)** (Hệ thống bắt buộc đăng nhập để xác định định danh người mua).

#### Mô tả quan hệ (Vẽ sơ đồ)

### Sơ đồ Lớp (Class Diagram)

Đây là các bảng trong Database, quan hệ như sau:

* 1. **User**
* **Attributes**: id, username, password, fullName, role.
* **Quan hệ**: 1 User có nhiều Order (1..n), 1 User có nhiều CartItem (1..n).
* 2. **Product**
* **Attributes**: id, name, price, description, image, stock.
* **Quan hệ**: 1 Product nằm trong nhiều OrderDetail (1..n), 1 Product nằm trong nhiều CartItem (1..n).
* **3. Cart (Giỏ hàng)**
* **Attributes**: id, quantity.
* **Quan hệ**: Chứa 1 User và 1 Product (Many-to-One).
* **4. Order (Đơn hàng)**
* **Attributes**: id, address, phone, totalMoney, status, date.
* **Quan hệ**: Thuộc về 1 User, Chứa nhiều OrderDetail.
* **5. OrderDetail (Chi tiết đơn hàng)**
* **Attributes**: id, price, quantity, totalPrice.
* **Quan hệ**: Thuộc về 1 Order, Chứa 1 Product.



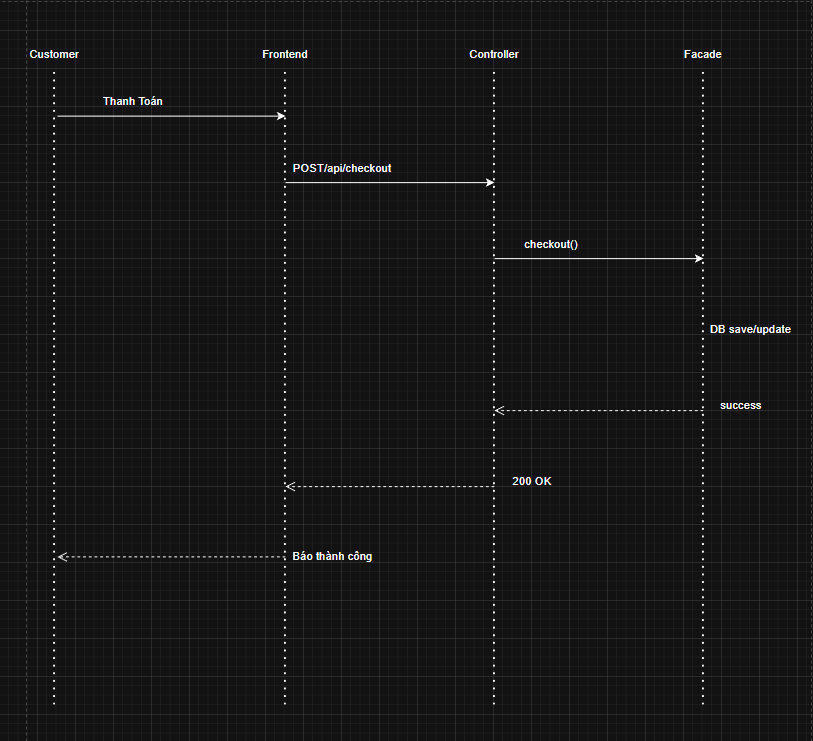
Ảnh . Class Diagram

### Sơ đồ Tuần tự (Sequence Diagram)

Đây là sơ đồ tuần tự (Sequence Diagram) rút gọn cho quy trình **Đặt hàng (Checkout)** dưới dạng hình ảnh để bạn dễ dàng đưa vào bài thuyết trình:

**Tóm tắt các bước trong ảnh:**

* **Actor (Khách hàng)**: Thực hiện thao tác nhấn nút đặt hàng trên giao diện React.
* **Frontend**: Gửi yêu cầu POST chứa thông tin đơn hàng (đã được xử lý qua OrderBuilder) đến Server.
* **Controller**: Tiếp nhận dữ liệu từ API và gọi đến lớp Facade để xử lý logic.
* **Facade (Xử lý DB)**: Đây là nơi thực hiện các nghiệp vụ quan trọng như lấy giỏ hàng, tạo đơn hàng, trừ số lượng tồn kho và xóa giỏ hàng.
* **Phản hồi thành công**: Kết quả được trả ngược về từ Facade qua Controller để Frontend hiển thị thông báo cho người dùng.



Ảnh . Sequence Diagram

Trong bối cảnh phát triển phần mềm hiện đại, việc áp dụng các mẫu thiết kế (Design Pattern) không chỉ là một kỹ thuật lập trình mà còn là giải pháp tối ưu để giải quyết các bài toán về cấu trúc hệ thống. Với dự án "Xây dựng website bán hàng LAPTOPVIP", nhóm chúng em đã tập trung triển khai các Design Pattern nhằm đảm bảo mã nguồn tuân thủ các nguyên lý thiết kế hướng đối tượng, từ đó nâng cao tính linh hoạt và khả năng bảo trì.

Cụ thể, hệ thống đã tích hợp hai mẫu thiết kế cốt lõi là **Builder Pattern** và **Singleton Pattern**, phục vụ cho các module quan trọng như quản lý đơn hàng và dịch vụ giỏ hàng.

### Builder

**a. Lý do áp dụng:** Trong một ứng dụng thương mại điện tử, đối tượng Đơn hàng (Order) là một thực thể phức tạp với nhiều thuộc tính khác nhau như: thông tin khách hàng, địa chỉ giao hàng, số điện thoại, phương thức thanh toán và danh sách sản phẩm. Việc khởi tạo đối tượng này theo cách truyền thống thường dẫn đến các hàm khởi tạo (constructor) quá dài hoặc tạo ra các đối tượng có trạng thái không ổn định trong quá trình thiết lập. Builder Pattern cho phép chúng ta xây dựng đối tượng đơn hàng theo từng bước rõ ràng, giúp mã nguồn dễ đọc và tránh sai sót dữ liệu.

**b. Triển khai thực tế trong dự án:** Hệ thống triển khai lớp OrderBuilder ở phía Frontend để đóng gói logic xây dựng đơn hàng trước khi gửi yêu cầu về Backend.

**Cấu trúc kỹ thuật:** Lớp OrderBuilder sử dụng cơ chế **Fluent Interface** (Method Chaining), cho phép kết nối các phương thức thiết lập thành một chuỗi duy nhất.

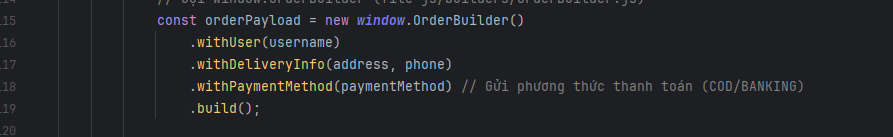
**Các phương thức chính:**

withUser(username): Thiết lập định danh người mua.

withDeliveryInfo(address, phone): Cung cấp thông tin nhận hàng.

withPaymentMethod(method): Lựa chọn hình thức thanh toán (COD hoặc Banking).

build(): Phương thức cuối cùng để kiểm tra tính hợp lệ (Validate) của dữ liệu trước khi hoàn tất đối tượng.



Ảnh . Minh Họa Order Builder

**c. Lợi ích đạt được:** Việc áp dụng Builder Pattern giúp tách biệt quá trình xây dựng đơn hàng phức tạp ra khỏi logic của trang Checkout.js, đồng thời cho phép hệ thống dễ dàng mở rộng thêm các thuộc tính mới cho đơn hàng trong tương lai mà không làm ảnh hưởng đến mã nguồn hiện tại.

### Singleton

**a. Lý do áp dụng:** Singleton Pattern được sử dụng cho các thành phần quản lý mà hệ thống chỉ cần duy nhất một thể hiện trong suốt vòng đời ứng dụng. Trong dự án này, các dịch vụ như quản lý giỏ hàng hay các controller xử lý API cần được duy trì duy nhất để đảm bảo tính đồng nhất của dữ liệu và tối ưu hóa tài nguyên hệ thống.

**b. Triển khai thực tế trong dự án:** Mẫu Singleton được áp dụng đồng bộ ở cả Frontend và Backend:

**Tại Frontend (ReactJS):** Dịch vụ giỏ hàng CartService được triển khai dưới dạng một đối tượng duy nhất và gắn trực tiếp vào thực thể toàn cục window. Điều này giúp các Component khác nhau như Detail.js (khi thêm hàng) và Cart.js (khi xem giỏ) cùng truy cập và chia sẻ chung một logic xử lý API duy nhất.



Ảnh . Singleton CartService

**c. Lợi ích đạt được:**

**Tính nhất quán:** Đảm bảo dữ liệu giỏ hàng và trạng thái đăng nhập được quản lý tập trung, tránh xung đột giữa các luồng xử lý khác nhau.

**Hiệu suất:** Giảm thiểu việc cấp phát bộ nhớ liên tục cho các đối tượng dịch vụ không thay đổi trạng thái, đặc biệt là các lớp xử lý nghiệp vụ nặng như CheckoutFacade.

Thông qua việc kết hợp Builder và Singleton, mã nguồn của dự án đã đạt được sự chuyên nghiệp trong thiết kế, đúng với mục tiêu đề ra trong việc vận dụng lý thuyết vào thực tiễn xây dựng ứng dụng thương mại điện tử.

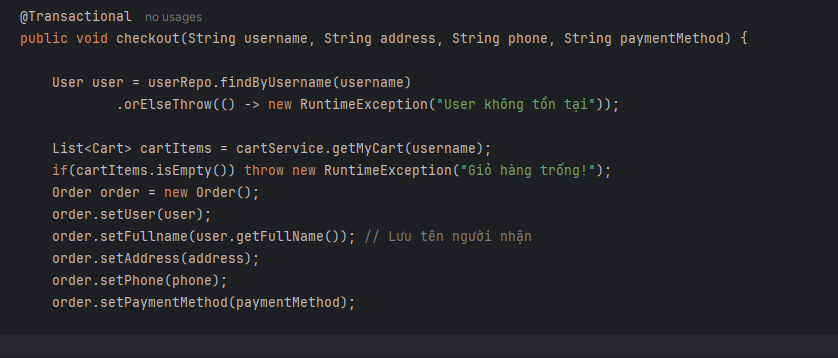
### Facade Pattern

a. Lý do áp dụng: Trong quy trình thanh toán (Checkout) của ứng dụng, hệ thống cần thực hiện một chuỗi các tác vụ phức tạp liên tiếp: xác thực người dùng, lấy thông tin giỏ hàng, tính toán tổng tiền, xử lý thanh toán, lưu đơn hàng, lưu chi tiết đơn hàng, trừ tồn kho và xóa giỏ hàng. Nếu để Controller gọi lẻ tẻ từng dịch vụ này, mã nguồn sẽ trở nên cồng kềnh, khó quản lý và phụ thuộc lẫn nhau quá nhiều (tightly coupled). Facade Pattern cung cấp một giao diện đơn giản hóa để che giấu sự phức tạp này.

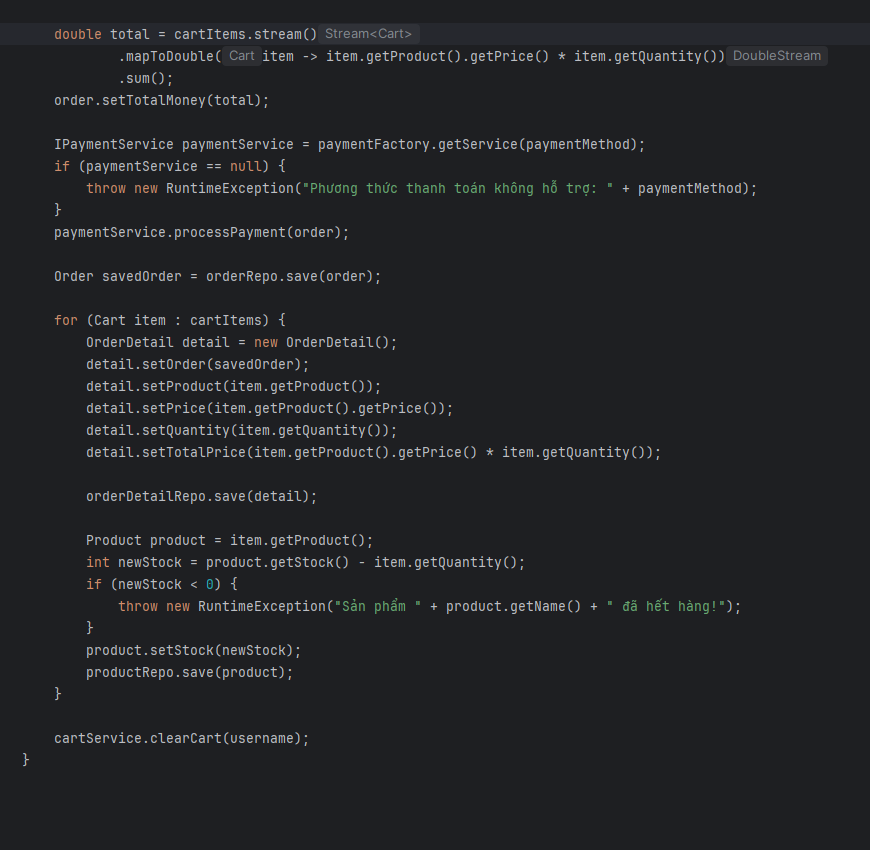
b. Triển khai thực tế trong dự án:

Vị trí: Lớp CheckoutFacade nằm trong package facade.

* Cấu trúc kỹ thuật: Lớp này đứng ra điều phối và gọi đến các Service con như UserService, CartService, OrderRepository, ProductRepository và PaymentFactory.
* Các phương thức chính: checkout(username, address, phone, paymentMethod): Đây là hàm duy nhất mà Controller cần gọi. Bên trong hàm này chứa toàn bộ logic nghiệp vụ từ lúc bắt đầu đặt hàng đến khi hoàn tất.



Ảnh . Kiểm tra khách hàng và giỏ hàng



Ảnh . Thanh toán, kiểm tra và cập nhật kho hàng

**c. Lợi ích đạt được:** Giúp CheckoutController trở nên cực kỳ gọn nhẹ và dễ đọc. Nó tách biệt hoàn toàn logic điều phối phức tạp ra khỏi tầng giao tiếp HTTP. Khi cần thay đổi quy trình đặt hàng (ví dụ: thêm bước gửi email xác nhận), ta chỉ cần sửa trong CheckoutFacade mà không ảnh hưởng đến Controller hay các Service con khác.

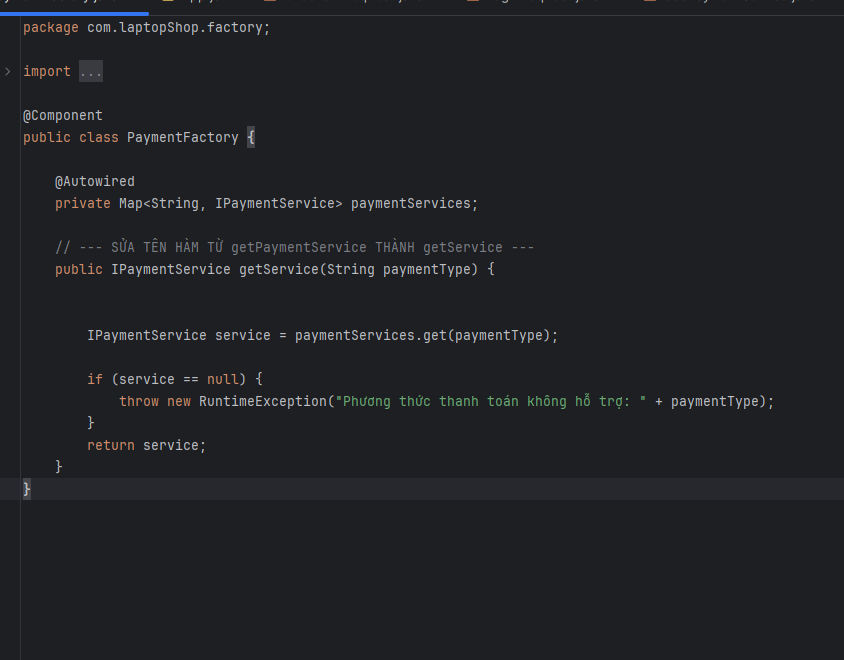
### Factory Method Pattern (Mẫu Nhà Máy)

**a.** Lý do áp dụng: Hệ thống hỗ trợ nhiều phương thức thanh toán khác nhau (COD, Chuyển khoản ngân hàng) và có thể mở rộng thêm trong tương lai (Ví điện tử, Thẻ quốc tế). Việc sử dụng các câu lệnh điều kiện if-else hoặc switch-case trực tiếp trong logic xử lý chính để khởi tạo đối tượng thanh toán sẽ vi phạm nguyên tắc Đóng/Mở (Open/Closed Principle) và làm mã nguồn khó bảo trì.

**b.** Triển khai thực tế trong dự án:

* Vị trí: Lớp PaymentFactory nằm trong package factory.
* Cấu trúc kỹ thuật: Sử dụng cơ chế Dependency Injection của Spring để quản lý các Bean dịch vụ thanh toán.
* Các phương thức chính**:**
* getService(paymentType): Nhận vào chuỗi ký tự (ví dụ "COD" hoặc "BANKING") và trả về đúng instance của IPaymentService tương ứng (là CodPaymentService hoặc BankingPaymentService).

**c. Lợi ích đạt được:** Tách biệt logic *khởi tạo* đối tượng thanh toán ra khỏi logic *sử dụng* đối tượng. Hệ thống dễ dàng mở rộng: nếu muốn thêm phương thức thanh toán MOMO, chỉ cần tạo class mới và đăng ký vào Factory mà không cần sửa đổi logic cốt lõi của tính năng đặt hàng.



Ảnh . Xây dựng phương thức thanh toán

### Repository Pattern

a. Lý do áp dụng: Việc truy cập trực tiếp vào cơ sở dữ liệu từ lớp Service (viết các câu lệnh SQL thô) sẽ khiến mã nguồn phụ thuộc chặt chẽ vào loại database đang sử dụng và khó kiểm thử (Unit Test). Repository Pattern tạo ra một lớp trung gian để trừu tượng hóa việc truy xuất dữ liệu.

**b.** Triển khai thực tế trong dự án:

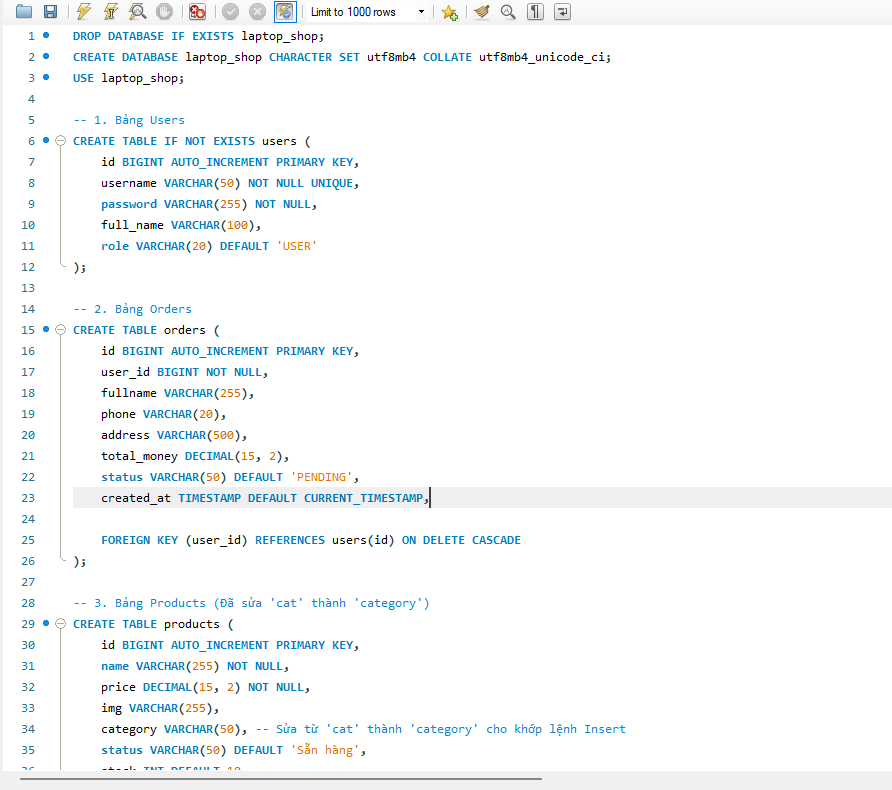
* Vị trí: Các Interface trong package repository (UserRepository, ProductRepository, OrderRepository...).
* Cấu trúc kỹ thuật: Sử dụng Spring Data JPA. Các interface này kế thừa từ JpaRepository.
* Các phương thức chính:
* Cung cấp sẵn các phương thức CRUD chuẩn: save(), findById(), findAll(), delete().
* Các phương thức truy vấn tùy biến (Derived Query Methods): findByUsername(), findByStatus().

**c**. Lợi ích đạt được: Tách biệt hoàn toàn logic nghiệp vụ (Business Logic) khỏi logic truy xuất dữ liệu (Data Access Logic). Lập trình viên không cần viết các câu lệnh SQL lặp đi lặp lại. Dễ dàng thay đổi hệ quản trị cơ sở dữ liệu (ví dụ từ MySQL sang PostgreSQL) mà ít ảnh hưởng đến code của ứng dụng nhất.

## Cách chạy chương trình

### Tạo cơ sở dữ liệu

Dán đoạn mã ở backend/src/main/sources/data.sql vào MySQL



Ảnh . Cơ sở dữ liệu

### Chạy chương trình

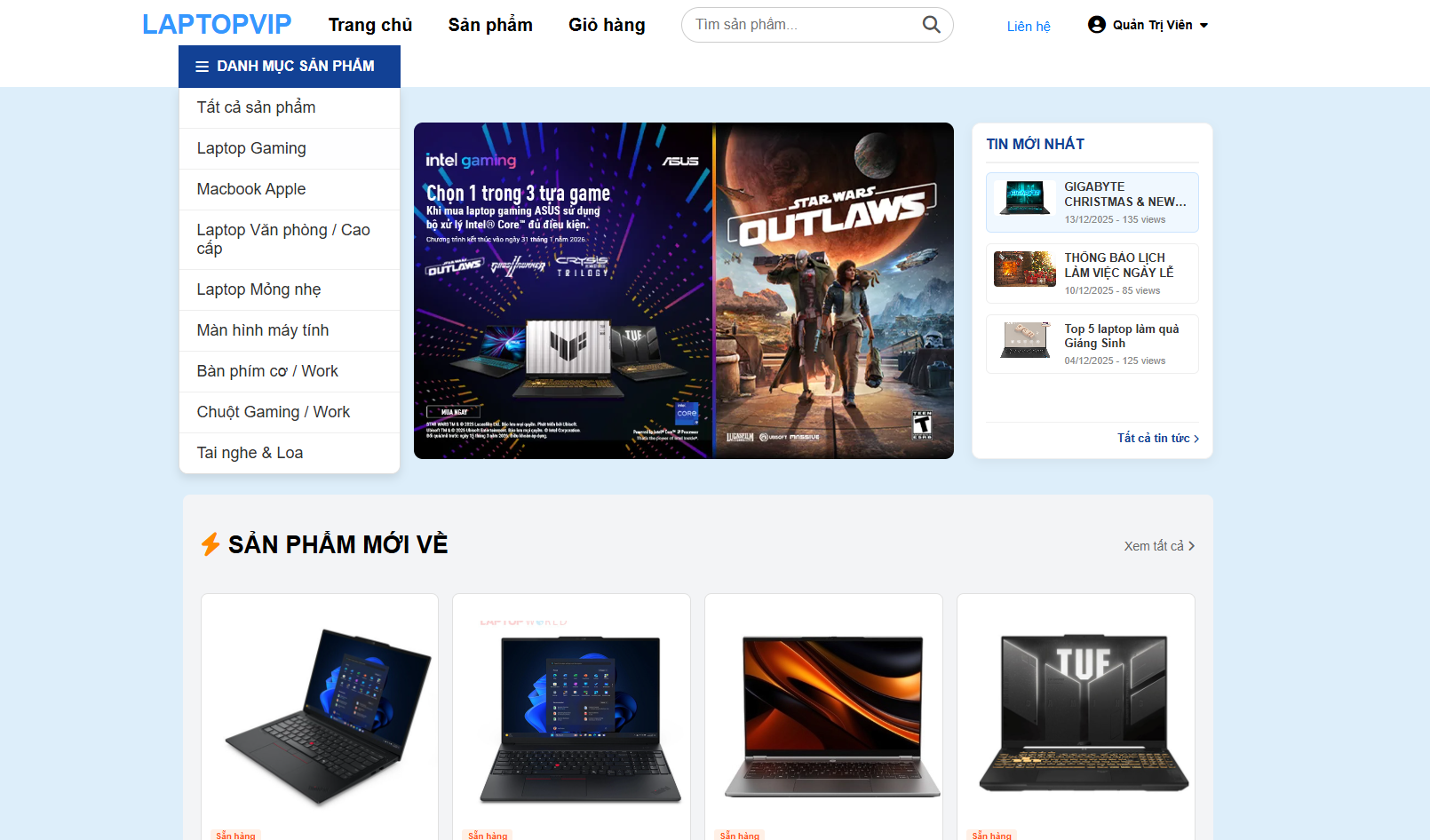
Vào thư mục vision và double click file Start.bat



Ảnh . Thư mục chạy dự án



Ảnh . Chạy dự án



Ảnh . Sau khi chạy thành công

# tìm hiểu thêm

## Template Method

### Giới thiệu chung

Template Method Pattern là một mẫu thiết kế hành vi (Behavioral Pattern) trong lập trình hướng đối tượng.

Mẫu thiết kế này định nghĩa bộ khung (skeleton) của một thuật toán trong một phương thức của lớp cha, đồng thời cho phép các lớp con cài đặt hoặc ghi đè một số bước cụ thể của thuật toán đó mà không làm thay đổi cấu trúc tổng thể.

Nói cách khác, Template Method giúp:

* Giữ nguyên trình tự xử lý của thuật toán
* Cho phép tùy biến chi tiết thực hiện thông qua kế thừa

### Ý nghĩa và Động cơ sử dụng

* Template Method Pattern ra đời nhằm giải quyết các vấn đề phổ biến sau:
* Nhiều lớp có quy trình xử lý giống nhau, chỉ khác ở một vài bước
* Code bị lặp lại ở nhiều lớp con
* Khó bảo trì khi logic chung thay đổi

**Ý nghĩa chính:**

Tái sử dụng mã nguồn (Code Reuse): gom phần logic chung vào lớp cha

Quản lý quy trình xử lý: đảm bảo các bước luôn chạy đúng thứ tự

Tùy biến có kiểm soát: lớp con chỉ được thay đổi những bước cho phép

Phân tách trách nhiệm rõ ràng:

* Lớp cha: điều khiển luồng thuật toán
* Lớp con: cài đặt chi tiết từng bước

### Kiến trúc và Thành phần

Template Method Pattern gồm các thành phần chính sau:

#### Abstract Class (Lớp trừu tượng)

* Định nghĩa Template Method
* Khai báo các bước của thuật toán dưới dạng:
* Phương thức trừu tượng (Primitive Operations)
* Phương thức móc (Hook Methods)

#### Template Method (Phương thức khuôn mẫu)

* Là phương thức chính
* Xác định trình tự cố định của các bước
* Thường được khai báo là final (trong Java) để lớp con không thể thay đổi luồng

#### Primitive Operations (Các bước nguyên thủy)

* Là các phương thức trừu tượng
* Bắt buộc lớp con phải cài đặt
* Thể hiện phần khác nhau giữa các biến thể

#### Hook Methods (Phương thức móc)

* Có cài đặt mặc định (thường rỗng)
* Lớp con có thể ghi đè hoặc không
* Cung cấp các điểm mở rộng linh hoạt

#### Concrete Class (Lớp cụ thể)

* Kế thừa từ Abstract Class
* Triển khai các bước chi tiết của thuật toán
* Không được thay đổi Template Method

### Đặc điểm của Template Method Pattern

* Sử dụng kế thừa để mở rộng hành vi
* Kiểm soát chặt chẽ trình tự thực hiện thuật toán
* Cho phép tùy chỉnh hành vi mà không ảnh hưởng đến cấu trúc
* Phù hợp để xây dựng framework hoặc thư viện dùng chung

### Ưu điểm và Nhược điểm

**Ưu điểm**

* Giảm trùng lặp mã nguồn
* Dễ bảo trì khi logic chung thay đổi
* Tăng tính nhất quán của quy trình xử lý
* Hỗ trợ đa hình một cách hiệu quả

**Nhược điểm**

* Phụ thuộc nhiều vào kế thừa
* Khó mở rộng nếu muốn thay đổi hoàn toàn thuật toán
* Dễ gây phức tạp nếu hệ thống có nhiều lớp con
* Có thể vi phạm nguyên lý Liskov nếu ghi đè không hợp lý

### Khi nào nên sử dụng Template Method Pattern

Sử dụng Template Method Pattern khi:

* Có một thuật toán gồm nhiều bước cố định
* Muốn cho phép lớp con tùy chỉnh từng bước cụ thể
* Cần một triển khai chung duy nhất cho bộ khung thuật toán
* Muốn đặt hành vi chung ở lớp cha để tránh lặp code
* Xây dựng các quy trình chuẩn, ví dụ:
* Quy trình pha chế đồ uống (Trà, Cà phê)
* Quy trình xử lý tài liệu (PDF, CSV)
* Vòng đời ứng dụng (Khởi tạo – Chạy – Kết thúc)

### Ví dụ về quy trình pha chế đồ uống (Trà, Cà phê) bằng python.

*from abc import ABC, abstractmethod*

*class DoUong(ABC):*

*def pha\_do\_uong(self):*

*self.dun\_nuoc()*

*self.pha\_che()*

*self.rot\_vao\_coc()*

*self.them\_gia\_vi()*

*def dun\_nuoc(self):*

*print("Dun nuoc soi")*

*@abstractmethod*

*def pha\_che(self):*

*pass*

*def rot\_vao\_coc(self):*

*print("Rot do uong vao coc")*

*def them\_gia\_vi(self):*

*pass*

*class Tra(DoUong):*

*def pha\_che(self):*

*print("Cho la tra vao nuoc soi")*

*def them\_gia\_vi(self):*

*print("Them chanh va duong")*

*class CaPhe(DoUong):*

*def pha\_che(self):*

*print("Cho bot ca phe vao nuoc soi")*

*def them\_gia\_vi(self):*

*print("Them sua")*

*print("Pha tra:")*

*tra = Tra()*

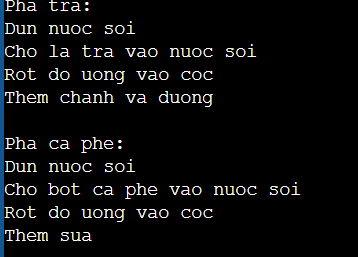
*tra.pha\_do\_uong()*

*print("\nPha ca phe:")*

*caphe = CaPhe()*

*caphe.pha\_do\_uong()*

**kết quả**

****

Ảnh . Kết quả

### Ví dụ về quy trình pha chế đồ uống (Trà, Cà phê) bằng Java.

*abstract class DoUong {*

*public final void phaDoUong() {*

*dunNuoc();*

*phaChe();*

*rotVaoCoc();*

*themGiaVi();*

*}*

*protected void dunNuoc() {*

*System.out.println("Dun nuoc soi");*

*}*

*protected abstract void phaChe();*

*protected void rotVaoCoc() {*

*System.out.println("Rot do uong vao coc");*

*}*

*// Hook method*

*protected void themGiaVi() {*

*// mac dinh khong lam gi*

*}*

*}*

*class Tra extends DoUong {*

*@Override*

*protected void phaChe() {*

*System.out.println("Cho la tra vao nuoc soi");*

*}*

*@Override*

*protected void themGiaVi() {*

*System.out.println("Them chanh va duong");*

*}*

*}*

*class CaPhe extends DoUong {*

*@Override*

*protected void phaChe() {*

*System.out.println("Cho bot ca phe vao nuoc soi");*

*}*

*@Override*

*protected void themGiaVi() {*

*System.out.println("Them sua");*

*}*

*}*

*public class Main {*

*public static void main(String[] args) {*

*System.out.println("Pha tra:");*

*DoUong tra = new Tra();*

*tra.phaDoUong();*

*System.out.println("\nPha ca phe:");*

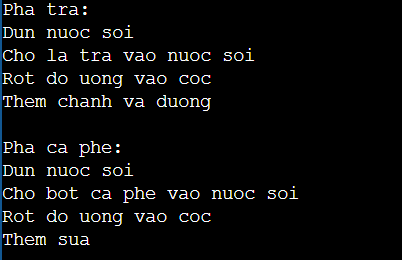
*DoUong caPhe = new CaPhe();*

*caPhe.phaDoUong();*

*}*

*}*

Kết quả

****

Ảnh . kết quả pha chế