

BỘ KHOA HỌC VÀ CÔNG NGHỆ
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG



BÁO CÁO BÀI TẬP LỚN

MÔN: THỰC TẬP CƠ SỞ

Giảng viên: TS. Nguyễn Duy Phương

Nhóm 1

B22DCCN482 Trịnh Quang Lâm

B22DCCN902 Lại Quang Vinh

B22DCCN122 Bùi Tiến Dũng

Link sản phẩm: <https://github.com/quanglam04/FoodStore>

Hà Nội, ngày 20 tháng 05 năm 2025

Mục lục

PHẦN 1: ĐẶT VÂN ĐÈ	1
PHẦN 2: KHẢO SÁT	2
PHẦN 3: TÌM HIỂU VỀ MÔ HÌNH MVC	3
PHẦN 4: CÁC MODULE CHÍNH	5
4.1. Controller	5
4.2. Service	5
4.3. Repository	6
4.4. Domain.....	6
4.5. Các module khác.....	7
PHẦN 5: TRIỂN KHAI	8
5.1. Ngôn ngữ, thư viện	8
5.2. Phương pháp thực hiện	8
5.2.a. Thiết kế cơ sở dữ liệu	8
5.2.b. Khởi tạo dự án.....	9
5.2.c. Xác thực cơ bản với Spring Security.....	12
5.2.d. Phân quyền truy cập.....	15
5.3.e. Thêm sản phẩm vào giỏ hàng, tạo đơn hàng, và xem lịch sử mua hàng.....	16
5.3.f. Quản lý người dùng, sản phẩm, đơn đặt hàng.....	17
5.2.h. Tìm kiếm, lọc sản phẩm với Spring JPA Specification	18
5.2.i. Tìm kiếm sản phẩm theo tên.....	19
5.2.j. Gửi email thông báo đặt hàng thành công cho người dùng	20
5.2.k. Xác thực email đăng ký	21
5.2.l. Lấy lại mật khẩu người dùng thông qua email	24
5.2.m. Tích hợp thanh toán VNPAY	25
5.2.n. Tích hợp login OAuth2 với tài khoản Google và Github	27
5.2.o. Tích hợp chat-bot AI	30
5.2.p. Tích hợp API Giao hàng nhanh trong việc tính toán chi phí vận chuyển.....	36
5.3. Công cụ hỗ trợ	42
PHẦN 6: KẾT QUẢ	43
6.1. Giao diện phía người dùng	43
6.1. Giao diện phía người quản trị	49
PHẦN 7: KẾT LUẬN	52
PHỤ LỤC 1: HƯỚNG DẪN CÀI ĐẶT VÀ CHẠY ỨNG DỤNG	53
Cách 1: Sử dụng Docker	53
Cách 2: Chạy thủ công bằng cách cấu hình các tham số	55
TÀI LIỆU THAM KHẢO	59

Danh sách bảng biểu

STT	Chú thích	Trang
Hình 1	Quy trình xử lý request trong mô hình MVC	3
Hình 2	Sơ đồ các bảng và mối liên hệ	9
Hình 3	Giao diện tại trang Spring initializer	10
Hình 4	Hình ảnh minh họa quá trình xử lý một yêu cầu HTTP	12
Hình 5	Hình ảnh mô phỏng luồng hoạt động của Spring Security	13
Hình 6	Mật khẩu mặc định được Spring Security tạo ra	14
Hình 7	Cấu hình Security	15
Hình 8	Xử lý Logic liên quan đến Order	16
Hình 9	Tạo mới người dùng	17
Hình 10	Xem chi tiết thông tin người dùng	18
Hình 11	Lọc theo type sản phẩm	19
Hình 12	Lọc các sản phẩm có giá nằm trong khoảng min và max	19
Hình 13	Tìm kiếm sản phẩm dựa theo tên	20
Hình 14	Cấu hình triển khai gửi email cho người dùng	20
Hình 15	Xử lý logic gửi Email khi có yêu cầu	21
Hình 16	Cấu hình dịch vụ gửi Email xác nhận	22
Hình 17	Gửi mã OTP đến Email người dùng khi đăng ký	23
Hình 18	Xác nhận mã OTP và lưu thông tin người dùng	23
Hình 19	Gửi Email khôi phục mật khẩu	24
Hình 20	Xác nhận Token và hiển thị thay đổi mật khẩu	25
Hình 21	Mô hình kết nối VNPAY	26
Hình 22	Luồng hoạt động khi sử dụng chat-bot AI	31
Hình 23	Kết quả sau khi tích hợp chat-bot	34
Hình 24	Kết quả sau khi chỉnh sửa prompt	35
Hình 25	Luồng xử lý khi thanh toán	36
Hình 26	Giao diện khi chọn vị trí nhận hàng	37
Hình 27	Giao diện khi bấm xem thông tin cá nhân	38
Hình 28	Giao diện sau khi tính toán chi phí vận chuyển	41
Hình 29	Giao diện màn hình đăng nhập	43
Hình 30	Giao diện màn hình đăng ký	43
Hình 31	Giao diện quên mật khẩu	44
Hình 32	Giao diện quản lý tài khoản	44
Hình 33	Giao diện lịch sử mua hàng	45
Hình 34	Giao diện đổi mật khẩu	45
Hình 35	Giao diện danh sách sản phẩm	46
Hình 36	Giao diện tìm kiếm, lọc sản phẩm	46
Hình 37	Giao diện xem chi tiết sản phẩm	47
Hình 38	Giao diện thông tin đơn hàng	47
Hình 39	Giao diện thanh toán	48
Hình 40	Giao diện thanh toán thành công	48
Hình 41	Email xác nhận mua hàng được gửi về mail người dùng	49
Hình 42	Số lượng người dùng, đơn hàng, sản phẩm hiện có	49

Hình 43	Giao diện quản lý người dùng	50
Hình 44	Giao diện quản lý sản phẩm	50
Hình 45	Giao diện quản lý đơn đặt hàng	51
Hình 46	Giao diện tại trang chủ Github	53
Hình 47	Giao diện sau khi clone thành công dự án về máy	53
Hình 48	Build Image → Container	54
Hình 49	Giao diện chạy container	54
Hình 50	Giao diện kết quả sau khi chạy với docker	55
Hình 51	Giao diện tại trang chủ Github	55
Hình 52	Giao diện sau khi clone thành công dự án về máy	56
Hình 53	Giao diện chạy ứng dụng với SpringBoot Dashboard	57
Hình 54	Giao diện khi chạy thành công	58
Hình 55	Truy cập ứng dụng	58

Danh mục từ viết tắt

Số thứ tự	Từ ngữ trích dẫn	Giải thích
1	MVC	Mô hình phát triển phần mềm Model – View – Controller
2	JDBC	Java Database Connectivity – Giao diện lập trình để kết nối và thao tác với cơ sở dữ liệu trong Java
3	JSP	JavaServer Pages – Công nghệ của Java dùng để tạo các trang web động
4	DTO	Data Transfer Object – đối tượng để dùng để truyền dữ liệu giữa các lớp hoặc tầng trong ứng dụng
5	CSRF	Cross-Site Request Forgery – tấn công giả mạo yêu cầu từ trang web khác
6	JPA	Java Persistence API – Giao diện lập trình dùng để quản lý dữ liệu quan hệ trong Java
7	API	Application Programming Interface – Giao diện lập trình ứng dụng giúp các phần mềm giao tiếp với nhau
8	URI	Uniform Resource Identifier – định danh tài nguyên trên Internet

Danh sách phân chia công việc

Thành viên	Phân chia nhiệm vụ	Đánh giá
Trịnh Quang Lâm	<ul style="list-style-type: none">- Quản lý source code- Viết báo cáo- Tích hợp chat-bot AI Gemini- Tích hợp API Giao hàng nhanh- Tích hợp login OAuth2 với Google/Github- Tích hợp thanh toán VNPay- Thiết kế cơ sở dữ liệu- Xử lý logic phía User	Hoàn thành
Lại Quang Vinh	<ul style="list-style-type: none">- Xử lý logic phía Admin bao gồm các thao tác thêm, sửa, cập nhật đối với các Người dùng, Sản phẩm, Đơn hàng- Viết báo cáo- Thiết kế cơ sở dữ liệu	Hoàn thành
Bùi Tiến Dũng	<ul style="list-style-type: none">- Sửa UI theo yêu cầu phía User và Admin- Viết báo cáo	Hoàn thành

PHẦN 1: ĐẶT VĂN ĐỀ

Trong thời đại tăng trưởng như hiện nay, để có thể đảm bảo cho tương lai của chính mình thì các bạn trẻ ngày càng có xu hướng dành phần nhiều thời gian của bản thân cho công việc. Do đó các bạn sẽ mất đi khoảng thời gian cần thiết để chăm chút cuộc sống hàng ngày - điển hình là mất đi thời gian nấu nướng hay mua thực phẩm cho những bữa ăn. Thế nhưng, ngay lúc này là thời đại cho công nghệ, mua bán hàng không chỉ được thực hiện tại một địa điểm vật lý cụ thể, các điểm bán hàng trên thị trường ngày nay còn được tiến hành theo 1 xu hướng mới, xu hướng tiên bộ của thời đại kỹ thuật thông tin - mua sắm trực tuyến. Giờ đây giới trẻ mua sắm thực phẩm, đặt đồ ăn trực tuyến trên các trang Web, thay vì có thể mất nhiều thời gian đi lòng vòng giữa các điểm (còn chưa kể đến về vấn đề an toàn giao thông, ùn tắc giao thông, ... càng lấy đi nhiều thời gian hơn nữa) thì có thể lên Internet tìm kiếm theo nhu cầu của mình rất nhanh chóng, rất tiện lợi. Xác định được mục tiêu và đối tượng cho sản phẩm, nhóm đã quyết định chọn chủ đề xây dựng một Website phục vụ việc bán hàng online các thực phẩm, món ăn sẵn,... cho các bạn trẻ, những nhân viên văn phòng không có thời gian. Website sẽ bao gồm những chức năng thiết yếu nhất của một trang web bán hàng gồm:

- **Đăng ký tài khoản, đăng nhập, lấy lại mật khẩu**
- **Đăng nhập** thông qua tài khoản Gmail/Github
- Thanh toán qua ví điện tử **VNPay**
- Tích hợp Giao hàng nhanh **tính chi phí vận chuyển**
- **Tìm kiếm, lọc** sản phẩm theo nhu cầu, **phân loại** và **thêm sản phẩm** vào giỏ hàng,...
- **Xem thông tin chi tiết** sản phẩm, thông tin đơn hàng sau khi thanh toán thanh công
- **Quản lý, duyệt** đơn hàng, **điều chỉnh trạng thái** các đơn hàng
- **Quản lý người dùng** (thêm, sửa, xóa, cập nhật)
- **Quản lý sản phẩm** (thêm, sửa, xóa, cập nhật)
- **Gửi thông báo xác nhận đặt hàng** qua email người dùng

PHẦN 2: KHẢO SÁT

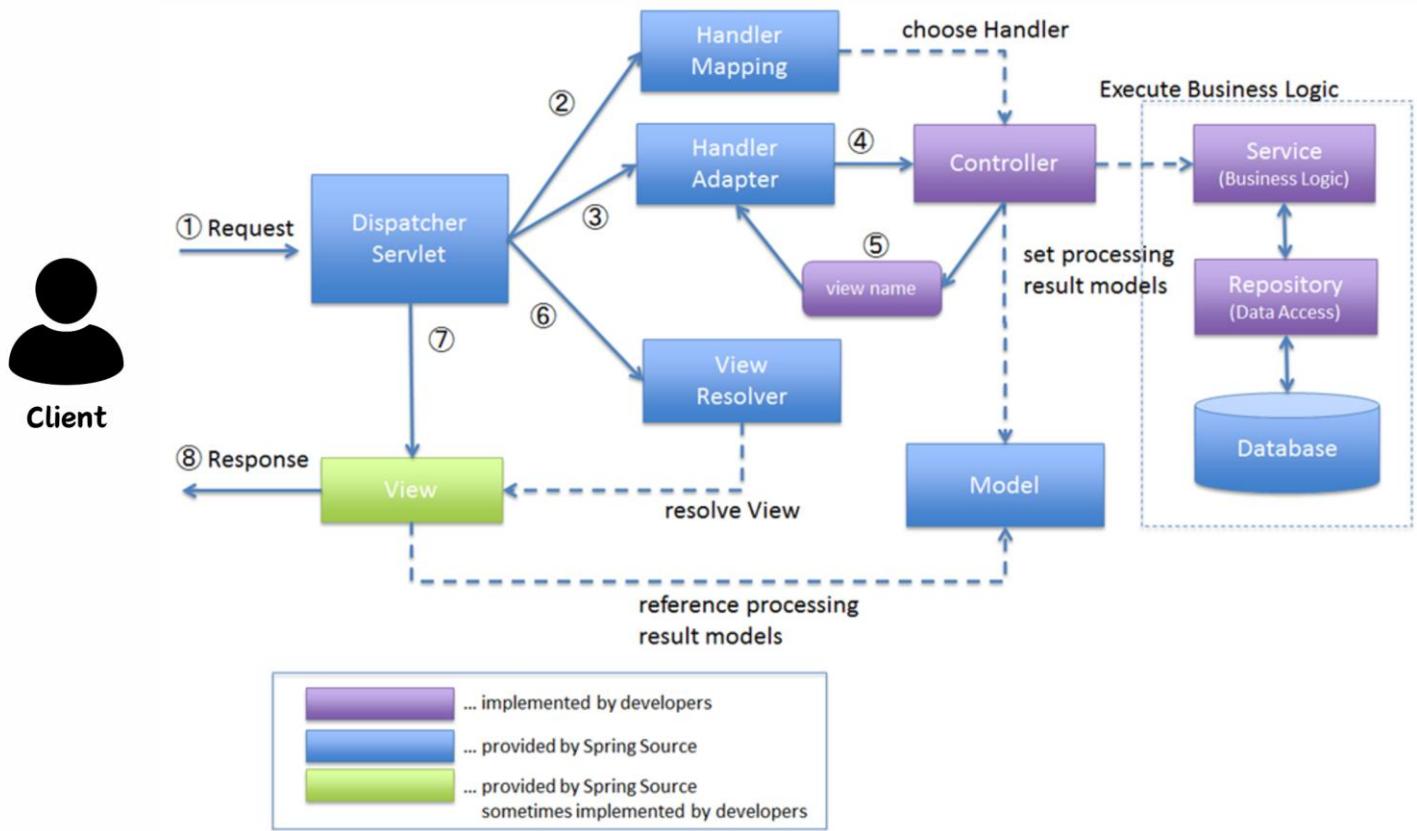
Với ý tưởng và những yêu cầu cụ thể đã nêu ra như trong Phần 1, nhóm đã tìm hiểu, thảo luận đặt ra những vấn đề và cách giải quyết ban đầu cho bài toán *Xây dựng Website bán hàng online*. Cụ thể trong bài này, cần phải sử dụng một hệ quản trị cơ sở dữ liệu để lưu trữ các thông tin. Dựa trên các yếu tố về khả năng tương thích tốt với Java, hiệu suất cao, bảo mật mạnh mẽ, khả năng mở rộng, hỗ trợ tính năng phù hợp cho thương mại điện tử và cộng đồng hỗ trợ lớn, nhóm 01 chúng em đã quyết định chọn hệ quản trị cơ sở dữ liệu MySQL cho bài toán lần này.

Về xây dựng hệ thống backend, nhóm chúng em quyết định sử dụng Spring Framework để hỗ trợ cho mô hình MVC thuần trong Java. Spring cung cấp các tính năng mạnh mẽ giúp quản lý luồng dữ liệu và điều hướng giữa các thành phần trong ứng dụng một cách dễ dàng, đồng thời tối ưu hóa quá trình phát triển nhờ khả năng tích hợp tốt với các công nghệ khác. Chúng em chọn sử dụng Spring Data JPA để thao tác với cơ sở dữ liệu, giúp đơn giản hóa việc quản lý các kết nối, truy vấn, và thao tác dữ liệu ở mức cao, thay vì phải làm việc trực tiếp với JDBC.

Về phía front-end, thay vì xây dựng từ đầu, nhóm đã quyết định lựa chọn sử dụng các template có sẵn và điều chỉnh lại theo yêu cầu bằng các kiến thức về HTML và CSS. Để tối ưu hóa thời gian và nâng cao hiệu suất, nhóm quyết định kết hợp Bootstrap - một framework CSS phổ biến - nhằm hỗ trợ trong việc thiết kế và bố trí giao diện. Đồng thời, chúng em sử dụng JSP cùng với JSTL làm template engine để render giao diện cho một số trang/phần của trang trực tiếp tại server (Server Side Rendering), giúp tạo ra giao diện đồng nhất và tăng tính tương tác cho ứng dụng.

Để việc hoạt động nhóm trở nên hiệu quả và đạt được năng suất cao nhất, nhóm sẽ sử dụng Git cho quá trình phát triển và lưu trữ source code trên server Github.

PHẦN 3: TÌM HIỂU VỀ MÔ HÌNH MVC



Hình 1: Quy trình xử lý request trong mô hình MVC

Mô hình MVC(Model – View - Controller) là một thiết kế phổ biến trong lập trình ứng dụng web, giúp tách biệt cách thành phần quản lý dữ liệu(Model), giao diện người dùng(View), và điều hướng logic(Controller). Spring Framework cung cấp một triển khai mạnh mẽ cho mô hình MVC thông qua thành phần **Spring Web MVC**, giúp tổ chức ứng dụng một cách rõ ràng, dễ bảo trì và mở rộng.

Bằng cách sử dụng DispatcherServlet làm trung tâm điều phối, Spring MVC hỗ trợ xử lý các yêu cầu HTTP từ người dùng, thực hiện logic nghiệp vụ và trả về kết quả giao diện thích hợp. Trên hình mô tả quá trình xử lý một request điển hình trong mô hình này, cụ thể như sau:

- Client gửi Request lên Server:** Người dùng gửi một yêu cầu HTTP đến ứng dụng thông qua trình duyệt. Request này được tiếp nhận bởi **DispatcherServlet**, đóng vai trò chịu trách nhiệm điều phối toàn bộ quá trình xử lý.
- Xác định Handler:** **DispatcherServlet** sử dụng **Handler Mapping** để tra cứu xem yêu cầu được ánh xạ đến phương thức nào trong Controller. Handler Mapping được định nghĩa dựa trên các cấu hình hoặc chú thích như **@RequestMapping** trong mã nguồn.
- Gọi Handler Adapter:** Khi xác định được Controller, DispatcherServlet chuyển request đến **Handler Adapter**, một thành phần trung gian đảm bảo DispatcherServlet có thể gọi đúng phương thức trong Controller.
- Controller xử lý logic:** Controller nhận request và thực thi các thao tác xử lý

nhu:

- Lấy tham số từ request
 - Gọi đến các lớp Service để thực hiện logic nghiệp vụ
 - Kết nối đến lớp Repository để truy xuất dữ liệu từ cơ sở dữ liệu nếu cần. Sau khi hoàn tất, Controller trả về một đối tượng Model chứa dữ liệu cần hiển thị và tên của View sẽ được sử dụng
5. **Tìm kiếm View qua View Resolver:** Dựa trên tên View mà Controller trả về, DispatcherServlet sử dụng **View Resolver** để ánh xạ tên này đến file giao diện thực tế(ví dụ: file JSP, Thymeleaf template,v.v..).
 6. **Kết hợp View và Model:** DispatcherServlet gửi dữ liệu từ Model đến View để render giao diện. View sẽ hiển thị các thông tin động từ Model dưới dạng trang web hoàn chỉnh.
 7. **Trả về Response cho Client:** Sau khi view được render hoàn thiện, nội dung HTML được gửi lại cho người dùng. Đây là kết quả cuối cùng, hiển thị nội dung mà người dùng yêu cầu.

Quy trình này không chỉ đảm bảo sự tách biệt rõ ràng giữa các thành phần mà còn giúp ứng dụng dễ dàng bảo trì và mở rộng. Spring MVC với DispatcherServlet đã tối ưu hóa quy trình xử lý yêu cầu, giúp việc phát triển ứng dụng web trở nên nhanh chóng và hiệu quả.

PHẦN 4: CÁC MODULE CHÍNH

4.1. Controller

Trong mô hình **MVC** (Model-View-Controller), các Controller đóng vai trò trung gian, chịu trách nhiệm tiếp nhận và điều phối các yêu cầu từ phía người dùng đến các tầng khác của ứng dụng. Cụ thể, trong ứng dụng lần này, các Controller nằm trong package `com.example.food_store.controller`. Với **View Engine** là JSP, các Controller sẽ tương tác với các trang JSP để hiển thị dữ liệu một cách trực quan cho người dùng.

Các Controller này được trang trí với annotation `@Controller`, giúp Spring nhận diện chúng như các thành phần xử lý yêu cầu HTTP. Ngoài ra, các phương thức trong Controller có thể sử dụng các annotation như `@RequestMapping`, `@GetMapping`, `@PostMapping` để định nghĩa các đường dẫn URL và các loại yêu cầu HTTP mà phương thức đó xử lý.

Khi nhận được yêu cầu từ phía người dùng, Controller sẽ thực hiện nhiệm vụ điều phối bằng cách chuyển yêu cầu xuống tầng **Service**. Tầng Service là nơi chứa các logic nghiệp vụ, xử lý các thao tác chính của ứng dụng. Service sẽ tiếp nhận các yêu cầu từ Controller, thực hiện các công việc cần thiết như lấy dữ liệu từ database hoặc xử lý các logic phức tạp, sau đó trả kết quả về cho Controller. Cuối cùng, Controller sẽ gửi dữ liệu đó đến View (JSP) để hiển thị cho người dùng.

Sự phân chia này giúp cho Controller chỉ tập trung vào việc điều phối, trong khi tầng Service đảm nhận logic nghiệp vụ, làm cho mã nguồn dễ bảo trì và tái sử dụng. Controller đóng vai trò làm cầu nối giữa View và Model thông qua tầng Service, giúp duy trì sự tách biệt rõ ràng giữa các phần trong ứng dụng, một yếu tố quan trọng của mô hình MVC.

4.2. Service

Trong mô hình **MVC**, tầng **Service** đóng vai trò quan trọng trong việc xử lý các logic nghiệp vụ của ứng dụng. Tầng này được thiết kế để nhận yêu cầu từ các Controller, thực hiện các thao tác nghiệp vụ và sau đó trả kết quả về cho Controller để hiển thị hoặc xử lý tiếp.

Cụ thể, tầng Service đảm nhiệm nhiệm vụ **triển khai các logic nghiệp vụ cốt lõi của ứng dụng**, bao gồm các thao tác như truy xuất dữ liệu từ tầng **Repository** (tầng dữ liệu), xử lý tính toán, và thực hiện các quy tắc nghiệp vụ trước khi gửi dữ liệu trở lại cho Controller. Việc sử dụng tầng Service giúp tách biệt rõ ràng giữa logic nghiệp vụ và logic điều hướng, giúp mã nguồn dễ bảo trì, mở rộng và tái sử dụng.

Các lớp trong tầng Service thường được đánh dấu bằng annotation `@Service` trong Spring. Annotation này giúp Spring Framework nhận diện và quản lý các lớp này như là các thành phần xử lý logic nghiệp vụ, hỗ trợ Dependency Injection để dễ dàng quản lý và sử dụng các đối tượng trong ứng dụng.

4.3. Repository

Trong mô hình **MVC**, tầng **Repository** được sử dụng để **làm việc với cơ sở dữ liệu**, đóng vai trò là lớp trung gian giữa tầng **Service** và tầng dữ liệu. Đây là nơi quản lý việc truy xuất, lưu trữ, cập nhật, và xóa dữ liệu trong cơ sở dữ liệu. Tầng Repository giúp tách biệt các thao tác dữ liệu khỏi logic nghiệp vụ, tạo nên một cấu trúc rõ ràng và dễ bảo trì cho ứng dụng.

Trong ứng dụng này, nhóm đã tạo các interface trong tầng Repository bằng cách kế thừa từ **JpaRepository<T, I>** của Spring Data JPA thay vì viết các lớp xử lý dữ liệu cụ thể. Điều này cho phép sử dụng các phương thức **CRUD** (Create, Read, Update, Delete) mặc định mà Spring Data JPA cung cấp, như **findAll(), save(), deleteById()**, v.v. Chúng ta cũng có thể định nghĩa thêm các phương thức truy vấn tùy chỉnh bằng cách đặt tên phương thức theo quy tắc của Spring Data JPA mà không cần viết mã SQL cụ thể.

Các interface trong tầng **Repository** được đánh dấu bằng annotation **@Repository**, giúp Spring Framework nhận diện chúng như các thành phần quản lý truy cập dữ liệu. Annotation này không chỉ tạo sự rõ ràng về vai trò của tầng Repository mà còn giúp xử lý các ngoại lệ liên quan đến dữ liệu, biến chúng thành các ngoại lệ của Spring Data để dễ quản lý trong ứng dụng.

4.4. Domain

Tầng **Domain** chứa các **Entity** và **DTO** (Data Transfer Object) để đại diện và quản lý dữ liệu trong ứng dụng. Trong bài tập này, các thành phần này nằm trong **package com.example.food_store.domain**.

- Entity là các lớp đại diện cho các bảng trong cơ sở dữ liệu. **Mỗi Entity thường tương ứng với một bảng**, với các trường (fields) trong lớp đại diện cho các cột trong bảng. Entity giúp ứng dụng kết nối trực tiếp với cơ sở dữ liệu, cho phép thao tác với dữ liệu thông qua **ORM** (Object-Relational Mapping) mà Spring Data JPA hỗ trợ.
- DTO là các lớp dùng để truyền dữ liệu giữa các tầng trong ứng dụng, đặc biệt là giữa Controller và Service. DTO giúp kiểm soát dữ liệu được truyền đi, chỉ truyền các thông tin cần thiết, giúp tăng hiệu suất và bảo mật.

Các Entity được đánh dấu bằng annotation **@Entity**, cho phép Spring và JPA nhận diện và quản lý chúng như các đối tượng dữ liệu.

Để chỉ định tên bảng trong cơ sở dữ liệu, chúng ta có thể sử dụng annotation **@Table(name = "ten_bang")**. Điều này sẽ xác định tên bảng cụ thể trong cơ sở dữ liệu mà Entity sẽ ánh xạ đến.

Trong các ứng dụng thực tế, các bảng trong cơ sở dữ liệu thường có mối quan hệ với nhau. Để phản ánh các mối quan hệ này trong tầng Domain, Spring Data JPA cung cấp các annotation sau:

- [@OneToOne](#): Thiết lập mối quan hệ một-một giữa hai Entity.
- [@OneToMany](#) và [@ManyToOne](#): Thiết lập mối quan hệ một-nhiều và nhiều-một giữa hai Entity.
- [@ManyToMany](#): Thiết lập mối quan hệ nhiều-nhiều giữa hai Entity.

Chi tiết hơn xem tại:

https://docs.jboss.org/hibernate/orm/6.4/introduction/html_single/Hibernate_Introduction.html#associations

4.5. Các module khác

- *Specifications*: thuộc package [com.example.food_store.service.specification](#) gồm các Specification để thực hiện những câu SQL phức tạp hơn khi lọc, tìm kiếm, sắp xếp,...cho sản phẩm.
- *SendEmail*: thuộc package [com.example.food_store.service](#) có nhiệm vụ xử lý các yêu cầu liên quan đến việc gửi Email như: xác thực email đăng ký, gửi Email xác nhận sau khi thanh toán, gửi Email lấy lại mật khẩu cho người dùng
- *Config*: thuộc [com.example.food_store.config](#) có vai trò quản lý các cấu hình cần thiết cho ứng dụng, đặc biệt là các cấu hình bảo mật và hiển thị. Các class trong module này thường được đánh dấu với annotation [@Configuration](#), giúp Spring tự động nhận diện và xử lý. Module này chứa các cấu hình về bảo mật, trong đó [WebSecurityConfiguration](#) thiết lập các quy tắc bảo mật, quản lý quyền truy cập và đăng nhập/đăng xuất cho người dùng. Để đảm bảo an toàn, module sử dụng [PasswordEncoder](#) để mã hóa mật khẩu trước khi lưu trữ trong cơ sở dữ liệu. Bên cạnh đó, [UserDetailsService](#) giúp quản lý thông tin đăng nhập của người dùng, hỗ trợ việc xác thực và cấp quyền. Ngoài ra, module Config cũng chứa các cấu hình để hiển thị view, đảm bảo giao diện người dùng được xử lý và hiển thị đúng cách.
- *WebApp*: Gồm các tài nguyên khác như ảnh, file CSS, Javascript, JSP, để render, trả về cho người dùng

PHẦN 5: TRIỂN KHAI

5.1. Ngôn ngữ, thư viện

- Frontend:
 - Ngôn ngữ: HTML, CSS, JavaScript
 - Thư viện: Bootstrap 5, JSP(JSTL), JQuery
- Backend:
 - Ngôn ngữ: Java
 - Thư viện: Spring Boot, Spring Data JPA, Spring Security, Spring Session
 - Cơ sở dữ liệu: MySQL

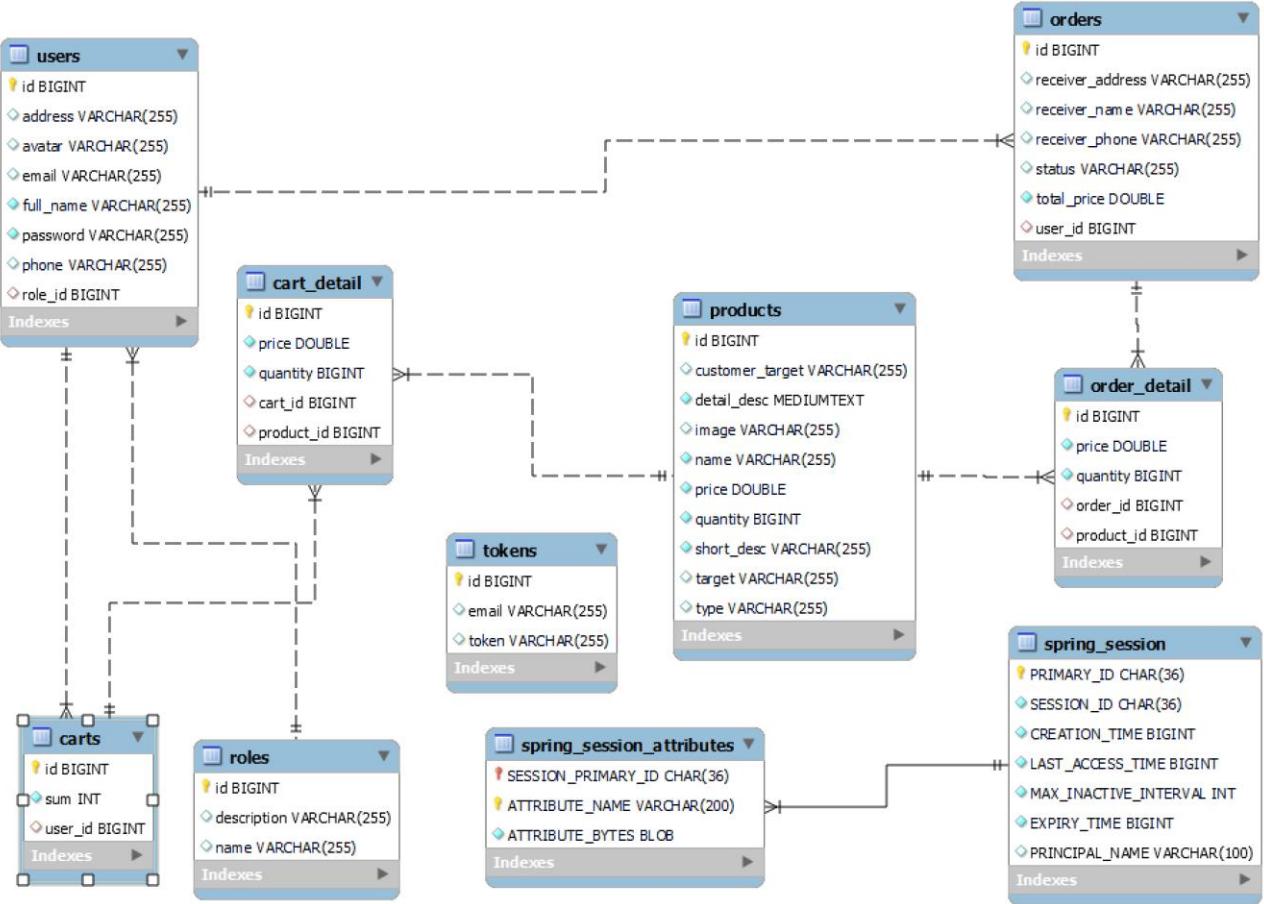
5.2. Phương pháp thực hiện

5.2.a. Thiết kế cơ sở dữ liệu

Cấu trúc cơ sở dữ liệu gồm 10 bảng. Trong đó gồm 2 phần chính gồm các bảng liên quan đến xác thực, định danh của người dùng, phân quyền và phần còn lại là các bảng liên quan đến sản phẩm, mua bán , đơn hàng,...

Các bảng chính gồm có `products` dùng để lưu thông tin về sản phẩm, `users` dùng để lưu thông tin về người dùng, `cart` và `cart_detail` lưu thông tin về giỏ hàng của người dùng, trong đó bảng `carts` lưu tổng giá trị của giỏ hàng và liên kết với `users` thông qua `user_id`, `cart_detail` dùng để lưu chi tiết từng sản phẩm trong giỏ hàng.

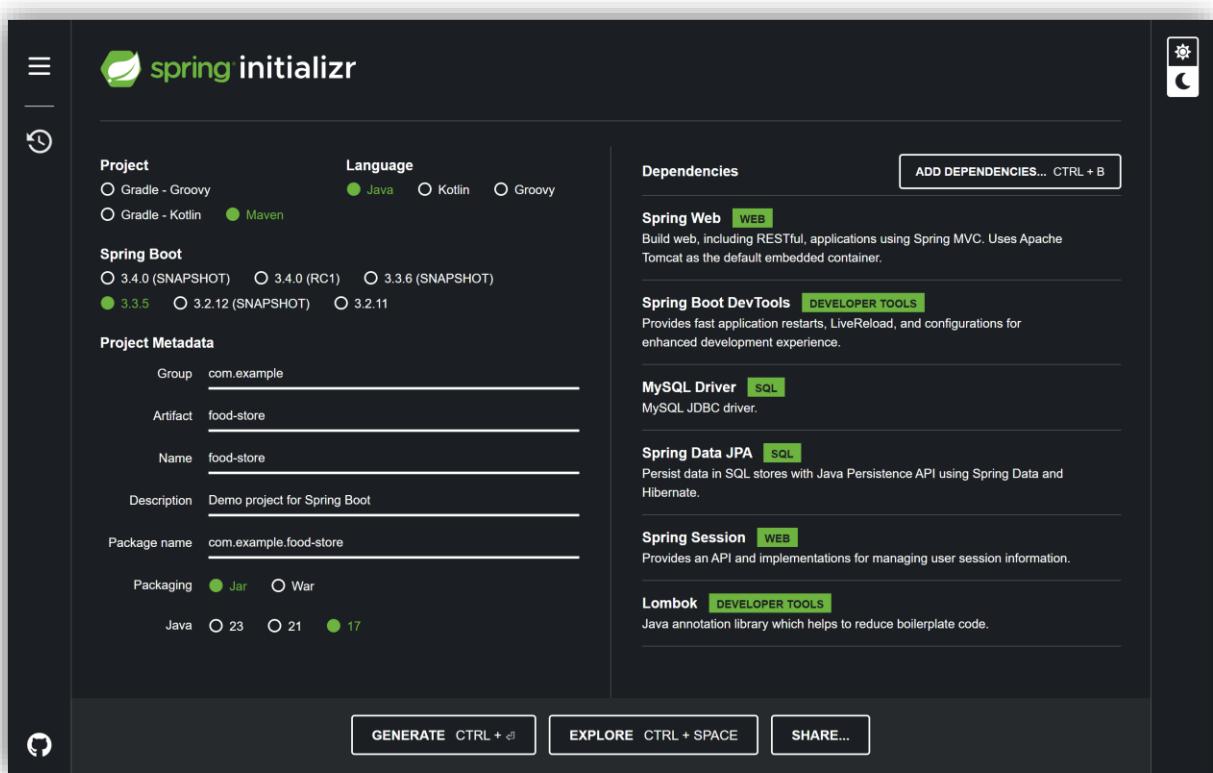
Các bảng liên quan đến xác thực, định danh người dùng phân quyền gồm có `users` lưu thông tin xác thực của người dùng, bao gồm email và mật khẩu. `roles` định nghĩa các vai trò người dùng để phục vụ cho việc phân quyền. `spring_session`, `spring_session_attributes` dùng để lưu thông tin về phiên đăng nhập của người dùng, phục vụ cho việc quản lý phiên làm việc trong ứng dụng. Ngoài ra còn có thêm bảng phụ như `tokens` dùng để lưu mã tokens khi người dùng yêu cầu lấy lại mật khẩu, bảng này giúp xác minh được danh tính người dùng trong quá trình khôi phục mật khẩu



Hình 2:Sơ đồ các bảng và mối liên hệ

5.2.b. Khởi tạo dự án

Để khởi tạo cấu trúc thư mục cho bài toán lần này, nhóm đã sử dụng [Spring Initialize](#), chọn các thiết lập về công cụ build Maven, phiên bản Spring Boot 3.3.4(do Project đã tạo từ trước đó và bây giờ Spring Boot đã update lên phiên bản 3.3.5 nên tại giao diện sẽ không thấy phiên bản 3.3.4), phiên bản Java 17 cùng các lựa chọn dependency trong file pom.xml sau khi khởi tạo như sau:



Hình 3: Giao diện tại trang Spring initializer

Các thư viện cần sử dụng:
`<dependencies>`

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

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

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

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

```
<dependency>
```

```

<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-devtools</artifactId>
<optional>true</optional>
</dependency>

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

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

<dependency>
<groupId>org.springframework.session</groupId>
<artifactId>spring-session-jdbc</artifactId>
</dependency>

<dependency>
<groupId>com.mysql</groupId>
<artifactId>mysql-connector-j</artifactId>
</dependency>

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

<dependency>
<groupId>org.apache.tomcat.embed</groupId>
<artifactId>tomcat-embed-jasper</artifactId>
</dependency>

<dependency>
<groupId>jakarta.servlet.jsp.jstl</groupId>
<artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
</dependency>

<dependency>
<groupId>org.glassfish.web</groupId>
<artifactId>jakarta.servlet.jsp.jstl</artifactId>
</dependency>

<dependency>
<groupId>org.hibernate.orm</groupId>
<artifactId>hibernate-jpamodelgen</artifactId>
<version>6.4.1.Final</version>
<scope>provided</scope>

```

```

</dependency>

<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-mail</artifactId>
<version>3.1.5</version>
</dependency>

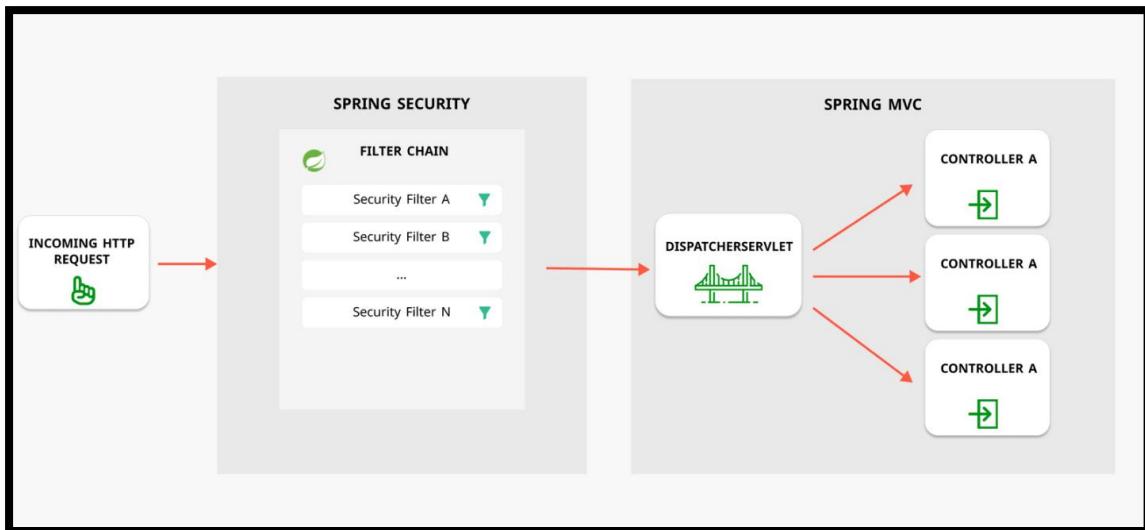
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-context-support</artifactId>
<version>6.1.5</version>
</dependency>

</dependencies>

```

5.2.c. Xác thực cơ bản với Spring Security

Trước khi đi vào phần này, ta cần tìm hiểu Spring Security là gì và tại sao cần sử dụng Spring Security.



Hình 4. Hình ảnh minh họa quá trình xử lý một yêu cầu HTTP

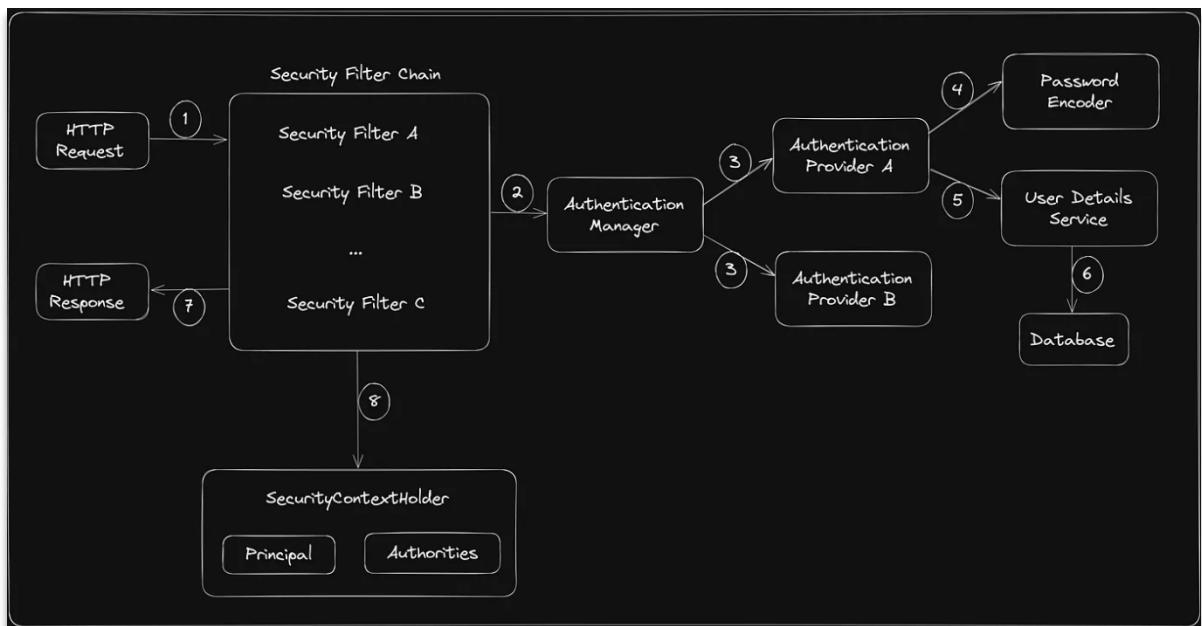
Spring Security là một framework mạnh mẽ trong hệ sinh thái Spring, chuyên về các giải pháp bảo mật cho các ứng dụng Java. Nó cung cấp các chức năng quan trọng như **xác thực (authentication)** và **phân quyền (authorization)** nhằm đảm bảo rằng chỉ những người dùng có quyền truy cập mới có thể sử dụng tài nguyên cụ thể của ứng dụng.

Việc sử dụng Spring Security rất quan trọng để bảo vệ ứng dụng khỏi các mối đe dọa bảo mật như truy cập trái phép, tấn công CSRF (Cross-Site Request Forgery), và các lỗ hổng khác. Với các ứng dụng chứa thông tin nhạy cảm, việc sử dụng Spring Security là một phần thiết yếu để đảm bảo an toàn dữ liệu người dùng và duy trì tính bảo mật của hệ thống.

Dựa vào hình ảnh trên, ta có thể thấy khi một yêu cầu HTTP đến ứng dụng, đầu tiên nó sẽ đi qua **Spring Security Filter Chain** (chuỗi bộ lọc bảo mật của Spring

Security). Filter Chain bao gồm nhiều lớp filter như Security Filter A, Spring Filter B,.. mỗi filter này có nhiệm vụ kiểm tra và xác minh yêu cầu, như xác thực người dùng hoặc xác nhận token bảo mật. Nếu yêu cầu vượt qua các bộ lọc bảo mật, nó sẽ được chuyển đến **DispatcherServlet** – thành phần điều phối của Spring MVC, chịu trách nhiệm chuyển hướng yêu cầu đến đúng **Controller** nhận trách nhiệm xử lý. Cuối cùng, Controller sẽ xử lý yêu cầu và trả về phản hồi cho người dùng.

Luồng hoạt động của Spring Security diễn ra như sau:



Hình 5: Hình ảnh mô phỏng luồng hoạt động của Spring Security

Spring Security xử lý xác thực thông qua một chuỗi các bước được thiết kế để đảm bảo tính bảo mật cho hệ thống. Khi một request được gửi đến, request này sẽ đi qua một chuỗi các bộ lọc bảo mật gọi là filter chain, và một trong số đó là **AuthenticationFilter** - bộ lọc chịu trách nhiệm chính cho việc xử lý xác thực. **AuthenticationFilter** kiểm tra thông tin xác thực (thường là username và password) trong request và nếu thông tin này hợp lệ, nó sẽ gọi đến **AuthenticationManager** để thực hiện xác thực. AuthenticationManager là trung gian quản lý xác thực và có thể ủy quyền cho một hoặc nhiều **AuthenticationProvider** để kiểm tra tính hợp lệ của thông tin người dùng. AuthenticationProvider là lớp quan trọng để thực hiện xác thực, trong đó bao gồm các thành phần như **UserDetailsService** và **PasswordEncoder**. UserDetailsService có nhiệm vụ tìm kiếm và trả về thông tin chi tiết của người dùng dựa trên username thông qua phương thức `loadUserByUsername()`, trong khi PasswordEncoder đảm bảo rằng mật khẩu do người dùng cung cấp được mã hóa và đối chiếu với mật khẩu đã lưu trong cơ sở dữ liệu. Nếu thông tin xác thực hợp lệ, quá trình xác thực thành công và Spring Security sẽ tạo ra một đối tượng **Authentication** chứa thông tin người dùng đã được xác thực cùng với các quyền (authorities) của họ. Đối tượng Authentication này sau đó được lưu vào **SecurityContext**, một bộ nhớ tạm trong RAM, giúp ứng dụng có thể truy cập lại thông tin bảo mật của phiên hiện tại ở bất kỳ đâu trong ứng dụng khi cần. SecurityContext cho phép sử dụng lại các thông

tin xác thực mà không cần xác thực lại, cải thiện hiệu năng hệ thống. Nhờ có cấu trúc này, Spring Security đảm bảo bảo vệ tài nguyên nhạy cảm một cách an toàn và linh hoạt, đồng thời cho phép mở rộng dễ dàng khi tích hợp các cơ chế xác thực mới.

Để tích hợp Spring Security vào dự án, ta cần thêm vào file **pom.xml** các dependency cần thiết:

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

Khi chạy lại ứng dụng, tại cửa sổ terminal, Spring Security đã tự động tạo nên người dùng và mật khẩu mặc định vì chưa thiết lập bất kỳ cấu hình nào

Tên người dùng: **user**

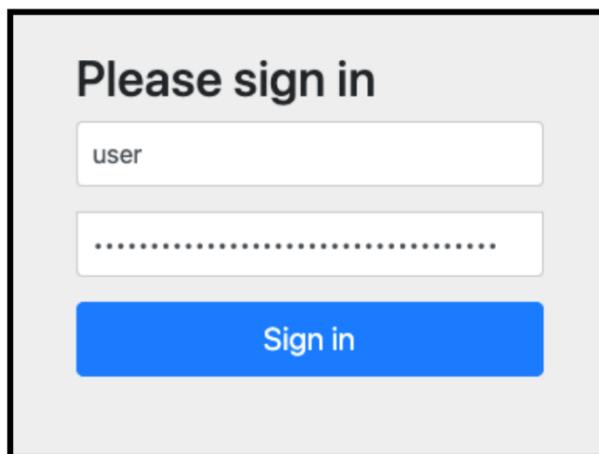
Mật khẩu: **<Mật khẩu được tạo tự động>**

```
2022-01-25 19:14:48.873 INFO 77545 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2022-01-25 19:14:48.885 INFO 77545 --- [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-01-25 19:14:48.885 INFO 77545 --- [ restartedMain] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.56]
2022-01-25 19:14:48.934 INFO 77545 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2022-01-25 19:14:48.935 INFO 77545 --- [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 957 ms
2022-01-25 19:14:49.345 INFO 77545 --- [ restartedMain] .s.s.UserDetailsServiceAutoConfiguration : Using generated security password: 3af71a15-606b-484c-90a1-a779ab2df271

2022-01-25 19:14:49.456 INFO 77545 --- [ restartedMain] o.s.s.web.DefaultSecurityFilterChain : Will secure any request with [org.springframework.security.web.context.request.async.WebAsyncManagerIntegrationFilter@1d937734, org.springframework.security.web.context.SecurityContextPersistenceFilter@4222b10d, org.springframework.security.web.header.HeaderWriterFilter@3a77be22, org.springframework.security.web.csrf.CsrfFilter@6e9c5ecd, org.springframework.security.web.authentication.logout.LogoutFilter@8c8db0ac8, org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter@125a8c70, org.springframework.security.web.authentication.logout.LogoutPageGeneratingFilter@1e1b420, org.springframework.security.web.authentication.www.BasicAuthenticationFilter@2c01f00b, org.springframework.security.web.savedrequest.RequestCacheAwareFilter@6d4595a1, org.springframework.security.web.servletapi.SecurityContextHolderAwareRequestFilter@38bbba35, org.springframework
```

Hình 6: Mật khẩu mặc định được Spring Security tạo ra

Bây giờ, ta đã có thể truy cập vào đường dẫn trang đăng nhập <http://localhost:8080/login> mặc định được tự động tạo của Spring và thử đăng nhập với tài khoản và mật khẩu trước đó:



5.2.d. Phân quyền truy cập

Sau khi đã xác thực người dùng thành công, việc phân quyền cho người dùng là rất quan trọng để đảm bảo chỉ những người có thẩm quyền mới được truy cập vào các tài nguyên nhất định. Trong bài toán lần này, có hai thực thể chính tham gia là người quản lý (**ROLE_ADMIN**) và người dùng thông thường (**ROLE_USER**). Spring sẽ tự động thêm tiền tố ROLE_ khi gán quyền, do đó, các vai trò sẽ được lưu trong **Context** dưới dạng **ROLE_USER** và **ROLE_ADMIN**.

Để kiểm tra quyền hạn của người dùng, có hai phương pháp chính:

1. **Method Security** – quản lý quyền hạn ngay tại cấp phương thức trong mã nguồn, xem chi tiết tại [đây](#).
2. **HTTP Request** – thiết lập quyền hạn trực tiếp trên các endpoint HTTP, xem chi tiết tại [đây](#).

Nhóm đã quyết định sử dụng **HTTP Request** vì các lý do sau. Thứ nhất, bài toán của chúng ta tương đối đơn giản, chỉ yêu cầu phân quyền ở mức cơ bản với hai vai trò chính là USER và ADMIN. Ở các hệ thống lớn hơn, có thể có nhiều cấp độ quản lý khác nhau, ví dụ như trong các ứng dụng thương mại điện tử có các vai trò quản lý từ phòng ban Marketing, IT, và nhiều người quản trị khác. Khi đó, phương pháp phân quyền sẽ cần được thực hiện cẩn thận và phức tạp hơn. Tuy nhiên, với hệ thống đơn giản như bài toán lần này, việc dùng HTTP Request giúp cấu hình phân quyền trực tiếp trên các endpoint mà không cần tạo thêm nhiều lớp logic hoặc cấu trúc phức tạp, từ đó tiết kiệm thời gian và giảm thiểu rủi ro.

Lý do thứ hai là sự thuận tiện trong cấu hình. Với HTTP Request, chúng ta có thể dễ dàng quy định các quy tắc truy cập ngay trong file cấu hình bảo mật của Spring Security. Điều này làm cho mã nguồn trở nên rõ ràng, dễ hiểu và dễ duy trì, đặc biệt phù hợp với nhóm phát triển chưa cần đến các phương pháp kiểm tra quyền hạn phức tạp hơn.

Chi tiết phần cấu hình:

```
● ● ●  
@Bean  
    SecurityFilterChain filterChain(HttpSecurity http) throws Exception {  
  
    http  
        .authorizeHttpRequests(authorize -> authorize  
            .dispatcherTypeMatchers(DispatcherType.FORWARD,  
                DispatcherType.INCLUDE)  
            .permitAll()  
  
            .requestMatchers("/product/**", "/", "/password/**", "/login/**", "/client/**",  
                "/css/**",  
                "/js/**",  
                "/register/**",  
                "/products/**",  
                "/images/**", "/send-request-to-mail", "reset-password/**",  
                "/process-reset-password/**", "/verify/**")  
            .permitAll()  
  
            .requestMatchers("/admin/**").hasRole("ADMIN")  
            .anyRequest().authenticated()  
    }  
}
```

Hình 7:Cấu hình Security

5.3.e. Thêm sản phẩm vào giỏ hàng, tạo đơn hàng, và xem lịch sử mua hàng

Để thực hiện các chức năng "Thêm sản phẩm vào giỏ hàng", "Tạo đơn hàng" và "Xem lịch sử mua hàng" cho người dùng, nhóm đã xây dựng các lớp Entity bao gồm **Order**, **Cart**, **CartDetail**, và **OrderDetail**. Các Entity này giúp lưu trữ thông tin chi tiết của giỏ hàng và các đơn đặt hàng trong hệ thống, đảm bảo dữ liệu người dùng được quản lý một cách hợp lý và có tổ chức. Cụ thể, Cart và CartDetail lưu trữ thông tin về các sản phẩm đã được thêm vào giỏ hàng của người dùng, trong khi Order và OrderDetail lưu trữ thông tin về các đơn hàng đã được tạo.

Tiếp theo, tạo các Controller để xử lý các yêu cầu mà người dùng gửi lên liên quan đến việc thêm sản phẩm vào giỏ hàng, tạo đơn hàng, và xem lịch sử mua hàng. Tại đây, nhóm đã sử dụng **HttpServletRequest** để lấy thông tin người dùng từ các request, bao gồm cả ID của người dùng. Với ID này, Controller có thể gọi đến các lớp Service để xử lý các logic nghiệp vụ.

Lớp Service không chỉ đảm nhiệm việc xử lý logic nghiệp vụ mà còn gọi đến lớp Repository để thao tác với cơ sở dữ liệu. Repository sẽ cập nhật các thông tin về giỏ hàng, đơn hàng và lịch sử mua hàng của người dùng trong cơ sở dữ liệu dựa trên ID của người dùng được lấy từ HttpServletRequest. Cách tiếp cận này giúp duy trì kiến trúc tách biệt giữa các tầng Controller, Service và Repository, làm cho mã nguồn dễ hiểu, dễ bảo trì và mở rộng.



```
● ● ●

@Service
public class OrderService {
    private final OrderRepository orderRepository;
    private final OrderDetailRepository orderDetailRepository;

    public OrderService(OrderRepository orderRepository, OrderDetailRepository orderDetailRepository) {
        this.orderDetailRepository = orderDetailRepository;
        this.orderRepository = orderRepository;
    }

    public Page<Order> fetchAllOrders(Pageable pageable) {
        return this.orderRepository.findAll(pageable);
    }

    public Optional<Order> fetchOrderById(long id) {
        return this.orderRepository.findById(id);
    }

    public List<Order> fetchOrderByUser(User user) {
        return this.orderRepository.findByUser(user);
    }
}
```

Hình 8:Xử lý các logic liên quan đến Order

Tham số **pageable** được truyền vào với mục đích để giới hạn các bản ghi được trả về, điều này sẽ phục vụ cho việc xử lý phân trang tại phía giao diện người dùng.

5.3.f. Quản lý người dùng, sản phẩm, đơn đặt hàng

Trong hệ thống quản lý của ứng dụng, chỉ những người dùng có quyền *ROLE_ADMIN* mới được phép truy cập vào trang quản trị để thực hiện các thao tác quản lý như thêm, sửa, xóa người dùng và sản phẩm. Để giới hạn quyền truy cập này, nhóm tạo các Controller xử lý các URL có dạng `/admin/**`, đảm bảo rằng chỉ những người dùng với quyền *ROLE_ADMIN* mới có thể truy cập vào các tài nguyên này. Cấu hình này được thiết lập trong file `SecurityConfiguration` nhằm bảo vệ các tài nguyên quan trọng và ngăn chặn truy cập trái phép từ người dùng thường.

```
.requestMatchers("/admin/**").hasRole("ADMIN")
```

Các logic liên quan đến việc thêm, sửa, xóa người dùng và sản phẩm sẽ được xử lý tại tầng Service. Ở đây, tầng Service chịu trách nhiệm thực hiện các nghiệp vụ liên quan và đảm bảo tính toàn vẹn của dữ liệu trước khi lưu trữ vào cơ sở dữ liệu. Tầng Service sẽ gọi đến tầng Repository để cập nhật dữ liệu vào cơ sở dữ liệu khi cần thiết, đảm bảo dữ liệu được lưu trữ một cách an toàn và chính xác.

Dưới đây là minh họa về quá trình tạo và xem chi tiết người dùng tại trang quản trị:

```
● ● ●

@RequestMapping(value = "/admin/user/create", method = RequestMethod.POST)
public String createUser(Model model, @ModelAttribute("newUser") @Valid User trinhlam,
    BindingResult newBindingResult,
    @RequestParam("avatarFile") MultipartFile file) {

    List<FieldError> errors = newBindingResult.getFieldErrors();
    for (FieldError error : errors) {
        System.out.println(error.getField() + " - " + error.getDefaultMessage());
    }

    if (newBindingResult.hasErrors() || this.userService.checkEmailExist(trinhlam.getEmail())) {

        if (this.userService.checkEmailExist(trinhlam.getEmail()))
            model.addAttribute("errorEmail", "Email đã tồn tại.");
        return "admin/user/create";
    }

    String avatar = this.uploadService.handleSaveUploadFile(file, "avatar");
    String hashPassword = this.passwordEncoder.encode(trinhlam.getPassword());
    trinhlam.setAvatar(avatar);
    trinhlam.setPassword(hashPassword);

    trinhlam.setRole(this.userService.getRoleByName(trinhlam.getRole().getName()));
    System.out.println(trinhlam.getRole());

    this.userService.handleSaveUser(trinhlam);
    System.out.println(trinhlam.getEmail());

    return "redirect:/admin/user";
}
```

Hình 9:Tạo mới người dùng

```

@RequestMapping("/admin/user/{id}")
public String getUserDetailPage(Model model, @PathVariable long id) {
    User user = this.userService.getUserById(id);
    model.addAttribute("id", id);
    model.addAttribute("user", user);

    return "admin/user/detail";
}

```

Hình 10:Xem chi tiết thông tin người dùng

5.2.h. Tìm kiếm, lọc sản phẩm với Spring JPA Specification

Trong phần tìm kiếm và lọc sản phẩm của bài toán lần này, có hai hướng tiếp cận chính để thực hiện truy vấn và lọc dữ liệu với Spring Data JPA:

1. Sử dụng truy vấn truyền thống với [@Query](#): Trong cách tiếp cận này, có thể sử dụng annotation [@Query](#) để viết truy vấn JPQL hoặc Native SQL trực tiếp trong repository. Ngoài ra, Spring Data JPA cung cấp khả năng tự động xây dựng truy vấn dựa trên tên phương thức với các từ khóa như `findById`, `findByName`, ... giúp đơn giản hóa việc tạo các truy vấn đơn giản mà không cần viết SQL thủ công
2. Sử dụng [Specification](#) với [JpaSpecificationExecutor](#): Cách tiếp cận này linh hoạt hơn, cho phép xây dựng các truy vấn động với nhiều điều kiện khác nhau bằng cách sử dụng Specification API của Spring Data JPA. Bằng cách kế thừa `JpaSpecificationExecutor`, repository có thể hỗ trợ việc truyền các điều kiện lọc dưới dạng Specification, cho phép các tiêu chí tìm kiếm phức tạp được xây dựng một cách linh hoạt, dễ dàng mở rộng

Trong bài toán lần này, nhóm đã quyết định sử dụng cách thứ hai, sử dụng Specification để xây dựng các điều kiện lọc. Dưới đây là các ví dụ minh họa cho cách tạo các điều kiện lọc bằng Specification:

- Lọc với một điều kiện: Để lọc các sản phẩm theo danh sách loại type, chúng ta định nghĩa một Specification như sau:



```
public static Specification<Product> matchListType(List<String> type) {  
    return (root, query, criteriaBuilder) ->  
criteriaBuilder.in(root.get(Product_.TYPE)).value(type);  
}
```

Hình 11:Lọc theo type của sản phẩm

- Lọc với nhiều điều kiện: Để lọc sản phẩm dựa trên một khoảng giá trị, có thể kết hợp nhiều điều kiện như sau:



```
public static Specification<Product> matchPrice(int min, int max) {  
    return (root, query, criteriaBuilder) -> criteriaBuilder.and(  
        criteriaBuilder.gt(root.get(Product_.PRICE), min),  
        criteriaBuilder.le(root.get(Product_.PRICE), max));  
}
```

Hình 12:Lọc các sản phẩm có giá nằm trong khoảng min và max

Để thực hiện lọc sản phẩm, chỉ cần gọi các phương thức được định nghĩa trong Repository và truyền các Specification tương ứng, giúp tách biệt rõ rang logic lọc dữ liệu và dễ dàng tái sử dụng các điều kiện lọc trong các ngữ cảnh khác nhau. Cách tiếp cận này mang lại sự linh hoạt cao, đặc biệt trong các trường hợp cần kết hợp nhiều điều kiện lọc phức tạp mà vẫn duy trì mà nguồn ngắn gọn và dễ hiểu

Page<Product>findAll(Specification<Product> spec, Pageable page);

5.2.i. Tìm kiếm sản phẩm theo tên

Tương tự như phần lọc sản phẩm thì phần chức năng tìm kiếm theo tên sản phẩm cũng định nghĩa một Specification để có thể truy vấn các sản phẩm từ database lên.



```
public static Specification<Product> nameLike(String name) {  
    return (root, query, criteriaBuilder) ->  
criteriaBuilder.like(root.get(Product_.NAME), "%" + name + "%");  
}
```

Hình 13:Tìm kiếm sản phẩm dựa theo tên

Để thực hiện phần auto complete trong chức năng tìm kiếm, ở phần controller liên quan đến chức năng này thực hiện lấy danh sách các tên sản phẩm đang có trong database hiện tại: `nameProducts` và trả ra cho giao diện. Nó sẽ gợi ý người dùng hoàn thành tên sản phẩm mà người dùng đang muốn tìm sau khi người dùng nhập từ một đến hai ký tự. Phần sử lý các sự kiện của phần auto complete có thể xem chi tiết tại [đây](#)

5.2.j. Gửi email thông báo đặt hàng thành công cho người dùng

Trong quá trình hoàn thiện chức năng mua hàng, một bước quan trọng là xác nhận đơn hàng để thông báo và gửi lời cảm ơn đến khách hàng. Việc này không chỉ giúp người dùng nắm bắt được trạng thái của đơn hàng mà còn tăng cường sự tin cậy và chuyên nghiệp cho ứng dụng. Để thực hiện chức năng này, hệ thống sẽ gửi một Email xác nhận sau khi khách hàng hoàn thành thanh toán

Cấu hình và triển khai gửi email xác nhận được trình bày dưới đây:



```
@Autowired  
private JavaMailSender mailSender;  
  
public void sendEmail(String toEmail, String subject, String  
body) {  
    SimpleMailMessage message = new SimpleMailMessage();  
    message.setFrom("foodstore247official@gmail.com");  
    message.setTo(toEmail);  
    message.setText(body);  
    message.setSubject(subject);  
  
    mailSender.send(message);  
}
```

Hình 14:Cấu hình triển khai gửi Email cho người dùng

Phương thức `sendEmail` này được thiết kế để gửi email xác nhận đến người dùng. Khi được gọi, phương thức này sẽ tạo một email với thông tin người nhận, tiêu đề và nội dung được chỉ định. Email sẽ được gửi từ địa chỉ: foodstore247official@gmail.com

Trong ItemController, tạo một controller để xử lý URL có endpoint `/afterOrder` để xử lý việc xác nhận đơn hàng. Khi khách hàng hoàn thành thanh toán, hệ thống sẽ lưu thông tin đơn hàng và gửi một email xác nhận như sau:

```
● ● ●

@GetMapping( "/afterOrder" )
public String getAfterOrderPage(HttpServletRequest request)
{
    HttpSession session = request.getSession(false);
    Long id = (long) session.getAttribute("id");

    User user = this.userService.getUserById(id);
    String email = user.getEmail();

    sendEmail.sendEmail(email, "Xác nhận đơn hàng",
                        "FoodStore chân thành cảm ơn bạn vì đã sử dụng
                        sản phẩm của chúng tôi!");

    return "client/cart/afterOrder";
}
```

Hình 15: Xử lý logic gửi Email khi có yêu cầu

Sau khi nhận yêu cầu từ endpoint `/afterOrder`, ứng dụng sẽ lấy email của người dùng và gọi phương thức `sendEmail` để gửi email xác nhận với lời cảm ơn về việc đã mua hàng.

5.2.k. Xác thực email đăng ký

Để đảm bảo rằng người dùng cung cấp một địa chỉ email hợp lệ khi đăng ký tài khoản, hệ thống sẽ yêu cầu xác nhận email. Điều này không chỉ giúp tránh đăng ký gian lận mà còn đảm bảo rằng dùng truy cập hợp pháp vào tài khoản mình trong tương lai. Xác nhận qua email là một phương pháp phổ biến và hiệu quả để thực hiện điều này.

Sau đây là các bước trong quá trình xác nhận email của người dùng khi đăng ký:

1. Cấu hình dịch vụ gửi email xác nhận: Trong lớp `SendEmailToVerify`, sử dụng `JavaMailSender` để gửi mã xác nhận OTP đến địa chỉ email của người dùng.

OTP là một chuỗi 6 chữ số được tạo ngẫu nhiên bằng phương thức getRandom().

```
● ● ●

@Service
public class SendEmailToVerify {
    public String getRandom() {
        Random rnd = new Random();
        int number = rnd.nextInt(999999);
        return String.format("%06d", number);
    }

    @Autowired
    private JavaMailSender mailSender;

    public void sendEmail(String toEmail, String subject, String
body) {
        SimpleMailMessage message = new SimpleMailMessage();
        message.setFrom("foodstore247official@gmail.com");
        message.setTo(toEmail);
        message.setText(body);
        message.setSubject(subject);

        mailSender.send(message);
    }
}
```

Hình 16:Cấu hình dịch vụ gửi email xác nhận

2. *Gửi mã OTP đến email người dùng khi đăng ký:* Khi người dùng nhập thông tin đăng ký và gửi lên, hệ thống sẽ tự động một mã OTP và gửi mã này qua email. Tại đây, controller `getVerifyPage` sẽ nhận email từ đối tượng `RegisterDTO`, tạo mã OTP và gửi email xác nhận

```

● ● ●

@PostMapping("/verify")
public String getVerifyPage(@ModelAttribute("registerUser")
@Valid RegisterDTO userDTO, BindingResult bindingResult,
Model model) {

    String email = userDTO.getEmail();
    String OTP = this.sendEmailToVerify.getRandom();
    this.sendEmailToVerify.sendEmail(email, "Xác nhận đăng ký", "Mã xác nhận đăng ký của bạn là: " + OTP);
    userDTO.setOTP(OTP);
    model.addAttribute("userDTO", userDTO);
    return "client/auth/verifyEmail";
}

```

Hình 17: Gửi mã OTP đến email người dùng khi đăng ký

3. Xác nhận mã OTP và lưu thông tin người dùng: Sau khi người dùng nhập mã OTP từ email, hệ thống sẽ kiểm tra xem mã OTP nhập vào có khớp với mã đã gửi hay không. Nếu mã OTP đúng, hệ thống sẽ lưu thông tin vào cơ sở dữ liệu

```

● ● ●

@PostMapping("/register")
public String handleRegister(@ModelAttribute("userDTO")
@Valid RegisterDTO userDTO,
BindingResult bindingResult,
@RequestParam("OTP_check") String OTP,
Model model) {

    String OTP_real = userDTO.getOTP();
    if (!OTP.equals(OTP_real)) {
        model.addAttribute("errorVerifyEmail", "Mã OTP không chính xác. Vui lòng nhập lại.");
        return "client/auth/verifyEmail";
    }

    User user = this.userService.registerDTOtoUser(userDTO);
    String hashPassword =
this.passwordEncoder.encode(userDTO.getPassword());
    user.setPassword(hashPassword);
    user.setRole(this.userService.getRoleByName("USER"));
    this.userService.handleSaveUser(user);
    return "client/homepage/registerSuccess";
}

```

Hình 18: Xác nhận mã OTP và lưu thông tin người dùng

5.2.1. Lấy lại mật khẩu người dùng thông qua email

Trong các hệ thống hiện đại, việc hỗ trợ người dùng khôi phục mật khẩu là tính năng quan trọng để tăng cường tính bảo mật và tính thuận tiện khi sử dụng. Để đảm bảo rằng chỉ người dùng chính chủ mới có thể thay đổi mật khẩu của mình, chúng ta có thể thực hiện quy trình xác minh qua email với mã định danh duy nhất (token). Dưới đây là toàn bộ quá trình cấu hình:

1. Gửi Email khôi phục mật khẩu: Khi người dùng yêu cầu lấy lại mật khẩu, hệ thống sẽ tạo một token duy nhất để xác nhận người dùng. Đoạn mã dưới đây thể hiện việc tạo token, lưu trữ trong cơ sở dữ liệu và gửi đường dẫn khôi phục mật khẩu qua email:

```
● ● ●

@PostMapping("/send-request-to-mail")
public String sendRequestToMail(@RequestParam("email") String
email) {
    String tokenEmail = UUID.randomUUID().toString();
    Token token = new Token();
    token.setEmail(email);
    token.setToken(tokenEmail);
    tokenService.saveToken(token);
    String resetLink = "http://localhost:8080/reset-password?
token=" + tokenEmail;
    sendEmail.sendEmail(email, "Xác nhận khôi phục mật khẩu",
"Nhấn vào đây để lấy lại mật khẩu: " + resetLink);
    return "redirect:/password";
}
```

Hình 19: Gửi Email khôi phục mật khẩu

Token này sẽ được lưu vào cơ sở dữ liệu để đảm bảo tính xác thực trong lần kiểm tra tiếp theo. Liên kết khôi phục mật khẩu được gửi qua email sẽ chứa token này và cho phép người dùng truy cập trang thay đổi mật khẩu

2. Xác nhận token và hiển thị thay đổi mật khẩu: Sau khi người dùng nhấn vào liên kết được gửi trong email, hệ thống sẽ xác thực token từ URL và tìm kiếm email người dùng tương ứng trong cơ sở dữ liệu. Nếu token hợp lệ, hệ thống sẽ trả về trang để người dùng nhập mật khẩu mới:

```

@GetMapping("/reset-password")
public String getResetPasswordPage(@RequestParam("token") String token, Model model) {
    String email = tokenService.getEmailByToken(token);
    User user = this.userService.getUserByEmail(email);
    Long id = user.getId();
    ResetPasswordDTO resetPasswordDTO = new
ResetPasswordDTO();
    resetPasswordDTO.setUserID(id);
    model.addAttribute("ResetPasswordDTO",
resetPasswordDTO);
    return "client/homepage/resetPassword";
}

```

Hình 20:Xác nhận token và hiển thị thay đổi mật khẩu

Trong các hệ thống lớn với số lượng yêu cầu cao, cách tiếp cận này có thể gặp vấn đề về token trùng lặp, gây sai logic. Để khắc phục nhược điểm này, ta có thể bổ sung biện pháp quản lý token với thời gian hết hạn hoặc sử dụng các phương pháp khác để tạo mã định danh duy nhất

5.2.m. Tích hợp thanh toán VNPay

Tài liệu tham khảo: <https://sandbox.vnpayment.vn/apis/>

Đầu tiên cần đăng ký tài khoản của môi trường test, và có được 3 tham số:

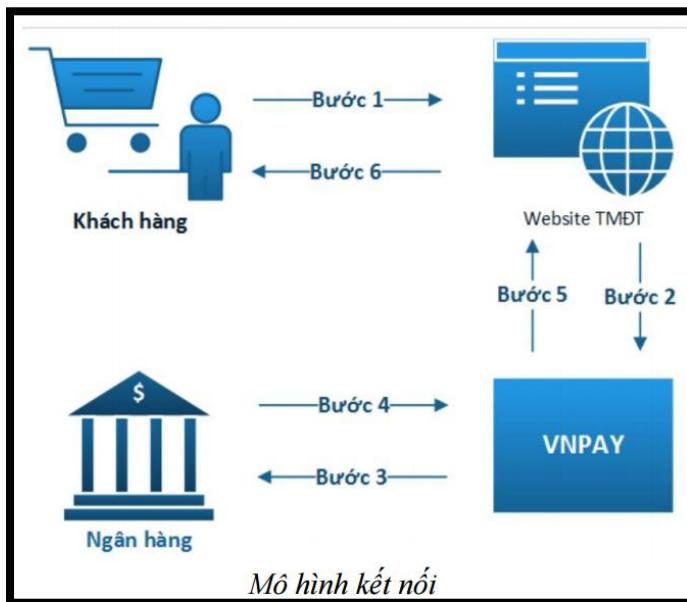
- 1. Terminal ID/ Mã website**
- 2. Secret Key/ Chuỗi bảo mật tạo bằng checksum**
- 3. Số thẻ dùng thử test**
- 4. Tài khoản để đăng nhập vào trang quản trị của merchant (chính là email/password khi đăng ký tài khoản)**

Ngân hàng	NCB
Số thẻ	9704198526191432198
Tên chủ thẻ	NGUYEN VAN A
Ngày phát hành	07/15
Mật khẩu OTP	123456

LOGIC nghiệp vụ:

- Đơn hàng, nếu lựa chọn hình thức thanh toán khi nhận hàng (COD), hoặc lựa chọn thanh toán qua VNPay, sẽ được khởi tạo trạng thái **PAYMENT_UNPAID** (chưa thanh toán)
- Đơn hàng, nếu lựa chọn thanh toán với VNPay, thanh toán thành công (**PAYMENT_SUCCEED**) và thanh toán thất bại (**PAYMENT_FAILED**)
- **Payment_ref** dùng để theo dõi đơn hàng

Giải thích mô hình tích hợp:



Hình 21: Mô hình kết nối

- Bước 1:** User chọn hình thức thanh toán với VNPAY, backend cần tạo ra URL thanh toán theo tiêu chuẩn VNPAY
- Bước 2:** User tiến hành thanh toán với giao diện của VNPAY
- Bước 3:** Do không có IPN, nên là sử dụng trực tiếp callback **vnp_ReturnUrl**. Sau khi nhận callback, backend thay đổi trạng thái order(từ UNPAID, chuyển thành PAYMENT_FAILED/PAYMENT_SUCCEEDED) dựa vào tham số **paymentRef**

Chú thích: IPN (Instant Payment Notification) được sử dụng để cập nhật tình trạng thanh toán cho giao dịch. Merchant cần gửi cho VNPAY URL này. Nhưng do tạo IPN cần liên hệ cho bên đối tác VNPAY cầu hình nên để đơn giản hóa, phần này sẽ không trình bày kỹ và sẽ không dùng đến.

Ý tưởng: tạo 1 URL thanh toán để redirect người dùng đến trang thanh toán cho VNPAY xử lý. Toàn bộ quy trình được tham khảo dựa trên code demo của VNPAY.

Chi tiết xem [tại đây](#)

Đến bước này tạm thời đã xong việc thanh toán, tuy nhiên trạng thái đơn hàng trong Database vẫn chưa được cập nhật. Do không có IPN, nên chúng ta sẽ sử dụng trực tiếp callback **vnp_ReturnUrl**. Sau khi nhận callback, backend thay đổi trạng thái order(từ Chưa thanh toán chuyển thành **Thanh toán thất bại/Thanh toán thành công**) dựa vào tham số **paymentRef**

URL callback minh họa:

```
http://localhost:8080/thanks?vnp_Amount=10000000&vnp_BankCode=NCB&vnp_BankTranNo=VNP14692534&vnp_CardType=ATM&vnp_OrderInfo=Thanh+toan+cho+ma+GD%3A037f1c72-4814-45a0-a088-2b786ae443cf&vnp_PayDate=20241123163354&vnp_ResponseCode=00&vnp_Tm  
nCode=VZG5&vnp_TransactionRef=
```

ctionNo=14692534&vnp_TransactionStatus=00&**vnp_TxnRef=037f1c72-4814-45a0-a088-2b786ae443cf&**

27

vnp_SecureHash=9eb0710828033da690da6a3c31667d47321b196f3a279581883
6ae6ecaae85cb07e9e02bf7fec0e6e694c960b545bbaac80bb03722c25a0e15f8896c
f714bd75

Ý tưởng: lấy trong URL callback 2 tham số **vnp_ResponseCode** và **vnp_TxnRef** bằng cách sử dụng **@RequestParam** tại Controller. Sau khi đã lấy được 2 tham số này ta tiến hành kiểm tra mã trạng thái trả về. Đối với mã **00** cho biết đã thanh toán thành công, sử dụng **vnp_TxnRef** để xác định order trong Database và tiến hành thay đổi trạng thái

5.2.n. Tích hợp login OAuth2 với tài khoản Google và Github

Tài liệu tham khảo <https://spring.io/guides/tutorials/spring-boot-oauth2>

Đầu tiên chúng ta cần cài đặt thư viện để hỗ trợ OAuth2

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

Giải thích mô hình tích hợp:

Bước 1: Cấu hình dự án: Cài đặt thư viện oauth2-client

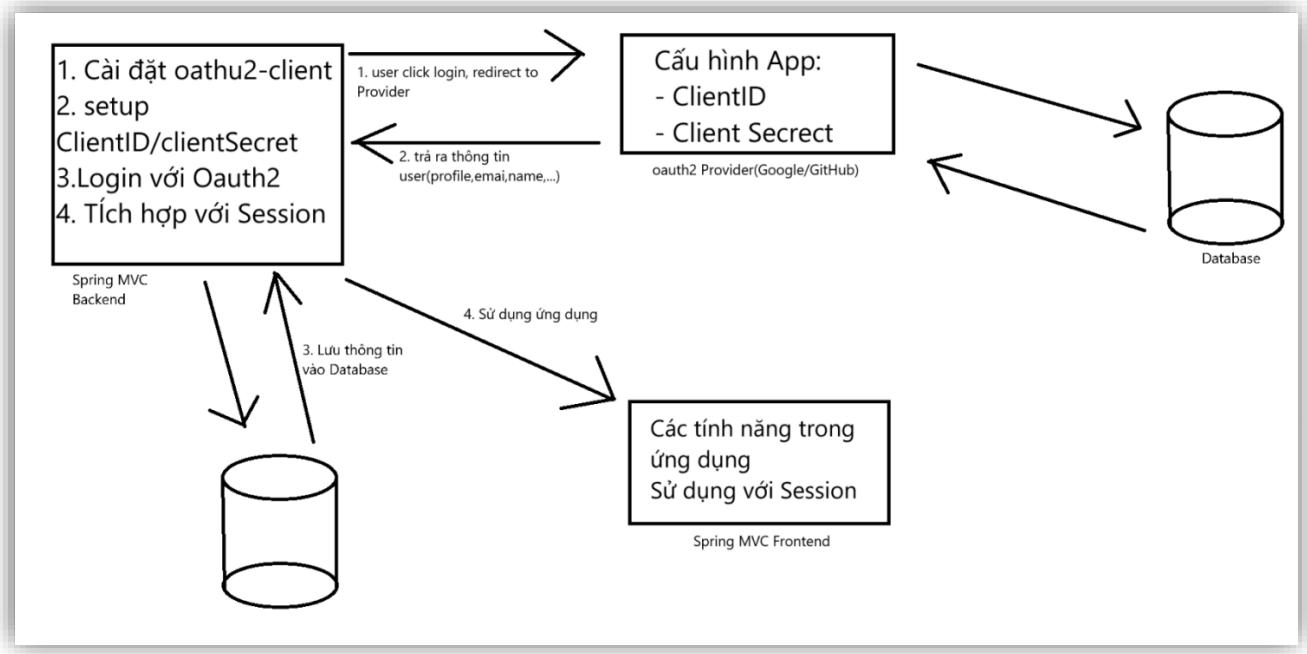
Bước 2: Cấu hình oauth2 Provider(Google/Github)

Mục tiêu ở bước này là để có được client-id và client-secret

Bước 3: Setup client-id và client-secret trong ứng dụng Spring

Bước 4: Xử lý kết quả login, lưu vào session

Chi tiết được trình bày trong hình vẽ sau:



Hình 20: Luồng hoạt động

Tiếp đến ta cần tạo Google Project, cấu hình như sau:

1. Cấu hình Google App

Authorized JavaScript origins: <https://localhost:8080>

Authorized redirect URIs: <http://localhost:8080/login/oauth2/code/google>

2. Cấu hình Java Spring

```
//application.properties
spring.security.oauth2.client.registration.google.client-id=<your client id>
spring.security.oauth2.client.registration.google.client-secret=<your client secret>
```

Chi tiết xem tại [đây](#)

Tiếp đến cần cấu hình Spring Security để hệ thống biết được người dùng đang đăng nhập là tài khoản tạo từ Email hay tài khoản đăng nhập với Google hoặc Github?

Trong bài này là </oauth2/authorization/google> và </oauth2/authorization/github>

```
.oauth2Login(oauth2 -> oauth2.loginPage(loginPage: "/login"))
```

Bây giờ này sinh 1 vấn đề. Làm sao để hệ thống biết được người dùng đang đăng nhập là tài khoản tạo từ Email hay tài khoản đăng nhập với Google hoặc Github? Để giải quyết điều này, ta sẽ bổ sung thêm trường **PROVIDER** vào bảng users với các thông tin có thể nhận: LOCAL, GOOGLE, GITHUB

Logic nghiệp vụ:

- User login với Google, đặt role mặc định là Role_User
- Khi login thành công, tiến hành check mail:
 - o Nếu Email chưa tồn tại, tiến hành tạo User với các tham số sau:
 - Email: lấy theo kết quả login trả về
 - fullName: lấy theo kết quả login trả về
 - avatar: default-google.png
 - role: tạo đối role, ứng với ROLE_USER
 - provider: GOOGLE

- Nếu email đã tồn tại, không cần tạo user → trả về null

Tương tự khi cấu hình để login với tài khoản github

1. Đầu tiên cần tạo Github OAuth App bằng cách truy cập đường link:

<https://github.com/settings/developers>

Authorized JavaScript origins: <http://localhost:8080>

Authorized redirect URIs: <http://localhost:8080/login/oauth2/code/github>

2. Cấu hình Java Spring:

```
spring.security.oauth2.client.registration.github.client-id=<your client id>
spring.security.oauth2.client.registration.github.client-secret=<your client secret>
```

Lưu ý: Do cơ chế bảo mật của Github vì thế để đăng nhập được thì người dùng cần publish thông tin về email và cần có 1 username công khai

Đến đây nay sinh một vấn đề, nếu người dùng sử dụng email để đăng ký tài khoản Github, vậy khi đăng nhập vào ta cần phân biệt giữa 2 tài khoản này bằng cách sử dụng trường **PROVIDER** đã định nghĩa trước đó. Ta tiến hành so sánh, nếu user.getProvider trùng với **PROVIDER** của người đăng nhập thì ta chấp nhận người dùng vào, còn không sẽ hiển thị thông báo: đã tồn tại tài khoản trong hệ thống. Do việc kiểm tra tài khoản được thực hiện trong Service, vì thế nếu muốn render ra màn hình thông báo cho người dùng, ta cần thực hiện như sau:

Xử lý lỗi đăng nhập OAuth2 bằng cách thiết lập:

CustomOAuth2AuthenticationFailureHandler

```
@Component
public class CustomOAuth2AuthenticationFailureHandler extends SimpleUrlAuthenticationFailureHandler {
    @Override
    public void onAuthenticationFailure(HttpServletRequest request, HttpServletResponse response,
        AuthenticationException exception)
        throws IOException, ServletException {
        String errorMessage = "Đăng nhập thất bại. Vui lòng thử lại.";
        if (exception instanceof CustomOAuth2Exception) {
            errorMessage = exception.getMessage();
        }
        // Redirect về trang login với query string chứa thông báo lỗi
        getRedirectStrategy().sendRedirect(request, response,
            "/login?error1=" + URLEncoder.encode(errorMessage, enc:"UTF-8"));
    }
}
```

Tiếp đến cấu hình trong **SecurityConfig**

```
.oauth2Login(oauth2 -> oauth2.loginPage(loginPage: "/login")
    .successHandler(customSuccessHandler())
    .failureHandler(customFailureHandler)
    .userInfoEndpoint(user -> user
        .userService(new CustomOAuth2UserService(userService)))
```

Xử lý User Service và kiểm tra tài khoản

```
String email = (String) attributes.get("email");
if (email != null && userService.getUserByEmail(email) != null) {
    throw new CustomOAuth2Exception("Tài khoản đã tồn tại.");
}
```

Cuối cùng hiển thị ra thông báo cho người dùng

```
<c:if test="${not empty param.error}">
    <div class="alert alert-danger">
        ${param.error}
    </div>
</c:if>
```

5.2.o. Tích hợp chat-bot AI

Để tăng thêm trải nghiệm cho người dùng, trong bài này nhóm 1 đã tích hợp thêm chat-bot AI nhằm tư vấn cho người dùng về sản phẩm, gợi ý thực đơn phù hợp và hỗ trợ giải đáp thắc mắc một cách nhanh chóng, chính xác. Điều đặc biệt là model AI sử dụng trong bài này không phải model train từ đầu mà là sử dụng API của bên thứ 3, cụ thể trong bài này là chat-bot Gemini của Google. Quy trình triển khai như sau:

Bước 1: Truy cập <https://aistudio.google.com/app/apikey> để lấy API Key

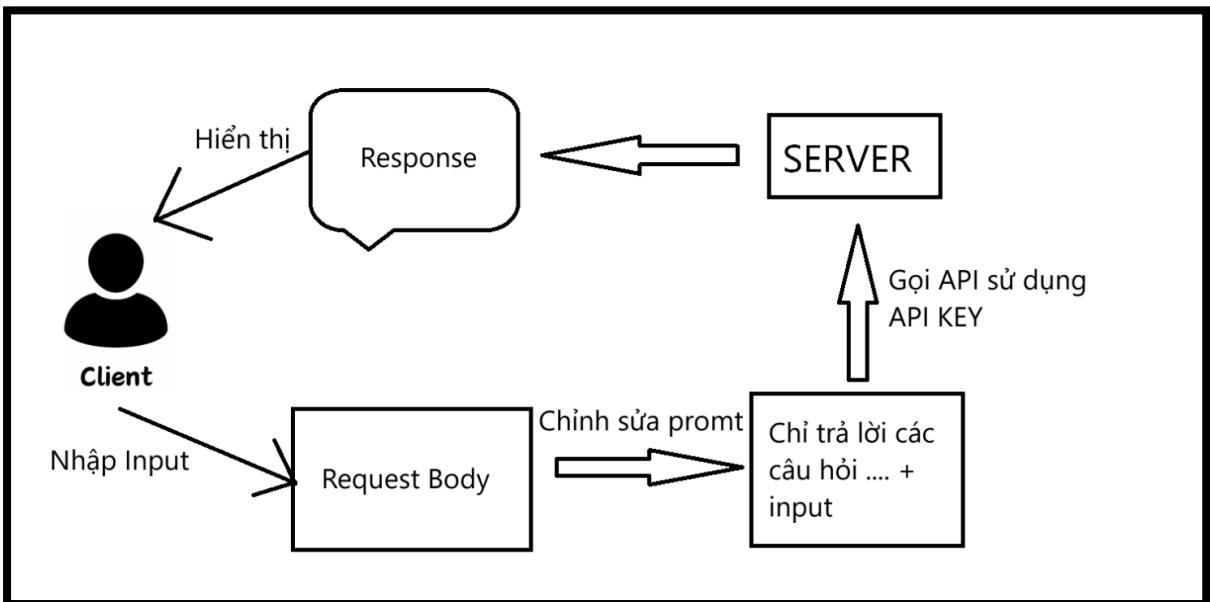
Bước 2: Tiến hành test API với Postman với Request Body truyền lên là :

```
{
  "contents": [
    [
      {
        "parts": [
          [
            {
              "text": "Explain how AI works"
            }
          ]
        ]
      }
    ]
  }
}
```

Với text chính là input mà người dùng nhập vào. Sau khi test thành công.
Tiến hành tích hợp vào trong ứng dụng

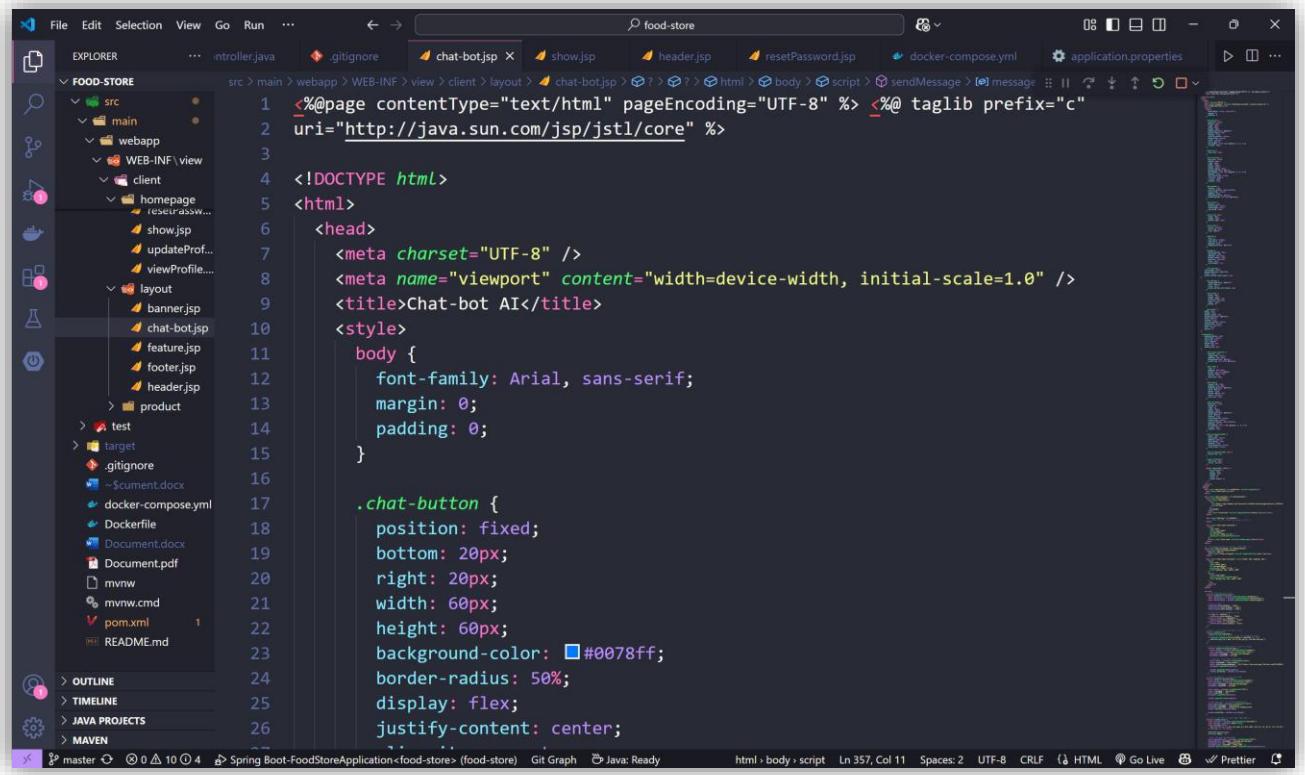
Bước 3: Tích hợp vào ứng dụng

Giải thích mô hình tích hợp



Hình 22: Luồng hoạt động

Đầu tiên tạo 1 file JSP chat-bot.jsp để xây dựng giao diện cho phần này với mục đích là sẽ sử dụng lại tại các giao diện khác.



```
<%@page contentType="text/html" pageEncoding="UTF-8" %> <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>Chat-bot AI</title>
<style>
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
}

.chat-button {
    position: fixed;
    bottom: 20px;
    right: 20px;
    width: 60px;
    height: 60px;
    background-color: #0078ff;
    border-radius: 50%;
    display: flex;
    justify-content: center;
}
```

Tiếp đến sử dụng JavaScript để gọi API với công cụ được tích hợp sẵn là fetch. Nhưng trước khi đó ta cần cấu hình proxy gián tiếp qua Controller, không gọi trực tiếp được

```

14  @RestController
15  @RequestMapping("/gemini-proxy")
16  public class GeminiController {
17
18      @PostMapping
19      public ResponseEntity<String> proxyToGemini(@RequestBody String requestBody) throws IOException {
20          String apiUrl = "https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:gene";
21          URL url = new URL(apiUrl);
22          HttpURLConnection conn = (HttpURLConnection) url.openConnection();
23          conn.setRequestMethod(method:"POST");
24          conn.setRequestProperty(key:"Content-Type", value:"application/json");
25          conn.setDoOutput(doOutput:true);
26
27          try (OutputStream os = conn.getOutputStream()) {
28              os.write(requestBody.toString().getBytes(StandardCharsets.UTF_8));
29          }
30
31          BufferedReader reader = new BufferedReader(new InputStreamReader(conn.getInputStream()), StandardCharsets.UTF_8);
32          StringBuilder responseStr = new StringBuilder();
33          String line;
34          while ((line = reader.readLine()) != null) {
35              responseStr.append(line);
36          }
37          reader.close();
38
39      }

```

Bây giờ chỉ cần sử dụng JavaScript để gọi API và lấy data trả về

```

1  <%@page contentType="text/html" pageEncoding="UTF-8" %> <%@ taglib prefix="c"
393
394
395
396    // lấy Data từ API của gemini
397    async function fetchGeminiResponse(userText) {
398        try {
399            const requestBody = {
400                contents: [{ parts: [{ text: userText }] }],
401            };
402
403            const response = await fetch("/gemini-proxy", {
404                method: "POST",
405                headers: { "Content-Type": "application/json" },
406                body: JSON.stringify(requestBody),
407            });
408
409            // xóa thông báo đang nhập
410            const typingIndicator = document.getElementById("typingIndicator");
411            if (typingIndicator) {
412                typingIndicator.remove();
413            }
414
415            if (!response.ok) throw new Error("Lỗi kết nối API");
416
417            const data = await response.text();

```

Tuy nhiên đến đây ứng dụng vẫn chưa thể chạy được do cấu hình CSRF từ phía Server. CSRF (**Cross-Site Request Forgery**) là một dạng tấn công bảo mật, trong đó

kết tấn công lợi dụng việc một người dùng đã đăng nhập để gửi yêu cầu trái phép đến máy chủ. CSRF Token được sử dụng để xác minh rằng yêu cầu đến từ một nguồn hợp lệ, giúp bảo vệ các thao tác **có thay đổi trạng thái** trên máy chủ, chẳng hạn như cập nhật thông tin, xóa dữ liệu, hoặc thực hiện giao dịch. CSRF Token được thiết kế để bảo vệ ứng dụng khỏi các cuộc tấn công giữa các trang web, nhưng khi gọi API Gemini – một dịch vụ bên ngoài không liên quan đến phiên đăng nhập của người dùng trên ứng dụng – việc kiểm tra CSRF Token là không cần thiết. Vì thế ta cần bỏ qua CSRF Token cho phép các request tới Gemini API, còn lại các endpoint khác vẫn cần yêu cầu CSRF protection. Để làm được điều này, ta cần cấu hình như sau:

```
.csrf(csrf -> csrf
    // Cho phép các request tới Gemini API mà không cần CSRF token
    .ignoringRequestMatchers(...patterns:"/gemini-proxy/**")
    // Các endpoint khác vẫn yêu cầu CSRF protection
    .csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse())))

```

Bây giờ, mở ứng dụng để kiểm tra thành quả

The screenshot shows a food store website named "FoodStore". The main page displays a grid of products including grapes, porridge, mashed vegetables, and Nestle NAN baby food. Each product card includes a price, a "Thêm vào giỏ hàng" button, and a brief description. A search bar and a login button are at the top right. In the bottom right corner, there is an AI ChatBot window with a conversation history and input fields.

Hình 23: Kết quả sau khi tích hợp bot-chat

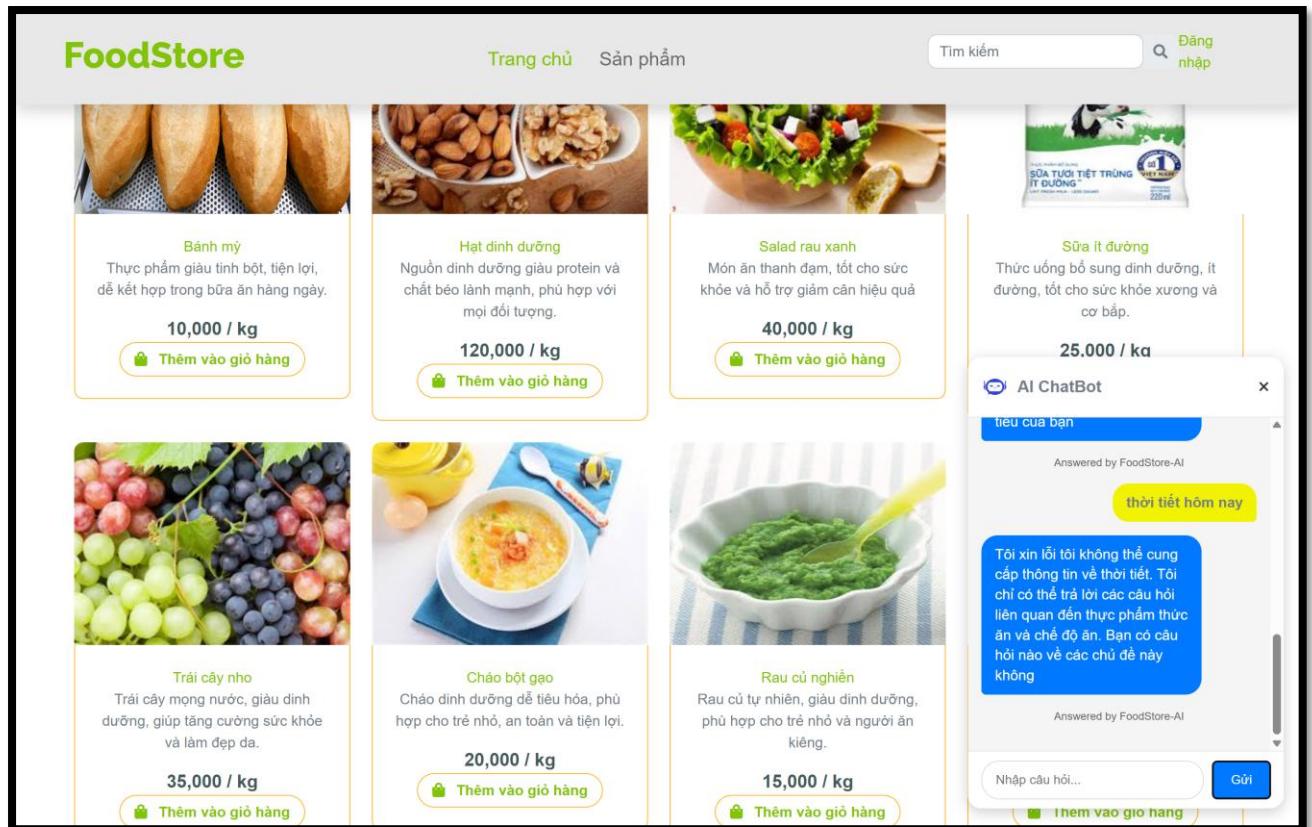
Tuy nhiên mục đích ban đầu của nhóm 1 tạo ra chat-bot này là chỉ để sử dụng hỏi đáp những vấn đề liên quan đến chủ đề thực phẩm, do đó cần giới hạn phạm vi, lĩnh vực mà người dùng có thể hỏi. Để làm được điều này, ta chỉ cần chỉnh sửa input trước khi gửi request lên server

```

const message = userInput.value.trim();
const messageCallAPI =
  "Chỉ trả lời các câu hỏi liên quan đến thực phẩm, thức ăn, chế độ ăn. Viết thành đoạn văn và loại bỏ tất cả ký tự đặc biệt khỏi
if (message === "") return;
addUserMessage(message);

```

Khi đó, nếu người dùng hỏi về một chủ đề khác ngoài thực phẩm, chat-bot sẽ không trả lời



Hình 24: Kết quả sau khi chỉnh lại prompt

Để tái sử dụng chat-bot tại các trang giao diện khác, ta sử dụng cú pháp trong jsp cho phép import 1 file jsp vào 1 file jsp khác

```

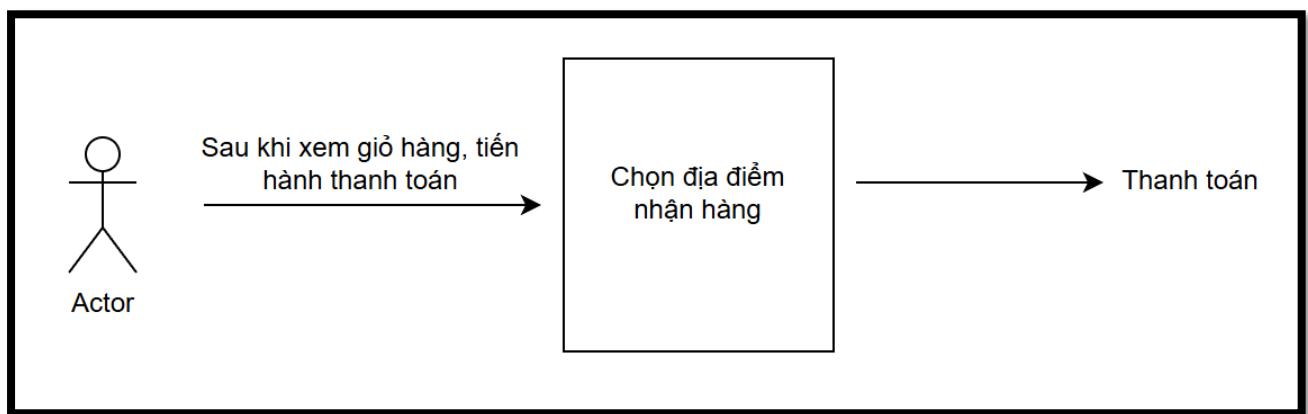
<jsp:include page="../layout/footer.jsp" />
<jsp:include page="../layout/chat-bot.jsp" />

```

5.2.p. Tích hợp API Giao hàng nhanh trong việc tính toán chi phí vận chuyển

Tính năng tích hợp API giao hàng nhanh giúp website tự động tính toán chi phí vận chuyển một cách nhanh chóng dựa trên vị trí người nhận, trọng lượng đơn hàng và loại hình giao hàng. Điều này không chỉ giúp khách hàng biết trước chi phí vận chuyển ngay trong quá trình đặt hàng, mà còn tối ưu hóa quy trình xử lý đơn của cửa hàng, đảm bảo giao hàng thực phẩm tươi sống đến tay người dùng đúng thời gian và chi phí hợp lý. Tài liệu tham khảo tại [đây](#)

Ý tưởng: trước khi đến bước chọn phương thức thanh toán, sẽ hiện ra 1 giao diện cho phép người dùng chọn vị trí nhận hàng. Luồng xử lý như sau:



Hình 25: Luồng xử lý khi thanh toán

Dưới đây là design của màn hình khi người dùng bấm thanh toán sau khi xem giỏ hàng

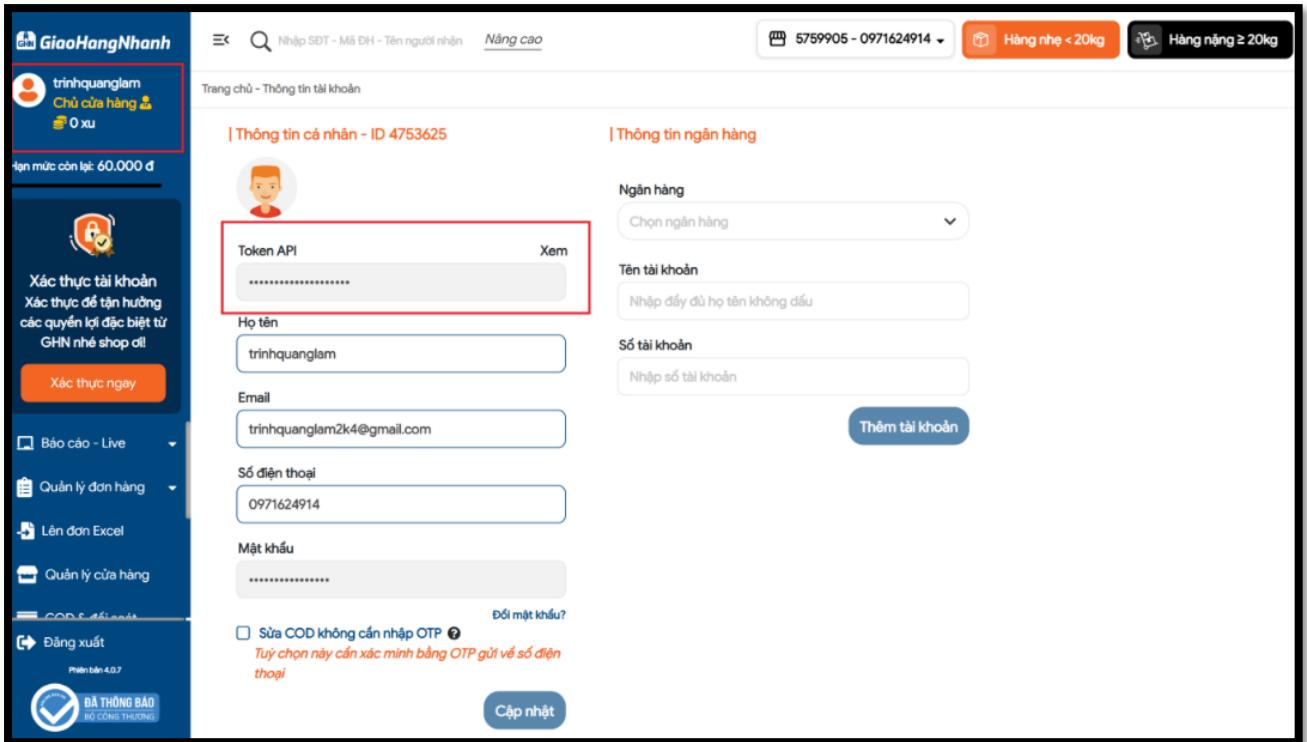
The screenshot shows a user interface for calculating delivery costs. At the top, there's a title 'Tính phí vận chuyển' (Calculate delivery cost) next to a green truck icon. Below it are three dropdown menus: 'Tỉnh/Thành phố' (Province/City) set to 'Hà Nội', 'Quận/Huyện' (District/County) with a placeholder 'Chọn quận/huyện' (Select district/county), and 'Phường/Xã' (Neighborhood/Village) with a placeholder 'Chọn phường/xã' (Select neighborhood/village). A 'Địa chỉ chi tiết' (Detailed address) input field contains the placeholder 'Nhập số nhà, tên đường...'. A large green button labeled 'Tính phí vận chuyển' (Calculate delivery cost) is centered below these fields. Below this section, a box displays the result with the title 'Kết quả tính phí vận chuyển' (Delivery cost calculation results) and the message 'Chưa có thông tin' (No information available).

Hình 26: Giao diện khi chọn vị trí nhận hàng

Các bước thực hiện:

Bước 1: Đăng ký tài khoản Giao hàng nhanh tại [đây](#)

Bước 2: Tại giao diện, chọn góc bên trái để xem thông tin cá nhân. Mục đích để lấy Token API phục vụ cho việc gọi API



Hình 27: Giao diện khi bấm xem thông tin cá nhân

Để lấy được tỉnh thành cũng như quận/huyện tương ứng với tỉnh thành đó, ta sẽ sử dụng API có sẵn của Giao hàng nhanh. Sau đây là 1 ví dụ về API lấy tất cả thông tin tỉnh, thành phố

```
curl --location 'https://online-gateway.ghn.vn/shiip/public-api/master-data/province' \
--header 'Content-Type: application/json' \
--header 'Token: ed67dc49-2835-11f0-8c8d-faf19a0e6e5b'
```

Tương tự để tính được chi phí vận chuyển, chúng ta sẽ sử dụng API bên Giao hàng nhanh được cung cấp sẵn và sử dụng Token API để gọi. Dưới đây là thông tin về API và các param được truyền vào

```
curl --location 'https://online-gateway.ghn.vn/shiip/public-api/v2/shipping-order/fee' \
--header 'Content-Type: application/json' \
--header 'Token: ed67dc49-2835-11f0-8c8d-faf19a0e6e5b' \
--header 'ShopId: 5759905' \
--data '{
    "from_district_id": 1542,
    "from_ward_code": "1B1507",
    "service_id": 53321,
    "service_type_id": null,
    "to_district_id": 2264
}'
```

```

    "to_ward_code": "90816,
    "height": 50,
    "length": 20,
    "weight": 200,
    "width": 20,
    "insurance_value": 10000,
    "cod_failed_amount": 2000,
    "coupon": null
  }

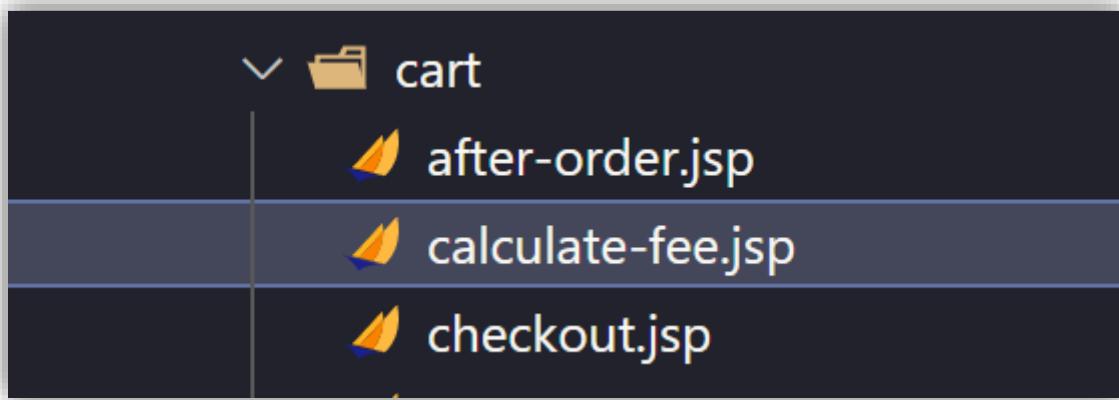
```

Giải thích về các tham số cần được truyền vào:

1. **from_district_id**: Quận huyện nơi gửi. Trong bài nhóm sẽ cố định một giá trị với giá trị **1542** tương ứng với quận Hà đông
2. **from_ward_code**: phường xã nơi gửi. Trong bài nhóm sẽ cố định một giá trị với giá trị "**1B1507**" tương ứng với Phường Mô Lao
3. **to_district_id**: Quận huyện nơi người nhận, mã này sẽ lấy từ người dùng
4. **to_ward_code**: Phường/xã nơi nhận cái này sẽ lấy từ người dùng
5. **service_id**: Trong bài này sẽ để là **53321** tương ứng với dịch vụ Giao hàng tiết kiệm (còn có các dịch vụ như giao hàng nhanh, giao hỏa tốc,... tuy nhiên trong bài này sẽ sử dụng dịch vụ phổ biến nhất)
6. **service_type_id**: tương với service_id **53321** sẽ là **3**
7. **height**: Trong bài này nhóm 1 sẽ cố định một giá trị là 50 cm tương ứng với gói hàng gửi đi có chiều cao 50 cm
8. **length**: Trong bài này nhóm 1 sẽ cố định một giá trị là 20 cm tương ứng với gói hàng gửi đi có chiều dài mặt đáy là 20cm
9. **weight**: Trong bài này nhóm 1 sẽ cố định một giá trị là 500g tương ứng với gói hàng gửi đi có cân nặng là 500 gam
10. **width**: Trong bài này nhóm 1 sẽ cố định một giá trị là 20 cm tương ứng với hàng gửi đi có chiều rộng mặt đáy là 20cm
11. **insurance_value**: Giá trị này Giao hàng nhanh sẽ dựa vào để tính chi phí bảo hiểm trong trường hợp hàng hóa bị mất hoặc hàng hóa không đúng như dự kiến. Giá trị này sẽ cộng tổng giá trị hàng hóa lại
12. **coupon**: Mã giảm giá (nếu có). Trong bài này nhóm 1 sẽ để trống
13. **cod_failed_amount**. Cái này là số tiền giao hàng nhanh sẽ thu nếu hàng giao không thành công. Trong bài này nhóm 1 chúng em sẽ để với giá trị 0

Địa chỉ người gửi trong bài này nhóm 1 chúng em sẽ để là địa chỉ của Học Viện Công nghệ Bưu chính Viễn Thông tương ứng với Quận Hà Đông, Phường Mô Lao
Các bước triển khai trong source code:

Bước 1: Tạo 1 file để hiển thị page khi người dùng bấm thanh toán

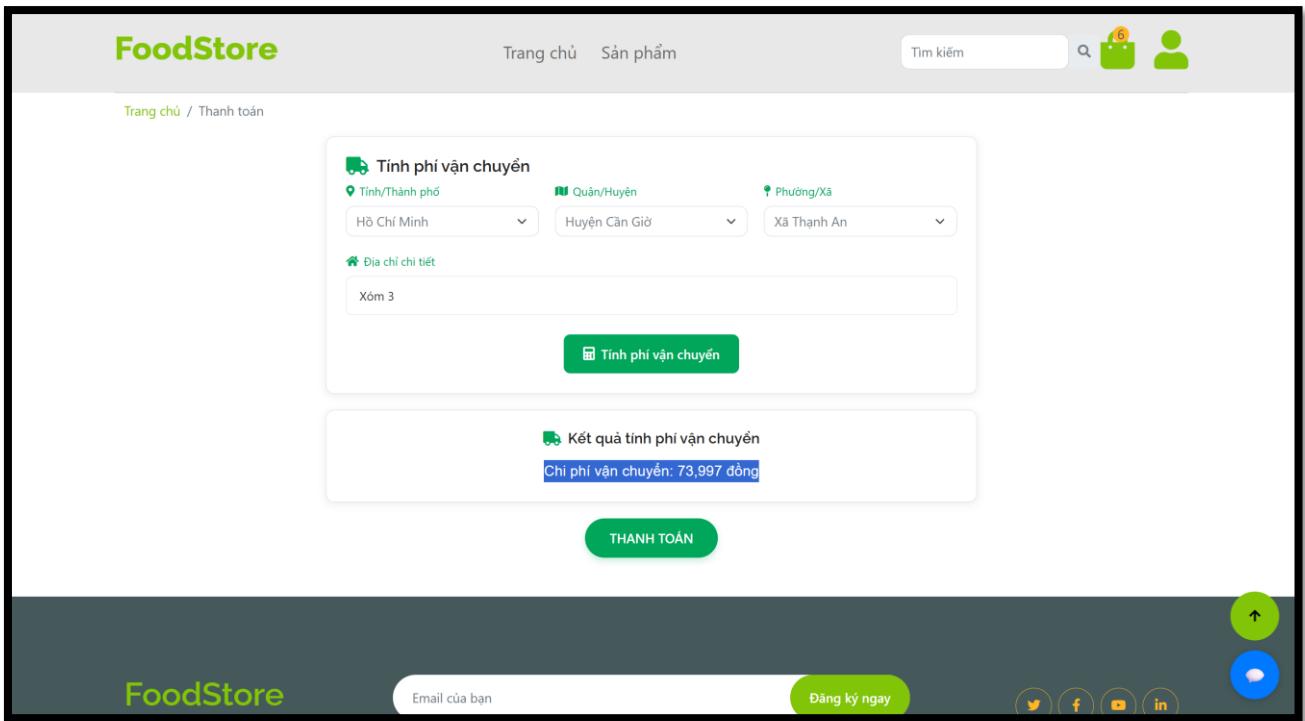


Bước 2: Dựa trên Design ở trên, xây dựng giao diện tương ứng

Ở bước này, xây dựng các hàm để gọi API lấy các giá trị tỉnh thành và thị xã tương ứng sử dụng JavaScript, sau đó fill thông tin vào trong các thẻ select

```
4     // Hàm gọi API để lấy danh sách quận/huyện dựa trên provinceID
5     function getDistricts(provinceId) {
6         // Kiểm tra nếu không có provinceID
7         if (!provinceId) {
8             console.error("Province ID is required");
9             return;
10        }
11        // Cấu hình request
12        const url =
13            "https://online-gateway.ghn.vn/shiip/public-api/master-data/district";
14        const token = "ed67dc49-2835-11f0-8c8d-faf19a0e6e5b";
15        // Sử dụng fetch API để gửi request
16        fetch(url, {
17            method: "POST",
18            headers: {
19                "Content-Type": "application/json",
20                Token: token,
21            },
22            body: JSON.stringify({
```

Bước 3: Sau khi người dùng chọn hết và bấm tính toán chi phí. Tiến hành gọi API với các giá trị và người dùng vừa chọn



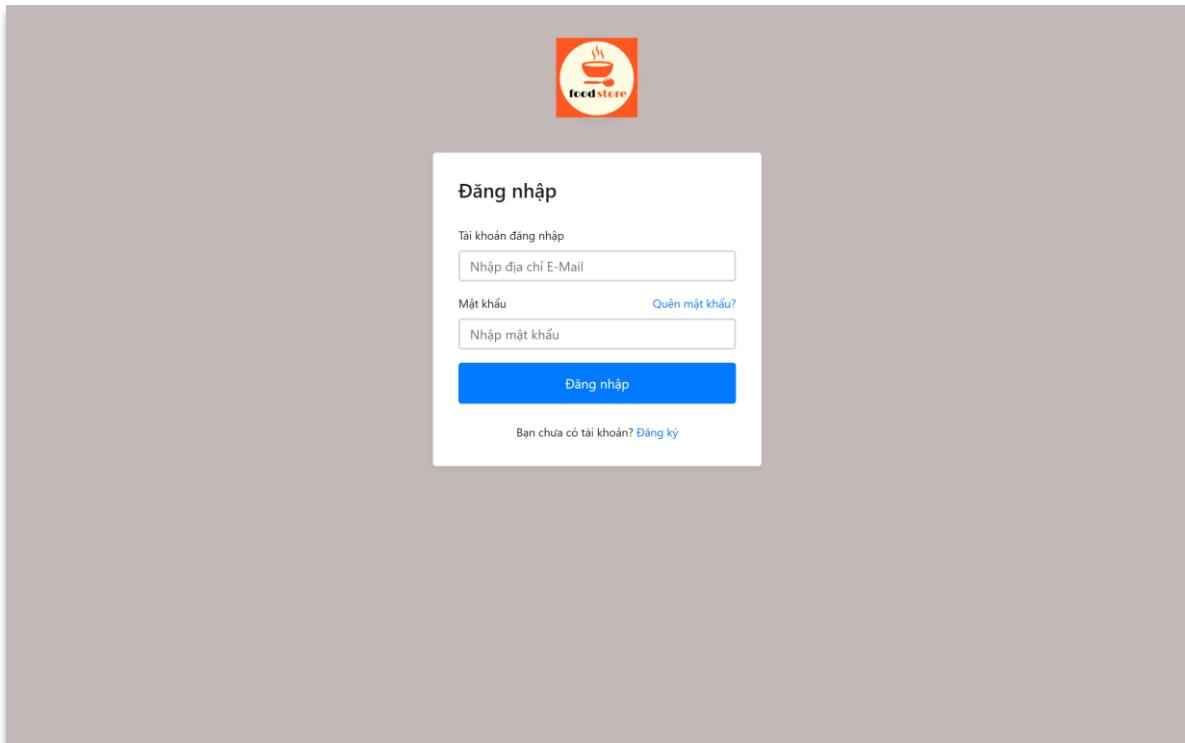
Hình 28: Giao diện sau khi tính toán chi phí
Có thể thấy chi phí vận chuyển từ Học viện Công nghệ Bưu chính viễn thông đến Xóm 3, Xã Thạnh An, Huyện Cần Giờ, Thành phố Hồ Chí Minh là **73,997đ**

5.3. Công cụ hỗ trợ

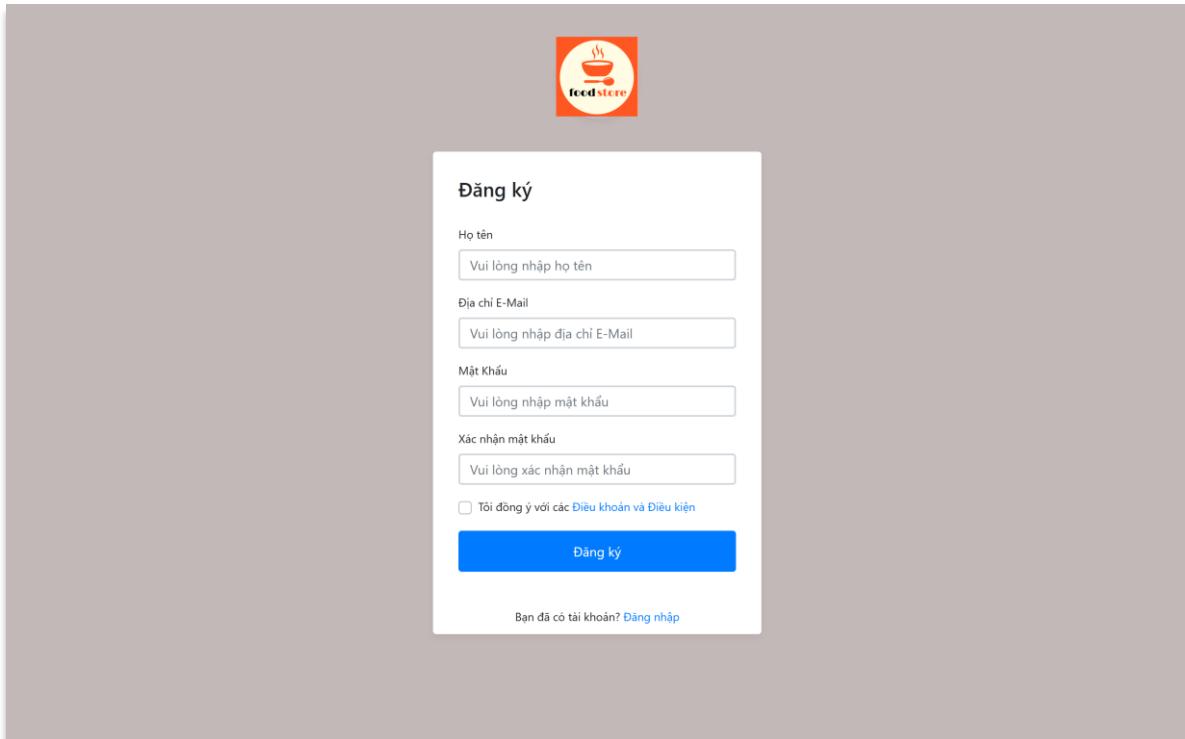
- **Spring Initializr:** Một công cụ được cung cấp bởi các nhà phát triển của Spring Framework giúp khởi tạo cấu trúc thư mục ban đầu cũng như các dependency cho dự án sử dụng Spring Boot một cách dễ dàng. Chi tiết xem tại [đây](#)
- **Visual Studio Code:** Một trình soạn thảo mã nguồn mở và miễn phí được phát triển bởi Microsoft. Đây là một công cụ mạnh mẽ, hỗ trợ nhiều ngôn ngữ lập trình khác nhau và được cộng đồng lập trình viên ưa chuộng nhờ vào tính linh hoạt và hiệu năng cao. Chi tiết xem tại [đây](#)
- **Git + Github:** Quản lý source code, tăng cường khả năng làm việc nhóm giúp quá trình phát triển nhanh và dễ dàng hơn. Chi tiết xem tại [đây](#)
- **Spring Boot Devtool:** Một module hỗ trợ phát triển nhanh của Spring Boot, cho phép tự động khởi động lại ứng dụng khi có thay đổi trong mã nguồn, tắt bộ nhớ đệm của template, và hỗ trợ tải lại trình duyệt tự động, giúp lập trình viên tiết kiệm thời gian và nâng cao hiệu quả khi phát triển ứng dụng. Chi tiết xem tại [đây](#)
- **Diagrams.net:** Giúp mô hình hóa quan hệ giữa các thực thể khi xây dựng cơ sở dữ liệu. Chi tiết xem tại [đây](#)

PHẦN 6: KẾT QUẢ

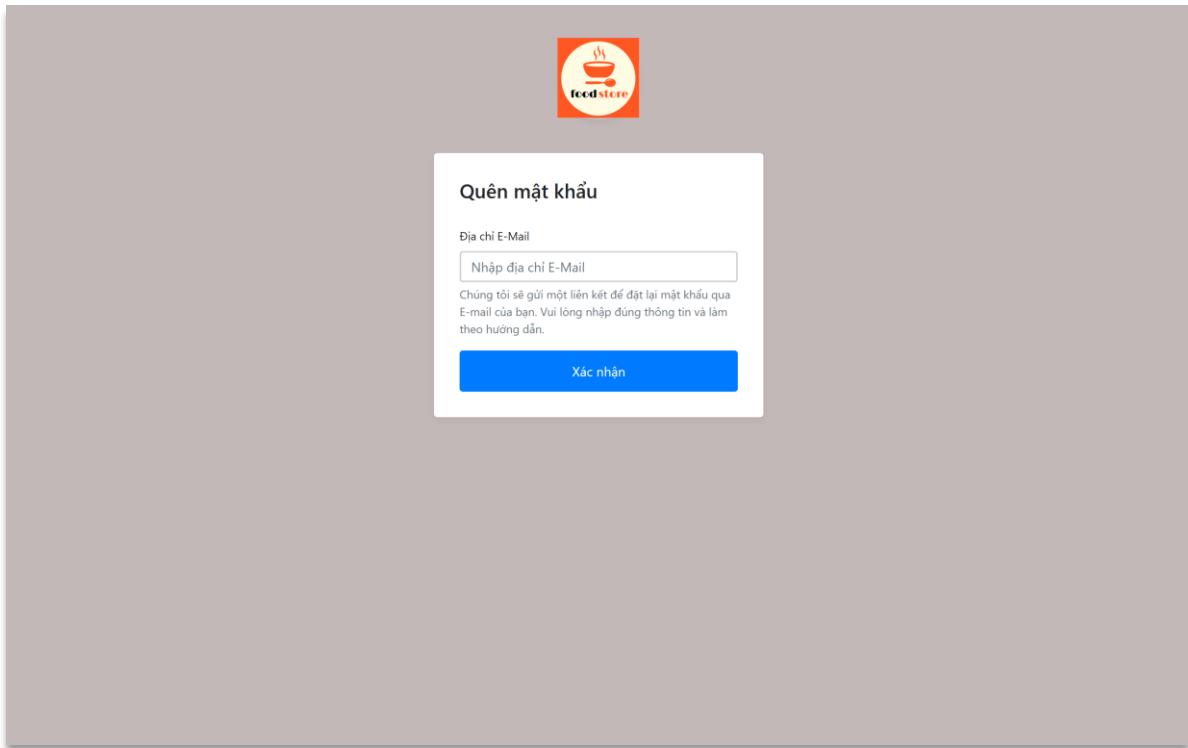
6.1. Giao diện phía người dùng



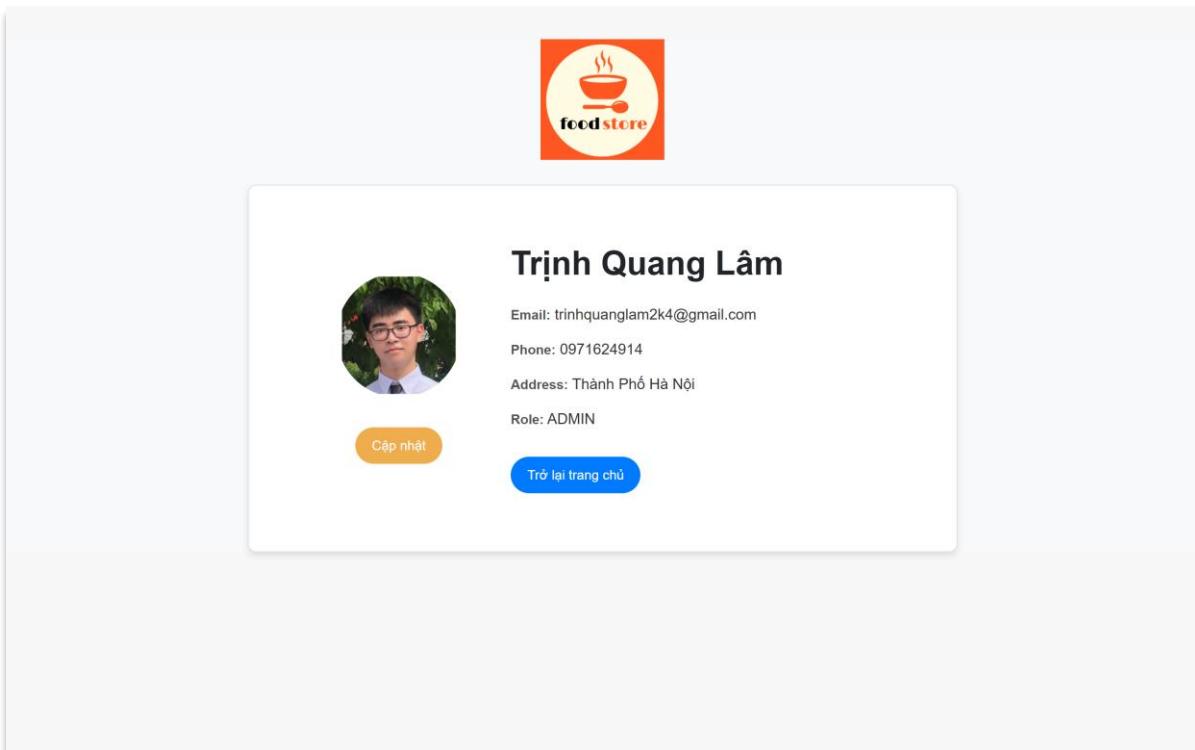
Hình 29. Giao diện màn hình đăng nhập



Hình 30. Giao diện giao diện màn hình đăng ký



Hình 31. Giao diện quên mật khẩu



Hình 32. Giao diện quản lý tài khoản

The screenshot shows the FoodStore website's shopping history page. At the top, there is a navigation bar with links for 'Trang chủ' and 'Sản phẩm'. On the right side of the navigation bar are search, cart, and user profile icons. Below the navigation bar, the URL 'Trang chủ / Lịch sử mua hàng' is displayed. The main content area is titled 'Lịch sử mua hàng' and contains a table showing two items purchased:

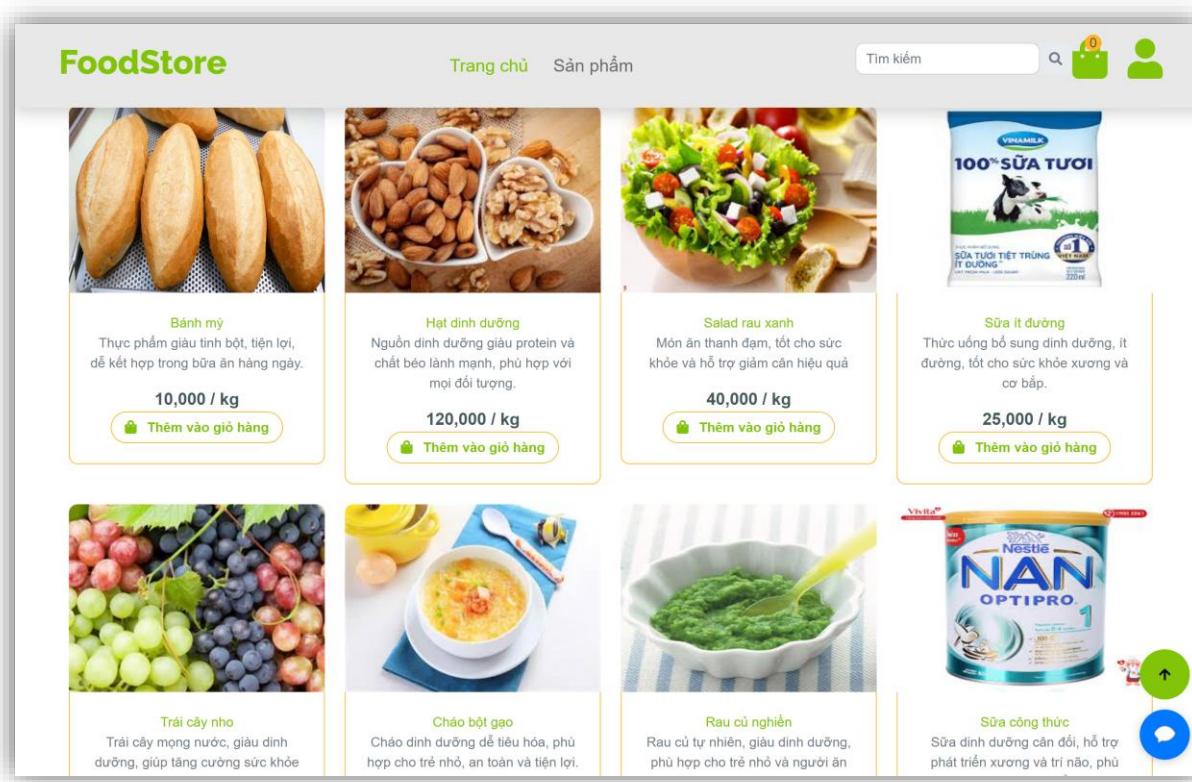
Sản phẩm	Tên	Giá cả	Số lượng	Thành tiền	Trạng thái
Số thứ tự : 12		120,000 đ			Chưa xử lý
	Bánh mỳ	10,000 đ	4	40,000 đ	
	Salad rau xanh	40,000 đ	2	80,000 đ	

Below the shopping history, there is a dark banner with the FoodStore logo, social media links (Twitter, Facebook, YouTube, LinkedIn), and a speech bubble icon.

Hình 33. Giao diện lịch sử mua hàng

The screenshot shows a password change form titled 'Đổi mật khẩu'. It features a logo for 'foodstore' with a coffee cup icon. The form has two input fields: 'Mật khẩu hiện tại' (Current password) and 'Mật khẩu mới' (New password), both with placeholder text 'Vui lòng nhập mật khẩu hiện tại' and 'Vui lòng nhập mật khẩu mới'. A blue 'Đổi mật khẩu' button is located at the bottom of the form.

Hình 34. Giao diện đổi mật khẩu



Hình 35. Giao diện danh sách sản phẩm

Mục đích sử dụng
 Giảm cân Tăng cơ Tăng chiều cao
 Tăng cân

Đối tượng
 Dành cho người tập thể hình
 Dành cho dân văn phòng
 Dành cho phụ nữ có thai
 Dành cho bé dưới 1 tuổi
 Dành cho người ăn kiêng Tất cả

Loại
 Rau Củ Trái cây
 Thực phẩm giàu protein
 Thực phẩm chứa tinh bột Thức uống

Mức giá
 Dưới 100 nghìn Từ 100 - 150 nghìn
 Từ 150 - 200 nghìn Trên 200 nghìn

Sắp xếp
 Giá tăng dần Giá giảm dần
 Không sắp xếp

Hình 36. Giao diện tìm kiếm, lọc sản phẩm

The screenshot shows a product detail page for 'Hạt dinh dưỡng' (Nutritious Nuts) on the FoodStore website. At the top, there is a navigation bar with links for 'Trang chủ' and 'Sản phẩm', a search bar, and user icons for shopping cart (0 items) and profile.

Product Information:

- Tên sản phẩm:** Hạt dinh dưỡng
- Giá:** 120,000 vnd/ Kg
- Rating:** ★★★★☆
- Mô tả:** Nguồn dinh dưỡng giàu protein và chất béo lành mạnh, phù hợp với mọi đối tượng.
- Thêm vào giỏ hàng:** Button to add to cart.

Category Filter:

- Thể loại:**
 - Rau (10)
 - Củ (0)
 - Trái cây (5)
 - Thực phẩm giàu protein (7)
 - Thực phẩm chứa tinh bột (0)
 - Thức uống (5)

Comments:

- Để lại bình luận:** Button to leave a comment.
- Upvote/Downvote:** Green up arrow and blue down arrow buttons.

Hình 37. Giao diện xem chi tiết sản phẩm

The screenshot shows a shopping cart and order summary on the FoodStore website.

Shopping Cart:

Sản phẩm	Tên	Giá cả	Số lượng	Thành tiền	Xử lý
	Hạt dinh dưỡng	120,000 đ	3	360,000 đ	

Thông Tin Đơn Hàng (Order Information):

Tạm tính:	360,000 đ
Phí vận chuyển	0 đ
Tổng số tiền	360,000 đ

XÁC NHẬN ĐẶT HÀNG (Confirm Order):

Hình 38. Giao diện thông tin đơn hàng

The screenshot shows the FoodStore website's payment page. At the top, there are navigation links for 'Trang chủ' and 'Sản phẩm'. A search bar and user icons are also present. The main content area displays a table of products:

Sản phẩm	Tên	Giá cả	Số lượng	Thành tiền
	Hạt dinh dưỡng	120,000 đ	3	360,000 đ

Below this, there are sections for 'Thông Tin Người Nhận' (Recipient Information) and 'Thông Tin Thanh Toán' (Payment Information). The recipient info includes:

- Tên người nhận: Trịnh Quang Lâm
- Địa chỉ người nhận: Ha Noi Capital Region
- Số điện thoại: 234

The payment information section shows:

- Phí vận chuyển: 0 đ
- Hình thức: Thanh toán khi nhận hàng (COD)
- Tổng số tiền: 360,000 đ

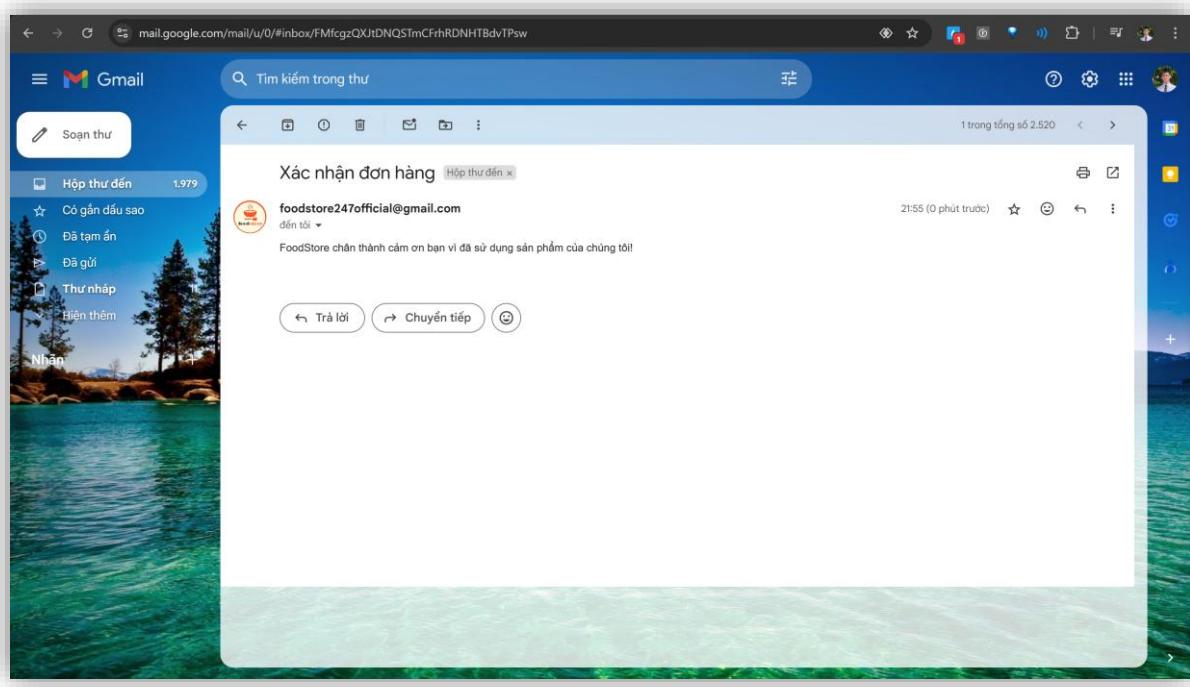
A green button labeled 'XÁC NHẬN THANH TOÁN' (Confirm Payment) is visible.

Hình 39. Giao diện thanh toán

The screenshot shows the FoodStore website's payment confirmation page. At the top, there are navigation links for 'Trang chủ' and 'Thanh toán'. A green banner at the top reads 'Cảm ơn quý khách đã sử dụng sản phẩm của chúng tôi!' (Thank you for using our products!). Below this, there is a large 'THANK YOU' logo.

The main content area features the FoodStore logo and a 'Thực phẩm tươi' (Fresh food) tag. It includes sections for 'Thông tin cửa hàng' (Store information), 'Tài khoản' (Account), and 'Thông tin liên hệ' (Contact information). Social media sharing icons for Twitter, Facebook, YouTube, and LinkedIn are located in the bottom right corner.

Hình 40. Giao diện thanh toán thành công



Hình 41. Email xác nhận mua hàng được gửi về mail người dùng

6.1. Giao diện phía người quản trị

FoodStore

Chức năng

- Bảng điều khiển
- Người dùng
- Sản phẩm
- Đơn hàng

Bảng điều khiển

Số lượng User: (2)

Số lượng đơn hàng: (2)

Số lượng sản phẩm: (27)

Hình 42. Số lượng người dùng, đơn hàng, sản phẩm hiện có

Quản lý người dùng

Bảng thông tin người dùng

ID	Email	Tên đầy đủ	Vai trò	Thao tác
1	trinhhquanglam2k4@gmail.com	Trịnh Quang Lâm	ADMIN	<button>Chi tiết</button> <button>Cập nhật</button> <button>Xóa</button>
8	nam085924@gmail.com	Nguyễn Văn Nam	USER	<button>Chi tiết</button> <button>Cập nhật</button> <button>Xóa</button>

[Quay lại](#)

« 1 »

Nhóm 1

Hình 43. Giao diện quản lý người dùng

Quản lý sản phẩm

Bảng thông tin sản phẩm

ID	Tên	Giá	Thao tác
1	Bánh mỳ	10,000	<button>Chi tiết</button> <button>Cập nhật</button> <button>Xóa</button>
2	Hạt dinh dưỡng	120,000	<button>Chi tiết</button> <button>Cập nhật</button> <button>Xóa</button>
3	Salad rau xanh	40,000	<button>Chi tiết</button> <button>Cập nhật</button> <button>Xóa</button>
4	Sữa ít đường	25,000	<button>Chi tiết</button> <button>Cập nhật</button> <button>Xóa</button>
5	Trái cây nho	35,000	<button>Chi tiết</button> <button>Cập nhật</button> <button>Xóa</button>

[Quay lại](#)

« 1 2 3 4 5 6 »

Nhóm 1

Hình 44. Giao diện quản lý sản phẩm

FoodStore

Chào mừng đến với Nhóm 1

CHỨC NĂNG

- Bảng điều khiển
- Người dùng
- Sản phẩm
- Đơn hàng

Quản lý đơn hàng

Bảng điều khiển / Đơn hàng

Bảng thông tin đơn hàng

ID	Tổng	Người dùng	Trạng thái	Thao tác
1	375,000	Trịnh Quang Lam	Đang vận chuyển	<button>Chi tiết</button> <button>Cập nhật</button> <button>Xóa</button>
7	730,000	Nguyễn Văn Nam	Đang vận chuyển	<button>Chi tiết</button> <button>Cập nhật</button> <button>Xóa</button>

Quay lại

Nhóm 1

Hình 45. Giao diện quản lý đơn đặt hàng

PHẦN 7: KẾT LUẬN

Hiện nay, trên nền tảng Internet đã xuất hiện rất nhiều trang web bán hàng thực phẩm. Tuy nhiên, phần lớn các trang này vẫn chưa đáp ứng đầy đủ nhu cầu của người sử dụng. Đặc biệt, đối với những người có bệnh lý hoặc cần tuân theo một chế độ ăn đặc biệt, việc chọn đúng loại thực phẩm cho bữa ăn trở nên vô cùng quan trọng.

Ứng dụng lần này đã khắc phục được những hạn chế còn tồn tại của trang web hiện nay bằng cách hướng đến một nhóm đối tượng khách hàng cụ thể. Thay vì phải chuyển đổi qua nhiều danh mục để tìm sản phẩm phù hợp, người dùng có chế độ ăn phức tạp nay chỉ cần chọn danh mục sản phẩm được thiết kế riêng cho nhu cầu của mình. Điều này không chỉ giúp rút ngắn thời gian lựa chọn mà còn mang lại trải nghiệm tiện lợi và tối ưu hơn cho khách hàng.

Báo cáo đã trình bày quá trình phát triển backend cho một ứng dụng web hoàn chỉnh sử dụng các công nghệ trong hệ sinh thái Spring, bao gồm Spring Boot, Spring Data JPA, Spring Security, và Spring Session, đồng thời tuân theo mẫu thiết kế kiến trúc Model – View – Controller (MVC) với Spring MVC. Báo cáo tập trung vào các luồng hoạt động chính của hệ thống, ý tưởng thiết kế và các thành phần quan trọng, nhằm tránh sự dài dòng và chi tiết thừa. Phần frontend được lấy từ template có sẵn thay vì phát triển từ đầu, vì vậy không được trình bày chi tiết trong báo cáo.

Bên cạnh những điểm đã đạt được, chẳng hạn như xây dựng một website hoàn chỉnh có các chức năng cơ bản phục vụ bán hàng, hỗ trợ phân quyền người dùng và tích hợp các công cụ mạnh mẽ từ Spring, hệ thống vẫn còn một số hạn chế:

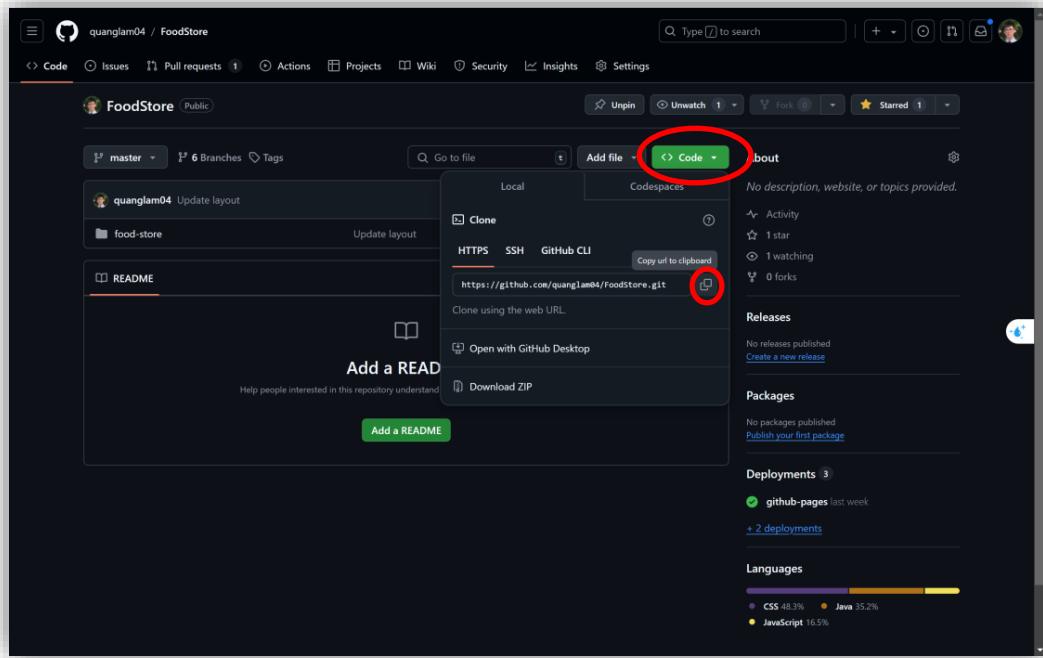
- Chưa đáp ứng được các yêu cầu phức tạp hơn
- Chưa giải quyết tình trạng reload trang mỗi khi xử lý request
- Chưa tối ưu cơ sở dữ liệu để xử lý khôi lượng lớn dữ liệu

PHỤ LỤC 1: HƯỚNG DẪN CÀI ĐẶT VÀ CHẠY ỨNG DỤNG

Cách 1: Sử dụng Docker

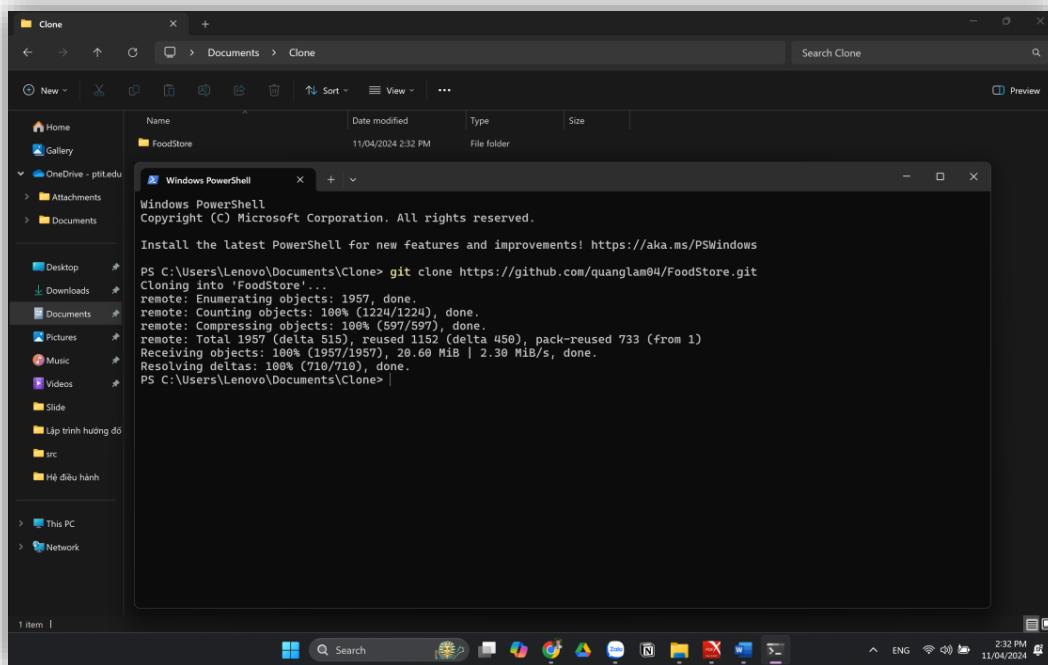
B1: Sử dụng Git để lấy sourcode về máy trên Github Server

- Bấm vào đường link sản phẩm, tại đây chọn biểu tượng button Code trên màn hình và copy đường dẫn tại cửa sổ HTTPs



Hình 46: Giao diện tại trang chủ Github

- Tại máy tính cá nhân, chọn folder sẽ lưu trữ. Sử dụng terminal và kèm theo câu lệnh `git clone <url>`



Hình 47: Giao diện sau khi clone thành công dự án về máy

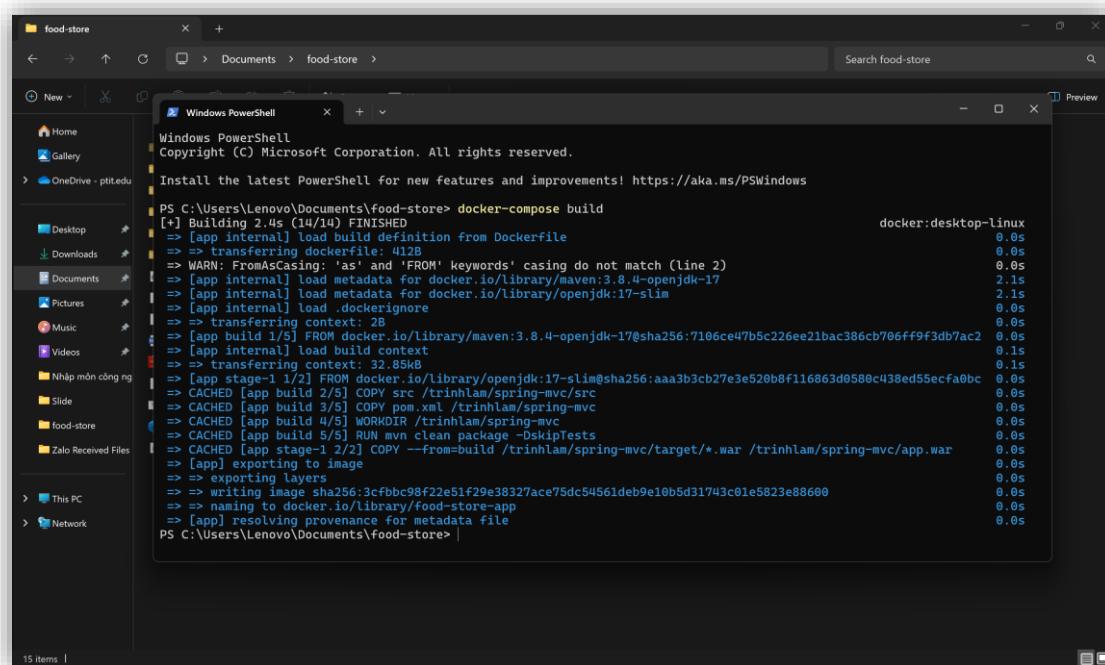
B2: Cài đặt Docker Desktop để chạy ứng dụng tại đây

B3: Tiến hành build Image và chạy

Trước tiên cần khởi động ứng dụng docker desktop. Sau đó vào thư mục dự án, sử dụng terminal và chạy lần lượt các câu lệnh sau:

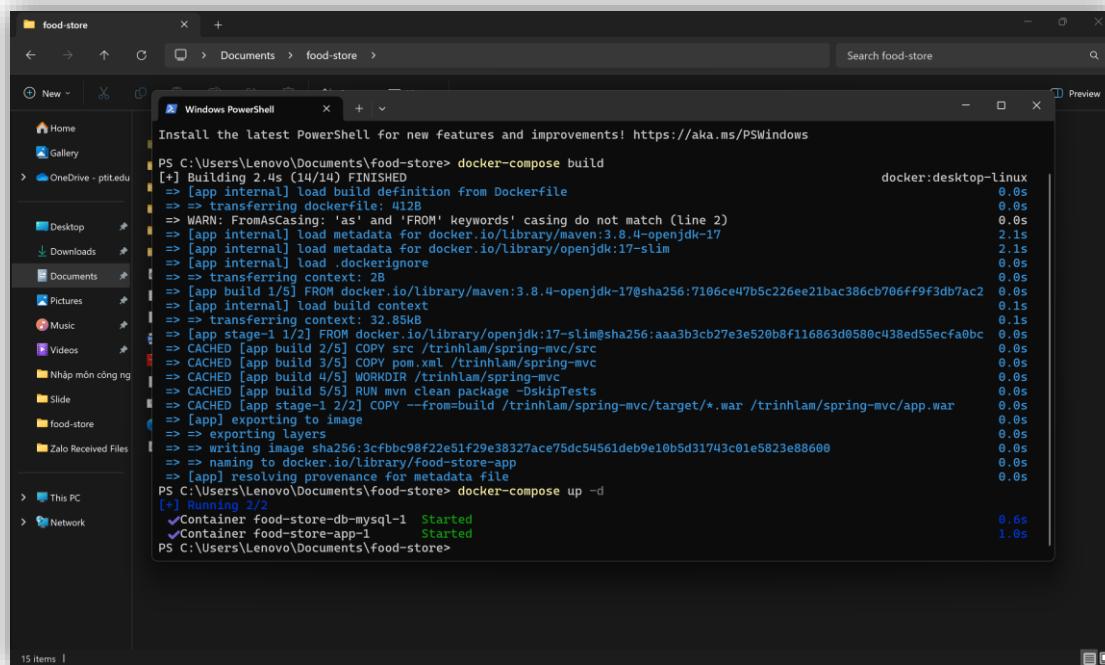
Nếu là lần đầu tiên chạy:

```
docker-compose build  
docker-compose up -d
```



```
PS C:\Users\Lenovo\Documents\food-store> docker-compose build  
[+] Building 2.4s (14/14) FINISHED  
=> [app internal] load build definition from Dockerfile  
=> => transferring dockerfile: 412B  
=> WARN: FromAsCasing: 'as' and 'FROM' keywords' casing do not match (line 2)  
=> [app internal] load metadata for docker.io/library/maven:3.8.4-openjdk-17  
=> [app internal] load metadata for docker.io/library/openjdk:17-slim  
=> [app internal] load .dockerrignore  
=> => transferring context: 2B  
=> [app built 1/5] FROM docker.io/library/maven:3.8.4-openjdk-17@sha256:7106ce47b5c226ee21bac386cb706ff9f3db7ac2  
=> [app internal] load build context  
=> => transferring context: 32.B5K8  
=> [app stage-1 1/2] FROM docker.io/library/openjdk:17-slim@sha256:aaa3b3cb27e3e520b8f116863d0580c438ed55ecfa0bc  
=> CACHED [app build 2/5] COPY src /trinhlam/spring-mvc/src  
=> CACHED [app build 3/5] COPY pom.xml /trinhlam/spring-mvc  
=> CACHED [app build 4/5] WORKDIR /trinhlam/spring-mvc  
=> CACHED [app build 5/5] RUN mvn clean package -DskipTests  
=> CACHED [app stage-1 2/2] COPY --from=build /trinhlam/spring-mvc/target/*.war /trinhlam/spring-mvc/app.war  
=> [app] exporting to image  
=> => exporting layers  
=> => writing image sha256:3cfbbc98f22e51f29e38327ace75dc54561deb9e10b5d31743c01e5823e88600  
=> => => naming to docker.io/library/food-store-app  
=> [app] resolving provenance for metadata file  
PS C:\Users\Lenovo\Documents\food-store>
```

Hình 48: Build Image → Container



```
PS C:\Users\Lenovo\Documents\food-store> docker-compose build  
[+] Building 2.4s (14/14) FINISHED  
=> [app internal] load build definition from Dockerfile  
=> => transferring dockerfile: 412B  
=> WARN: FromAsCasing: 'as' and 'FROM' keywords' casing do not match (line 2)  
=> [app internal] load metadata for docker.io/library/maven:3.8.4-openjdk-17  
=> [app internal] load metadata for docker.io/library/openjdk:17-slim  
=> [app internal] load .dockerrignore  
=> => transferring context: 2B  
=> [app built 1/5] FROM docker.io/library/maven:3.8.4-openjdk-17@sha256:7106ce47b5c226ee21bac386cb706ff9f3db7ac2  
=> [app internal] load build context  
=> => transferring context: 32.B5K8  
=> [app stage-1 1/2] FROM docker.io/library/openjdk:17-slim@sha256:aaa3b3cb27e3e520b8f116863d0580c438ed55ecfa0bc  
=> CACHED [app build 2/5] COPY src /trinhlam/spring-mvc/src  
=> CACHED [app build 3/5] COPY pom.xml /trinhlam/spring-mvc  
=> CACHED [app build 4/5] WORKDIR /trinhlam/spring-mvc  
=> CACHED [app build 5/5] RUN mvn clean package -DskipTests  
=> CACHED [app stage-1 2/2] COPY --from=build /trinhlam/spring-mvc/target/*.war /trinhlam/spring-mvc/app.war  
=> [app] exporting to image  
=> => exporting layers  
=> => writing image sha256:3cfbbc98f22e51f29e38327ace75dc54561deb9e10b5d31743c01e5823e88600  
=> => => naming to docker.io/library/food-store-app  
=> [app] resolving provenance for metadata file  
PS C:\Users\Lenovo\Documents\food-store> docker-compose up -d  
[+] Running 2/2  
  ✓ Container food-store-db-mysql-1 Started  
  ✓ Container food-store-app-1 Started  
PS C:\Users\Lenovo\Documents\food-store>
```

Hình 49: Giao diện chạy container

Nếu đã từng chạy trước đó, chỉ cần khởi động lại container: docker-compose up -d

B4: Truy cập localhost 8080

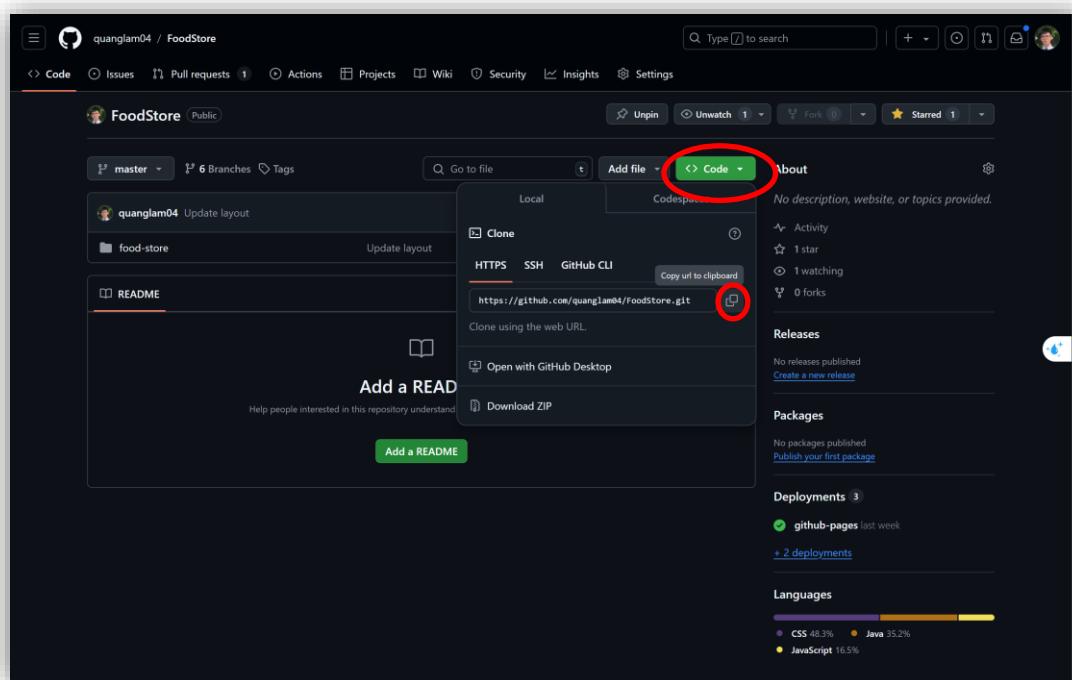


Hình 50: Giao diện kết quả sau khi chạy với docker

Cách 2: Chạy thử công bằng cách cấu hình các tham số

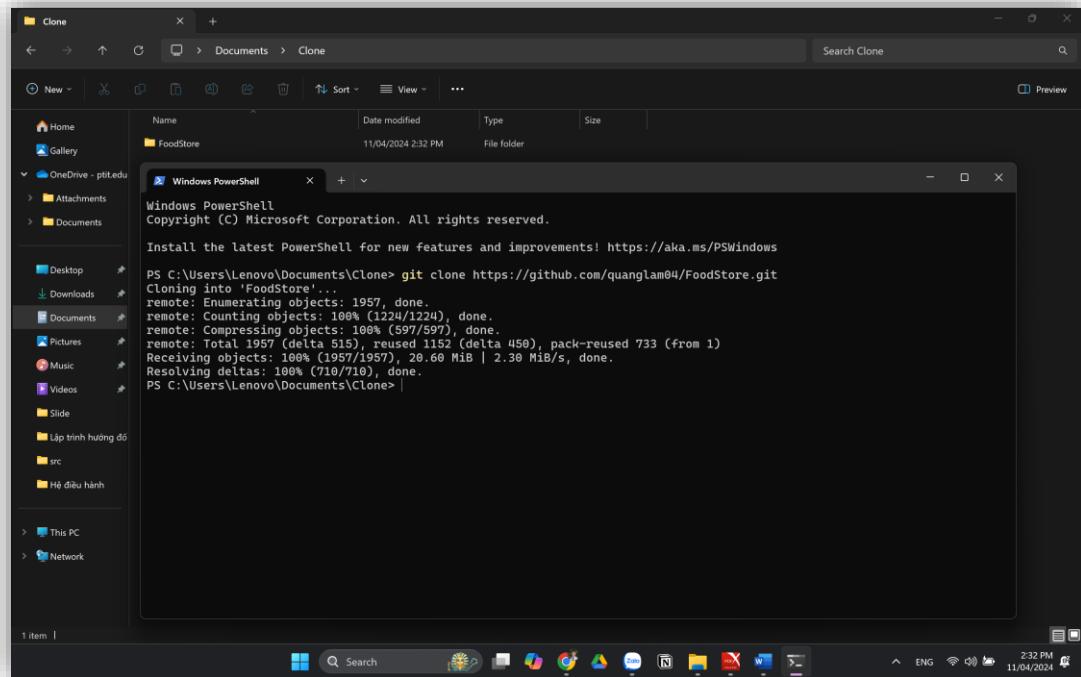
B1: Sử dụng Git để lấy sourcode về máy trên Github Server

- Bấm vào đường link sản phẩm, tại đây chọn biểu tượng button Code trên màn hình và copy đường dẫn tại cửa sổ HTTPs



Hình 51: Giao diện tại trang chủ Github

- Tại máy tính cá nhân, chọn folder sẽ lưu trữ. Sử dụng terminal và kèm theo câu lệnh
`git clone <url>`



Hình 52: Giao diện sau khi clone thành công dự án về máy

B2: Cài đặt môi trường

- [Java version 17](#)
- [Visual Studio Code](#)
- [MySQL Server version 8.0 trở lên](#)

B3: Cài đặt các Extension cần thiết

- [Extension pack for java](#) : hỗ trợ code/debug java
- [Spring boot Extension pack](#) : hỗ trợ chạy dự án Java Spring
- [Trailing Spaces](#) : Kiểm tra khoảng trắng có tồn tại trong file không

B4: Thiết lập cơ sở dữ liệu

- Tạo cơ sở dữ liệu có tên “[foodstore](#)” trong MySQL dùng MySQL Workbench

[CREATE DATABASE foodstore](#)

- Mở file [application.properties](#) trong thư mục [src/main/resources](#) và thay đổi thiết lập tương ứng với thông tin về MySQL trên máy gồm tên đăng nhập và mật khẩu kết nối

tới MySQL Server

```
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://${MYSQL_HOST}:3306/foodstore
spring.datasource.username=< username >
spring.datasource.password=< password >
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

B5: Thiết lập cấu hình để gửi Email

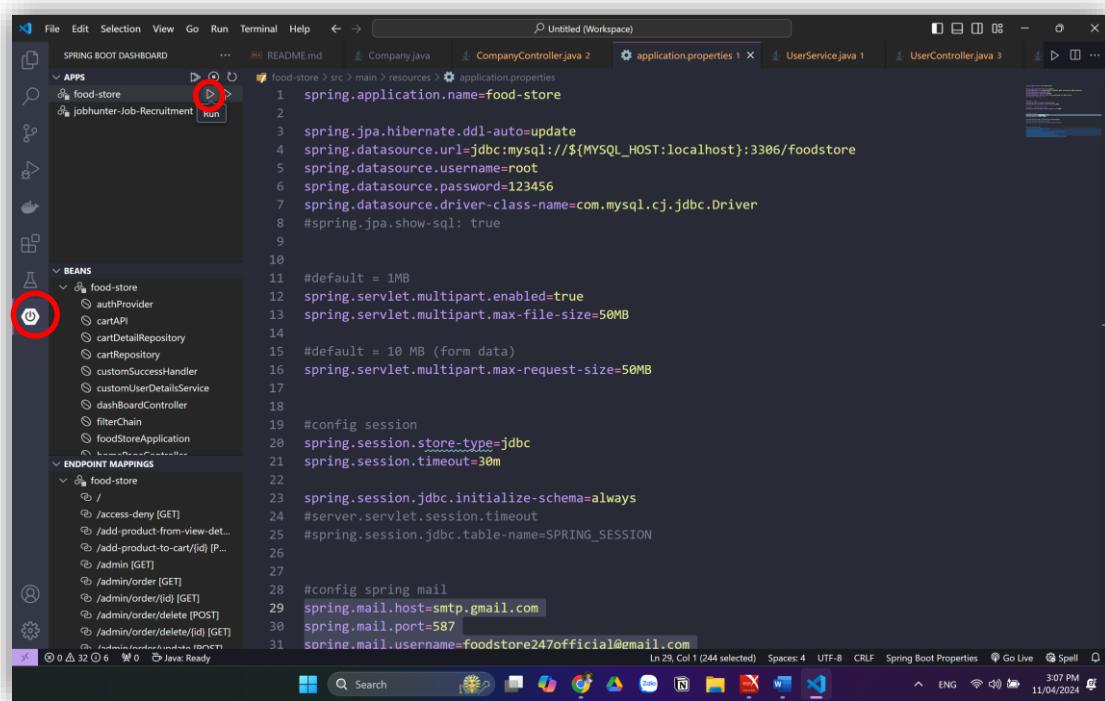
- Để có thể gửi mail cho người dùng sau khi đặt hàng sử dụng **JavaMailSender**, ta cần khai báo các thông tin để JavaMail có thể xác thực với mail Server. Đối với Gmail, thiết lập trong **application.properties** như sau:

```
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=< Địa chỉ Email >
spring.mail.password=< Mật khẩu >
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

***Chú ý:** mật khẩu mail sử dụng trong bài sẽ là một App Password do Google tạo ra. Để lấy mật khẩu này, tham khảo hướng dẫn tại địa chỉ:
<https://support.google.com/accounts/answer/185833?hl=en>

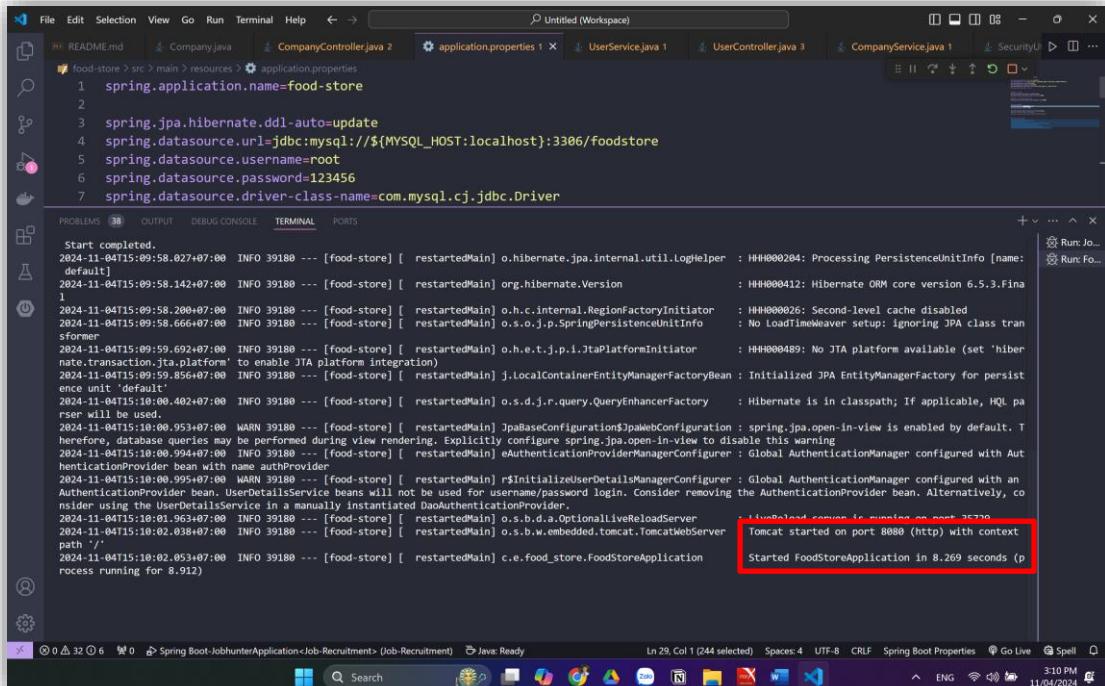
B6: Chạy ứng dụng

- Tại cửa sổ VS Code, chọn vào biểu tượng SpringBoot ở thanh bên trái, tại cửa sổ Apps sẽ hiện ra các Project sẵn có. Án vào biểu tượng run để chạy ứng dụng



Hình 53: Giao diện chạy ứng dụng với SpringBoot Dashboard

- Giao diện khi chạy ứng dụng thành công



```
spring.application.name=food-store
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://${MYSQL_HOST}:3306/foodstore
spring.datasource.username=root
spring.datasource.password=123456
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

Start completed.
2024-11-04T15:09:58.027+07:00 INFO 39180 --- [food-store] o.h.license.LicenseHelper : HHH000204: Processing PersistenceUnitInfo [name: default]
2024-11-04T15:09:58.142+07:00 INFO 39180 --- [food-store] org.hibernate.Version : HHH000412: Hibernate ORM core version 6.5.3.Final
2024-11-04T15:09:58.200+07:00 INFO 39180 --- [food-store] o.h.c.internal.RegionFactoryInitiator : HHH00026: Second-level cache disabled
2024-11-04T15:09:58.200+07:00 INFO 39180 --- [food-store] o.s.o.o.p.SpringPersistenceUnitInfo : No LoadTimeWeaver setup; ignoring JPA class transformer
2024-11-04T15:09:59.692+07:00 INFO 39180 --- [food-store] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000489: No JTA platform available (set 'hibernate.transaction.jta.platform' to enable JTA platform integration)
2024-11-04T15:09:59.856+07:00 INFO 39180 --- [food-store] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'.
2024-11-04T15:10:00.402+07:00 INFO 39180 --- [food-store] o.s.d.j.r.query.QueryEnhancerFactory : Hibernate is in classpath; If applicable, HQL parser will be used.
2024-11-04T15:10:00.953+07:00 WARN 39180 --- [food-store] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
2024-11-04T15:10:00.994+07:00 INFO 39180 --- [food-store] e.AuthenticationProviderManagerConfigurer : Global AuthenticationManager configured with AuthenticationProvider bean with name authProvider
2024-11-04T15:10:00.995+07:00 WARN 39180 --- [food-store] r$InitializeUserDetailsManagerConfigurer : Global AuthenticationManager configured with an AuthenticationProvider bean. UserDetailsService beans will not be used for username/password login. Consider removing the AuthenticationProvider bean. Alternatively, consider using the UserDetailsService or a manually instantiated DaoAuthenticationProvider
2024-11-04T15:10:01.905+07:00 INFO 39180 --- [food-store] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
2024-11-04T15:10:02.038+07:00 INFO 39180 --- [food-store] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path '/'
2024-11-04T15:10:02.053+07:00 INFO 39180 --- [food-store] c.e.food_store.FoodStoreApplication : Started FoodStoreApplication in 8.269 seconds (process running for 8.912)
```

Hình 54: Giao diện khi chạy thành công

- Tại cửa sổ trình duyệt Web, truy cập vào địa chỉ chỉ <http://localhost:8080/>



Hình 55: Truy cập ứng dụng

TÀI LIỆU THAM KHẢO

[1] <https://www.baeldung.com>

[2] <https://spring.io>

[3] Template: <https://themewagon.github.io/fruitables/>

[4] <https://backendstory.com/spring-security-authentication-architecture-explained-in-depth/>

[5] <https://howtodoinjava.com/spring-boot/spring-boot-jsp-view-example/>

[6] https://docs.jboss.org/hibernate/orm/6.4/introduction/html_single/Hibernate_Introduction.html#associations

[7] <https://techmaster.vn/posts/37151/phan-trang-va-sap-xep-du-lieu-voi-spring-data-jpa>

[8] https://www.w3schools.com/howto/howto_js_autocomplete.asp