

**BỘ THÔNG TIN VÀ TRUYỀN THÔNG**  
**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**



**BÁO CÁO BÀI TẬP LỚN**  
**MÔN : LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG**  
**Giảng viên: Nguyễn Mạnh Sơn**

**Nhóm 1**

B22DCCN482 Trịnh Quang Lâm

B22DCCN381 Lê Đức Huy

B22DCCN866 Vương Đức Trọng

B22DCCN038 Nguyễn Viết Tuấn Anh

B22DCCN434 Vũ Nhân Kiên

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

# Mục lục

<b>PHẦN 1: ĐẶT VẤN ĐỀ.....</b>	<b>1</b>
<b>PHẦN 2: KHẢO SÁT .....</b>	<b>2</b>
<b>PHẦN 3: TÌM HIỂU VỀ MÔ HÌNH MVC .....</b>	<b>3</b>
<b>PHẦN 4: CÁC MODULE CHÍNH .....</b>	<b>5</b>
4.1. Controller .....	5
4.2. Service .....	5
4.3. Repository .....	6
4.4. Domain.....	6
4.5. Các module khác.....	7
<b>PHẦN 5: TRIỂN KHAI .....</b>	<b>8</b>
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.....	14
5.2.e. Thêm sản phẩm vào giỏ hàng, tạo đơn hàng, và xem lịch sử mua hàng.....	15
5.2.f. Quản lý người dùng, sản phẩm, đơn đặt hàng.....	16
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
5.3. Công cụ hỗ trợ .....	26
<b>PHẦN 6: KẾT QUẢ .....</b>	<b>27</b>
6.1. Giao diện phía người dùng.....	27
6.1. Giao diện phía người quản trị .....	33
<b>PHẦN 7: KẾT LUẬN.....</b>	<b>36</b>
<b>PHỤ LỤC 1: HƯỚNG DẪN CÀI ĐẶT VÀ CHẠY ỨNG DỤNG .....</b>	<b>37</b>
<b>TÀI LIỆU THAM KHẢO .....</b>	<b>41</b>

## PHẦN 1: ĐẶT VẤN 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**
- **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 thành 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

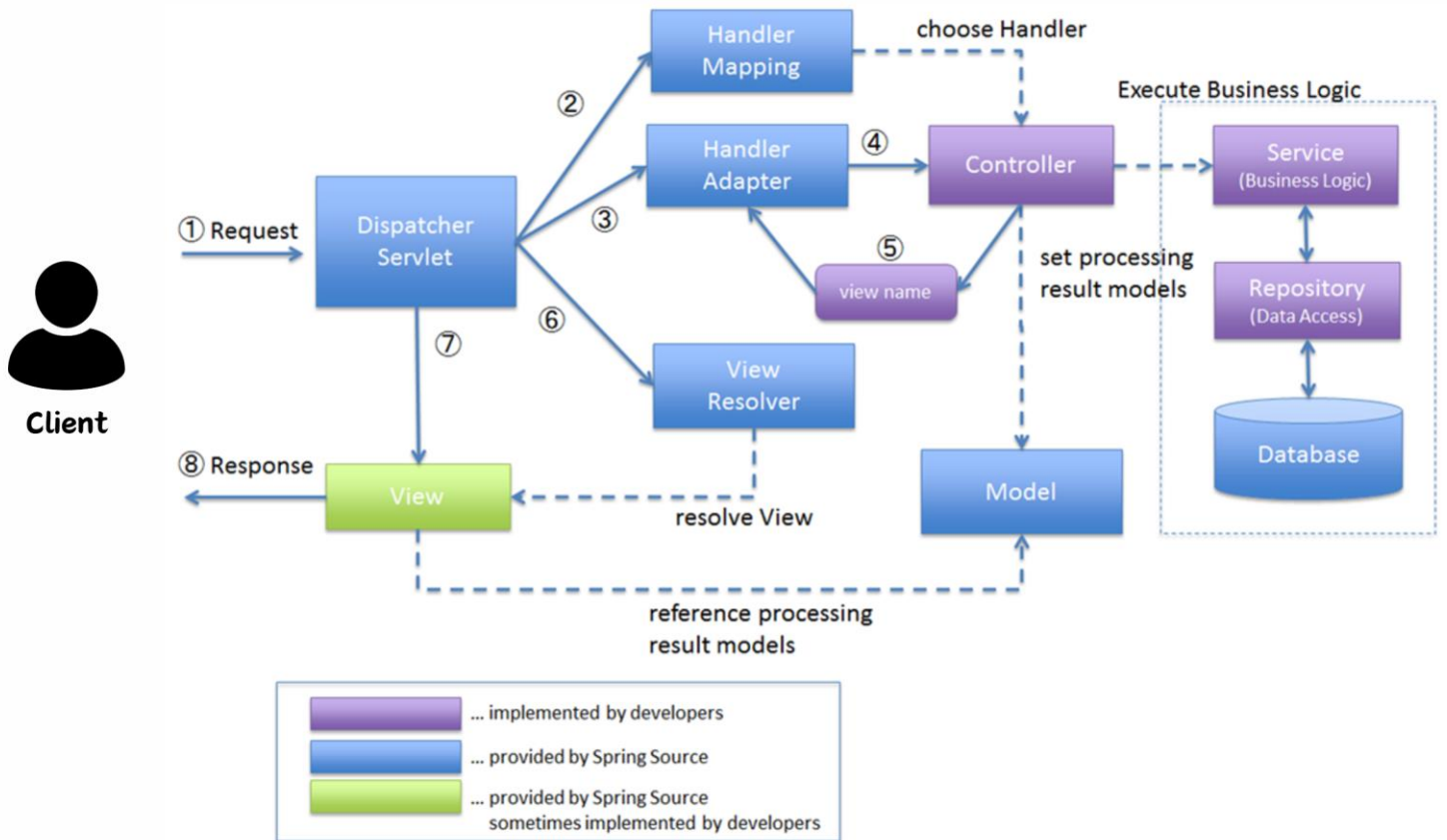
Với ý tưởng và những yêu cầu cụ thể đã nêu ra như trong Phần 1, nhóm đã tìm hiểu, thảo luận đặt ra những vấn đề và cách giải quyết ban đầu cho bài toán *Xây dựng Website bán hàng online*. Cụ thể trong bài này, cần phải sử dụng một hệ quản trị cơ sở dữ liệu để lưu trữ các thông tin. Dựa trên các yếu tố 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



#### 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:

1. **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.
2. **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.
3. **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.

4. **Controller xử lý logic:** Controller nhận request và thực thi các thao tác xử lý như:
  - 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](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

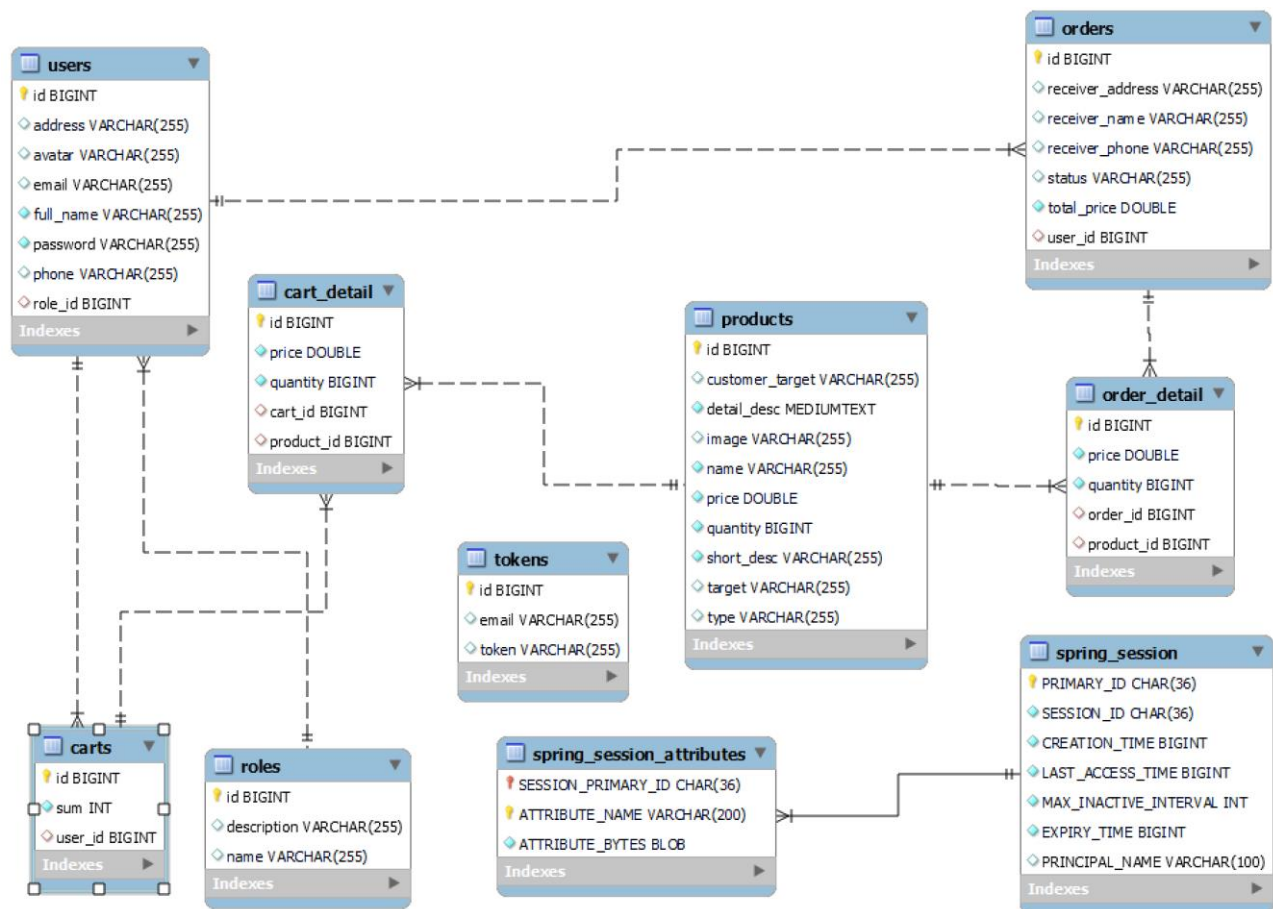
### 5.2. Phương pháp thực hiện

#### 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



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:

*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>

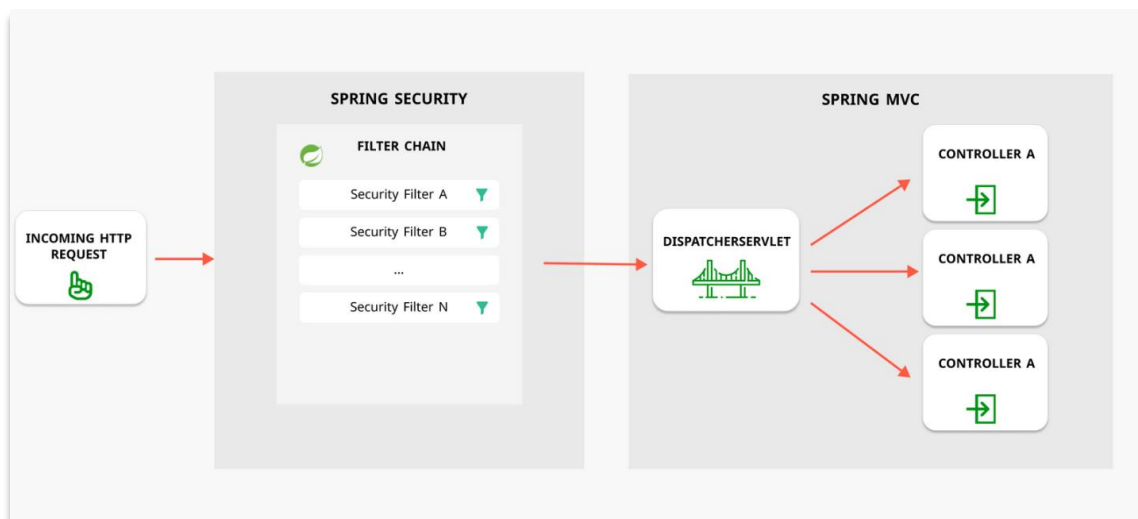
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-context-support</artifactId>
<version>6.1.5</version>
</dependency>

</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.



Hình1. Hình ảnh minh họa quá trình xử lý một yêu cầu HTTP

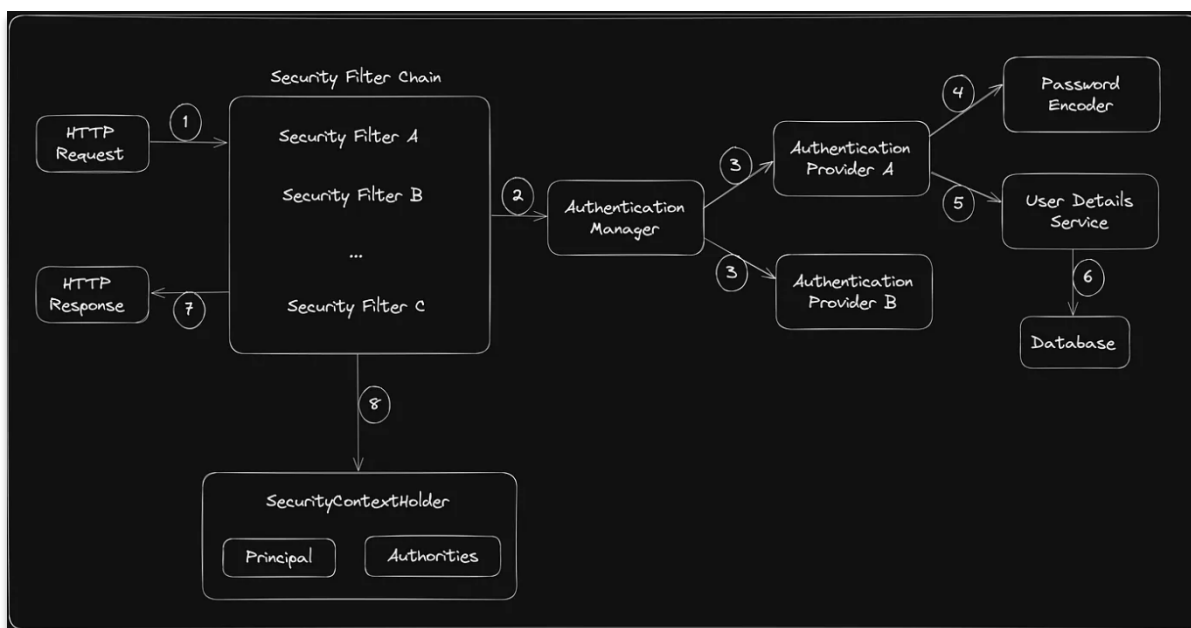
Spring Security là một framework mạnh mẽ trong hệ sinh thái Spring, chuyên về các giải pháp bảo mật cho các ứng dụng Java. Nó cung cấp các chức năng quan trọng như **xác thực (authentication)** và **phân quyền (authorization)** nhằm đảm bảo rằng chỉ những người dùng có quyền truy cập mới có thể sử dụng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

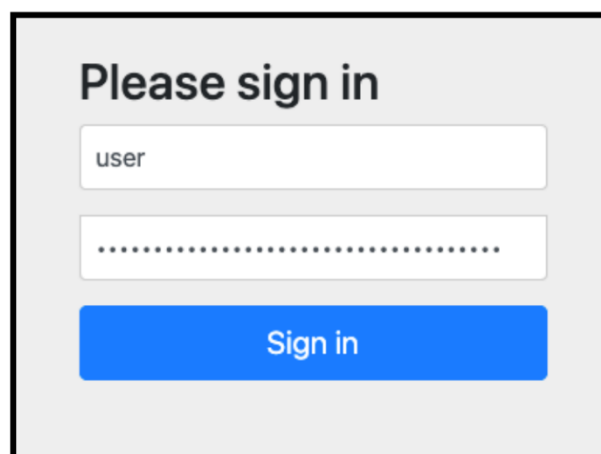
Tên người dùng: **user**

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.s.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-a7790b2df271
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@3a778e22, org.springframework.security.web.csrf.CsrfFilter@6e9c5ecd, org.springframework.security.web
.authentication.logout.LogoutFilter@78cb0ac8, org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter@125a8c70, org.springframework
.security.web.authentication.ui.DefaultLoginPageGeneratingFilter@12b43cf0, org.springframework.security.web.authentication.ui
.DefaultLogoutPageGeneratingFilter@61eb420, org.springframework.security.web.authentication.www.BasicAuthenticationFilter@2c01f00b, org.springframework.security.web
.savedrequest.RequestCacheAwareFilter@6d4595a1, org.springframework.security.web.servletapi.SecurityContextHolderAwareRequestFilter@38bba35, org.springframework
```

*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())
}
```

*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);
    }
}
```

*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 tringlam,
    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(tringlam.getEmail())) {

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

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

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

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

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

*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";
}

```

*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);
}

```

*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));
}

```

*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 ràng 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 mã 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 + "%");
}

```

*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 xử 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);
}

```

*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](mailto: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";
}
```

*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 người 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);
    }
}

```

*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";
}

```

*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";
}

```

*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";
}
```

*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";
}

```

*Xác nhận token và hiển thị thay đổi mật khẩu*

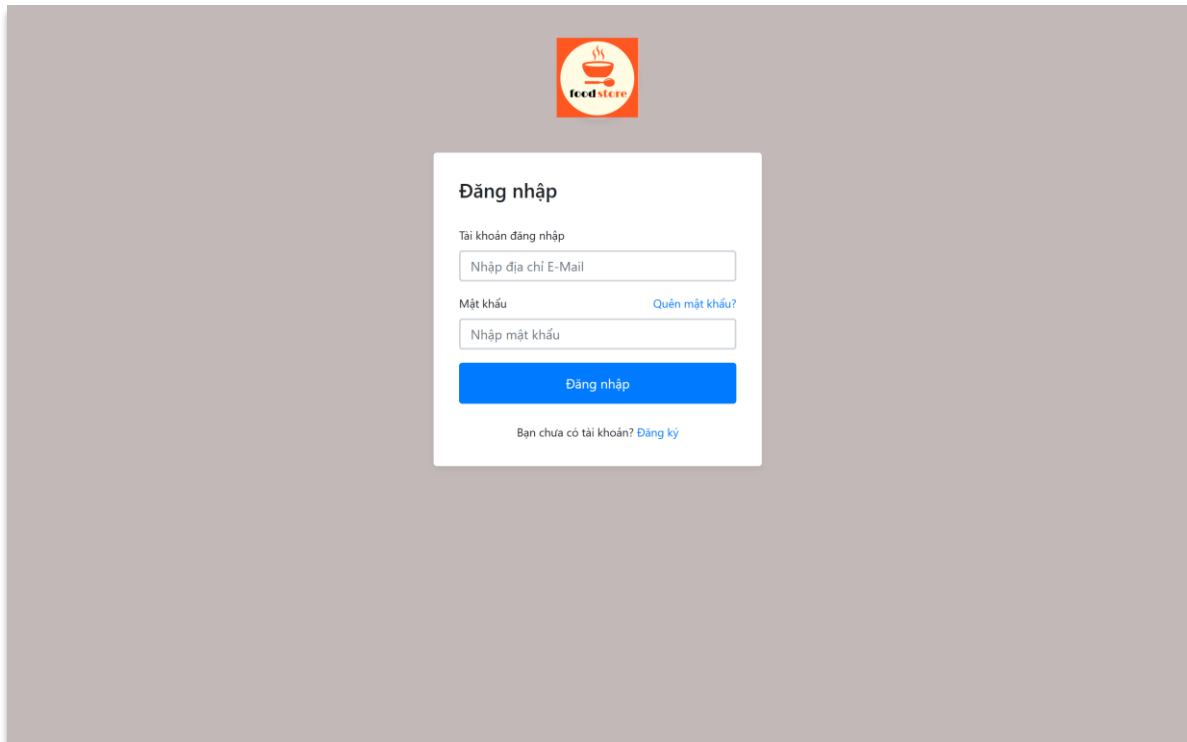
Trong các hệ thống lớn với số lượng yêu cầu cao, cách tiếp cận này có thể gặp vấn đề về token trùng lặp, gây sai logic. Để khắc phục nhược điểm này, ta có thể bổ sung biện pháp quản lý token với thời gian hết hạn hoặc sử dụng các phương pháp khác để tạo mã định danh duy nhất

### 5.3. 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 hoá quan hệ giữa các thực thể khi xây dựng cơ sở dữ liệu. Chi tiết xem tại [đây](#)

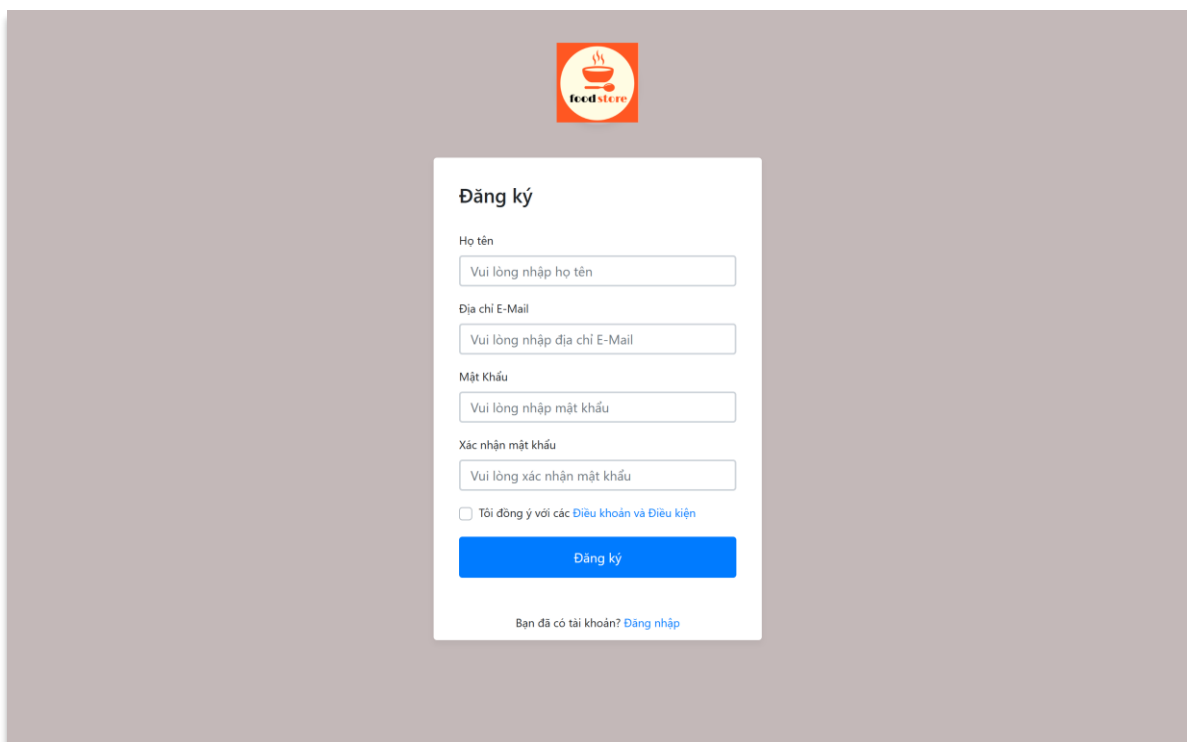
## PHẦN 6: KẾT QUẢ

### 6.1. Giao diện phía người dùng



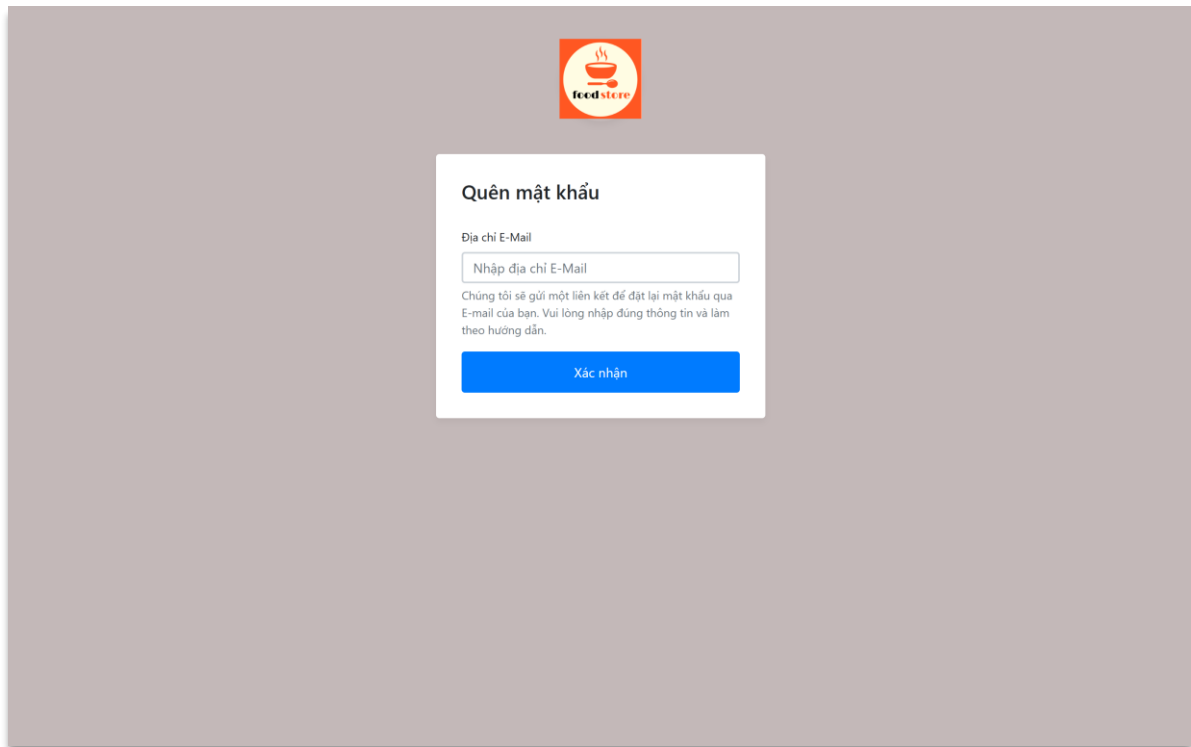
The screenshot shows a login interface for 'feedstore'. At the top center is the 'feedstore' logo, which consists of a red circle containing a white bowl with steam rising from it, and the word 'feedstore' in red below it. Below the logo is a white rectangular box with the title 'Đăng nhập' (Login) in bold. Inside this box, there is a section titled 'Tài khoản đăng nhập' (Login account) with a text input field labeled 'Nhập địa chỉ E-Mail'. Below this is a section titled 'Mật khẩu' (Password) with a text input field labeled 'Nhập mật khẩu'. To the right of the password field is a blue link that says 'Quên mật khẩu?' (Forgot password?). Below the password field is a blue button labeled 'Đăng nhập' (Login). At the bottom of the box is a link that says 'Bạn chưa có tài khoản? Đăng ký' (Don't have an account? Register).

Hình 1. Đăng nhập

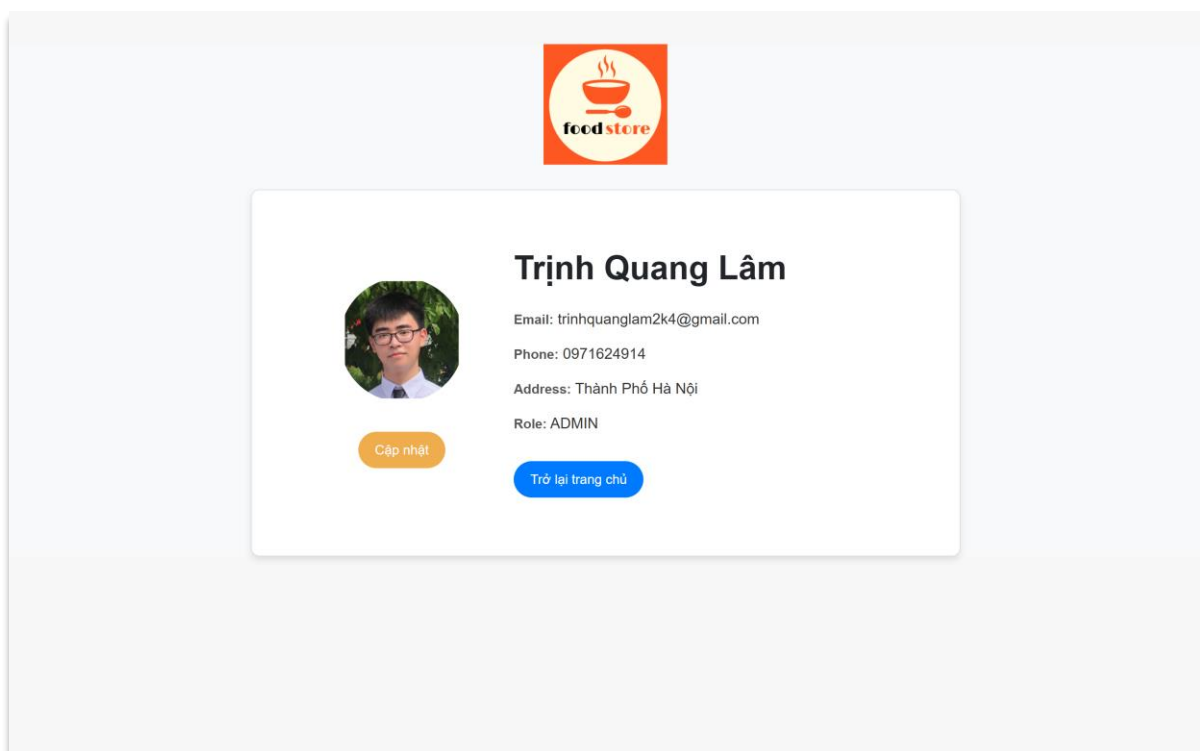


The screenshot shows a registration interface for 'feedstore'. At the top center is the 'feedstore' logo, which consists of a red circle containing a white bowl with steam rising from it, and the word 'feedstore' in red below it. Below the logo is a white rectangular box with the title 'Đăng ký' (Register) in bold. Inside this box, there are four text input fields: the first is labeled 'Họ tên' (Last name) with placeholder text 'Vui lòng nhập họ tên'; the second is labeled 'Địa chỉ E-Mail' with placeholder text 'Vui lòng nhập địa chỉ E-Mail'; the third is labeled 'Mật Khẩu' (Password) with placeholder text 'Vui lòng nhập mật khẩu'; and the fourth is labeled 'Xác nhận mật khẩu' (Confirm password) with placeholder text 'Vui lòng xác nhận mật khẩu'. Below these fields is a checkbox with the text 'Tôi đồng ý với các Điều khoản và Điều kiện' (I agree with the Terms and Conditions). Below the checkbox is a blue button labeled 'Đăng ký' (Register). At the bottom of the box is a link that says 'Bạn đã có tài khoản? Đăng nhập' (Do you have an account? Login).







Hình 2. Đăng ký



Hình 3. Quên mật khẩu




Hình 4. Quản lý tài khoản

FoodStore					
Trang chủ Sản phẩm					
Home / Lịch sử mua hàng					
Lịch sử mua hàng					
Sản phẩm	Tên	Giá cả	Số lượng	Thành tiền	Trạng thái
Số thứ tự : 01		95,000 đ			PENDING
	Sữa tươi giàu canxi	35,000 đ	1	35,000 đ	
	Rau cải xoong	20,000 đ	3	60,000 đ	
Số thứ tự : 02		485,000 đ			SHIPPING
	Bánh mì	15,000 đ	8	120,000 đ	
	Salad rau xanh	50,000 đ	7	350,000 đ	
	Sữa chua	15,000 đ	1	15,000 đ	
Số thứ tự : 03		150,000 đ			PENDING
	Hạt dinh dưỡng	150,000 đ	1	150,000 đ	

Hình 5. Lịch sử mua hàng

Đổi mật khẩu



Mật khẩu hiện tại

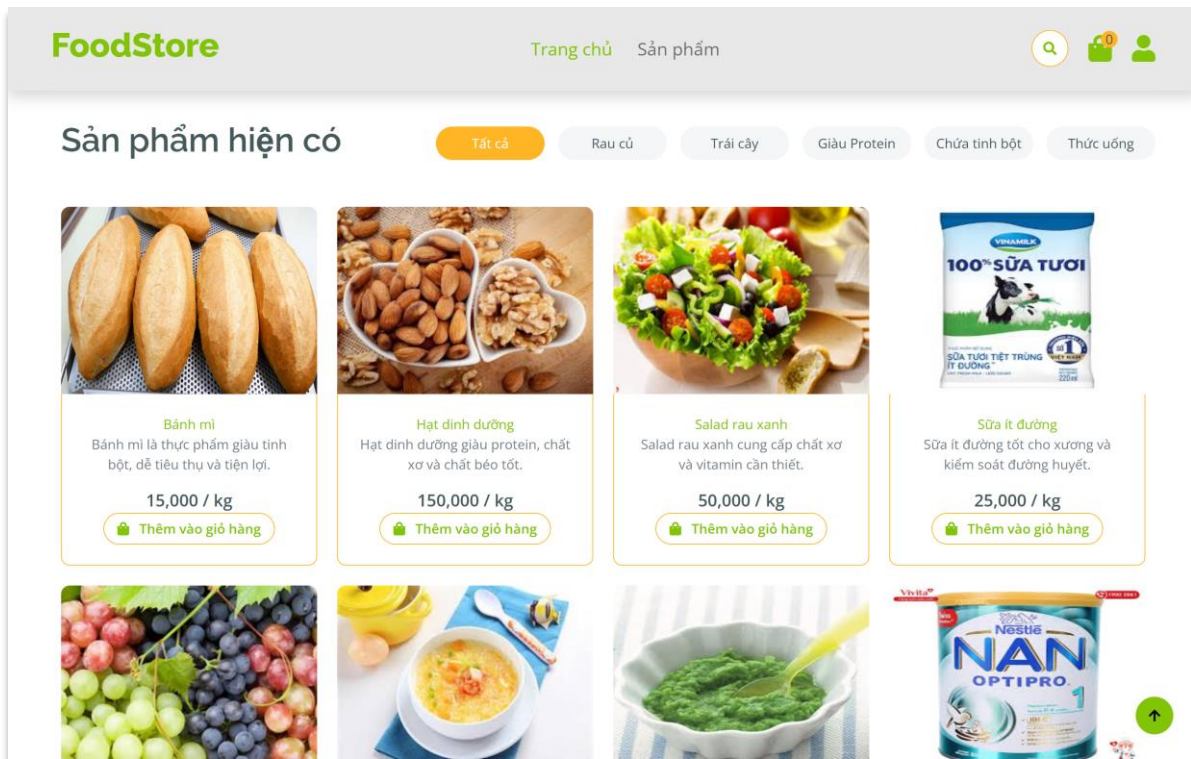
Vui lòng nhập mật khẩu hiện tại

Mật khẩu mới

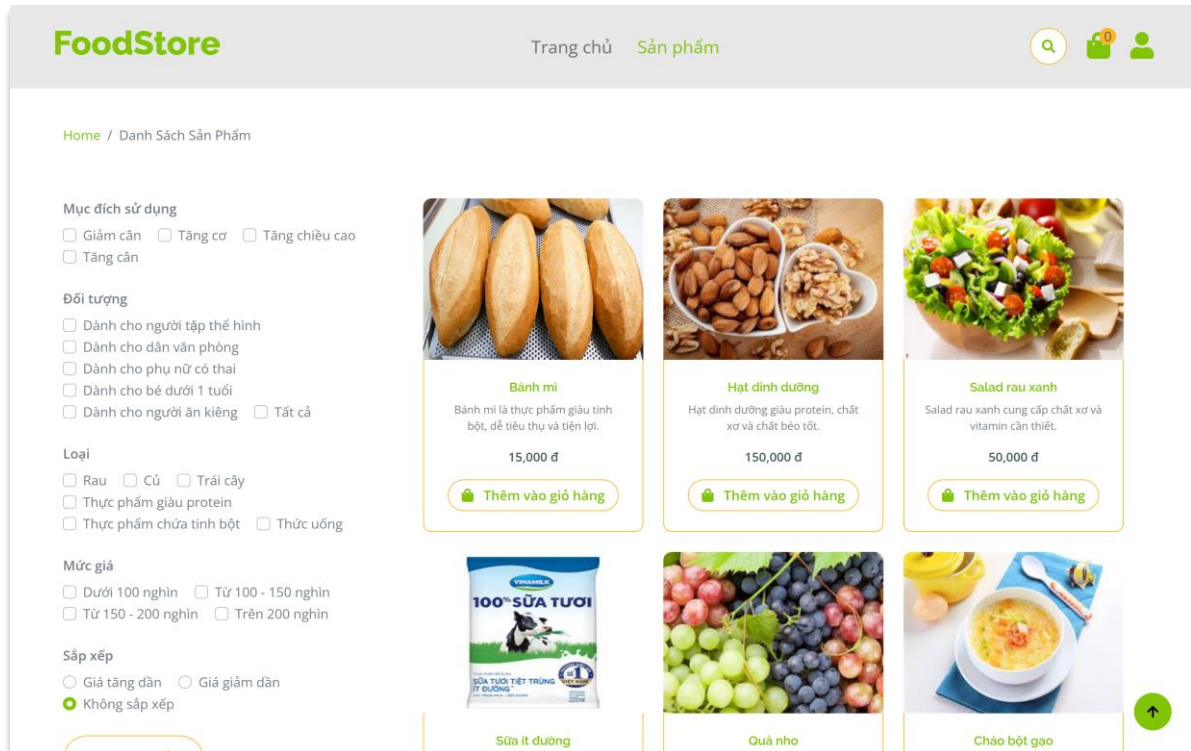
Vui lòng nhập mật khẩu mới

Đổi mật khẩu

Hình 6. Đổi mật khẩu

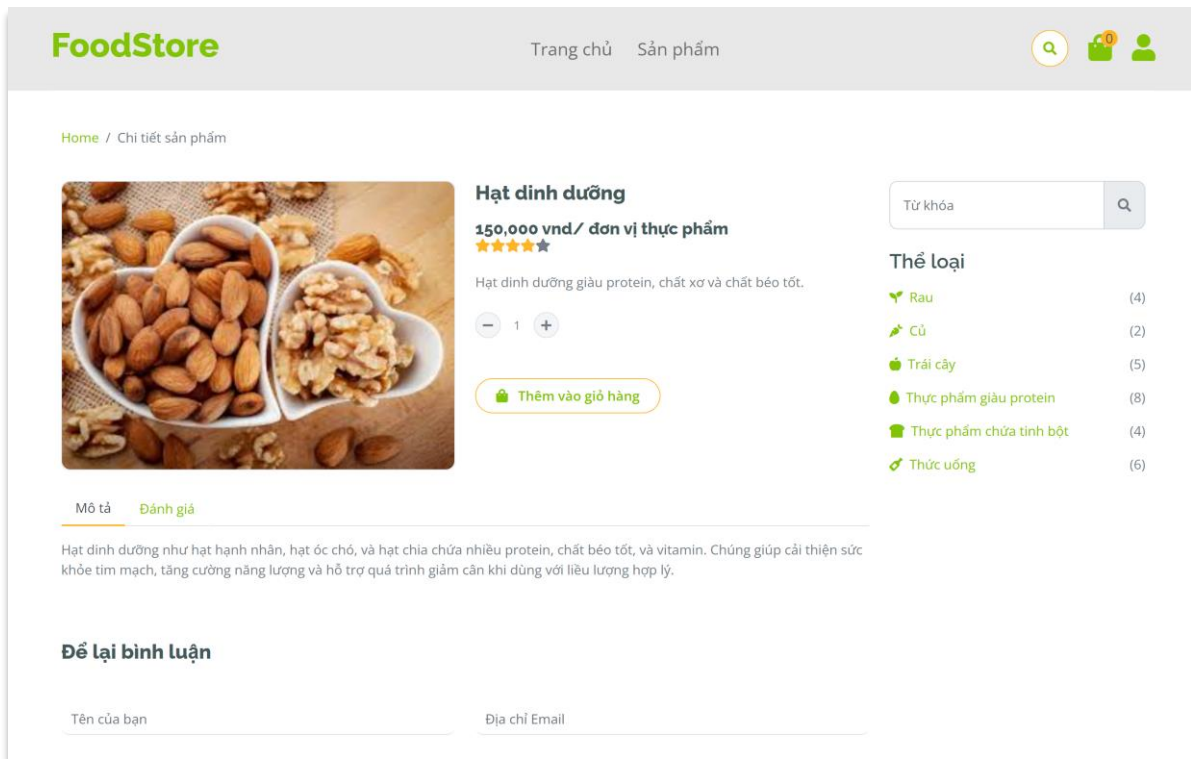


Hình 7. Danh sách sản phẩm

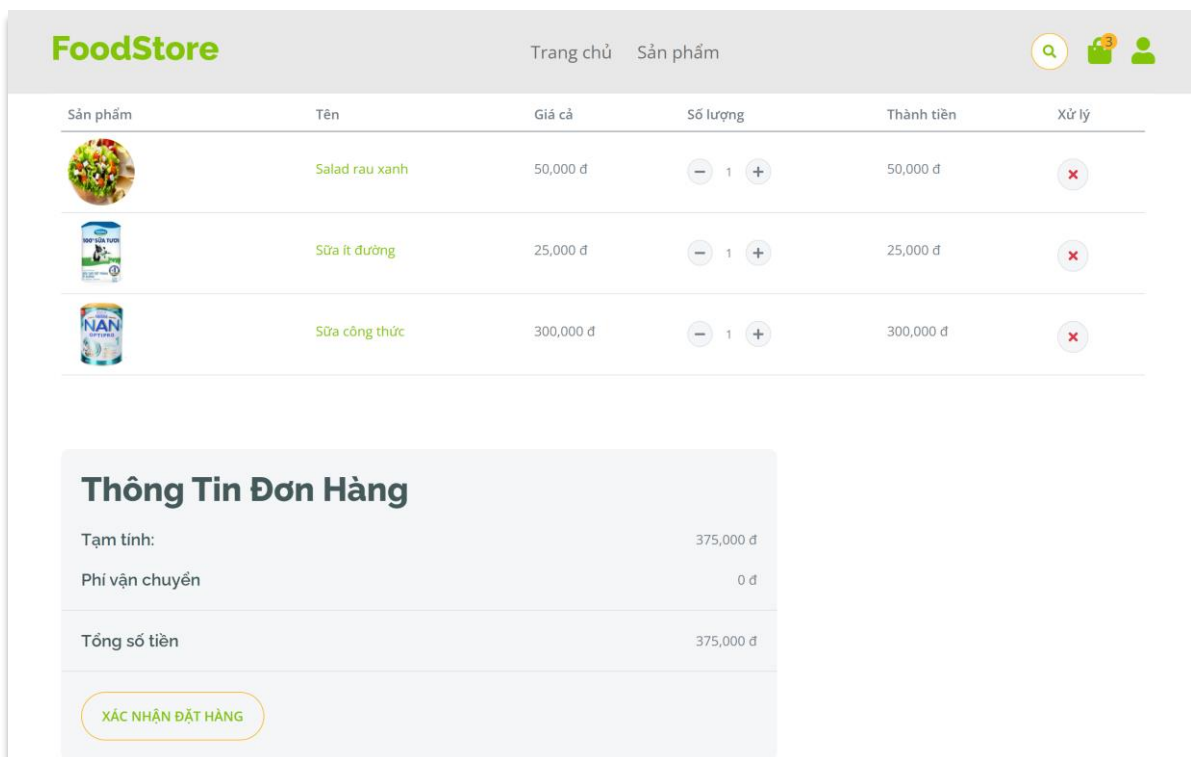


Hình 8. Tìm kiếm, lọc sản phẩm





Hình 9. Xem chi tiết sản phẩm



Hình 10. Thông tin đơn hàng

FoodStore

[Trang chủ](#)
[Sản phẩm](#)

Sản phẩm	Tên	Giá cả	Số lượng	Thành tiền
	Salad rau xanh	50,000 đ	1	50,000 đ
	Sữa ít đường	25,000 đ	1	25,000 đ
	Sữa công thức	300,000 đ	1	300,000 đ

Thông Tin Người Nhận

Tên người nhận

Trình Quang Lâm

Địa chỉ người nhận

Thành Phố Hà Nội

Số điện thoại

0971624914

[← Quay lại giỏ hàng](#)

Thông Tin Thanh Toán

Phí vận chuyển

0 đ

Hình thức

Thanh toán khi nhận hàng (COD)

Tổng số tiền

375,000 đ

XÁC NHẬN THANH TOÁN

Hình 11. Thanh toán

FoodStore

[Trang chủ](#)
[Sản phẩm](#)

[Home](#) / [Thanh toán](#)

Cảm ơn quý khách đã sử dụng sản phẩm của chúng tôi!

FoodStore

Fresh products

Your Email

Subscribe Now

Shop Info

[About Us](#)
[Contact Us](#)
[Privacy Policy](#)

Account

[My Account](#)
[Shop details](#)
[Shopping Cart](#)

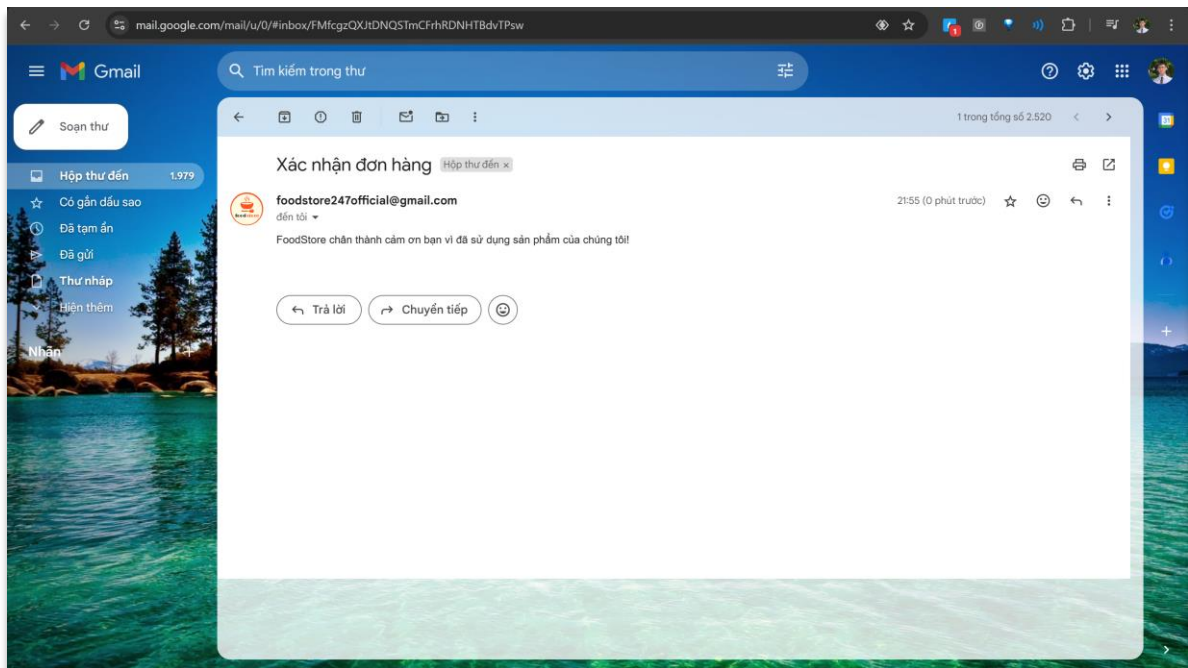
Contact

Address: Km10, Đường Nguyễn Trãi, Q. Hà Đông, Hà Nội

Email: trinquanglam2k4@gmail.com

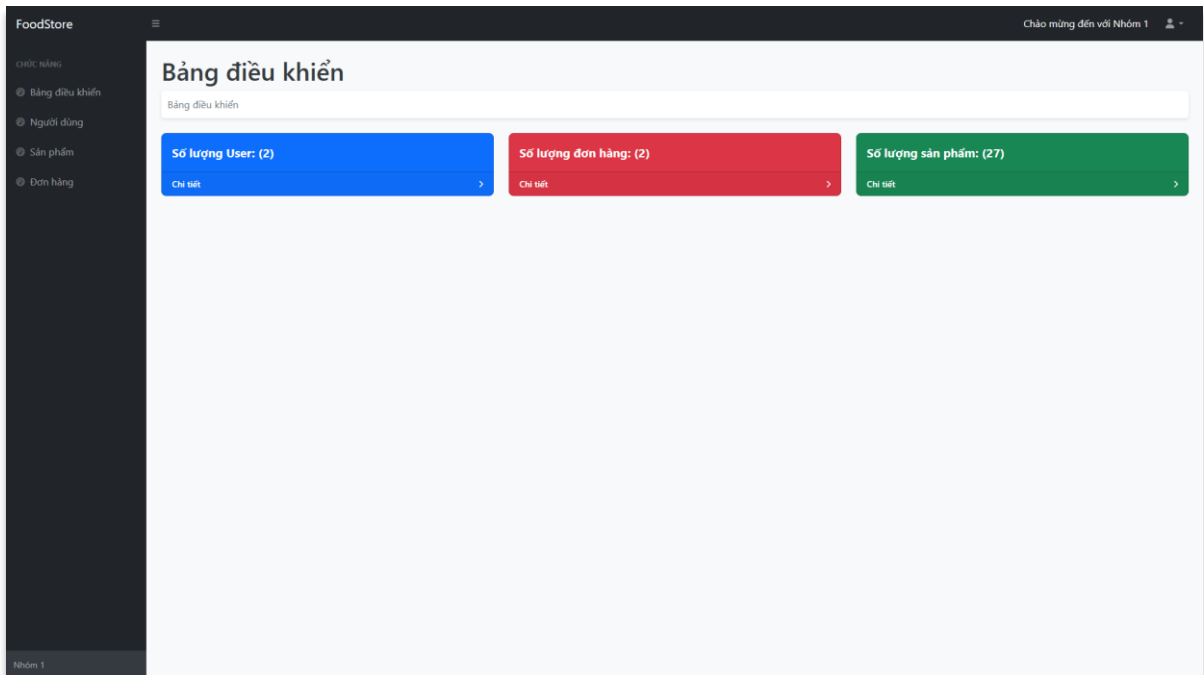
Hình 12. Thanh toán thành công

32

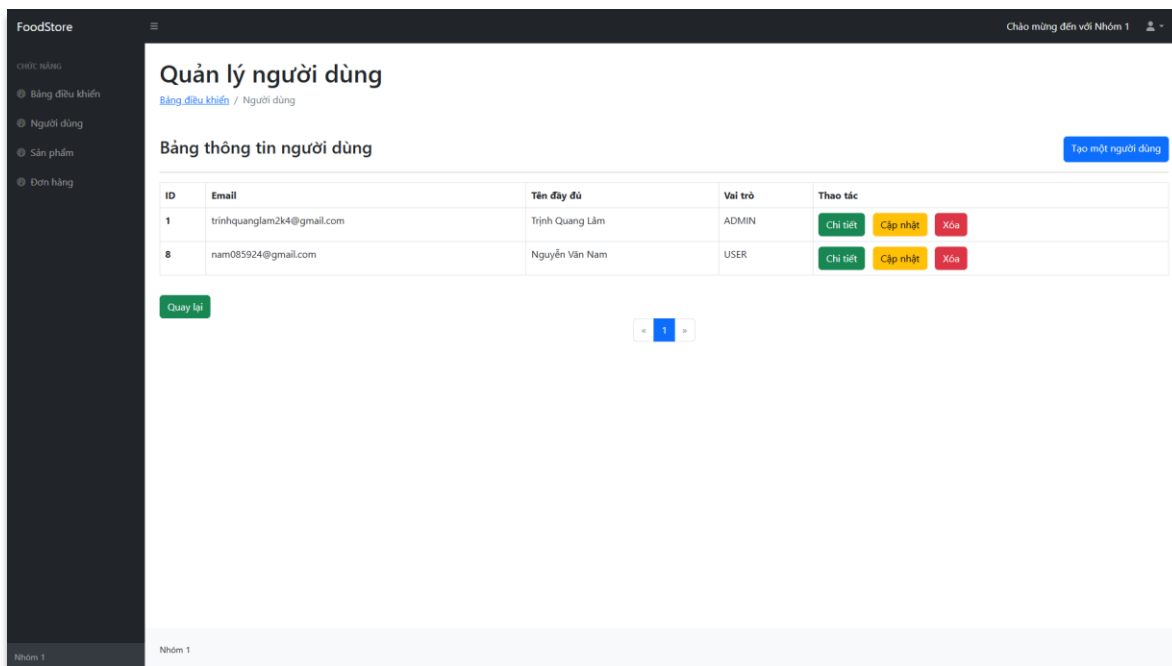


Hình 13. Xác nhận mua hàng

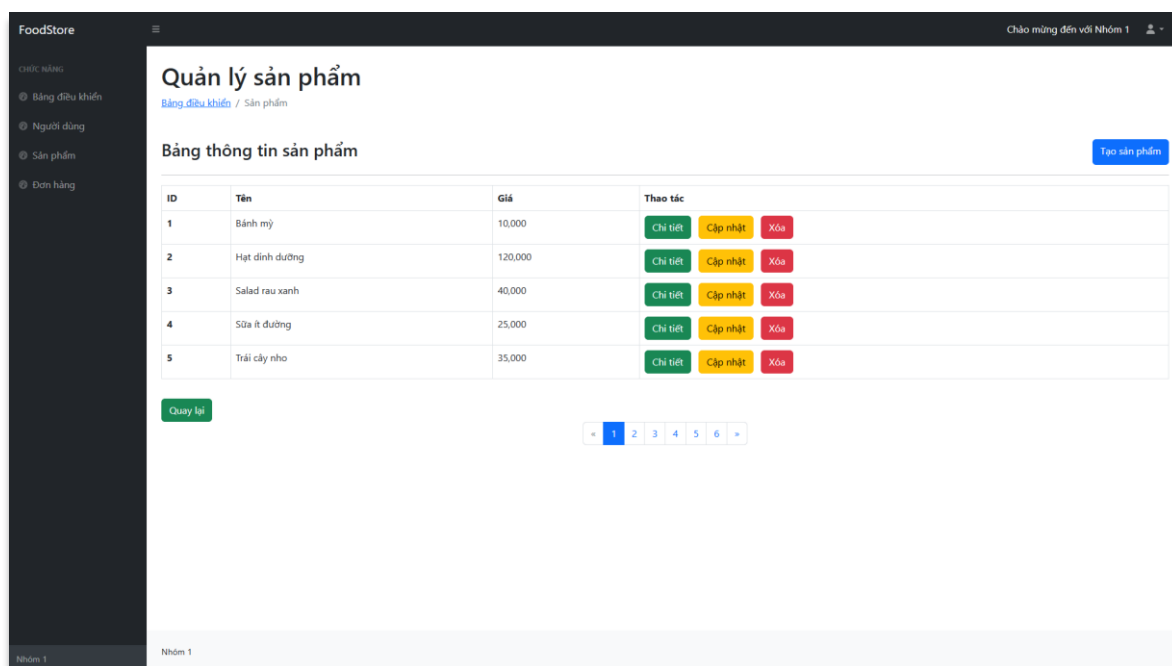
## 6.1. Giao diện phía người quản trị



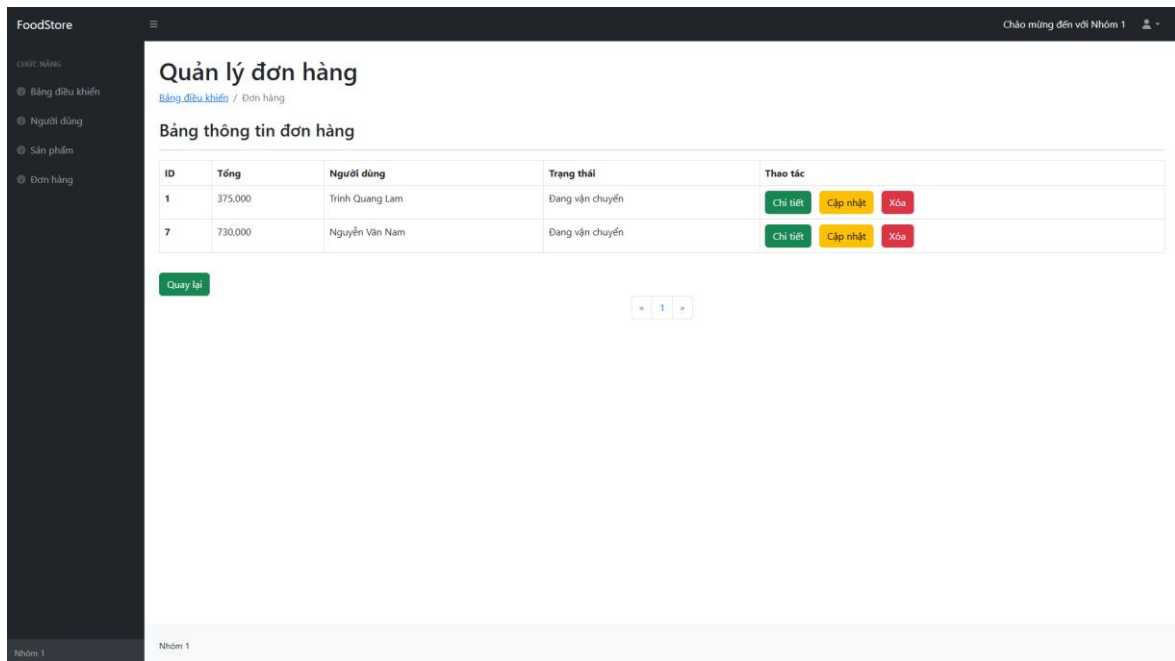
Hình 14. Số lượng người dùng, đơn hàng, sản phẩm hiện có



Hình 15. Quản lý người dùng



Hình 16. Quản lý sản phẩm



Hình 17. 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.

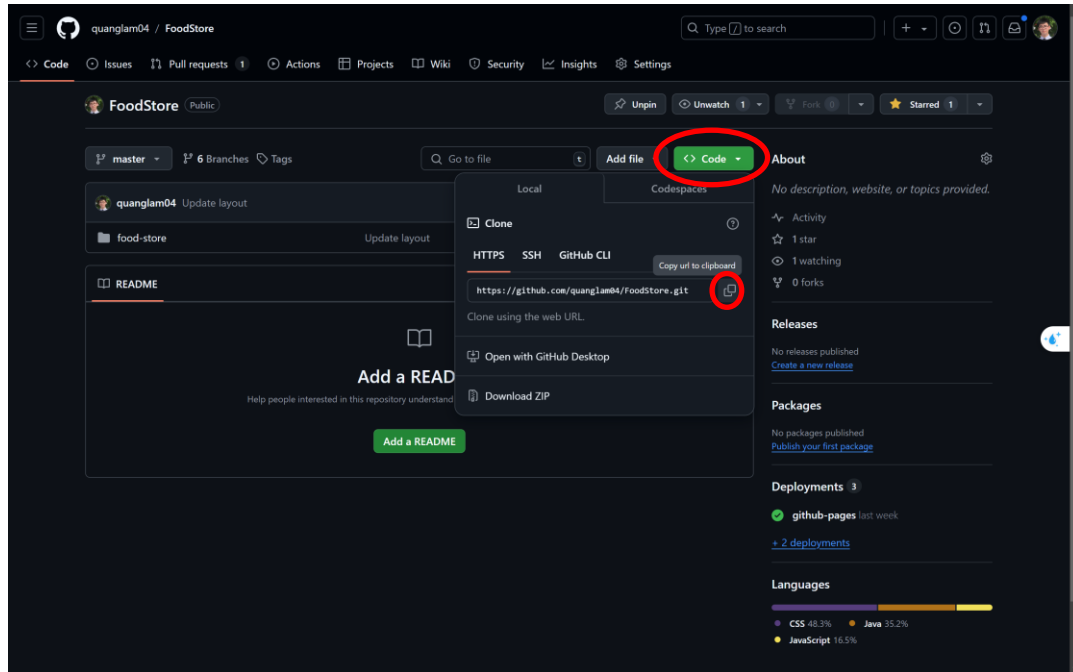
Bên cạnh những điểm đã đạt được, chẳng hạn như xây dựng một website hoàn chỉnh có các chức năng cơ bản phục vụ bán hàng, hỗ trợ phân quyền người dùng và tích hợp các công cụ mạnh mẽ từ Spring, hệ thống vẫn còn một số hạn chế:

- 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 khối lượng lớn dữ liệu

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

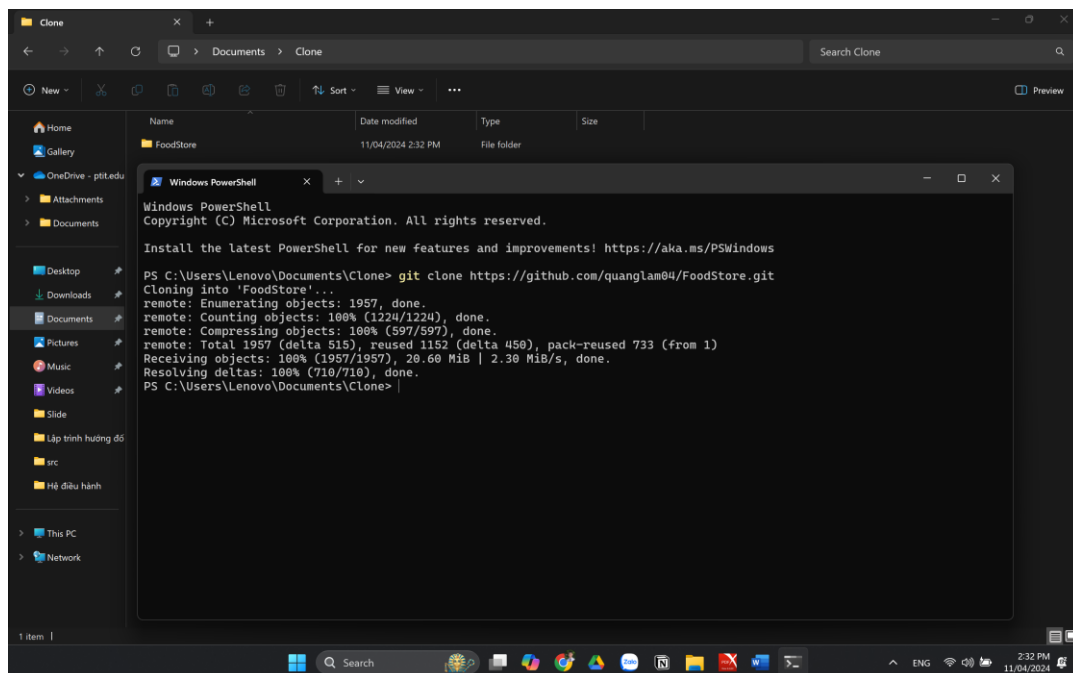
### B1: Sử dụng Git để lấy sourcode về máy trên Github Server

- 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 18: 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 19: 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 trống 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:localhost}: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
```

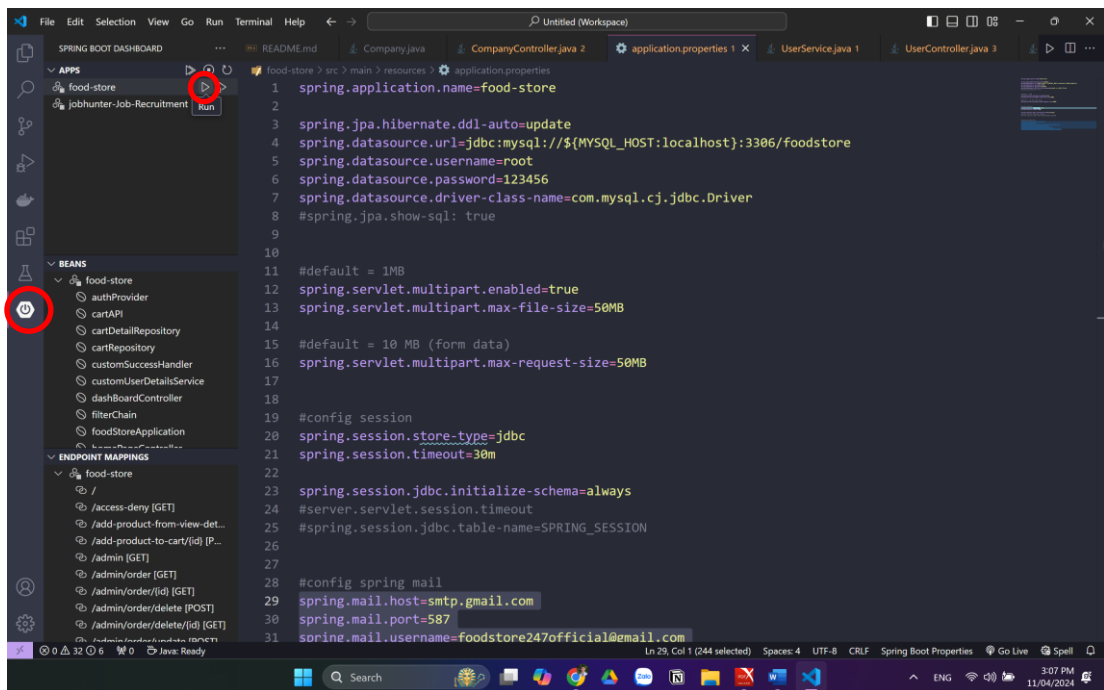
*\*Chú ý:* mật khẩu mail sử dụng trong bài sẽ là một App Password do Google tạo ra. Để lấy mật khẩu này, tham khảo hướng dẫn tại địa chỉ:

<https://support.google.com/accounts/answer/185833?hl=en>

## B6: Chạy ứng dụng

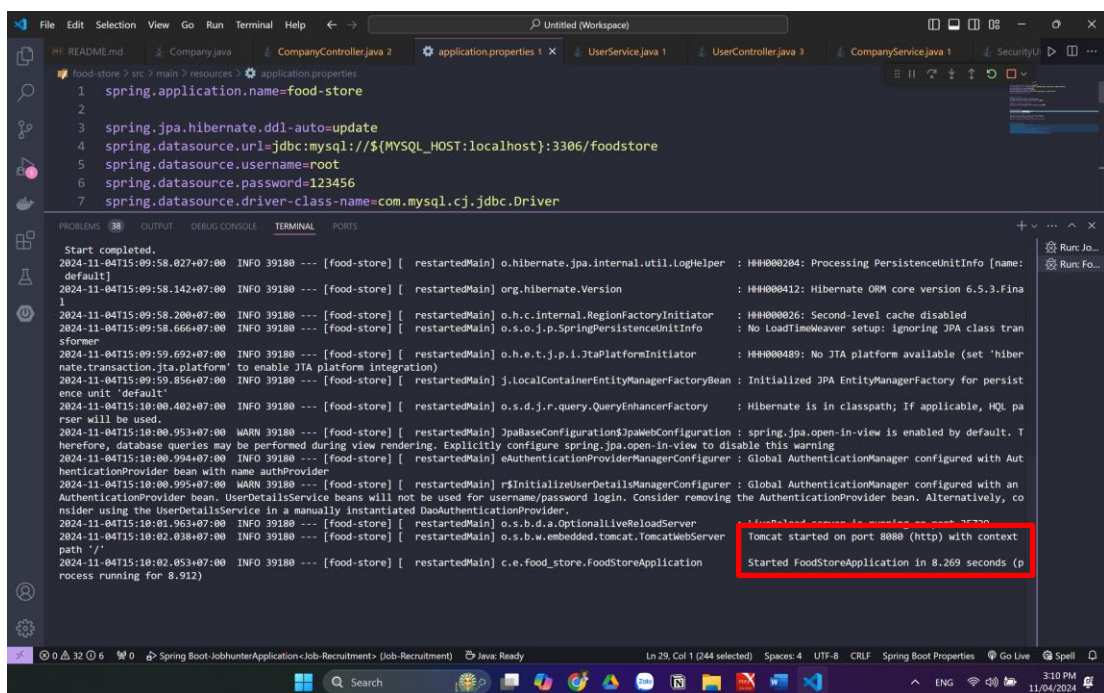


- Tại cửa sổ VS Code, chọn vào biểu tượng SpringBoot ở thanh bên trái, tại cửa sổ Apps sẽ hiện ra các Project sẵn có. Ấn vào biểu tượng run để chạy ứng dụng



Hình 20: Chạy ứng dụng

- Giao diện khi chạy ứng dụng thành công



Hình 21: Chạy thành công ứng dụng

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



Hình 21: 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](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](https://www.w3schools.com/howto/howto_js_autocomplete.asp)