**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC NÔNG LÂM TP.HCM**

**KHOA CÔNG NGHỆ THÔNG TIN**

-----&-----



**Tiểu luận tốt nghiệp**

**ỨNG DỤNG CÁC CÔNG NGHỆ REACT, SPRING BOOT, REDIS TRONG KIẾN TRÚC MICROSERVICE ĐỂ XÂY DỰNG WEBSITE BÁN NÔNG SẢN TRỰC TUYẾN.**

**Giảng viên hướng dẫn: Sinh viên thực hiện:**

**Ts. Nguyễn Thị Phương Phạm Lê Đạt 21130022**

**Trâm**

**BỘ GIÁO DỤC VÀ ĐÀO TẠO CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM**

**TRƯỜNG ĐH NÔNG LÂM TP.HCM Độc lập - Tự do - Hạnh phúc**

**I. NHIỆM VỤ LUẬN VĂN CỬ NHÂN**

Họ tên sinh viên: **PHẠM LÊ ĐẠT**

Giới tính: **Nam**

Ngày tháng năm sinh: 22/02/2003

Nơi sinh: Bình Phước

Chuyên ngành: Công nghệ thông tin

Ngành: Công nghệ thông tin

Email: 2110022@st.hcmuaf.edu.vn

**II. TÊN ĐỀ TÀI:** ỨNG DỤNG CÁC CÔNG NGHỆ REACT, SPRING BOOT, REDIS TRONG KIẾN TRÚC MICROSERVICE ĐỂ XÂY DỰNG WEBSITE BÁN NÔNG SẢN TRỰC TUYẾN.

**III. NHIỆM VỤ VÀ NỘI DUNG**

**1. Nhiệm vụ**  
Nghiên cứu tập trung thiết kế và triển khai website bán nông sản trực tuyến dựa trên kiến trúc Microservice, đảm bảo tính linh hoạt và khả năng mở rộng. Sử dụng React để xây dựng giao diện người dùng động, Spring Boot cho backend, và Redis để tối ưu hiệu suất, giảm tải cơ sở dữ liệu, nâng cao tốc độ truy cập, nhằm tạo hệ thống ổn định, đáp ứng tốt nhu cầu người dùng.

**2. Nội dung**  
Nội dung nghiên cứu gồm: tìm hiểu React (component, state, props, Hooks, React Router, lazy loading, code splitting) để phát triển giao diện thân thiện, hiệu quả; xây dựng backend bằng Spring Boot với kiến trúc Microservice, dùng Spring Cloud quản lý dịch vụ, Spring Security bảo mật API, tích hợp RESTful APIs/GraphQL; sử dụng Redis cho caching, quản lý phiên người dùng, xử lý sự kiện bất đồng bộ, cải thiện tốc độ phản hồi. Hệ thống sẽ được thử nghiệm để đảm bảo ổn định và hiệu quả.

**IV. NGÀY GIAO NHIỆM VỤ**:

**V. NGÀY HOÀN THÀNH NHIỆM VỤ**:

**VI. HỌ VÀ TÊN CÁN BỘ HƯỚNG DẪN**: Ts. Nguyễn Thị Phương Trâm

**LỜI CAM ĐOAN**

Tôi xin cam đoan đề tài “ ỨNG DỤNG CÁC CÔNG NGHỆ REACT, SPRING BOOT, REDIS TRONG KIẾN TRÚC MICROSERVICE ĐỂ XÂY DỰNG WEBSITE BÁN NÔNG SẢN TRỰC TUYẾN” là một công trình nghiên cứu độc lập, không sao chép các đề tài khác.

Đề tài là một sản phẩm do cá nhân tôi đã nỗ lực nghiên cứu, trong bài có sự tham khảo của một số tài liệu có nguồn gốc rõ ràng. Tôi cam đoan chịu hoàn toàn trách nhiệm về bản quyền.

**Sinh viên thực hiện**

Phạm Lê Đạt

**DANH MỤC CÁC BẢNG**

[Bảng 2.1: So sánh giữa Monolithic và Microservice 15](#_Toc32669)

[Bảng 2.2: So sánh RESTful API và SOAP API 21](#_Toc4660)

[Bảng 2.3: So sánh giữa Redis và Memcache 31](#_Toc10427)

[Bảng 2.4: So sánh giữa React, Angular và Vuejs 36](#_Toc22193)

[Bảng 2.5: Mô tả trạng thái đơn hàng 53](#_Toc8928)

[Bảng 3.1: Kiểm thử truy cập đồng thời endpoint log-in 67](#_Toc31765)

[Bảng 3.2: Kiểm thử truy cập đồng thời endpoint log-in 68](#_Toc14175)

[Bảng 3.3: Kiểm thử truy cập đồng thời endpoint log-in 68](#_Toc28725)

**DANH MỤC HÌNH ẢNH**

[Hình ảnh 2.1: Minh họa về kiến trúc Microservices 13](#_Toc21313)

[Hình ảnh 2.2: So sánh microservice với monolithic 15](#_Toc6115)

[Hình ảnh 2.3: API Gateway kết nối tới các service 18](#_Toc15544)

[Hình ảnh 2.4: Mô tả cơ bản về cách RESTful API hoạt động 20](#_Toc14071)

[Hình ảnh 2.5: Đoạn mã cấu hình xác thực OAuth 2.0 với Google 27](#_Toc2867)

[Hình ảnh 2.6: Đoạn mã cấu hình Spring Security để xác minh JWT 28](#_Toc3591)

[Hình ảnh 2.7: Đoạn mã trên cấu hình xác thực Basic Authentication 28](#_Toc28061)

[Hình ảnh 2.8: Minh họa hoạt động cơ bảng của redis 30](#_Toc8852)

[Hình ảnh 3.1: Sơ đồ kiến trúc hệ thống 39](#_Toc9396)

[Hình ảnh 3.2: Công nghệ được sử dụng ở frontend 40](#_Toc9913)

[Hình ảnh 2.9: Công nghệ được sử dụng ở backend 41](#_Toc1327)

[Hình ảnh 3.3: Các microservices của đề tài 43](#_Toc814)

[Hình ảnh 3.4: Logo của feign client và kafka 44](#_Toc10301)

[Hình ảnh 3.5: Lược đồ use case tổng quát của hệ thống 45](#_Toc23129)

[Hình ảnh 3.6: Lược đồ usecase của xác minh 46](#_Toc9432)

[Hình ảnh 3.7: Lược đồ usecase của quản lý đơn hàng 46](#_Toc5328)

[Hình ảnh 3.8: Lược đồ usecase xem sản phẩm 47](#_Toc16570)

[Hình ảnh 3.9: Lược đồ usecase quản lý người dùng 47](#_Toc19194)

[Hình ảnh 3.10: Lược đồ usecase quản lý đơn hàng 48](#_Toc15864)

[Hình ảnh 3.11: Lược đồ usecase quản lý sản phẩm 48](#_Toc15537)

[Hình ảnh 3.12: Lược đồ usecase quản lý phân quyền 49](#_Toc21144)

[Hình ảnh 3.13: Lược đồ quan hệ user 49](#_Toc20253)

[Hình ảnh 3.14: Lược đồ quan hệ sản phẩm 50](#_Toc402)

[Hình ảnh 3.15: Lược đồ quan hệ đơn hàng 51](#_Toc2579)

[Hình ảnh 3.16: Lược đồ tuần tự luồng đặt hàng 52](#_Toc2335)

[Hình ảnh 3.17: Lược đồ tuần tự luồng chỉnh sửa email/sđt 52](#_Toc6591)

[Hình ảnh 3.18: Lược đồ tuần tự luồng hủy đặt hàng 53](#_Toc30790)

[Hình ảnh 3.19: Chuyển đổi mã trạng thái từ GHTK sang bảng mã của hệ thống 55](#_Toc15979)

[Hình ảnh 3.20: Giao diện trang chủ 56](#_Toc19970)

[Hình ảnh 3.21: Giao diện đăng nhập 57](#_Toc9111)

[Hình ảnh 3.22: Giao diện cửa hàng 58](#_Toc22143)

[Hình ảnh 3.23: Giao diện chi tiết sản phẩm 59](#_Toc8386)

[Hình ảnh 3.24: Giao diện giỏ hàng 60](#_Toc24372)

[Hình ảnh 3.25: Giao diện thanh toán 61](#_Toc14745)

[Hình ảnh 3.26: Giao diện hồ sơ thông tin 61](#_Toc24813)

[Hình ảnh 3.27: Giao diện lịch sử đơn hàng 62](#_Toc8283)

[Hình ảnh 3.28: Giao diện lịch sử đơn hàng 62](#_Toc30462)

[Hình ảnh 3.29: Giao diện thống kê 63](#_Toc21694)

[Hình ảnh 3.30: Giao diện quản lý đơn hàng 64](#_Toc30006)

[Hình ảnh 3.31: Giao diện quản lí phân quyền 65](#_Toc2710)

[Hình ảnh 3.32: Giao diện quản lý sản phẩm 66](#_Toc14927)

[Hình ảnh 3.33: Giao diện thêm phân loại trọng lượng 67](#_Toc7943)

[Hình ảnh 3.34: Giao diện thêm loại sản phẩm 67](#_Toc2792)

[Hình ảnh 3.35: Giao diện quản lý hồ sơ 68](#_Toc22548)

DANH SÁCH CHỮ VIẾT TẮT

|  |  |  |
| --- | --- | --- |
| Viết tắt | Ý nghĩa đầy đủ | Giải thích ngắn gọn |
| API | Application Programming Interface | Giao diện lập trình cho phép các ứng dụng giao tiếp với nhau. |
| REST | Representational State Transfer | Kiến trúc API nhẹ, dựa trên HTTP. |
| SOAP | Simple Object Access Protocol | Giao thức nhắn tin dùng XML, thường dùng trong hệ thống bảo mật cao. |
| GHTK | Giao Hàng Tiết Kiệm | Đơn vị vận chuyển nội địa phổ biến tại Việt Nam. |
| ISPDS | Intelligent Sales and Product Delivery System | Hệ thống thông minh hỗ trợ bán hàng và phân phối sản phẩm. |
| WebUI | Web User Interface | Giao diện người dùng trên trình duyệt web. |
| SSL | Secure Sockets Layer | Giao thức mã hóa cũ, giúp bảo mật thông tin khi truyền qua mạng. |
| TLS | Transport Layer Security | Phiên bản nâng cấp và an toàn hơn của SSL, dùng phổ biến hiện nay (HTTPS). |
| SDT | Số điện thoại | Thông tin số điện thoại của khách hàng hoặc liên hệ. |

**MỤC LỤC**

**[DANH MỤC CÁC BẢNG 5](#_Toc29667)**

**[DANH MỤC HÌNH ẢNH 5](#_Toc5105)**

**[DANH SÁCH CHỮ VIẾT TẮT 6](#_Toc18389)**

**[MỞ ĐẦU 8](#_Toc19572)**

**[1. LÝ DO CHỌN ĐỀ TÀI 8](#_Toc2149)**

**[2. MỤC TIÊU VÀ PHẠM VI NGHIÊN CỨU 9](#_Toc14100)**

**[3. KẾT QUẢ DỰ KIẾN ĐẠT ĐƯỢC 9](#_Toc945)**

[3.1 Về phía Người mua hàng 9](#_Toc21325)

[3.2 Về phía quản trị viên 10](#_Toc17868)

[3.3 Đánh giá hiệu năng 10](#_Toc1354)

**[CHƯƠNG 1: TỔNG QUAN ĐỀ TÀI 10](#_Toc4368)**

[1.1 Phân tích, đánh giá các công trình nghiên cứu đã có liên quan đến đề tài nghiên cứu 10](#_Toc8552)

[1.2 Những vấn đề còn tồn tại 11](#_Toc24081)

[1.3 Nội dung nghiên cứu 12](#_Toc1268)

**[CHƯƠNG 2: PHƯƠNG PHÁP VÀ NGHIÊN CỨU 13](#_Toc18403)**

**[2.1 Microservices 13](#_Toc11713)**

**[2.2 API Gateway, RESTful API 19](#_Toc12244)**

[2.2.1 API Gateway 19](#_Toc3977)

[2.2.2 RESTful API 22](#_Toc31696)

**[2.3 Spring Boot 25](#_Toc7478)**

**[2.4 Redis 31](#_Toc11525)**

[2.4.1 Giới thiệu về redis 32](#_Toc28032)

[2.4.2 Ưu điểm của Redis 33](#_Toc14468)

[2.4.3 Nhược điểm của Redis 33](#_Toc23950)

[2.4.4 So sánh Redis và Memcached 34](#_Toc6629)

[2.4.5 Kết luận về Redis 34](#_Toc26993)

**[2.5 React 35](#_Toc6217)**

[2.5.1 Giới thiệu về React 35](#_Toc29113)

[2.5.2 Những đặc trưng của React 36](#_Toc14381)

[2.5.3 Ưu điểm của React 37](#_Toc7799)

[2.5.4 Nhược điểm của React 37](#_Toc26843)

[2.5.5 So sánh React, Angular và Vue.js 38](#_Toc21337)

[2.5.6 Kết luận về React 39](#_Toc12940)

**[CHƯƠNG 3: GIẢI PHÁP CHO BÀI TOÁN/VẤN ĐỀ/MÔ HÌNH 39](#_Toc1496)**

**[3.1 Phát biểu bài toán trong đề tài 39](#_Toc4727)**

[3.1.1 Bài toán nghiên cứu 39](#_Toc3458)

[3.1.2 Giải pháp nghiên cứu 40](#_Toc23639)

[3.1.3 Kết luận 40](#_Toc2024)

**[3.2 Giải pháp cụ thể để giải quyết mô hình/bài toán 40](#_Toc29886)**

**[3.3 Hiện thực giải pháp 42](#_Toc6525)**

[3.3.1 Mô hình chương trình 42](#_Toc5162)

[3.3.2 Mô hình hoá yêu cầu 47](#_Toc12441)

[3.3.3 Hiện thực chương trình phần mềm 57](#_Toc8277)

**[CHƯƠNG 4: KẾT QUẢ, KẾT LUẬN VÀ KIẾN NGHỊ 72](#_Toc697)**

**[4.1 KẾT QUẢ 72](#_Toc1209)**

[4.1.1 Kiến thức đã tìm hiểu được 72](#_Toc2405)

[4.1.2 Kết quả phát triển giao diện người dùng 73](#_Toc2251)

[4.1.3 Chương trình đã làm được 73](#_Toc6667)

[4.1.4 Hiệu suất hệ thống 74](#_Toc20710)

[4.1.5 Những điểm hạn chế 74](#_Toc25009)

**[TÀI LIỆU THAM KHẢO 75](#_Toc28585)**

MỞ ĐẦU

1. LÝ DO CHỌN ĐỀ TÀI

Trong thời đại công nghệ số bùng nổ, khi mọi khoảng cách giữa con người dần được xóa nhòa nhờ những cú nhấp chuột, tôi chọn đề tài "Ứng dụng các công nghệ React, Spring Boot, Redis trong kiến trúc Microservice để xây dựng website bán nông sản trực tuyến" không chỉ để bắt kịp xu hướng mà còn để tạo ra một dấu ấn sáng tạo, mang giá trị thực tiễn sâu sắc. Ý tưởng này bắt nguồn từ thực trạng: người tiêu dùng ngày càng khao khát sự tiện lợi trong việc tiếp cận nông sản tươi ngon, trong khi người nông dân lại cần một cầu nối để đưa sản phẩm của mình ra thị trường rộng lớn hơn. Đề tài của tôi chính là lời giải cho bài toán đó, kết hợp giữa công nghệ hiện đại và nhu cầu thiết yếu của cuộc sống.

Trước hết, đây là một cuộc chơi của sự đổi mới. Kiến trúc Microservice không chỉ đơn thuần là một xu hướng kỹ thuật, mà là cách tôi chia nhỏ hệ thống thành những mảnh ghép độc lập, linh hoạt như những cánh đồng được chăm sóc riêng biệt – dễ dàng mở rộng, bảo trì và thích nghi với mọi thay đổi của thị trường. React, với sức mạnh của mình, biến giao diện website thành một bức tranh sống động, nơi người dùng có thể "chạm" vào từng sản phẩm với trải nghiệm mượt mà, gần gũi. Spring Boot, như một người quản lý tài ba, dựng nên hậu trường vững chắc, đảm bảo mọi dịch vụ từ quản lý đơn hàng đến thanh toán đều hoạt động trơn tru. Và Redis – "người hùng thầm lặng" – giúp website luôn nhanh nhẹn, sẵn sàng phục vụ hàng nghìn khách hàng cùng lúc mà không chút chậm trễ.

Sau cùng, đây là cơ hội để tôi thử thách và hoàn thiện chính mình. Việc chinh phục những công nghệ tiên tiến như React, Spring Boot, Redis và kiến trúc Microservice không chỉ giúp tôi nắm bắt xu hướng công nghệ thông tin hiện đại, mà còn rèn luyện tư duy sáng tạo, khả năng tối ưu hóa hệ thống và giải quyết vấn đề thực tế. Tôi tin rằng, qua đề tài này, tôi không chỉ tạo ra một sản phẩm công nghệ, mà còn viết nên câu chuyện về sự kết nối, đổi mới và giá trị cộng đồng.

Tóm lại, lý do tôi chọn đề tài này là để hòa quyện giữa đam mê công nghệ và khát vọng tạo ra điều gì đó ý nghĩa. Một website bán nông sản trực tuyến không chỉ là một nền tảng kỹ thuật, mà còn là giấc mơ về một tương lai nơi công nghệ và nông nghiệp cùng nhau vươn xa.

1. MỤC TIÊU VÀ PHẠM VI NGHIÊN CỨU

Tôi tập trung vào mục tiêu triển khai kiến trúc Microservices cùng với các công nghệ tiên tiến như Spring Boot, Cloud API Gateway, Redis và React để xây dựng một nền tảng thương mại điện tử có khả năng chịu tải cao, đồng thời có khả năng mở rộng và thu hẹp các dịch vụ một cách hiệu quả. Trước hết, tôi sẽ tiến hành nghiên cứu sâu sắc về kiến trúc, các ứng dụng và những thách thức gắn liền với Microservices. Những kiến thức này sẽ tạo ra một nền tảng vững chắc, giúp tôi định hướng trong việc triển khai đề tài một cách hiệu quả.

Về phía Front End, đề tài sử dụng ReactJs cho hai dự án, bao gồm: giao diện quản trị giúp quản trị viên có thể quản lí hệ tương tác để quản lí hệ thống và giao diện khách hàng giúp người dùng có thể tương tác với hệ thống để thực hiện các chức năng mua, tìm kiếm, xem chi tiết và đánh giá sản phẩm, cùng nhiều chức năng hữu ích khác.

Về phần Back End, đề tài sử dụng Spring boot, Mysql để xây dựng nên hệ thống RESTful API. Áp dụng Kafka làm trung gian lưu trữ các message để các service có thể giao tiếp bất đồng bộ với nhau. Tích hợp api bên ngoài như VNPAY dùng cho việc thanh toán trực tiếp, GHTK để thực hiện vận chuyển các đơn hàng (cả hai đều đang được dùng trong môi trường thử nghiệm). Đồng thời để cải thiện tốc độ xử lý, đề tài sử dụng các công nghệ hiện đại như Redis để tối ưu và tăng hiệu suất của hệ thống.

3. KẾT QUẢ DỰ KIẾN ĐẠT ĐƯỢC

Xây dựng sàn thương mại điện tử có khả năng chịu tải cao, tự động mở rộng và

thu nhỏ một cách hiệu quả, cùng lúc có thể đáp ứng được số lượng người dùng với các chức năng đáp ứng được các nhu cầu cơ bản của một sàn thương mại điện tử và tạo ra sự tiện lợi khi sử dụng cho người dùng

1. Về phía Người mua hàng

Người mua hàng có thể thực hiện nhiều thao tác khác nhau trên hệ thống để thuận tiện trong quá trình mua sắm trực tuyến. Đầu tiên, họ có thể đăng nhập, đăng ký, đăng xuất tài khoản, khôi phục mật khẩu khi cần và tìm kiếm sản phẩm và sử dụng các bộ lọc để tìm kiếm chính xác hơn. Khi đã đăng nhập, người mua hàng có thể thay đổi thông tin cá nhân, bình luận về các sản phẩm, thêm sản phẩm vào giỏ hàng, xem giỏ hàng, tiến hành mua hàng và thanh toán. Quá trình thanh toán được thực hiện một cách nhanh chóng và thuận tiện. Sau quá trình đặt hàng, người mua có thể theo dõi, xem chi tiết và cập nhật trạng thái đơn hàng.

1. Về phía quản trị viên

Hệ thống quản trị cho phép các quản trị viên theo dõi thống kê doanh thu theo ngày, quý, năm và xuất báo cáo sang Excel, đồng thời quản lý danh sách sản phẩm bán chạy và khách hàng tiềm năng. Moderator nắm quyền cao nhất, có thể tạo tài khoản nhân viên, quản lí tài khoản, hồ sơ ; quản lý vai trò, quyền hạn và toàn bộ chức năng khác. Các nhân viên cấp dưới được phân công vai trò cụ thể: Nhân viên kho hàng đảm nhận quản lý sản phẩm, có quyền thêm, sửa, xóa sản phẩm và xem báo cáo sản phẩm bán chạy. Mỗi nhân viên chỉ được truy cập các chức năng liên quan đến vai trò được gán, đảm bảo phân quyền rõ ràng và hiệu quả.

1. Đánh giá hiệu năng

Hệ thống được thiết kế để có thể đáp ứng được lượng lớn người dùng truy cập

cùng một lúc, với tốc độ phản hồi nhanh và đảm bảo các chức năng thực hiện đúng. Độ ổn định của hệ thống là một yếu tố quan trọng, giúp người dùng có trải nghiệm tốt và tin cậy khi sử dụng dịch vụ.

1. TỔNG QUAN ĐỀ TÀI
2. Phân tích, đánh giá các công trình nghiên cứu đã có liên quan đến đề tài nghiên cứu

Một công trình nghiên cứu đáng chú ý của Ruxue Bai và Xueqian Song tại Hội nghị Quốc tế lần thứ 5 về Khoa học Thông tin, Hệ thống Song song và Phân tán (ISPDS) năm 2024 đã đề xuất một phương pháp sáng tạo trong việc áp dụng kiến trúc microservice vào thương mại điện tử, tập trung vào việc phân tách logic kinh doanh phức tạp thành các RestAPI microservices và WebUI microservices thông qua kết hợp phân vùng ngang, dọc và thiết kế hướng miền. Điểm mạnh của nghiên cứu này nằm ở việc quản lý hiệu quả các dịch vụ phân tán trên nhiều máy chủ, cụm, hoặc đám mây bằng cách sử dụng các quy tắc gọi giữa các dịch vụ được cấu hình động, đảm bảo hiệu suất cao, tính sẵn sàng và khả năng xử lý đồng thời. Đây là những yếu tố quan trọng trong thương mại điện tử. Tuy nhiên, công trình này chủ yếu tập trung vào thương mại điện tử nói chung, chưa đề cập đến các thách thức đặc thù của lĩnh vực nông sản trực tuyến như sản phẩm dễ hư hỏng, giao hàng nhanh, hay biến động theo mùa vụ, do đó chưa hoàn toàn phù hợp với bối cảnh của đề tài này. Dù vậy, phương pháp phân tách microservice được đề xuất trong nghiên cứu là một nền tảng lý thuyết và thực tiễn mạnh mẽ, mà tiểu luận này có thể tham khảo để phát triển hệ thống bán nông sản trực tuyến, đặc biệt trong việc tối ưu hóa giao tiếp giữa các dịch vụ như Dịch vụ sản phẩm và Dịch vụ đơn hàng [1].

Một nghiên cứu khác của Nahid Nawal Nayim và các đồng tác giả, trình bày tại Hội nghị Quốc tế lần thứ 26 về Công nghệ Thông tin và Máy tính (ICCIT) năm 2023, thực hiện đánh giá hiệu suất giữa kiến trúc monolithic và microservice trong bối cảnh một startup thương mại điện tử. Nghiên cứu này phát triển hai ứng dụng phần mềm sử dụng cùng nền tảng công nghệ, một theo kiến trúc monolithic và một theo microservice, để so sánh hiệu suất và chi phí. Kết quả cho thấy kiến trúc monolithic có thể giảm chi phí cơ sở hạ tầng và cải thiện hiệu suất trong một số điều kiện nhất định, thách thức quan niệm phổ biến về ưu thế của microservices. Điểm mạnh của công trình nằm ở việc sử dụng mô phỏng thực tế, thử nghiệm áp lực, và mô hình kinh tế để phân tích chi tiết, cung cấp cơ sở dữ liệu thực tiễn cho các doanh nghiệp khởi nghiệp. Tuy nhiên, nghiên cứu chưa xem xét các yếu tố đặc thù của ngành nông sản, như biến động tồn kho hoặc tích hợp logistics, làm hạn chế khả năng áp dụng trực tiếp vào đề tài này. Dù vậy, kết quả này mang lại giá trị tham khảo cho tiểu luận trong việc cân nhắc lựa chọn kiến trúc phù hợp khi phát triển website bán nông sản trực tuyến [2].

1. Những vấn đề còn tồn tại

Mặc dù các công trình nghiên cứu liên quan đã cung cấp những đóng góp giá trị cho việc phát triển hệ thống thương mại điện tử, nhưng vẫn tồn tại một số hạn chế đáng chú ý khi áp dụng vào bối cảnh bán nông sản trực tuyến. Nghiên cứu của Ruxue Bai và Xueqian Song (2024) đề xuất một phương pháp sáng tạo sử dụng kiến trúc microservice, tập trung vào việc phân tách logic kinh doanh thành các RESTAPI và WebUI microservices, đảm bảo hiệu suất cao và khả năng xử lý đồng thời. Tuy nhiên, công trình này chưa đề cập đến các thách thức đặc thù của lĩnh vực nông sản, như tính chất dễ hư hỏng của sản phẩm, yêu cầu giao hàng nhanh, hay sự biến động theo mùa vụ. Những yếu tố này đòi hỏi các giải pháp tối ưu hóa riêng biệt về quản lý tồn kho và tích hợp logistics, vốn chưa được nghiên cứu sâu trong công trình. Tương tự, nghiên cứu của Nahid Nawal Nayim và các đồng tác giả (2023) cung cấp cái nhìn thực tiễn khi so sánh hiệu suất giữa kiến trúc monolithic và microservice, chỉ ra rằng kiến trúc monolithic có thể tối ưu chi phí và hiệu suất trong một số trường hợp. Tuy nhiên, nghiên cứu này cũng không xem xét các đặc điểm riêng của ngành nông sản, chẳng hạn như tích hợp hệ thống logistics thời gian thực hoặc quản lý biến động tồn kho theo mùa. Ngoài ra, cả hai nghiên cứu đều chưa đề cập đến các yếu tố như trải nghiệm người dùng trên giao diện web hoặc tích hợp các công nghệ mới như trí tuệ nhân tạo để dự đoán nhu cầu nông sản. Những hạn chế này cho thấy cần có thêm các nghiên cứu chuyên sâu, tập trung vào các đặc thù của lĩnh vực nông sản trực tuyến, để xây dựng một hệ thống hiệu quả và phù hợp hơn với bối cảnh thực tế.

1. Nội dung nghiên cứu

Để giải quyết các hạn chế kỹ thuật trong việc phát triển hệ thống bán nông sản trực tuyến, tiểu luận cần tập trung vào các nội dung nghiên cứu sau, sử dụng Redis cho cache dữ liệu, Giao Hàng Tiết Kiệm (GHTK) cho vận chuyển, và VNPay cho thanh toán. Thứ nhất, để xử lý vấn đề tối ưu hóa thời gian phản hồi cho các giao dịch thời gian thực, cần triển khai Redis để cache dữ liệu sản phẩm, tồn kho, và trạng thái đơn hàng, áp dụng cơ chế thời gian tồn tại (TTL) và invalidate cache dựa trên sự kiện để đảm bảo dữ liệu luôn cập nhật, đặc biệt khi tồn kho nông sản biến động nhanh theo mùa vụ. Thứ hai, nhằm khắc phục hạn chế về tích hợp logistics, tích hợp API của GHTK sẽ hỗ trợ đồng bộ hóa trạng thái đơn hàng và tối ưu hóa thời gian giao hàng nhanh cho sản phẩm dễ hư hỏng, kết hợp với cơ chế xử lý lỗi API, như retry hoặc thông báo, để đảm bảo tính ổn định khi GHTK gặp sự cố. Thứ ba, để giải quyết vấn đề bảo mật giao dịch, tích hợp VNPay với giao thức mã hóa SSL/TLS và xác thực mạnh mẽ sẽ đảm bảo an toàn cho các giao dịch trực tuyến, đồng thời cần xây dựng quy trình xử lý các trường hợp thanh toán thất bại hoặc hoàn tiền để nâng cao trải nghiệm người dùng. Thứ tư, để xử lý hạn chế về khả năng mở rộng của hệ thống khi lưu lượng giao dịch tăng đột biến, như trong mùa vụ hoặc chiến dịch khuyến mãi, cần thực hiện thử nghiệm áp lực (stress testing) trên hệ thống tích hợp Redis, GHTK, và VNPay, đồng thời xem xét áp dụng kiến trúc microservice cho các dịch vụ độc lập như quản lý sản phẩm và đơn hàng để tăng tính linh hoạt. Thứ năm, để đảm bảo tính nhất quán dữ liệu, cần nghiên cứu cơ chế đồng bộ hóa giữa Redis, GHTK, và VNPay, đảm bảo tồn kho được cập nhật ngay sau khi thanh toán VNPay thành công và đơn hàng được chuyển cho GHTK. Cuối cùng, để cải thiện trải nghiệm người dùng trên giao diện web, cần thiết kế giao diện thân thiện, tối ưu hóa tốc độ tải trang bằng Redis và đảm bảo tính tương thích với các thiết bị khác nhau. Những nội dung này sẽ giúp xây dựng một hệ thống thương mại điện tử nông sản hiệu quả, đáp ứng các yêu cầu đặc thù về tốc độ, bảo mật, khả năng mở rộng, và trải nghiệm người dùng.

1. PHƯƠNG PHÁP VÀ NGHIÊN CỨU
2. Microservices

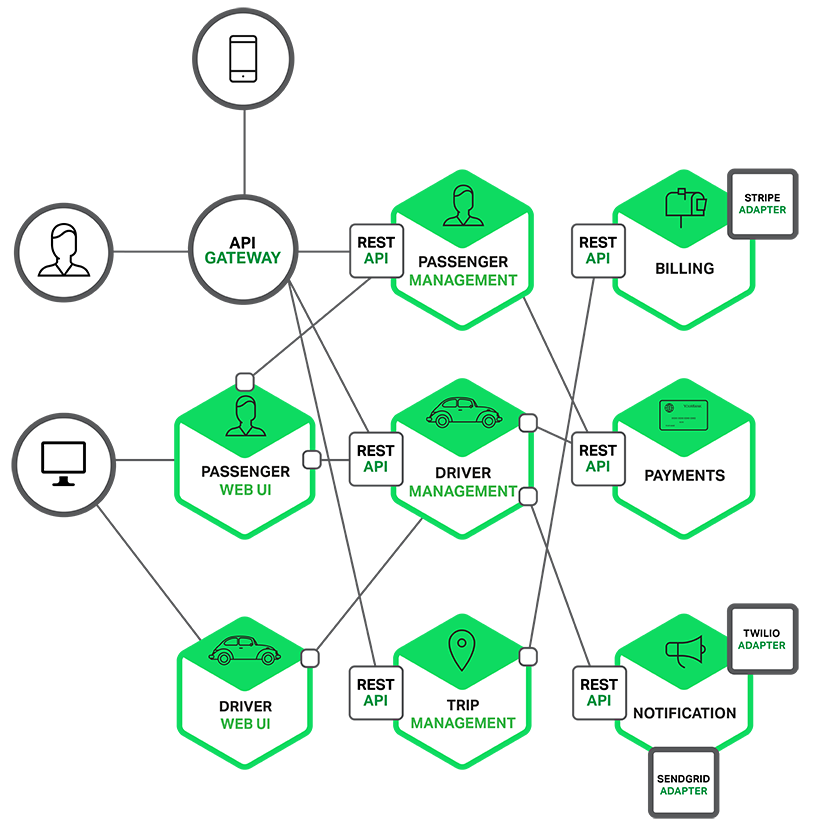
Kiến trúc microservices được lựa chọn cho hệ thống bán nông sản trực tuyến vì các ưu điểm phù hợp với đặc thù của lĩnh vực này. Thứ nhất, microservices cho phép phân tách các chức năng hệ thống thành các dịch vụ độc lập, như dịch vụ sản phẩm, dịch vụ đơn hàng, và dịch vụ thanh toán, giúp tối ưu hóa quản lý và triển khai riêng lẻ từng thành phần. Điều này đặc biệt quan trọng khi xử lý các yêu cầu riêng biệt của ngành nông sản, chẳng hạn như cập nhật tồn kho nhanh chóng hoặc tích hợp logistics thời gian thực. Thứ hai, microservices hỗ trợ khả năng mở rộng linh hoạt, cho phép tăng cường tài nguyên cho các dịch vụ cụ thể (ví dụ: dịch vụ đơn hàng trong mùa vụ cao điểm) mà không ảnh hưởng đến toàn bộ hệ thống, khắc phục hạn chế về lưu lượng giao dịch tăng đột biến. Thứ ba, kiến trúc này giúp giảm thiểu rủi ro khi tích hợp các công nghệ như Redis, GHTK, và VNPay, vì mỗi dịch vụ có thể được phát triển và kiểm tra độc lập, đảm bảo tính ổn định và dễ dàng xử lý lỗi. So với kiến trúc monolithic, microservices cung cấp tính linh hoạt cao hơn trong việc bảo trì và nâng cấp hệ thống, đặc biệt khi cần tích hợp thêm các dịch vụ logistics hoặc thanh toán mới trong tương lai. Do đó, microservices là lựa chọn tối ưu để xây dựng một hệ thống thương mại điện tử nông sản hiệu quả và bền vững.

1. Giới thiệu về Microservices

Microservices là một phong cách kiến trúc phần mềm hiện đại, bắt nguồn từ ý tưởng của kiến trúc hướng dịch vụ (SOA), vốn xuất hiện từ những năm 1990. SOA hướng đến việc phân chia ứng dụng thành các dịch vụ nhỏ, độc lập để tăng tính linh hoạt và khả năng tái sử dụng. Tuy nhiên, SOA thường gặp khó khăn trong thực tế do các dịch vụ lớn và sự phụ thuộc chặt chẽ giữa chúng.Từ giữa những năm 2000, các tập đoàn công nghệ như Amazon, Netflix và Twitter đã tiên phong áp dụng Microservices để giải quyết nhu cầu về khả năng mở rộng, tính mô-đun và triển khai liên tục. Sự thành công của họ đã tạo động lực để Microservices lan tỏa sang nhiều lĩnh vực phát triển phần mềm khác. Đến thập niên 2010, kiến trúc này trở thành xu hướng chủ đạo, thay thế dần kiến trúc nguyên khối (monolithic) truyền thống, giúp các tổ chức xây dựng ứng dụng phức tạp, linh hoạt và dễ thích nghi.

Trong Microservices, logic ứng dụng được chia thành các dịch vụ độc lập, gọi là microservices, mỗi dịch vụ đảm nhiệm một chức năng cụ thể. Không giống ứng dụng nguyên khối tích hợp toàn bộ logic, mỗi microservice có thể được phát triển bằng ngôn ngữ lập trình khác nhau, triển khai và mở rộng riêng lẻ. Chúng giao tiếp với nhau qua các giao thức như HTTP, RPC hoặc hàng đợi tin nhắn. Cách tiếp cận này tăng cường tính mô-đun, tách biệt chức năng, giúp việc phát triển, kiểm thử và bảo trì ứng dụng trở nên đơn giản và hiệu quả hơn.

Microservices mang lại lợi ích vượt trội như khả năng mở rộng linh hoạt, hỗ trợ triển khai liên tục và tích hợp với các hệ thống khác. Kiến trúc này đặc biệt quan trọng trong việc xây dựng các hệ thống phần mềm phức tạp, đáp ứng nhanh chóng nhu cầu của thị trường và người dùng. Hiện nay, Microservices tiếp tục phát triển, đóng vai trò nền tảng trong ngành công nghiệp phần mềm.



1. Minh họa về kiến trúc Microservices
2. Những đặc trưng của Microservices

Kiến trúc Microservices cung cấp một cách tiếp cận linh hoạt và hiệu quả để quản lý và phát triển các ứng dụng phức tạp, đặc biệt là đối với các nền tảng thương mại điện tử nơi tính mở rộng và linh hoạt là rất quan trọng. Dưới đây là một số lợi ích chính của kiến trúc Microservices đối với thương mại điện tử:

1. Tính mở rộng độc lập

Kiến trúc Microservices cho phép mỗi dịch vụ được mở rộng độc lập, giúp tối ưu hóa tài nguyên trong các nền tảng thương mại điện tử. Ví dụ, trong các sự kiện mua sắm lớn như Black Friday, chỉ dịch vụ thanh toán hoặc tìm kiếm sản phẩm cần tăng số lượng instance, thay vì mở rộng toàn bộ hệ thống. Điều này giảm chi phí vận hành, đảm bảo hiệu suất cao cho các chức năng quan trọng và duy trì trải nghiệm mượt mà cho người dùng.

1. Tùy chọn công nghệ linh hoạt

Microservices cho phép sử dụng các công nghệ khác nhau cho từng dịch vụ, phù hợp với yêu cầu cụ thể của nền tảng thương mại điện tử. Chẳng hạn, dịch vụ gợi ý sản phẩm có thể dùng Python với machine learning, trong khi dịch vụ thanh toán dùng Java để đảm bảo bảo mật. Tính linh hoạt này giúp tối ưu hóa hiệu suất, dễ dàng tích hợp công nghệ mới và hỗ trợ các nhóm phát triển làm việc với công cụ họ thành thạo.

1. Giao tiếp qua API

Các microservices giao tiếp thông qua các API chuẩn như REST, gRPC hoặc message queues, đảm bảo tính tương thích và linh hoạt trong hệ thống thương mại điện tử. Ví dụ, dịch vụ giỏ hàng gửi yêu cầu qua API đến dịch vụ thanh toán để xử lý giao dịch. Điều này cho phép tích hợp dễ dàng với các dịch vụ bên thứ ba như cổng thanh toán hoặc vận chuyển, đồng thời hỗ trợ giao tiếp bất đồng bộ để xử lý lưu lượng lớn.

1. Triển khai độc lập

Mỗi microservice có thể được triển khai riêng lẻ mà không ảnh hưởng đến các dịch vụ khác, hỗ trợ quy trình CI/CD trong thương mại điện tử. Ví dụ, cập nhật giao diện danh mục sản phẩm không làm gián đoạn dịch vụ xử lý đơn hàng. Điều này giúp các nền tảng nhanh chóng tung ra tính năng mới, sửa lỗi kịp thời và duy trì tính cạnh tranh trong môi trường kinh doanh trực tuyến.

1. Quản lý dữ liệu phân tán

Mỗi microservice thường có cơ sở dữ liệu riêng, giảm xung đột dữ liệu và tăng hiệu suất trong thương mại điện tử. Ví dụ, dịch vụ sản phẩm sử dụng MongoDB để xử lý dữ liệu phi cấu trúc, trong khi dịch vụ giao dịch dùng PostgreSQL để đảm bảo tính toàn vẹn. Tính năng này cho phép tối ưu hóa dữ liệu theo chức năng, nhưng đòi hỏi cơ chế đồng bộ hóa để tránh bất đồng bộ.

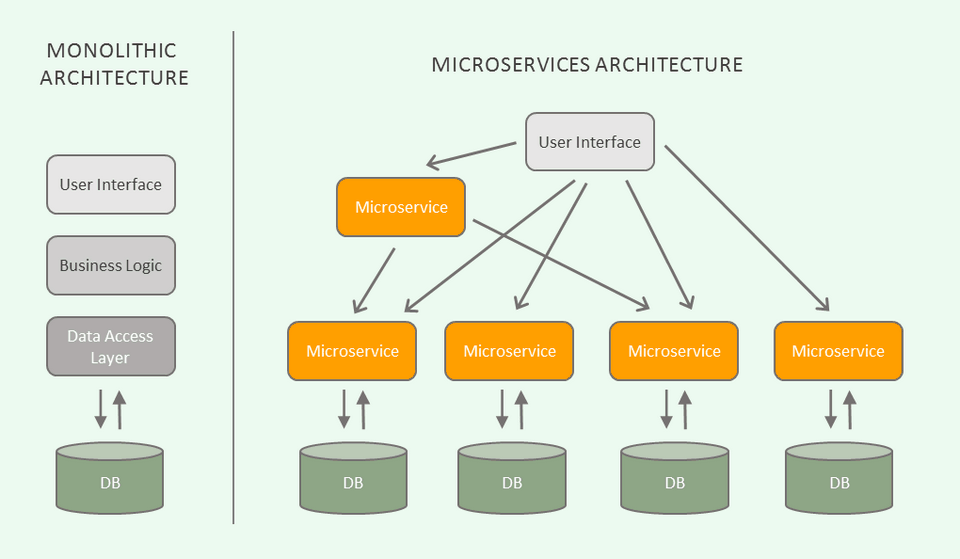
1. Khả năng phục hồi

Microservices đảm bảo nếu một dịch vụ gặp sự cố, các dịch vụ khác vẫn hoạt động, tăng độ tin cậy cho nền tảng thương mại điện tử. Chẳng hạn, nếu dịch vụ đánh giá sản phẩm bị lỗi, người dùng vẫn có thể đặt hàng và thanh toán. Các cơ chế như circuit breaker giúp giảm thiểu rủi ro, đảm bảo trải nghiệm người dùng không bị gián đoạn trong các tình huống tải cao.

1. So sánh kiến trúc Microservices và kiến trúc nguyên khối (Monolithic)

Chương này sẽ so sánh hai kiến trúc phần mềm phổ biến là Microservices và

Monolithic architecture. Đây là hai mô hình phát triển ứng dụng có ảnh hưởng đáng kể đến cách thiết kế, triển khai, và quản lý các hệ thống phần mềm.



1. So sánh microservice với monolithic

|  |  |  |
| --- | --- | --- |
| Tiêu chí | Monolithic Architecture | Microservices Architecture |
| Định nghĩa | Tất cả các thành phần của ứng dụng (frontend, backend, database) được phát triển và triển khai như một khối duy nhất. | Ứng dụng được chia thành nhiều dịch vụ nhỏ, mỗi dịch vụ đảm nhiệm một chức năng riêng biệt và giao tiếp qua API. |
| Scalability và Maintenance | Khó mở rộng và bảo trì khi ứng dụng lớn dần. Mọi thay đổi đều cần build và deploy lại toàn bộ ứng dụng, gây downtime. | Dễ dàng mở rộng và bảo trì từng dịch vụ riêng biệt, giảm downtime nhưng cần công cụ và kỹ năng quản lý phức tạp. |
| Độ phức tạp và Chi phí phát triển | Ban đầu đơn giản, phát triển nhanh. Tuy nhiên khi mở rộng sẽ tăng độ phức tạp và chi phí. | Ban đầu phức tạp và tốn kém hơn do cần hạ tầng phức tạp, nhưng hiệu quả hơn về lâu dài trong bảo trì và mở rộng. |
| Ứng dụng phù hợp | Phù hợp với dự án nhỏ đến trung bình, ít yêu cầu về mở rộng. | Phù hợp với hệ thống lớn, có yêu cầu cao về mở rộng và linh hoạt, nhất là trong môi trường phát triển phức tạp. |

1. So sánh giữa **Monolithic và Microservice**
2. Những khó khăn khi triển khai Microservices
3. Sự phức tạp trong quản lý và điều hành:

Khi chia nhỏ ứng dụng thành nhiều service, có thể dẫn đến sự phức tạp về quản

lý và điều hành. Số lượng service tăng lên và có thể trở nên khó kiểm soát, đặc biệt khi phải xử lý các sự phụ thuộc giữa các service.

1. Tăng chi phí

Với kiến trúc Microservices, giao tiếp giữa các service phải thông qua mạng, điều này có thể tăng chi phí về hạ tầng như thuê dịch vụ Cloud,…. và ảnh hưởng đến hiệu suất của hệ thống cũng như xuất hiện nhiều rủi ro tiềm tàng về mạng. Nếu chuyển đổi từ mô hình Monolithic sang Microservices thì chi phí sẽ cao.

1. Triển khai phức tạp hơn

Triển khai và quản lý các service đòi hỏi sự phối hợp và quản lý khá phức tạp và phải đảm bảo rằng các service hoạt động một cách đồng nhất và tương tác tốt với

nhau.

1. Kết luận về microservices

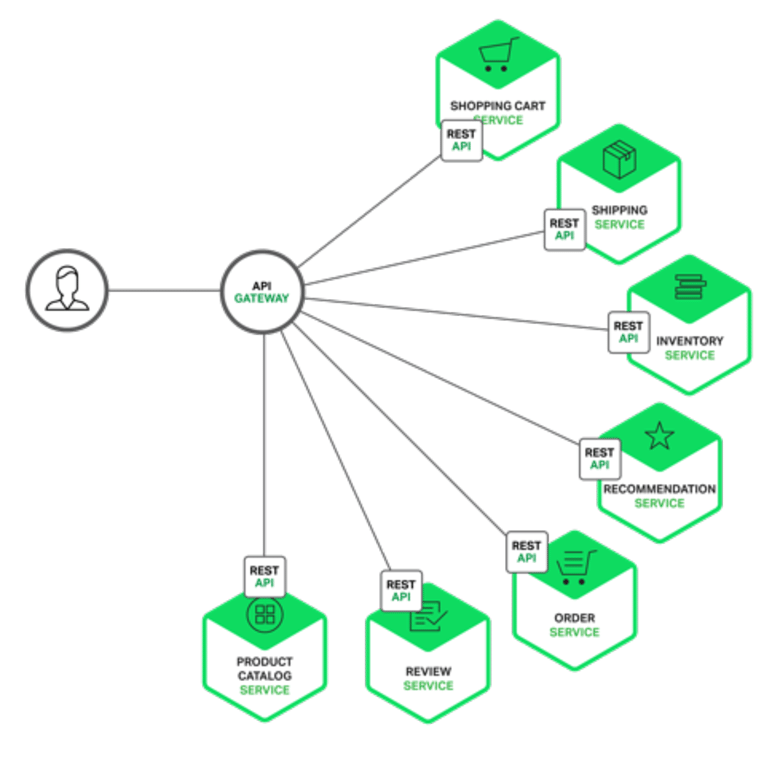
Kiến trúc Microservices là một mô hình phát triển phần mềm hiện đại, chia ứng dụng thành các dịch vụ nhỏ, độc lập, mỗi dịch vụ đảm nhiệm một chức năng cụ thể và giao tiếp qua API. Mô hình này mang lại lợi thế vượt trội về khả năng mở rộng và bảo trì, cho phép cập nhật hoặc mở rộng từng dịch vụ riêng lẻ, giảm downtime và tăng tính linh hoạt, đặc biệt phù hợp với các hệ thống lớn và môi trường phát triển phức tạp. Tuy nhiên, Microservices cũng đối mặt với thách thức lớn, bao gồm sự phức tạp trong quản lý nhiều dịch vụ, chi phí hạ tầng cao do giao tiếp qua mạng, và yêu cầu triển khai đồng bộ để đảm bảo hiệu suất. Việc chuyển đổi từ kiến trúc Monolithic sang Microservices đòi hỏi đầu tư lớn về thời gian, chi phí và kỹ năng kỹ thuật. Do đó, Microservices là lựa chọn tối ưu khi dự án yêu cầu quy mô lớn và tính linh hoạt cao, nhưng cần được cân nhắc kỹ lưỡng dựa trên nguồn lực và mục tiêu của tổ chức.

1. API Gateway, RESTful API

Trong kiến trúc microservices, các thành phần như API Gateway, REST API và Spring Cloud Gateway đóng vai trò quan trọng trong việc đảm bảo giao tiếp hiệu quả và an toàn giữa người dùng và các dịch vụ. API Gateway được ví như “người gác cổng” – là điểm truy cập duy nhất cho toàn bộ hệ thống, chịu trách nhiệm tiếp nhận, định tuyến các yêu cầu từ phía client đến đúng microservice tương ứng. Ngoài ra, nó còn thực hiện các chức năng như xác thực, phân quyền, giới hạn tốc độ truy cập và ghi log, giúp giảm tải cho các dịch vụ bên trong và nâng cao khả năng quản lý hệ thống. REST API đóng vai trò là chuẩn giao tiếp giữa các dịch vụ, sử dụng giao thức HTTP và hỗ trợ đầy đủ các thao tác CRUD (Create, Read, Update, Delete), giúp các microservice trao đổi dữ liệu một cách nhất quán và dễ dàng mở rộng. Trong hệ sinh thái Spring, Spring Cloud Gateway là một lựa chọn phổ biến để triển khai API Gateway, cho phép xử lý các yêu cầu một cách phi đồng bộ, định tuyến linh hoạt và tích hợp sâu với các dịch vụ Spring Boot, từ đó hỗ trợ xây dựng hệ thống microservices hiệu quả, hiện đại và dễ bảo trì.

1. API Gateway

API Gateway là một thành phần trung gian trong kiến trúc hệ thống, đặc biệt phổ biến trong mô hình Microservices, hoạt động như một điểm vào duy nhất (single entry point) cho các yêu cầu từ client đến các dịch vụ backend. Nó quản lý, định tuyến, và xử lý các yêu cầu API, đồng thời cung cấp các chức năng như xác thực, phân quyền, giám sát, và cân bằng tải. API Gateway đóng vai trò như một "người gác cổng", che giấu sự phức tạp của hệ thống backend và đảm bảo giao tiếp mượt mà giữa client và các dịch vụ.



1. API Gateway kết nối tới các service
2. Ưu điểm của API Gateway

API Gateway mang lại nhiều lợi ích vượt trội trong kiến trúc hệ thống, đặc biệt trong mô hình Microservices. Nó tăng cường bảo mật thông qua các cơ chế xác thực và phân quyền, chẳng hạn như sử dụng OAuth hoặc JWT để bảo vệ các dịch vụ backend khỏi truy cập trái phép, như trong trường hợp AWS API Gateway kiểm soát truy cập vào các API của Amazon. Đồng thời, API Gateway đơn giản hóa giao tiếp bằng cách cung cấp một điểm vào duy nhất cho client, giảm độ phức tạp khi tương tác với nhiều dịch vụ, ví dụ như Netflix sử dụng API Gateway để xử lý yêu cầu từ ứng dụng mobile đến hàng trăm microservices. Về hiệu suất, nó hỗ trợ cân bằng tải bằng cách phân phối yêu cầu đến các dịch vụ backend, như Kong Gateway phân bổ lưu lượng đến các server khác nhau, tối ưu hóa hiệu suất và đảm bảo khả năng mở rộng. Ngoài ra, API Gateway cung cấp khả năng giám sát lưu lượng truy cập, ghi log và phân tích hiệu suất, chẳng hạn như Apigee của Google giúp phát hiện sự cố qua dashboard thời gian thực. Cuối cùng, nó tăng tính linh hoạt thông qua hỗ trợ chuyển đổi giao thức và định dạng dữ liệu, như AWS API Gateway chuyển đổi JSON sang XML để tích hợp với các dịch vụ bên thứ ba.

1. Nhược điểm của API Gateway

Mặc dù API Gateway mang lại nhiều lợi ích, nó cũng tồn tại một số nhược điểm đáng lưu ý. API Gateway có thể trở thành điểm lỗi tập trung, khiến toàn bộ hệ thống bị ảnh hưởng nếu xảy ra sự cố, như khi AWS API Gateway gặp lỗi khiến các dịch vụ phụ thuộc tạm dừng hoạt động, đòi hỏi thiết kế dự phòng cẩn thận như sử dụng nhiều vùng (regions) trên đám mây. Việc thiết lập và quản lý API Gateway yêu cầu kỹ năng kỹ thuật cao, chẳng hạn như cấu hình Kong Gateway để tích hợp hàng chục microservices, làm tăng độ phức tạp trong vận hành. Bên cạnh đó, chi phí hạ tầng có thể tăng do triển khai và duy trì API Gateway trên nền tảng đám mây, như chi phí sử dụng AWS API Gateway tăng tỷ lệ với lưu lượng yêu cầu. Ngoài ra, việc thêm một lớp trung gian có thể gây ra độ trễ nhỏ, chẳng hạn như khi API Gateway xử lý yêu cầu phức tạp từ Netflix, làm tăng thời gian phản hồi nếu không được tối ưu hóa. Cuối cùng, quản lý cấu hình và cập nhật API Gateway cho nhiều dịch vụ backend trở nên khó khăn khi hệ thống mở rộng, như khi Apigee phải đồng bộ cấu hình cho hàng trăm API, đòi hỏi nỗ lực bảo trì đáng kể.

1. Kết luận về API Gateway

Tóm lại, API Gateway là một thành phần quan trọng trong kiến trúc hệ thống hiện đại, đặc biệt trong mô hình Microservices, đóng vai trò như một điểm vào duy nhất để quản lý và định tuyến yêu cầu từ client đến các dịch vụ backend. Với khả năng tăng cường bảo mật, đơn giản hóa giao tiếp, cân bằng tải, giám sát hiệu suất, và hỗ trợ tích hợp linh hoạt, API Gateway mang lại lợi ích đáng kể trong việc xây dựng các hệ thống phân tán quy mô lớn. Tuy nhiên, nó cũng đi kèm với các thách thức như nguy cơ trở thành điểm lỗi tập trung, độ phức tạp trong quản lý, chi phí vận hành cao, độ trễ tiềm tàng, và khó khăn trong bảo trì khi hệ thống mở rộng. Do đó, việc triển khai API Gateway cần được cân nhắc kỹ lưỡng, đảm bảo thiết kế dự phòng và tối ưu hóa phù hợp với yêu cầu kỹ thuật và nguồn lực của tổ chức, nhằm tận dụng tối đa lợi ích và giảm thiểu rủi ro.

1. RESTful API

RESTful API (Representational State Transfer) là một chuẩn kiến trúc phần mềm được giới thiệu lần đầu tiên bởi Roy Fielding trong luận án tiến sĩ của ông vào năm 2000. REST cung cấp một cách tiếp cận đơn giản và hiệu quả để thiết kế các API web, dựa trên giao thức HTTP và mô hình tài nguyên. Một API được coi là RESTful nếu tuân thủ đầy đủ các nguyên tắc của REST, bao gồm: sử dụng phương thức HTTP tiêu chuẩn (GET, POST, PUT, DELETE), định danh tài nguyên qua URI, không lưu trạng thái (stateless), khả năng phân tầng (layered system), và sử dụng định dạng dữ liệu chuẩn như JSON hoặc XML.

RESTful API ngày càng trở thành tiêu chuẩn phổ biến trong các hệ thống phân tán và đặc biệt phù hợp với kiến trúc microservices, nơi các dịch vụ nhỏ cần giao tiếp hiệu quả và độc lập với nhau thông qua mạng.



1. Mô tả cơ bản về cách RESTful API hoạt động
2. Ưu điểm của RESTful API

RESTful API mang lại nhiều lợi ích vượt trội trong việc xây dựng hệ thống phân tán. Thứ nhất, nó đơn giản và dễ sử dụng nhờ tận dụng các phương thức HTTP quen thuộc, giúp lập trình viên dễ dàng hiểu và triển khai. Ví dụ, một dịch vụ REST sử dụng GET /products/1 để truy xuất thông tin sản phẩm có ID là 1, điều này rất trực quan và dễ hiểu.

Thứ hai, REST hỗ trợ tính mở rộng và tương thích cao vì không yêu cầu lưu trạng thái phía server, nhờ đó các request độc lập và có thể được xử lý bởi nhiều node trong môi trường phân tán. Ví dụ, trong các hệ thống như Amazon hoặc Facebook, RESTful API giúp tăng khả năng mở rộng và phân phối tải hiệu quả giữa các server.

Thứ ba, REST sử dụng định dạng dữ liệu phổ biến như JSON, thuận tiện cho việc tích hợp với các frontend framework hiện đại như React, Angular hoặc Vue. Ngoài ra, RESTful API hỗ trợ caching, giúp cải thiện hiệu suất truy cập tài nguyên nhờ khả năng lưu trữ các response hợp lệ tại phía client hoặc proxy.

Cuối cùng, REST không bị ràng buộc với ngôn ngữ lập trình hoặc nền tảng, tạo điều kiện dễ dàng để xây dựng hệ thống đa nền tảng (cross-platform). Ví dụ, một REST API có thể được xây dựng bằng Java và tiêu thụ bởi một ứng dụng di động viết bằng Swift hoặc Kotlin.

1. Nhược điểm của RESTful API

Mặc dù RESTful API phổ biến và dễ triển khai, nó cũng tồn tại một số hạn chế cần lưu ý. Do REST là một giao thức stateless, nên mỗi request từ client đều phải chứa đủ thông tin để server xử lý, điều này có thể làm tăng kích thước và độ trễ của mỗi yêu cầu, đặc biệt trong các hệ thống cần duy trì trạng thái phiên (session).

Ngoài ra, REST không thích hợp cho các ứng dụng cần giao tiếp theo thời gian thực hoặc đẩy dữ liệu liên tục từ server đến client, như ứng dụng chat hoặc game. Trong những trường hợp đó, các giải pháp như WebSocket hoặc GraphQL có thể phù hợp hơn.

Thêm vào đó, REST không có chuẩn chính thức cho việc mô tả API như các dịch vụ SOAP (với WSDL), mặc dù các công cụ như Swagger/OpenAPI hiện nay đã phần nào giải quyết vấn đề này. Việc quản lý phiên bản API (API versioning) và kiểm soát truy cập (authentication/authorization) cũng cần được thiết kế thủ công trong REST.

Cuối cùng, REST hoạt động tốt trong các hệ thống đơn giản hoặc trung bình, nhưng khi số lượng endpoint và mức độ tương tác giữa các service tăng lên, việc duy trì và mở rộng REST API có thể trở nên khó khăn, đặc biệt khi thiếu tài liệu hoặc quy ước rõ ràng.

1. So sánh RESTful và SOAP

SOAP (Simple Object Access Protocol) là một giao thức dựa trên XML được sử dụng để trao đổi thông điệp giữa các ứng dụng qua mạng. Dưới đây là bảng so sánh chi tiết giữa REST và SOAP để có thấy được sự khác biệt giữa 2 giao thức này

|  |  |  |
| --- | --- | --- |
| Tiêu chí | RESTful API | SOAP API |
| Giao thức sử dụng | HTTP, HTTPS | HTTP, HTTPS, SMTP, TCP |
| Định dạng dữ liệu | JSON, XML, YAML, v.v. | XML |
| Khả năng mở rộng | Dễ dàng mở rộng do cấu trúc linh hoạt | Khó mở rộng hơn vì tuân theo cấu trúc nghiêm ngặt của chuẩn SOAP |
| Khả năng caching | Hỗ trợ tốt caching nhờ đặc điểm stateless và HTTP caching | Không hỗ trợ caching hiệu quả |
| Bảo mật | Sử dụng OAuth, JWT, HTTPS hoặc xác thực HTTP | Hỗ trợ bảo mật mạnh mẽ hơn với WS-Security, SSL, HTTPS |
| Trạng thái phiên (State) | Stateless – không lưu trạng thái phía server | Có thể Stateless hoặc Stateful |
| Khả năng tương thích | Tương thích với hầu hết các ngôn ngữ lập trình và công nghệ hiện đại | Hạn chế hơn, yêu cầu thư viện SOAP cụ thể |
| Độ phức tạp khi triển khai | Đơn giản, dễ sử dụng và học tập | Phức tạp, yêu cầu hiểu biết sâu về XML và các chuẩn SOAP |
| Tiêu chuẩn | Không tuân theo tiêu chuẩn cụ thể | Tuân theo các tiêu chuẩn WSDL, XSD |
| Ứng dụng phổ biến | Web APIs, hệ thống Microservices, ứng dụng Mobile hiện đại | Các hệ thống nghiệp vụ quy mô lớn như ngân hàng, bảo hiểm, dịch vụ tài chính |

1. So sánh RESTful API và SOAP API
2. Kết luận về RESTful API

Tổng kết lại, RESTful API là một chuẩn kiến trúc giao tiếp hiệu quả, đơn giản và dễ triển khai, phù hợp với các hệ thống microservices và ứng dụng web hiện đại. Nhờ sử dụng các giao thức HTTP phổ biến, REST hỗ trợ mạnh mẽ việc tích hợp giữa các thành phần khác nhau trong hệ thống, đồng thời dễ dàng mở rộng và bảo trì. Tuy nhiên, nó cũng đòi hỏi thiết kế cẩn thận để xử lý các vấn đề như lưu trạng thái, realtime, và bảo mật. Do đó, RESTful API nên được áp dụng trong các bối cảnh phù hợp, và có thể kết hợp với các công nghệ khác như GraphQL hoặc WebSocket khi cần đáp ứng yêu cầu cao hơn về tính tương tác hoặc thời gian thực.

1. Spring Boot

Spring Boot là một framework Java mã nguồn mở được phát triển bởi Pivotal Software (nay thuộc VMware), ra đời vào tháng 4 năm 2014, và được xây dựng trên nền tảng Spring Framework. Mục tiêu chính của Spring Boot là đơn giản hóa quá trình cấu hình và phát triển ứng dụng backend, đặc biệt là các hệ thống phân tán như kiến trúc Microservice. Trước khi Spring Boot ra đời, việc cấu hình các ứng dụng Spring truyền thống khá phức tạp, đòi hỏi lập trình viên phải cấu hình thủ công nhiều thành phần như datasource, web server, security,... Với triết lý “convention over configuration” và “auto-configuration”, Spring Boot giúp loại bỏ phần lớn cấu hình dư thừa, cung cấp các thiết lập mặc định thông minh để lập trình viên có thể tập trung vào logic nghiệp vụ thay vì xử lý cấu hình. Ngoài ra, Spring Boot tích hợp sẵn các máy chủ web nhúng như Tomcat, Jetty, cho phép triển khai ứng dụng một cách độc lập mà không cần cài đặt thêm máy chủ ngoài. Nhờ vào sự tiện lợi, khả năng mở rộng, dễ tích hợp và cộng đồng hỗ trợ mạnh mẽ, Spring Boot đã nhanh chóng trở thành lựa chọn hàng đầu trong việc xây dựng các ứng dụng Java hiện đại.

1. Ưu điểm của Spring Boot

Spring Boot mang lại nhiều ưu điểm nổi bật trong phát triển ứng dụng Java hiện đại. Trước hết, nó giúp đơn giản hóa quá trình cấu hình bằng cách cung cấp các thiết lập mặc định thông minh (auto-configuration), cho phép lập trình viên triển khai ứng dụng mà không cần cấu hình thủ công phức tạp. Spring Boot hỗ trợ tích hợp máy chủ web nhúng như Tomcat, Jetty hoặc Undertow, giúp ứng dụng có thể chạy độc lập chỉ với một file JAR. Bên cạnh đó, Spring Boot tương thích chặt chẽ với hệ sinh thái Spring, dễ dàng mở rộng và tích hợp với các công nghệ khác như JPA, Security, WebSocket, RESTful API, v.v. Ngoài ra, nó còn hỗ trợ tốt cho kiến trúc microservice, đi kèm với hệ thống quản lý phụ thuộc mạnh mẽ (Maven, Gradle), công cụ giám sát (Actuator), và khả năng kiểm thử mạnh, giúp tăng tốc độ phát triển, triển khai và bảo trì ứng dụng hiệu quả hơn.

1. Lợi ích của Spring Boot trong xây dựng ứng dụng kiến trúc Microservice
2. Modularity (Tính Module)

Spring Boot khuyến khích tổ chức mã nguồn theo cấu trúc mô-đun rõ ràng, phân tách thành các thành phần như Controller, Service, Repository,... Điều này giúp quản lý mã nguồn hiệu quả, dễ dàng phân chia hệ thống thành các microservice độc lập, tăng khả năng bảo trì và mở rộng.

1. Dependency Injection (DI)

Spring Boot tận dụng cơ chế Dependency Injection mạnh mẽ của Spring, sử dụng các annotation như @Autowired, @Component, @Service để quản lý phụ thuộc. Cơ chế này giảm thiểu sự kết dính giữa các thành phần, tăng tính tái sử dụng và đơn giản hóa việc kiểm thử, rất phù hợp với kiến trúc microservice.

1. Scalability (Khả năng mở rộng)

Spring Boot hỗ trợ xây dựng các ứng dụng microservice có khả năng mở rộng cao. Khi kết hợp với Spring Cloud, việc triển khai, quản lý và mở rộng từng service độc lập trở nên dễ dàng, đáp ứng nhu cầu phát triển nhanh chóng của hệ thống.

1. Flexibility (Tính linh hoạt)

Spring Boot cung cấp khả năng tích hợp linh hoạt với nhiều công nghệ backend như JPA, MongoDB, Kafka, RabbitMQ,... Điều này cho phép các nhà phát triển tùy chỉnh hệ thống theo yêu cầu cụ thể của dự án, đảm bảo tính đa dạng và hiệu quả.

1. Dễ kiểm thử

Với các công cụ như @SpringBootTest, @WebMvcTest và khả năng tích hợp tốt với JUnit/TestNG, Spring Boot hỗ trợ kiểm thử từng microservice một cách hiệu quả. Điều này giúp đảm bảo chất lượng mã nguồn và giảm thiểu lỗi trong quá trình phát triển.

1. Các thành phần nền tảng của Spring Boot
2. Controller

Controller là lớp chịu trách nhiệm xử lý các yêu cầu HTTP từ phía client và trả về phản hồi tương ứng. Được đánh dấu bằng annotation @RestController, Controller định nghĩa các endpoint (đường dẫn) thông qua các annotation như @GetMapping, @PostMapping, @PutMapping, hoặc @DeleteMapping. Mỗi phương thức trong Controller xử lý một loại yêu cầu cụ thể, gọi đến các lớp Service để thực hiện logic nghiệp vụ và trả về dữ liệu hoặc trạng thái phù hợp. Cấu trúc này tách biệt rõ ràng tầng giao tiếp (presentation) và logic nghiệp vụ, giúp mã nguồn dễ bảo trì, mở rộng và kiểm thử.

1. Service

Service là tầng trung gian chứa logic nghiệp vụ cốt lõi của ứng dụng. Các lớp Service, được đánh dấu bằng annotation @Service, đảm nhận việc xử lý các quy trình nghiệp vụ, phối hợp giữa các thành phần như Controller và Repository. Service nhận dữ liệu từ Controller, thực hiện các tác vụ cần thiết (tính toán, xử lý logic, gọi Repository), và trả kết quả về Controller. Tính mô-đun của Service giúp mã nguồn dễ tái sử dụng, dễ kiểm thử và giảm sự phụ thuộc giữa các thành phần.

1. Repository

Repository là tầng tương tác trực tiếp với cơ sở dữ liệu, chịu trách nhiệm quản lý dữ liệu (CRUD: Create, Read, Update, Delete). Được đánh dấu bằng annotation @Repository, các lớp Repository thường sử dụng Spring Data JPA để đơn giản hóa việc truy vấn và thao tác với cơ sở dữ liệu. Repository cung cấp các phương thức trừu tượng để truy xuất dữ liệu mà không cần viết truy vấn SQL thủ công, giúp tăng hiệu suất phát triển và đảm bảo tính nhất quán. Tầng này tách biệt logic dữ liệu khỏi logic nghiệp vụ, hỗ trợ bảo trì và mở rộng dễ dàng.

1. Middleware (filter và interceptor)

Middleware trong Spring Boot, bao gồm Filter và Interceptor, được sử dụng để xử lý các yêu cầu trước hoặc sau khi chúng đến Controller.

Filter: Hoạt động ở tầng servlet, Filter xử lý các tác vụ như xác thực, logging, hoặc nén dữ liệu trước khi yêu cầu được chuyển đến Controller hoặc sau khi phản hồi được gửi về client. Filter được cấu hình thông qua @WebFilter hoặc các bean trong Spring Boot.

Interceptor: Hoạt động ở tầng ứng dụng (Spring context), Interceptor can thiệp vào quá trình xử lý yêu cầu trước/sau khi Controller được gọi, thường dùng cho các tác vụ như kiểm tra quyền truy cập, logging, hoặc xử lý lỗi. Interceptor được định nghĩa bằng cách implement HandlerInterceptor hoặc sử dụng @Component với cấu hình phù hợp.  
Middleware giúp tăng tính linh hoạt, bảo mật và khả năng tùy chỉnh của ứng dụng.

1. Cơ chế bảo mật của Spring Boot

Trong bối cảnh phát triển ứng dụng hiện đại, đặc biệt là kiến trúc microservice, bảo mật là một yếu tố then chốt để đảm bảo an toàn dữ liệu và quyền truy cập. Spring Boot, với sự hỗ trợ của Spring Security, cung cấp một hệ thống bảo mật mạnh mẽ, linh hoạt và dễ tùy chỉnh. Trong đó, xác thực (Authentication) đóng vai trò cốt lõi, cho phép xác minh danh tính người dùng hoặc hệ thống trước khi cấp quyền truy cập vào các tài nguyên của ứng dụng. Phần này sẽ phân tích cơ chế xác thực của Spring Boot, các phương thức triển khai, và vai trò của nó trong việc xây dựng các ứng dụng an toàn.

1. Authentication (Xác thực)

Xác thực là quá trình kiểm tra danh tính của người dùng hoặc thực thể truy cập ứng dụng, đảm bảo rằng chỉ những người dùng hợp lệ mới có thể sử dụng các tài nguyên được bảo vệ. Spring Security, tích hợp sẵn trong Spring Boot, cung cấp một framework bảo mật toàn diện, cho phép cấu hình xác thực thông qua các annotation như @EnableWebSecurity và các class cấu hình như SecurityFilterChain. Cơ chế này hỗ trợ nhiều phương thức xác thực, từ cơ bản đến hiện đại, phù hợp với các yêu cầu đa dạng của ứng dụng microservice.

1. Các phương thức xác thực

Spring Boot hỗ trợ nhiều phương thức xác thực, giúp các nhà phát triển linh hoạt lựa chọn tùy theo nhu cầu dự án.

1. Xác thực dựa trên biểu mẫu (Form-based Authentication)

Xác thực dựa trên biểu mẫu là phương thức phổ biến trong các ứng dụng web, cho phép người dùng đăng nhập thông qua giao diện với thông tin username và password. Trong bối cảnh API, Spring MVC sử dụng các Controller (được đánh dấu bằng @RestController) để xử lý yêu cầu đăng nhập, trong khi Spring Security đảm bảo xác thực an toàn. Phương thức này thường được dùng trong các ứng dụng web kết hợp API, nơi cần cung cấp giao diện đăng nhập. Đoạn mã dưới đây minh họa cách cấu hình xác thực dựa trên biểu mẫu:

1. OAuth 2.0 và OpenID Connect

Spring Boot hỗ trợ xác thực thông qua các nhà cung cấp bên thứ ba (như Google, GitHub) hoặc hệ thống Single Sign-On (SSO) thông qua OAuth 2.0 và OpenID Connect. Trong Spring Boot API, các endpoint OAuth được định nghĩa bằng @RestController để xử lý callback từ nhà cung cấp. Phương thức này đặc biệt hữu ích trong các hệ thống phân tán, nơi người dùng chỉ cần đăng nhập một lần để truy cập nhiều microservice. Thư viện spring-security-oauth2-client giúp đơn giản hóa việc tích hợp.



1. Đoạn mã cấu hình xác thực OAuth 2.0 với Google

Đoạn mã trên cấu hình xác thực OAuth 2.0 với Google, cho phép người dùng đăng nhập thông qua tài khoản Google và chuyển hướng đến /api/home sau khi đăng nhập thành công.

1. JSON Web Token (JWT)

Trong kiến trúc microservice, JWT là lựa chọn phổ biến để xác thực không trạng thái (stateless). Sau khi người dùng đăng nhập, server tạo một token JWT chứa thông tin xác thực và gửi về client. Client sử dụng token này trong các yêu cầu tiếp theo. Spring Security cung cấp các bộ lọc để xác minh token, như sau:



1. Đoạn mã cấu hình Spring Security để xác minh JWT

Đoạn mã trên cấu hình Spring Security để xác minh JWT trong các yêu cầu API, đảm bảo bảo mật cho các endpoint trong ứng dụng microservice.

1. Basic Authentication

Basic Authentication là phương thức xác thực đơn giản, sử dụng header HTTP (Authorization: Basic ) để gửi thông tin username và password. Trong Spring Boot API, các endpoint RESTful được bảo vệ bằng Spring Security, và phương thức này thường được sử dụng với HTTPS để đảm bảo an toàn, phù hợp cho các API nội bộ hoặc đơn giản.



1. Đoạn mã trên cấu hình xác thực Basic Authentication

Đoạn mã trên cấu hình xác thực Basic Authentication, yêu cầu thông tin xác thực qua header HTTP cho mọi yêu cầu API.

1. Tích hợp cơ sở dữ liệu và mã hóa mật khẩu

Trong Spring Boot API, thông tin người dùng thường được lưu trữ trong cơ sở dữ liệu. Spring Security sử dụng interface UserDetailsService để tải thông tin người dùng từ cơ sở dữ liệu thông qua Spring Data JPA. Để bảo vệ mật khẩu, Spring Boot tích hợp BCryptPasswordEncoder, mã hóa mật khẩu trước khi lưu trữ để chống lại các cuộc tấn công như brute force. Các endpoint như /api/register có thể được định nghĩa để hỗ trợ đăng ký người dùng.

1. Khả năng hỗ trợ mạnh mẽ cho kiến trúc Microservice

Spring Boot cung cấp khả năng hỗ trợ vượt trội cho kiến trúc microservice, mang lại một hệ sinh thái toàn diện để phát triển và quản lý các dịch vụ nhỏ, độc lập. Hầu hết các khái niệm cốt lõi của Spring Boot như dependency injection, annotations (@RestController, @Bean), filters, và interceptors đều được áp dụng hiệu quả trong việc xây dựng ứng dụng microservice. Về cơ bản, Spring Boot tích hợp mạnh mẽ các công nghệ phổ biến trong microservice như RESTful APIs, message queuing (RabbitMQ, Kafka), và hệ thống publish/subscribe phân tán (Redis), giúp các nhà phát triển dễ dàng triển khai các hệ thống phân tán an toàn và hiệu quả.

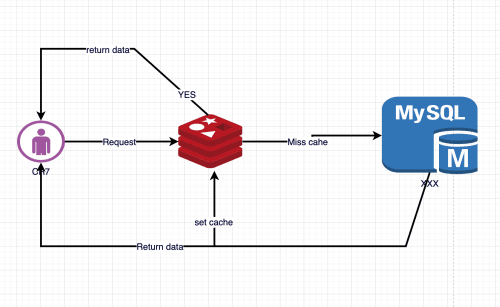
Trong Spring Boot, các microservice giao tiếp với nhau thông qua nhiều phương thức, chẳng hạn như HTTP/REST hoặc message-based communication sử dụng RabbitMQ hoặc Kafka. Spring Cloud, một phần mở rộng của Spring Boot, cung cấp các công cụ như Spring Cloud Gateway để định tuyến yêu cầu, Spring Cloud Netflix Eureka để khám phá dịch vụ, và Spring Cloud Config để quản lý cấu hình tập trung. Những tính năng này giúp Spring Boot không chỉ đơn giản hóa việc phát triển microservice mà còn đảm bảo khả năng mở rộng và quản lý dễ dàng trong môi trường phân tán.

1. Redis

Trong các hệ thống phân tán như kiến trúc microservice, việc quản lý dữ liệu nhanh chóng, hiệu quả và đáng tin cậy là yếu tố quyết định đến hiệu suất và trải nghiệm của ứng dụng. Yêu cầu đặt ra là cần một công cụ lưu trữ dữ liệu có tốc độ truy xuất cao, hỗ trợ giao tiếp bất đồng bộ giữa các dịch vụ, và tích hợp dễ dàng với các framework như Spring Boot. Redis – một cơ sở dữ liệu NoSQL in-memory mã nguồn mở – chính là sự lựa chọn lý tưởng đáp ứng các yêu cầu này. Với khả năng lưu trữ key-value trong RAM, hỗ trợ các cấu trúc dữ liệu đa dạng, và tích hợp mạnh mẽ với Spring Data Redis, Redis không chỉ tối ưu hóa hiệu suất mà còn đơn giản hóa việc phát triển các microservice cần caching, publish/subscribe, hoặc quản lý session.

1. Giới thiệu về redis

Redis (Remote Dictionary Server) là một cơ sở dữ liệu NoSQL mã nguồn mở, được phát triển bởi Salvatore Sanfilippo vào năm 2009, hoạt động như một kho lưu trữ key-value trong bộ nhớ (in-memory) với tốc độ truy xuất cực nhanh. Redis được thiết kế để hỗ trợ các ứng dụng yêu cầu hiệu suất cao, sử dụng các cấu trúc dữ liệu đa dạng như strings, lists, sets, hashes, và sorted sets. Nhờ khả năng lưu trữ dữ liệu trong RAM và hỗ trợ các tính năng như publish/subscribe, caching, và persistence, Redis trở thành một công cụ quan trọng trong các hệ thống phân tán, đặc biệt là kiến trúc microservice, nơi các dịch vụ nhỏ cần giao tiếp nhanh chóng và đáng tin cậy.



1. Minh họa hoạt động cơ bảng của redis

Redis lưu trữ dữ liệu dưới dạng key-value trong bộ nhớ RAM, cho phép truy xuất dữ liệu với độ trễ cực thấp (thường dưới 1ms). Các ứng dụng tương tác với Redis thông qua các lệnh đơn giản như SET, GET, LPUSH, hoặc SUBSCRIBE. Trong Spring Boot, Redis được tích hợp dễ dàng qua thư viện Spring Data Redis, cho phép các microservice sử dụng Redis như một cache, hàng đợi tin nhắn, hoặc cơ chế pub/sub. Ví dụ, một microservice có thể sử dụng Redis để lưu trữ dữ liệu tạm thời của giỏ hàng người dùng hoặc phát hành thông báo đến các dịch vụ khác.

1. ****Ưu điểm của Redis****

Redis mang lại nhiều lợi ích vượt trội trong việc xây dựng hệ thống phân tán.  
Thứ nhất, hiệu suất cao là ưu điểm nổi bật nhờ lưu trữ dữ liệu trong bộ nhớ. Ví dụ, một API Spring Boot sử dụng Redis làm cache để lưu kết quả truy vấn /products, giảm tải cho cơ sở dữ liệu chính và tăng tốc độ phản hồi.  
 Thứ hai, Redis hỗ trợ các cấu trúc dữ liệu phong phú, cho phép xử lý nhiều kịch bản phức tạp như xếp hạng (sorted sets) hoặc danh sách công việc (lists). Điều này rất hữu ích trong microservice, chẳng hạn khi một dịch vụ cần quản lý hàng đợi tác vụ bất đồng bộ.  
 Thứ ba, Redis cung cấp cơ chế publish/subscribe, phù hợp cho giao tiếp thời gian thực giữa các microservice. Ví dụ, một microservice xử lý đơn hàng có thể gửi thông báo qua Redis đến dịch vụ thông báo để cập nhật trạng thái cho người dùng.  
 Thứ tư, Redis hỗ trợ persistence (lưu trữ dữ liệu xuống đĩa), đảm bảo dữ liệu không bị mất hoàn toàn khi server khởi động lại, mặc dù chủ yếu hoạt động trong RAM.  
 Cuối cùng, Redis tích hợp dễ dàng với Spring Boot thông qua Spring Data Redis, cho phép các nhà phát triển cấu hình nhanh chóng trong các hệ thống microservice. Ví dụ, một ứng dụng thương mại điện tử có thể sử dụng Redis để lưu trữ session người dùng, cải thiện trải nghiệm đa nền tảng.

1. Nhược điểm của Redis

Mặc dù Redis rất mạnh mẽ, nó cũng tồn tại một số hạn chế. Do lưu trữ dữ liệu chủ yếu trong RAM, dung lượng lưu trữ của Redis bị giới hạn bởi bộ nhớ vật lý, khiến nó không phù hợp cho các ứng dụng cần lưu trữ lượng lớn dữ liệu lâu dài.  
 Thứ hai, Redis không hỗ trợ truy vấn phức tạp như các cơ sở dữ liệu quan hệ (SQL), vì vậy nó không thể thay thế hoàn toàn các hệ thống như MySQL hoặc PostgreSQL trong microservice.  
 Thứ ba, việc duy trì persistence và replication có thể làm tăng độ phức tạp vận hành, đặc biệt trong các hệ thống phân tán với nhiều node Redis. Ví dụ, cấu hình Redis Sentinel hoặc Redis Cluster đòi hỏi kiến thức chuyên sâu để đảm bảo tính sẵn sàng cao.  
 Cuối cùng, Redis không được tối ưu cho các tác vụ yêu cầu xử lý giao dịch phức tạp (multi-step transactions) như trong ngân hàng, vì cơ chế khóa của nó đơn giản hơn so với các cơ sở dữ liệu truyền thống.

1. So sánh Redis và Memcached

Memcached là một hệ thống caching in-memory khác, thường được so sánh với Redis. Dưới đây là bảng so sánh chi tiết giữa Redis và Memcached:

|  |  |  |
| --- | --- | --- |
| Tiêu chí | Redis | Memcached |
| Loại dữ liệu | Hỗ trợ nhiều cấu trúc (strings, lists, sets, hashes, sorted sets) | Chỉ hỗ trợ key-value đơn giản |
| Hiệu suất | Rất cao, độ trễ dưới 1ms | Cao, nhưng thấp hơn Redis trong các kịch bản phức tạp |
| Persistence | Hỗ trợ lưu trữ xuống đĩa (RDB, AOF) | Không hỗ trợ persistence |
| Publish/Subscribe | Hỗ trợ mạnh mẽ, phù hợp cho giao tiếp thời gian thực | Không hỗ trợ pub/sub |
| Tích hợp Spring Boot | Dễ dàng qua Spring Data Redis | Hỗ trợ cơ bản, ít tích hợp chặt chẽ |
| Khả năng mở rộng | Hỗ trợ cluster và replication (Redis Cluster, Sentinel) | Hỗ trợ phân tán đơn giản, không có cluster chính thức |
| Bảo mật | Hỗ trợ ACL, mã hóa TLS, và xác thực qua mật khẩu | Bảo mật cơ bản, không hỗ trợ ACL phức tạp |
| Ứng dụng phổ biến | Cache, pub/sub, session store, hàng đợi trong microservice | Cache đơn giản trong ứng dụng web |
| Độ phức tạp triển khai | Trung bình, cần cấu hình cho persistence và cluster | Đơn giản, nhưng hạn chế tính năng |

1. So sánh giữa Redis và Memcache
2. Kết luận về Redis

Tổng kết lại, Redis là một công cụ mạnh mẽ và linh hoạt, đặc biệt phù hợp với kiến trúc microservice nhờ hiệu suất cao, hỗ trợ đa dạng cấu trúc dữ liệu, và tích hợp dễ dàng với Spring Boot. Trong các hệ thống phân tán, Redis đóng vai trò quan trọng trong caching, giao tiếp bất đồng bộ, và quản lý session, giúp tăng tốc độ và khả năng mở rộng của các microservice. Tuy nhiên, Redis cần được thiết kế cẩn thận để xử lý các hạn chế về dung lượng bộ nhớ, truy vấn phức tạp, và vận hành cluster. Do đó, Redis nên được sử dụng kết hợp với các cơ sở dữ liệu khác (như PostgreSQL) và công nghệ như RabbitMQ hoặc Kafka khi cần đáp ứng yêu cầu cao hơn về giao tiếp thời gian thực hoặc lưu trữ lâu dài. Trong Spring Boot, việc tích hợp Redis qua Spring Data Redis giúp các nhà phát triển xây dựng các microservice hiệu quả, an toàn, và dễ bảo trì.

1. React

Giao diện sử dụng chính là phần quan trọng nhất, quyết định đến trải nghiệm

người dùng. Việc xây dựng được một giao diện người dùng đẹp mắt, tiện dụng, tốc độ phản hồi nhanh và tối ưu chính là yêu cầu hàng đầu khi xây dựng một website. Vì thế, việc lựa chọn một framework UI để sử dụng rất quan trọng, nhất là framework phải đáp ứng được yêu cầu là có nhiều thư viện UI đẹp mắt và tiện dụng, có thể tối ưu code bằng cách tái sử dụng component, có đầy đủ các công nghệ như lazyloading để tối ưu tốc độ. Với ReactJs – một framework UI nổi tiếng được Facebook tạo ra chính là sự lựa chọn phù hợp dành cho yêu cầu trên.

1. Giới thiệu về React

Redis (Remote Dictionary Server) là một cơ sở dữ liệu NoSQL mã nguồn mở được phát triển bởi Salvatore Sanfilippo vào năm 2009, tập trung vào lưu trữ dữ liệu key-value trong bộ nhớ (in-memory) để đảm bảo hiệu suất cao. Ra mắt lần đầu như một giải pháp caching, Redis nhanh chóng trở thành một trong những công cụ phổ biến nhất trong lĩnh vực phát triển hệ thống phân tán nhờ vào tính hiệu quả và dễ sử dụng.

Một trong những đặc điểm nổi bật của Redis là khả năng lưu trữ dữ liệu trong RAM, cho phép truy xuất với độ trễ cực thấp (thường dưới 1ms). Redis sử dụng các cấu trúc dữ liệu đa dạng như strings, lists, sets, hashes, và sorted sets, giúp xử lý linh hoạt nhiều kịch bản, từ caching đến quản lý hàng đợi. Trong Spring Boot, Redis được tích hợp dễ dàng thông qua Spring Data Redis, cho phép các microservice tận dụng Redis để lưu trữ dữ liệu tạm thời, giao tiếp bất đồng bộ, hoặc quản lý session. Điều này không chỉ cải thiện hiệu suất mà còn mang lại trải nghiệm mượt mà trong các hệ thống phân tán.

Ngoài ra, Redis còn có sự hỗ trợ mạnh mẽ từ cộng đồng lớn và phong phú về tài liệu. Nhiều thư viện và công cụ bổ sung đã được phát triển để mở rộng khả năng của Redis, chẳng hạn như Redis Sentinel cho tính sẵn sàng cao và Redis Cluster cho khả năng mở rộng phân tán. Với những ưu điểm này, Redis đã và đang được sử dụng rộng rãi trong các hệ thống microservice hiện đại, từ các ứng dụng thương mại điện tử đến các nền tảng thời gian thực quy mô lớn.

1. Những đặc trưng của React

React giúp việc phát triển giao diện người dùng trở nên trực quan và hiệu quả hơn. Thay vì thao tác trực tiếp với DOM thật, các nhà phát triển chỉ cần thiết kế các thành phần giao diện (component) cho từng trạng thái của ứng dụng, và React sẽ tự động cập nhật giao diện khi dữ liệu thay đổi, đảm bảo trải nghiệm người dùng mượt mà và tốc độ phản hồi nhanh chóng.

Dựa trên component: React xây dựng giao diện người dùng bằng cách sử dụng các component độc lập, mỗi component quản lý trạng thái (state) và logic riêng của nó. Các component này có thể được tái sử dụng và kết hợp để tạo ra giao diện phức tạp, giúp mã nguồn dễ bảo trì và mở rộng. Vì logic giao diện được viết bằng JavaScript thay vì các template tĩnh, việc truyền dữ liệu giữa các component và tích hợp với backend (như Spring Boot API) trở nên đơn giản.

“Học một lần, viết mọi nơi”: React cho phép nhà phát triển tạo các tính năng mới mà không cần viết lại mã hiện có. React hỗ trợ cả kết xuất phía client và phía server (server-side rendering) thông qua các framework như Next.js, giúp tối ưu hóa hiệu suất và SEO cho các ứng dụng web.

React Hook: Hook là các hàm đặc biệt của React, cho phép tích hợp trạng thái và các tính năng vòng đời vào các functional component. Các hook tích hợp như useState, useEffect, useContext, và useMemo giúp đơn giản hóa việc quản lý trạng thái và xử lý các tác vụ phụ (side effects) mà không cần sử dụng class component. Ví dụ, useEffect có thể được dùng để gọi API từ Spring Boot khi component được render.

Virtual DOM: Virtual DOM là một biểu diễn ảo của DOM thật, được React sử dụng để tối ưu hóa quá trình cập nhật giao diện. Khi trạng thái của một component thay đổi, React tạo một Virtual DOM mới, so sánh với phiên bản trước đó (quá trình reconciliation), và chỉ cập nhật các phần thay đổi lên DOM thật. Điều này giảm thiểu thao tác trực tiếp trên DOM, cải thiện đáng kể hiệu suất của ứng dụng web.

JSX (JavaScript Syntax Extension): JSX là một phần mở rộng cú pháp của JavaScript, cho phép viết mã giao diện tương tự HTML trong các file JavaScript. JSX giúp mã dễ đọc và bảo trì, đồng thời tích hợp chặt chẽ với logic JavaScript. Mặc dù không bắt buộc, JSX là lựa chọn phổ biến để định nghĩa các component trong React.

TSX (TypeScript + JSX): TSX là sự kết hợp giữa JSX và TypeScript, mang lại khả năng kiểm tra kiểu tĩnh trong quá trình phát triển. TSX cho phép viết các component React với cú pháp giống HTML, nhưng được tăng cường bởi các kiểm tra kiểu của TypeScript, giúp giảm lỗi và tăng độ tin cậy. Ví dụ, TSX đảm bảo rằng các props truyền vào component có đúng kiểu dữ liệu, đặc biệt hữu ích khi tích hợp với API Spring Boot trả về JSON.

1. ****Ưu điểm của React****

React mang lại nhiều lợi ích vượt trội trong việc phát triển giao diện người dùng hiện đại.Thứ nhất, hiệu suất cao nhờ cơ chế Virtual DOM, cho phép chỉ cập nhật các phần giao diện thay đổi, giảm thao tác trực tiếp trên DOM thật. Ví dụ, trong một ứng dụng thương mại điện tử, khi người dùng lọc danh sách sản phẩm, React chỉ render lại danh sách thay vì toàn bộ trang, cải thiện tốc độ phản hồi.  
 Thứ hai, kiến trúc component-based giúp chia nhỏ giao diện thành các thành phần độc lập, dễ tái sử dụng và bảo trì. Ví dụ, một component ProductCard có thể được sử dụng ở nhiều trang như trang chủ hoặc giỏ hàng.  
 Thứ ba, cộng đồng lớn và hệ sinh thái phong phú cung cấp nhiều thư viện bổ trợ như React Router (điều hướng), Redux (quản lý trạng thái), và Axios (gọi API). Điều này giúp React dễ dàng tích hợp với Spring Boot API để lấy dữ liệu JSON từ endpoint như /api/products.  
 Thứ tư, hỗ trợ TypeScript qua TSX tăng độ tin cậy bằng cách kiểm tra kiểu tĩnh, đặc biệt hữu ích khi làm việc với dữ liệu phức tạp từ API. Ví dụ, TSX đảm bảo rằng props của component ProductList khớp với cấu trúc JSON từ Spring Boot.  
 Cuối cùng, tính linh hoạt của React cho phép sử dụng trong nhiều kịch bản, từ SPA đến server-side rendering với Next.js, đáp ứng nhu cầu của các hệ thống microservice.

1. ****Nhược điểm của React****

Mặc dù React phổ biến và linh hoạt, nó cũng tồn tại một số hạn chế. Thứ nhất, vì React chỉ là một thư viện (không phải framework đầy đủ), nó yêu cầu thêm thư viện bên ngoài cho các tính năng như điều hướng (React Router) hoặc quản lý trạng thái (Redux), làm tăng độ phức tạp khi thiết lập dự án. Ví dụ, để gọi API Spring Boot và quản lý dữ liệu toàn cục, nhà phát triển cần cấu hình thêm Redux hoặc Context API.  
 Thứ hai, khúc học tập ban đầu của React có thể khó khăn với người mới, đặc biệt với các khái niệm như JSX, Virtual DOM, hoặc React Hook. Ví dụ, việc sử dụng useEffect để gọi API đòi hỏi hiểu về vòng đời component.  
 Thứ ba, React thiếu chuẩn chính thức cho cấu trúc dự án hoặc cách quản lý trạng thái, dẫn đến sự khác biệt giữa các dự án và khó khăn khi bảo trì nếu không có tài liệu rõ ràng.  
 Cuối cùng, React không tối ưu cho SEO nếu không sử dụng server-side rendering (như Next.js), vì nội dung động cần thời gian để render trên client, có thể ảnh hưởng đến thứ hạng tìm kiếm.

1. So sánh React, Angular và Vue.js

Angular (phát triển bởi Google) và Vue.js (phát triển bởi Evan You) là hai framework phổ biến khác để xây dựng giao diện người dùng. Dưới đây là bảng so sánh chi tiết giữa React, Angular, và Vue.js:

|  |  |  |  |
| --- | --- | --- | --- |
| Tiêu chí | React | Angular | Vue.js |
| Loại | Thư viện JavaScript | Framework đầy đủ | Framework tiến bộ (progressive) |
| Hiệu suất | Cao nhờ Virtual DOM | Tốt, nhưng chậm hơn do DOM thật và framework nặng | Cao, tương tự React nhờ Virtual DOM |
| Độ phức tạp | Trung bình, cần thêm thư viện cho tính năng đầy đủ | Cao, yêu cầu hiểu TypeScript và cấu trúc phức tạp | Thấp, dễ học và triển khai nhanh |
| Tích hợp Spring Boot | Dễ dàng qua fetch/Axios, gọi API RESTful | Dễ dàng qua HttpClient, phù hợp với API phức tạp | Dễ dàng qua Axios/fetch, tương tự React |
| Quản lý trạng thái | Redux, Context API (thêm thư viện) | NgRx, dịch vụ tích hợp | Vuex, Pinia (tích hợp sẵn) |
| SEO | Cần Next.js cho server-side rendering | Tốt nhờ server-side rendering tích hợp | Tốt với Nuxt.js cho server-side rendering |
| Cộng đồng | Rất lớn, nhiều thư viện (React Router, Redux) | Lớn, nhưng nhỏ hơn React | Lớn, đang phát triển nhanh |
| Khả năng mở rộng | Linh hoạt, phù hợp SPA và hệ thống nhỏ đến trung bình | Tốt cho hệ thống lớn, phức tạp | Linh hoạt, phù hợp từ dự án nhỏ đến lớn |
| Bảo mật | Dựa vào thư viện bên thứ ba (như Helmet.js) | Tích hợp bảo mật mạnh (XSS, CSRF) | Tích hợp bảo mật cơ bản, cần thêm thư viện |
| Ứng dụng phổ biến | SPA, ứng dụng web (Facebook, Netflix) | Hệ thống doanh nghiệp (Google, Microsoft) | Ứng dụng web nhẹ, SPA (Alibaba, GitLab) |
| Độ phức tạp triển khai | Trung bình, cần cấu hình thêm thư viện | Cao, cần cấu hình nhiều | Thấp, dễ dàng triển khai |

1. So sánh giữa React, Angular và Vuejs
2. ****Kết luận về React****

Tổng kết lại, React là một thư viện mạnh mẽ và linh hoạt để xây dựng giao diện người dùng, đặc biệt phù hợp với các ứng dụng web hiện đại và hệ thống microservice khi tích hợp với Spring Boot API. Nhờ Virtual DOM, kiến trúc component-based, và hỗ trợ TSX, React cung cấp hiệu suất cao và khả năng bảo trì tốt. Tuy nhiên, việc yêu cầu thêm thư viện bên ngoài và khúc học tập ban đầu đòi hỏi nhà phát triển phải thiết kế cẩn thận. So với Angular và Vue.js, React nổi bật nhờ tính linh hoạt và cộng đồng lớn, nhưng có thể cần thêm công cụ để đáp ứng các yêu cầu phức tạp như Angular hoặc sự đơn giản của Vue.js. Do đó, React nên được sử dụng trong các dự án ưu tiên hiệu suất và tích hợp với backend RESTful, đồng thời kết hợp với Next.js hoặc Redux khi cần SEO hoặc quản lý trạng thái phức tạp.

1. GIẢI PHÁP CHO BÀI TOÁN/VẤN ĐỀ/MÔ HÌNH
2. Phát biểu bài toán trong đề tài
3. Bài toán nghiên cứu

Trong bối cảnh thương mại điện tử ngày càng phát triển, các cửa hàng bán nông sản trực tuyến như của đề tài cần một hệ thống hiệu quả để quản lý sản phẩm, người dùng, và đơn hàng, đồng thời đảm bảo khả năng mở rộng linh hoạt và hiệu suất cao, đặc biệt khi số lượng truy cập tăng đột biến, nếu sử dụng kiến trúc monolithic, hệ thống có thể gặp các vấn đề như khả năng mở rộng hạn chế do việc mở rộng một phần sẽ ảnh hưởng toàn bộ hệ thống, gây lãng phí tài nguyên và giảm hiệu suất, triển khai và tích hợp khó khăn khi thêm tính năng mới hoặc cập nhật dịch vụ, dẫn đến rủi ro cao và thời gian ngừng hoạt động, cùng với đó là quản lý và bảo trì phức tạp khi hệ thống ngày càng mở rộng, làm tăng độ khó trong vận hành và cập nhật.

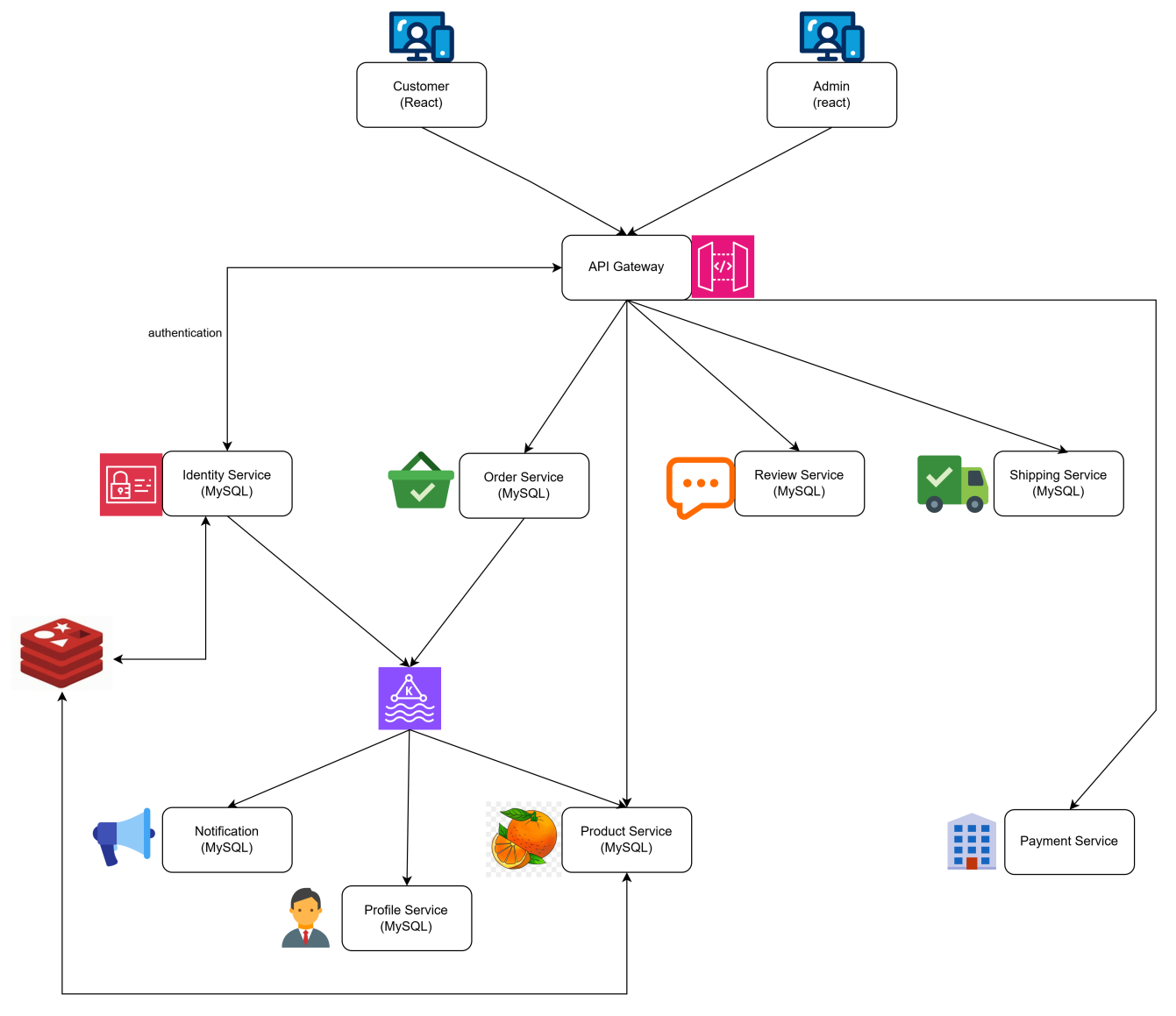
1. Giải pháp nghiên cứu

Để giải quyết các thách thức trên, đề tài xây dựng website bán nông sản trực tuyến dựa trên kiến trúc Microservices nhằm đảm bảo khả năng mở rộng linh hoạt và quản lý hiệu quả, hệ thống được phân tách thành các dịch vụ độc lập như Dịch vụ sản phẩm, Dịch vụ tài khoản, và Dịch vụ đơn hàng, cho phép mở rộng từng phần mà không ảnh hưởng đến các dịch vụ khác, sử dụng React cho frontend để giao diện mượt mà và Spring Boot cho backend để phát triển các dịch vụ nhanh chóng, đồng thời tích hợp Redis làm hệ thống cache nhằm giảm tải cho cơ sở dữ liệu MySQL và tăng tốc độ truy xuất dữ liệu, đảm bảo hiệu suất cao ngay cả khi số lượng người dùng tăng đột biến, giải pháp này giúp cửa hàng dễ dàng triển khai tính năng mới, bảo trì hệ thống hiệu quả, và duy trì chi phí vận hành ở mức thấp.

1. Kết luận

Đề tài chứng minh rằng việc áp dụng kiến trúc Microservices kết hợp với React, Spring Boot, và Redis là giải pháp phù hợp để xây dựng website bán nông sản trực tuyến, đáp ứng yêu cầu mở rộng linh hoạt, hiệu suất cao, và dễ bảo trì cho cửa hàng, đồng thời đảm bảo chi phí vận hành thấp, đáp ứng nhu cầu ngày càng tăng của người dùng một cách bền vững.

1. Giải pháp cụ thể để giải quyết mô hình/bài toán



1. Sơ đồ kiến trúc hệ thống

Để giải quyết bài toán xây dựng website bán nông sản trực tuyến với khả năng mở rộng và hiệu suất cao, đề tài triển khai các giải pháp cụ thể như sau, trước tiên hệ thống được thiết kế dựa trên kiến trúc Microservices với các dịch vụ độc lập gồm Dịch vụ sản phẩm để quản lý danh sách sản phẩm, tồn kho, và thông tin chi tiết, Dịch vụ tài khoản để xử lý đăng ký, đăng nhập, và quản lý thông tin người dùng, Dịch vụ đơn hàng để theo dõi trạng thái đơn hàng và lịch sử giao dịch, Dịch vụ thanh toán tích hợp với VNPay để đảm bảo giao dịch an toàn, và Dịch vụ giao hàng liên kết với GHTK để quản lý vận chuyển, các dịch vụ này được xây dựng bằng Spring Boot và giao tiếp qua Spring Cloud Gateway và Feign Client để đảm bảo luồng xử lý mượt mà, frontend được phát triển bằng React nhằm cung cấp giao diện thân thiện và tốc độ tải nhanh, với các tính năng như trang chủ, tìm kiếm, giỏ hàng, thanh toán, hồ sơ thông tin, và lịch sử đơn hàng, Redis được tích hợp làm hệ thống cache để lưu trữ dữ liệu truy cập thường xuyên như danh sách sản phẩm và thông tin người dùng, giúp giảm tải cho cơ sở dữ liệu MySQL và tăng tốc độ phản hồi, hệ thống sử dụng Kafka để xử lý các tác vụ bất đồng bộ như cập nhật tồn kho sau khi đặt hàng, cập nhật số điện thoại và email khi xác minh OTP hợp lệ, v.v. Điều này giúp đảm bảo tính nhất quán dữ liệu. Docker được áp dụng để triển khai các dịch vụ, đảm bảo tính đồng nhất giữa các môi trường phát triển và sản xuất, đồng thời hỗ trợ mở rộng dễ dàng khi lượng người dùng tăng, các biện pháp bảo mật như mã hóa mật khẩu, sử dụng JWT để xác thực, và OTP trong cập nhật thông tin nhạy cảm cũng được áp dụng để bảo vệ dữ liệu người dùng, qua đó hệ thống đáp ứng yêu cầu về hiệu suất, mở rộng linh hoạt, và dễ bảo trì cho cửa hàng.

1. Hiện thực giải pháp
2. Mô hình chương trình

Để đạt được mục tiêu đưa ra ban đầu, đề tài sẽ chia hệ thống thành 3 mục tiêu

quan trọng bao gồm: Front End, Back End và Hiệu suất.

1. Front-End



1. Công nghệ được sử dụng ở frontend

· **React**: Làm nền tảng chính để tạo giao diện động và hiệu quả, cung cấp khả năng tái sử dụng thành phần, giúp đề tài xây dựng các trang web linh hoạt và dễ bảo trì.

· **Tailwind CSS**: Được tích hợp để thiết kế giao diện linh hoạt và hiện đại, sử dụng các lớp tiện ích tùy chỉnh, hỗ trợ đề tài tạo ra giao diện đẹp mắt và dễ điều chỉnh.

· **React Router**: Hỗ trợ điều hướng mượt mà giữa các trang như trang chủ, giỏ hàng, và lịch sử đơn hàng, đảm bảo trải nghiệm người dùng liền mạch và dễ sử dụng.

· **Axios**: Được áp dụng để thực hiện các yêu cầu HTTP đến backend, đảm bảo giao tiếp dữ liệu nhanh chóng, ổn định, và đáng tin cậy giữa frontend và backend.

· **Recharts**: Được sử dụng để tạo biểu đồ trực quan như thống kê doanh thu và các sản phẩm bán chạy, giúp quản trị viên dễ dàng phân tích dữ liệu và đưa ra quyết định hiệu quả.

Đề tài sử dụng các công nghệ tiên tiến cho frontend để xây dựng website bán nông sản trực tuyến, bao gồm React làm nền tảng chính để tạo giao diện động và hiệu quả với khả năng tái sử dụng thành phần, Tailwind CSS được tích hợp để thiết kế giao diện linh hoạt và hiện đại với các lớp tiện ích tùy chỉnh, React Router hỗ trợ điều hướng giữa các trang như trang chủ, giỏ hàng, và lịch sử đơn hàng một cách mượt mà, Axios được áp dụng để thực hiện các yêu cầu HTTP đến backend, đảm bảo giao tiếp dữ liệu nhanh chóng và ổn định, Recharts được sử dụng để tạo biểu đồ trực quan như thống kê doanh thu và các sản phẩm bán chạy, giúp quản trị viên dễ dàng phân tích dữ liệu, các công nghệ này kết hợp với nhau tạo ra một giao diện thân thiện, tốc độ cao, và hỗ trợ tốt cho trải nghiệm người dùng cũng như quản lý hệ thống.

1. Back-End



1. Công nghệ được sử dụng ở backend

· **Spring**: Làm nền tảng chính để phát triển các dịch vụ Microservices, cung cấp khả năng xử lý mạnh mẽ và tích hợp dễ dàng với các công nghệ khác như cơ sở dữ liệu và message queue, hỗ trợ đề tài xây dựng các dịch vụ quản lý sản phẩm, đơn hàng, và tài khoản một cách hiệu quả.

· **OAuth 2.0**: Được sử dụng để thực hiện xác thực và ủy quyền an toàn, cho phép người dùng và quản trị viên đăng nhập qua tài khoản bên thứ ba hoặc thông tin cá nhân, giảm rủi ro dữ liệu bị xâm phạm.

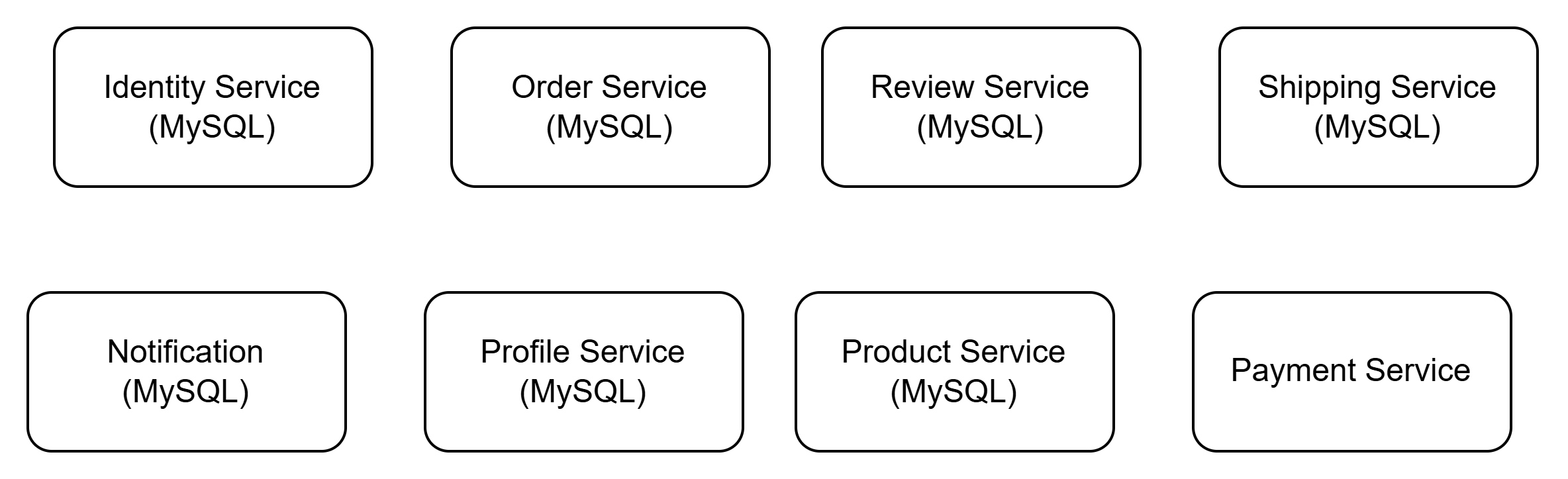
· **Brevo**: Hỗ trợ gửi thông báo qua email tự động hóa như xác nhận đơn hàng, thông tin giao hàng, hoặc cập nhật tài khoản, đảm bảo khách hàng nhận được thông tin kịp thời và chính xác.

· **Twilio**: Được tích hợp để gửi tin nhắn SMS chứa mã OTP, tăng cường bảo mật trong các thao tác nhạy cảm như đăng ký, đổi mật khẩu, hoặc xác nhận giao hàng, hỗ trợ xác thực hai yếu tố.

· **JWT**: Được áp dụng để tạo và xác thực token, đảm bảo giao tiếp an toàn giữa các dịch vụ Microservices và người dùng, ngăn chặn các cuộc tấn công giả mạo.

Đề tài áp dụng các công nghệ mạnh mẽ cho backend để đảm bảo hiệu suất và bảo mật của website bán nông sản trực tuyến, bao gồm Spring làm nền tảng chính để phát triển các dịch vụ Microservices với khả năng xử lý mạnh mẽ và tích hợp dễ dàng, OAuth 2.0 được sử dụng để thực hiện xác thực và ủy quyền an toàn cho người dùng và quản trị viên, Brevo hỗ trợ gửi thông báo qua email để xác nhận thông tin người dùng và cập nhật thông báo về đơn hàng, Twilio được tích hợp để gửi tin nhắn SMS nhằm xác nhận thông tin số điện thoại khách hàng với mã OTP , JWT được áp dụng để tạo và xác thực token, đảm bảo giao tiếp an toàn giữa các dịch vụ và người dùng, các công nghệ này kết hợp với nhau giúp đề tài xây dựng một backend linh hoạt, bảo mật cao, và hỗ trợ tốt cho quản lý sản phẩm, đơn hàng, và người dùng.

1. Xây dựng API microservice



1. Các microservices của đề tài

Đề tài thiết kế hệ thống website bán nông sản trực tuyến dựa trên kiến trúc Microservices với 8 service độc lập để đảm bảo tính linh hoạt và hiệu quả, bao gồm:

· **Dịch vụ sản phẩm (profile)**: Quản lý danh sách sản phẩm, thông tin chi tiết (tên, giá, nguồn gốc, xuất xứ), và tồn kho, hỗ trợ quản trị viên thêm, sửa, xóa sản phẩm, đồng thời cung cấp dữ liệu cho frontend để hiển thị trên trang chủ, tìm kiếm, và chi tiết sản phẩm.

· **Dịch vụ tài khoản (user)**: Xử lý đăng ký, đăng nhập, và quản lý thông tin người dùng (họ tên, email, số điện thoại, địa chỉ), hỗ trợ cập nhật thông tin cá nhân và phân quyền cho người dùng (khách hàng, quản trị viên).

· **Dịch vụ đơn hàng (order)**: Theo dõi trạng thái đơn hàng (đang xử lý, giao hàng thành công), lưu trữ lịch sử giao dịch, cung cấp thông tin cho giao diện lịch sử đơn hàng, hỗ trợ khách hàng yêu cầu trả hàng hoặc xem chi tiết, đồng thời gửi thông báo qua Dịch vụ thông báo khi đơn hàng được khách hàng đặt và khi quản trị viên tiếp nhận, ngoài ra dịch vụ này còn cung cấp các báo cáo và thống kê như doanh thu, sản phẩm bán chạy, và hiệu suất đơn hàng, sử dụng Recharts trên frontend để hiển thị dữ liệu trực quan, hỗ trợ quản trị viên đưa ra quyết định kinh doanh hiệu quả.

· **Dịch vụ thanh toán (payment)**: Tích hợp với VNPay để xử lý giao dịch an toàn, hỗ trợ các phương thức thanh toán như COD và chuyển khoản, đảm bảo khách hàng hoàn tất thanh toán nhanh chóng và minh bạch.

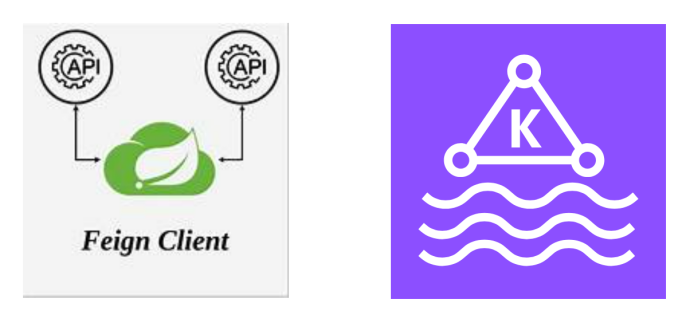
· **Dịch vụ giao hàng (shipping)**: Liên kết với GHTK để quản lý vận chuyển, cập nhật trạng thái giao hàng, và cung cấp thông tin giao hàng (địa chỉ, phí vận chuyển) cho khách hàng và quản trị viên.

· **Dịch vụ thông báo (notification)**: Quản lý việc gửi thông báo qua email và SMS, sử dụng Brevo để gửi email thông báo khi đơn hàng được khách hàng đặt và khi quản trị viên tiếp nhận, đồng thời sử dụng Twilio để gửi SMS thông báo, đảm bảo khách hàng nhận được cập nhật kịp thời, ngoài ra dịch vụ này còn xử lý việc xác thực thông tin người dùng qua email và số điện thoại, gửi mã OTP qua email (Brevo) và SMS (Twilio) để xác minh danh tính trong các thao tác như đăng ký, đăng nhập, hoặc đổi mật khẩu, tăng cường bảo mật cho hệ thống.

· **Dịch vụ hồ sơ (profile)**: Quản lý hồ sơ chi tiết của khách hàng, bao gồm lịch sử mua hàng, địa chỉ giao hàng mặc định, và tùy chọn thanh toán ưu tiên, hỗ trợ khách hàng cá nhân hóa trải nghiệm mua sắm và quản lý thông tin hiệu quả.

· **Dịch vụ đánh giá (review)**: Quản lý đánh giá và nhận xét của khách hàng về sản phẩm, hỗ trợ khách hàng gửi đánh giá sau khi nhận hàng, đồng thời cung cấp dữ liệu cho frontend để hiển thị đánh giá trên trang chi tiết sản phẩm, giúp tăng độ tin cậy và hỗ trợ quyết định mua hàng.

1. Giao tiếp giữa các microservice



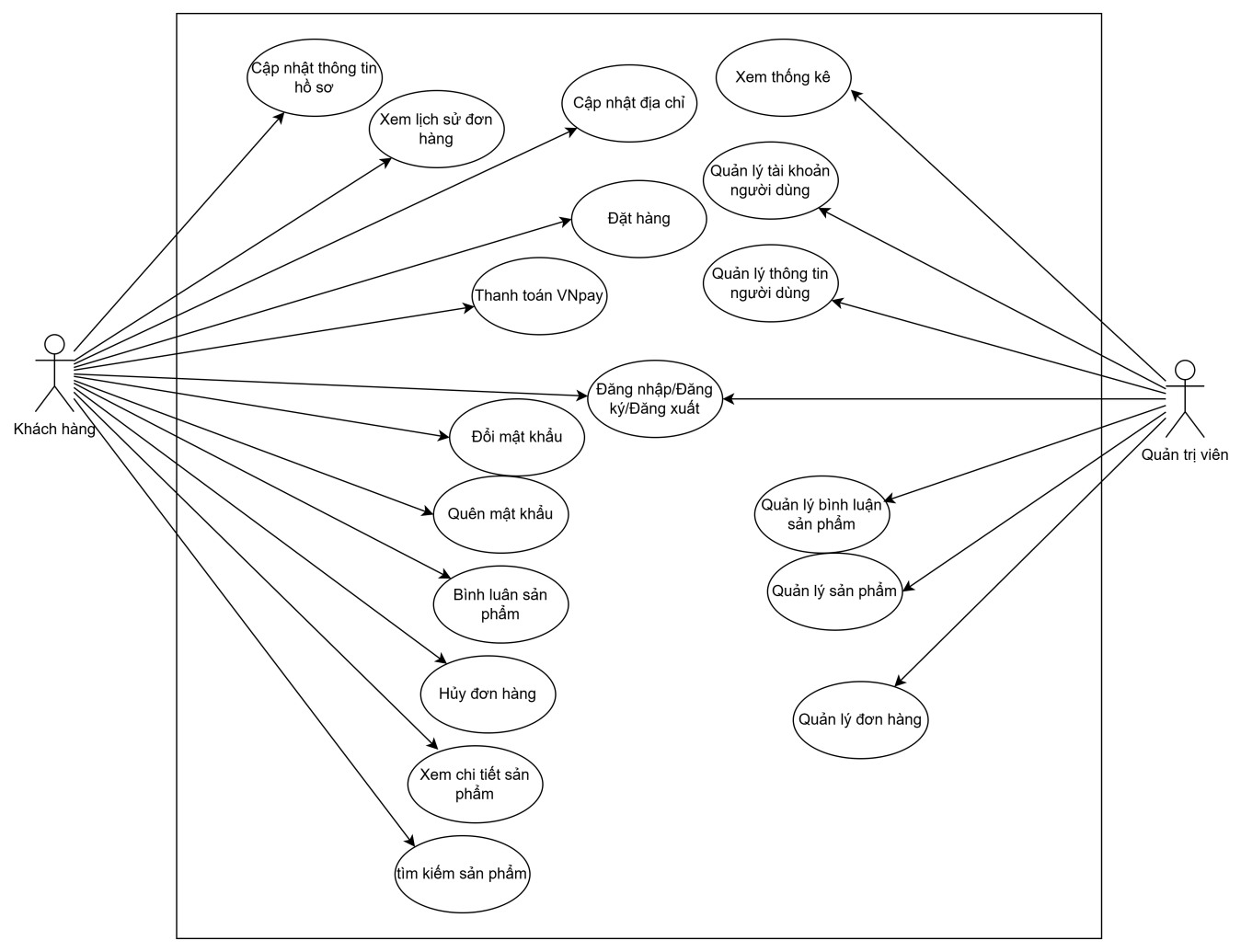
1. Logo của feign client và kafka

Đề tài thiết kế hệ thống giao tiếp giữa các microservice trong website bán nông sản trực tuyến để tối ưu hóa hiệu suất và tính linh hoạt, sử dụng Feign Client để thực hiện các yêu cầu đồng bộ giữa các microservice, chẳng hạn kiểm tra số lượng sản phẩm còn lại trong Dịch vụ sản phẩm trước khi Dịch vụ đơn hàng cho phép khách hàng đặt hàng, đảm bảo tính chính xác và tránh đặt hàng vượt quá tồn kho, đồng thời áp dụng Kafka để xử lý các tác vụ bất đồng bộ, chẳng hạn Dịch vụ đơn hàng gửi sự kiện qua Kafka để Dịch vụ sản phẩm cập nhật số lượng tồn kho khi khách hàng đặt hàng, hủy hàng, hoặc trả hàng, đảm bảo tính nhất quán dữ liệu mà không làm gián đoạn luồng xử lý chính, cách tiếp cận này kết hợp giao tiếp đồng bộ và bất đồng bộ giúp đề tài xây dựng một hệ thống mạnh mẽ, dễ mở rộng, và xử lý hiệu quả các yêu cầu phức tạp trong môi trường Microservices.

1. Hiệu suất

Đề tài tối ưu hóa hiệu suất hệ thống website bán nông sản trực tuyến bằng cách sử dụng Redis làm hệ thống cache, lưu trữ thông tin sản phẩm bao gồm tên, giá, nguồn gốc, xuất xứ, và tồn kho trong Redis, giúp giảm tải truy vấn trực tiếp đến cơ sở dữ liệu MySQL, tăng tốc độ truy xuất dữ liệu cho các trang như trang chủ, tìm kiếm, và chi tiết sản phẩm, đồng thời lưu trữ thông tin người dùng như họ tên, email, số điện thoại, địa chỉ trong Redis, hỗ trợ truy xuất nhanh thông tin tài khoản khi khách hàng đăng nhập, xem hồ sơ, hoặc thực hiện các thao tác liên quan đến tài khoản, giảm thời gian phản hồi của hệ thống, cách tiếp cận này giúp đề tài đảm bảo hiệu suất cao ngay cả khi số lượng người dùng tăng đột biến, mang lại trải nghiệm mượt mà và hiệu quả cho khách hàng và quản trị viên.

1. Mô hình hoá yêu cầu
2. Lược đồ use case

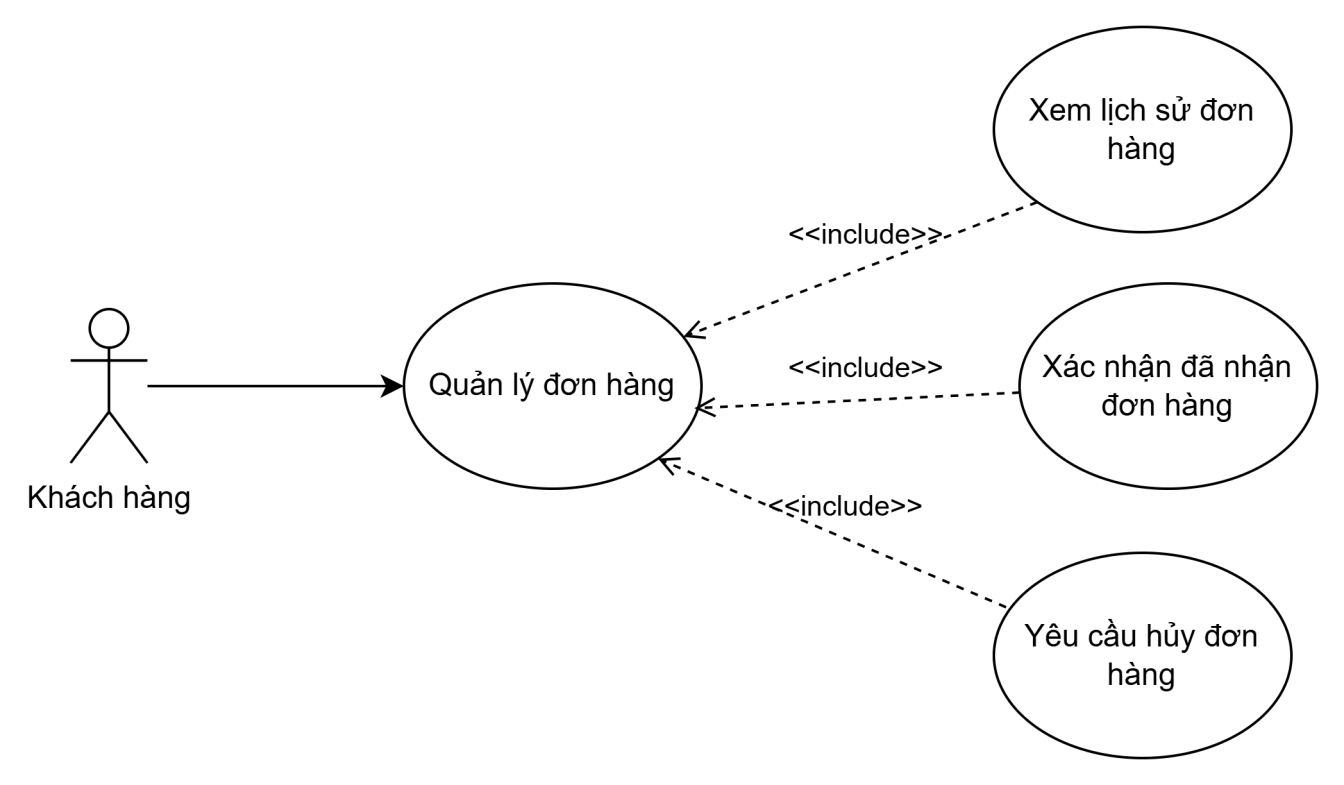


1. Lược đồ use case tổng quát của hệ thống

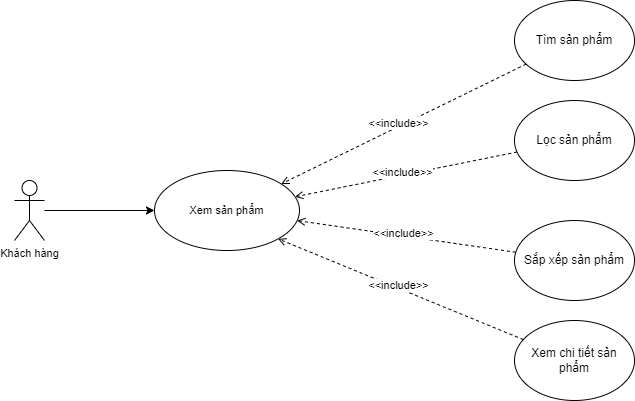
%3CmxGraphModel%3E%3Croot%3E%3CmxCell%20id%3D%220%22%2F%3E%3CmxCell%20id%3D%221%22%20parent%3D%220%22%2F%3E%3CmxCell%20id%3D%222%22%20value%3D%22%22%20style%3D%22endArrow%3Dclassic%3Bhtml%3D1%3Brounded%3D0%3BentryX%3D0.5%3BentryY%3D0%3BentryDx%3D0%3BentryDy%3D0%3BexitX%3D0.5%3BexitY%3D1%3BexitDx%3D0%3BexitDy%3D0%3BentryPerimeter%3D0%3B%22%20edge%3D%221%22%20source%3D%228%22%20target%3D%2213%22%20parent%3D%221%22%3E%3CmxGeometry%20width%3D%2250%22%20height%3D%2250%22%20relative%3D%221%22%20as%3D%22geometry%22%3E%3CmxPoint%20x%3D%22919%22%20y%3D%22597.4%22%20as%3D%22sourcePoint%22%2F%3E%3CmxPoint%20x%3D%22780%22%20y%3D%22676.4%22%20as%3D%22targetPoint%22%2F%3E%3C%2FmxGeometry%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%223%22%20value%3D%22%22%20style%3D%22sketch%3D0%3Bpoints%3D%5B%5B0%2C0%2C0%5D%2C%5B0.25%2C0%2C0%5D%2C%5B0.5%2C0%2C0%5D%2C%5B0.75%2C0%2C0%5D%2C%5B1%2C0%2C0%5D%2C%5B0%2C1%2C0%5D%2C%5B0.25%2C1%2C0%5D%2C%5B0.5%2C1%2C0%5D%2C%5B0.75%2C1%2C0%5D%2C%5B1%2C1%2C0%5D%2C%5B0%2C0.25%2C0%5D%2C%5B0%2C0.5%2C0%5D%2C%5B0%2C0.75%2C0%5D%2C%5B1%2C0.25%2C0%5D%2C%5B1%2C0.5%2C0%5D%2C%5B1%2C0.75%2C0%5D%5D%3BoutlineConnect%3D0%3BfontColor%3D%23232F3E%3BfillColor%3D%23DD344C%3BstrokeColor%3D%23ffffff%3Bdashed%3D0%3BverticalLabelPosition%3Dbottom%3BverticalAlign%3Dtop%3Balign%3Dcenter%3Bhtml%3D1%3BfontSize%3D12%3BfontStyle%3D0%3Baspect%3Dfixed%3Bshape%3Dmxgraph.aws4.resourceIcon%3BresIcon%3Dmxgraph.aws4.identity\_and\_access\_management%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22332%22%20y%3D%22589.4%22%20width%3D%2258%22%20height%3D%2258%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%224%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3BstartArrow%3Dclassic%3BstartFill%3D1%3B%22%20edge%3D%221%22%20source%3D%225%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%3E%3CArray%20as%3D%22points%22%3E%3CmxPoint%20x%3D%22430%22%20y%3D%22780%22%2F%3E%3C%2FArray%3E%3CmxPoint%20x%3D%22344%22%20y%3D%22780%22%20as%3D%22targetPoint%22%2F%3E%3C%2FmxGeometry%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%225%22%20value%3D%22Identity%20Service%26lt%3Bdiv%26gt%3B(MySQL)%26lt%3B%2Fdiv%26gt%3B%22%20style%3D%22rounded%3D1%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22390%22%20y%3D%22587.4%22%20width%3D%22120%22%20height%3D%2260%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%226%22%20value%3D%22Notification%26lt%3Bbr%26gt%3B%26lt%3Bdiv%26gt%3B(MySQL)%26lt%3B%2Fdiv%26gt%3B%22%20style%3D%22rounded%3D1%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22390%22%20y%3D%22917.4%22%20width%3D%22120%22%20height%3D%2260%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%227%22%20value%3D%22Product%20Service%26lt%3Bdiv%26gt%3B(MySQL)%26lt%3B%2Fdiv%26gt%3B%22%20style%3D%22rounded%3D1%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22760%22%20y%3D%22917.4%22%20width%3D%22120%22%20height%3D%2260%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%228%22%20value%3D%22Order%20Service%26lt%3Bdiv%26gt%3B(MySQL)%26lt%3B%2Fdiv%26gt%3B%22%20style%3D%22rounded%3D1%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22665%22%20y%3D%22589.4%22%20width%3D%22120%22%20height%3D%2260%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%229%22%20value%3D%22Review%20Service%26lt%3Bdiv%26gt%3B(MySQL)%26lt%3B%2Fdiv%26gt%3B%22%20style%3D%22rounded%3D1%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22920.0000000000001%22%20y%3D%22587.4%22%20width%3D%22120%22%20height%3D%2260%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2210%22%20value%3D%22Shipping%20Service%26lt%3Bdiv%26gt%3B(MySQL)%26lt%3B%2Fdiv%26gt%3B%22%20style%3D%22rounded%3D1%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%221160%22%20y%3D%22587.4%22%20width%3D%22120%22%20height%3D%2260%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2211%22%20value%3D%22Payment%20Service%22%20style%3D%22rounded%3D1%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%221170%22%20y%3D%22919.4%22%20width%3D%22120%22%20height%3D%2260%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2212%22%20style%3D%22rounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3BentryX%3D0.5%3BentryY%3D0%3BentryDx%3D0%3BentryDy%3D0%3B%22%20edge%3D%221%22%20source%3D%2213%22%20target%3D%2218%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2213%22%20value%3D%22%22%20style%3D%22sketch%3D0%3Bpoints%3D%5B%5B0%2C0%2C0%5D%2C%5B0.25%2C0%2C0%5D%2C%5B0.5%2C0%2C0%5D%2C%5B0.75%2C0%2C0%5D%2C%5B1%2C0%2C0%5D%2C%5B0%2C1%2C0%5D%2C%5B0.25%2C1%2C0%5D%2C%5B0.5%2C1%2C0%5D%2C%5B0.75%2C1%2C0%5D%2C%5B1%2C1%2C0%5D%2C%5B0%2C0.25%2C0%5D%2C%5B0%2C0.5%2C0%5D%2C%5B0%2C0.75%2C0%5D%2C%5B1%2C0.25%2C0%5D%2C%5B1%2C0.5%2C0%5D%2C%5B1%2C0.75%2C0%5D%5D%3BoutlineConnect%3D0%3BfontColor%3D%23232F3E%3BfillColor%3D%238C4FFF%3BstrokeColor%3D%23ffffff%3Bdashed%3D0%3BverticalLabelPosition%3Dbottom%3BverticalAlign%3Dtop%3Balign%3Dcenter%3Bhtml%3D1%3BfontSize%3D12%3BfontStyle%3D0%3Baspect%3Dfixed%3Bshape%3Dmxgraph.aws4.resourceIcon%3BresIcon%3Dmxgraph.aws4.managed\_streaming\_for\_kafka%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22590%22%20y%3D%22787.4%22%20width%3D%2258%22%20height%3D%2258%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2214%22%20value%3D%22%22%20style%3D%22endArrow%3Dclassic%3Bhtml%3D1%3Brounded%3D0%3BentryX%3D0.5%3BentryY%3D0%3BentryDx%3D0%3BentryDy%3D0%3BexitX%3D0.5%3BexitY%3D1%3BexitDx%3D0%3BexitDy%3D0%3BentryPerimeter%3D0%3B%22%20edge%3D%221%22%20source%3D%225%22%20target%3D%2213%22%20parent%3D%221%22%3E%3CmxGeometry%20width%3D%2250%22%20height%3D%2250%22%20relative%3D%221%22%20as%3D%22geometry%22%3E%3CmxPoint%20x%3D%22452%22%20y%3D%22557.4%22%20as%3D%22sourcePoint%22%2F%3E%3CmxPoint%20x%3D%22332%22%20y%3D%22687.4%22%20as%3D%22targetPoint%22%2F%3E%3C%2FmxGeometry%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2215%22%20value%3D%22%22%20style%3D%22endArrow%3Dclassic%3Bhtml%3D1%3Brounded%3D0%3BentryX%3D0.5%3BentryY%3D0%3BentryDx%3D0%3BentryDy%3D0%3BexitX%3D0.5%3BexitY%3D1%3BexitDx%3D0%3BexitDy%3D0%3BexitPerimeter%3D0%3B%22%20edge%3D%221%22%20source%3D%2213%22%20target%3D%226%22%20parent%3D%221%22%3E%3CmxGeometry%20width%3D%2250%22%20height%3D%2250%22%20relative%3D%221%22%20as%3D%22geometry%22%3E%3CmxPoint%20x%3D%22480%22%20y%3D%22687.4%22%20as%3D%22sourcePoint%22%2F%3E%3CmxPoint%20x%3D%22360%22%20y%3D%22817.4%22%20as%3D%22targetPoint%22%2F%3E%3C%2FmxGeometry%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2216%22%20value%3D%22%22%20style%3D%22endArrow%3Dclassic%3Bhtml%3D1%3Brounded%3D0%3BexitX%3D0.5%3BexitY%3D1%3BexitDx%3D0%3BexitDy%3D0%3BexitPerimeter%3D0%3BentryX%3D0.5%3BentryY%3D0%3BentryDx%3D0%3BentryDy%3D0%3B%22%20edge%3D%221%22%20source%3D%2213%22%20target%3D%227%22%20parent%3D%221%22%3E%3CmxGeometry%20width%3D%2250%22%20height%3D%2250%22%20relative%3D%221%22%20as%3D%22geometry%22%3E%3CmxPoint%20x%3D%22650%22%20y%3D%22757.4%22%20as%3D%22sourcePoint%22%2F%3E%3CmxPoint%20x%3D%22820%22%20y%3D%22747.4%22%20as%3D%22targetPoint%22%2F%3E%3CArray%20as%3D%22points%22%2F%3E%3C%2FmxGeometry%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2217%22%20value%3D%22%22%20style%3D%22points%3D%5B%5D%3Baspect%3Dfixed%3Bhtml%3D1%3Balign%3Dcenter%3Bshadow%3D0%3Bdashed%3D0%3BfillColor%3D%23FF6A00%3BstrokeColor%3Dnone%3Bshape%3Dmxgraph.alibaba\_cloud.sms\_short\_message\_service%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22856.66%22%20y%3D%22592.9%22%20width%3D%2263.34%22%20height%3D%2258%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2218%22%20value%3D%22Profile%20Service%26lt%3Bdiv%26gt%3B(MySQL)%26lt%3B%2Fdiv%26gt%3B%22%20style%3D%22rounded%3D1%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22556.33%22%20y%3D%22990%22%20width%3D%22120%22%20height%3D%2260%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2219%22%20value%3D%22%22%20style%3D%22shape%3Dimage%3BverticalLabelPosition%3Dbottom%3BverticalAlign%3Dtop%3BimageAspect%3D0%3Baspect%3Dfixed%3Bimage%3Ddata%3Aimage%2Fsvg%2Bxml%2CPHN2ZyB4bWxucz0iaHR0cDovL3d3dy53My5vcmcvMjAwMC9zdmciIHZlcnNpb249IjEiIHZpZXdCb3g9IjAgMCA0OCA0OCIgZW5hYmxlLWJhY2tncm91bmQ9Im5ldyAwIDAgNDggNDgiPiYjeGE7ICAgIDxwb2x5Z29uIGZpbGw9IiNGRjk4MDAiIHBvaW50cz0iMjQsMzcgMTksMzEgMTksMjUgMjksMjUgMjksMzEiLz4mI3hhOyAgICA8ZyBmaWxsPSIjRkZBNzI2Ij4mI3hhOyAgICAgICAgPGNpcmNsZSBjeD0iMzMiIGN5PSIxOSIgcj0iMiIvPiYjeGE7ICAgICAgICA8Y2lyY2xlIGN4PSIxNSIgY3k9IjE5IiByPSIyIi8%2BJiN4YTsgICAgPC9nPiYjeGE7ICAgIDxwYXRoIGZpbGw9IiNGRkI3NEQiIGQ9Ik0zMywxM2MwLTcuNi0xOC01LTE4LDBjMCwxLjEsMCw1LjksMCw3YzAsNSw0LDksOSw5czktNCw5LTlDMzMsMTguOSwzMywxNC4xLDMzLDEzeiIvPiYjeGE7ICAgIDxwYXRoIGZpbGw9IiM0MjQyNDIiIGQ9Ik0yNCw0Yy02LjEsMC0xMCw0LjktMTAsMTFjMCwwLjgsMCwyLjMsMCwyLjNsMiwxLjd2LTVsMTItNGw0LDR2NWwyLTEuN2MwLDAsMC0xLjUsMC0yLjNjMC00LTEtOC02LTlsLTEtMiBIMjR6Ii8%2BJiN4YTsgICAgPGcgZmlsbD0iIzc4NDcxOSI%2BJiN4YTsgICAgICAgIDxjaXJjbGUgY3g9IjI4IiBjeT0iMTkiIHI9IjEiLz4mI3hhOyAgICAgICAgPGNpcmNsZSBjeD0iMjAiIGN5PSIxOSIgcj0iMSIvPiYjeGE7ICAgIDwvZz4mI3hhOyAgICA8cG9seWdvbiBmaWxsPSIjZmZmIiBwb2ludHM9IjI0LDQzIDE5LDMxIDI0LDMyIDI5LDMxIi8%2BJiN4YTsgICAgPHBvbHlnb24gZmlsbD0iI0QzMkYyRiIgcG9pbnRzPSIyMywzNSAyMi4zLDM5LjUgMjQsNDMuNSAyNS43LDM5LjUgMjUsMzUgMjYsMzQgMjQsMzIgMjIsMzQiLz4mI3hhOyAgICA8cGF0aCBmaWxsPSIjNTQ2RTdBIiBkPSJNMjksMzFMMjksMzFsLTUsMTJsLTUtMTJjMCwwLTExLDItMTEsMTNoMzJDNDAsMzMsMjksMzEsMjksMzF6Ii8%2BJiN4YTs8L3N2Zz4%3D%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22490%22%20y%3D%22986.83%22%20width%3D%2266.33%22%20height%3D%2266.33%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2220%22%20value%3D%22%22%20style%3D%22shape%3Dimage%3BverticalLabelPosition%3Dbottom%3BverticalAlign%3Dtop%3BimageAspect%3D0%3Baspect%3Dfixed%3Bimage%3Ddata%3Aimage%2Fsvg%2Bxml%%2BJiN4YTsgICAgPC9nPiYjeGE7ICAgIDxnIGZpbGw9IiM3ODkwOUMiPiYjeGE7ICAgICAgICA8Y2lyY2xlIGN4PSIzNyIgY3k9IjM2IiByPSIyIi8%2BJiN4YTsgICAgICAgIDxjaXJjbGUgY3g9IjEzIiBjeT0iMzYiIHI9IjIiLz4mI3hhOyAgICA8L2c%2BJiN4YTsgICAgPHBhdGggZmlsbD0iIzM3NDc0RiIgZD0iTTQxLDI1aC03Yy0wLjYsMC0xLTAuNC0xLTF2LTdjMC0wLjYsMC40LTEsMS0xaDUuM2MwLjQsMCwwLjgsMC4zLDAuOSwwLjdsMS43LDUuMmMwLDAuMSwwLjEsMC4yLDAuMSwwLjNWMjQgQzQyLDI0LjYsNDEuNiwyNSw0MSwyNXoiLz4mI3hhOyAgICA8cG9seWdvbiBmaWxsPSIjRENFREM4IiBwb2ludHM9IjIxLjgsMTMuOCAxMy45LDIxLjcgMTAuMiwxNy45IDgsMjAuMSAxMy45LDI2IDI0LDE1LjkiLz4mI3hhOzwvc3ZnPg%3D%3D%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%221085%22%20y%3D%22580.9%22%20width%3D%2275%22%20height%3D%2275%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2221%22%20value%3D%22%22%20style%3D%22shape%3Dimage%3BverticalLabelPosition%3Dbottom%3BverticalAlign%3Dtop%3BimageAspect%3D0%3Baspect%3Dfixed%3Bimage%3Ddata%3Aimage%2Fsvg%2Bxml%%2BJiN4YTs8L3N2Zz4%3D%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22594.9999999999999%22%20y%3D%22580.9%22%20width%3D%2270%22%20height%3D%2270%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2222%22%20value%3D%22%22%20style%3D%22shape%3Dimage%3BverticalLabelPosition%3Dbottom%3BlabelBackgroundColor%3Ddefault%3BverticalAlign%3Dtop%3Baspect%3Dfixed%3BimageAspect%3D0%3Bimage%3Dhttps%3A%2F%2Fpng.pngtree.com%2Fpng-clipart%2F20190118%2Fourmid%2Fpngtree-cartoon-vegetable-orange-agricultural-products-harvest-agriculture-png-image\_449405.jpg%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22690%22%20y%3D%22912.4%22%20width%3D%2270%22%20height%3D%2270%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2223%22%20value%3D%22%22%20style%3D%22shape%3Dimage%3BverticalLabelPosition%3Dbottom%3BverticalAlign%3Dtop%3BimageAspect%3D0%3Baspect%3Dfixed%3Bimage%3Ddata%3Aimage%2Fsvg%2Bxml%2CPHN2ZyB4bWxucz0iaHR0cDovL3d3dy53My5vcmcvMjAwMC9zdmciIHZlcnNpb249IjEiIHZpZXdCb3g9IjAgMCA0OCA0OCIgZW5hYmxlLWJhY2tncm91bmQ9Im5ldyAwIDAgNDggNDgiPiYjeGE7ICAgIDxwb2x5Z29uIGZpbGw9IiNDNUNBRTkiIHBvaW50cz0iNDIsNDIgNiw0MiA2LDkgMjQsMiA0Miw5Ii8%2BJiN4YTsgICAgPHJlY3QgeD0iNiIgeT0iNDIiIGZpbGw9IiM5RkE4REEiIHdpZHRoPSIzNiIgaGVpZ2h0PSIyIi8%2BJiN4YTsgICAgPHJlY3QgeD0iMjAiIHk9IjM1IiBmaWxsPSIjQkYzNjBDIiB3aWR0aD0iOCIgaGVpZ2h0PSI5Ii8%2BJiN4YTsgICAgPGcgZmlsbD0iIzE1NjVDMCI%2BJiN4YTsgICAgICAgIDxyZWN0IHg9IjMxIiB5PSIyNyIgd2lkdGg9IjYiIGhlaWdodD0iNSIvPiYjeGE7ICAgICAgICA8cmVjdCB4PSIyMSIgeT0iMjciIHdpZHRoPSI2IiBoZWlnaHQ9IjUiLz4mI3hhOyAgICAgICAgPHJlY3QgeD0iMTEiIHk9IjI3IiB3aWR0aD0iNiIgaGVpZ2h0PSI1Ii8%2BJiN4YTsgICAgICAgIDxyZWN0IHg9IjMxIiB5PSIzNSIgd2lkdGg9IjYiIGhlaWdodD0iNSIvPiYjeGE7ICAgICAgICA8cmVjdCB4PSIxMSIgeT0iMzUiIHdpZHRoPSI2IiBoZWlnaHQ9IjUiLz4mI3hhOyAgICAgICAgPHJlY3QgeD0iMzEiIHk9IjE5IiB3aWR0aD0iNiIgaGVpZ2h0PSI1Ii8%2BJiN4YTsgICAgICAgIDxyZWN0IHg9IjIxIiB5PSIxOSIgd2lkdGg9IjYiIGhlaWdodD0iNSIvPiYjeGE7ICAgICAgICA8cmVjdCB4PSIxMSIgeT0iMTkiIHdpZHRoPSI2IiBoZWlnaHQ9IjUiLz4mI3hhOyAgICAgICAgPHJlY3QgeD0iMzEiIHk9IjExIiB3aWR0aD0iNiIgaGVpZ2h0PSI1Ii8%2BJiN4YTsgICAgICAgIDxyZWN0IHg9IjIxIiB5PSIxMSIgd2lkdGg9IjYiIGhlaWdodD0iNSIvPiYjeGE7ICAgICAgICA8cmVjdCB4PSIxMSIgeT0iMTEiIHdpZHRoPSI2IiBoZWlnaHQ9IjUiLz4mI3hhOyAgICA8L2c%2BJiN4YTs8L3N2Zz4%3D%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%221110%22%20y%3D%22920.4%22%20width%3D%2260%22%20height%3D%2260%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3C%2Froot%3E%3C%2FmxGraphMode



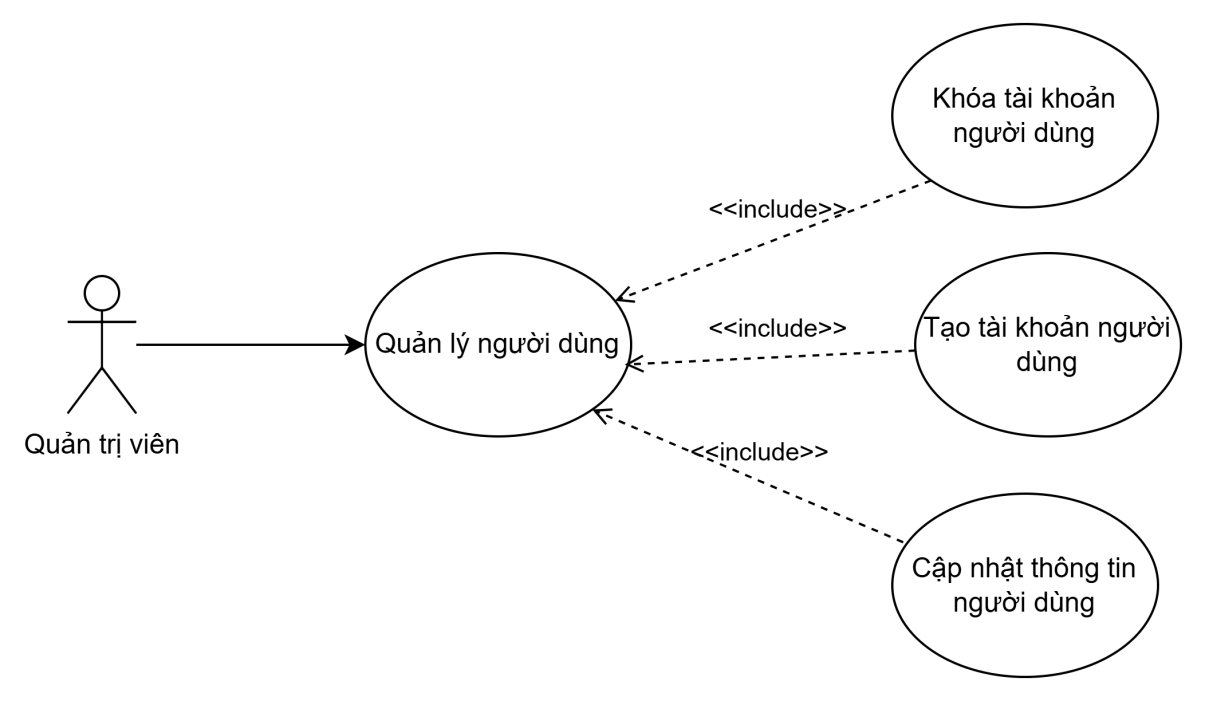
1. Lược đồ usecase của xác minh



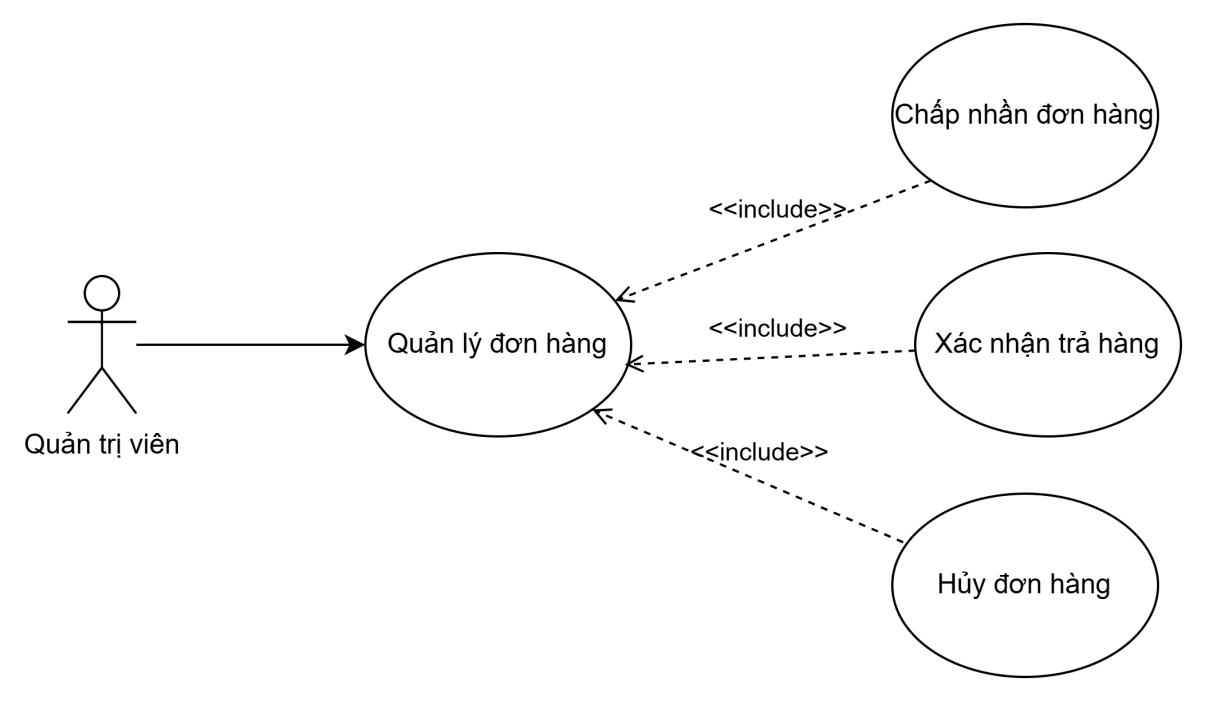
1. Lược đồ usecase của quản lý đơn hàng



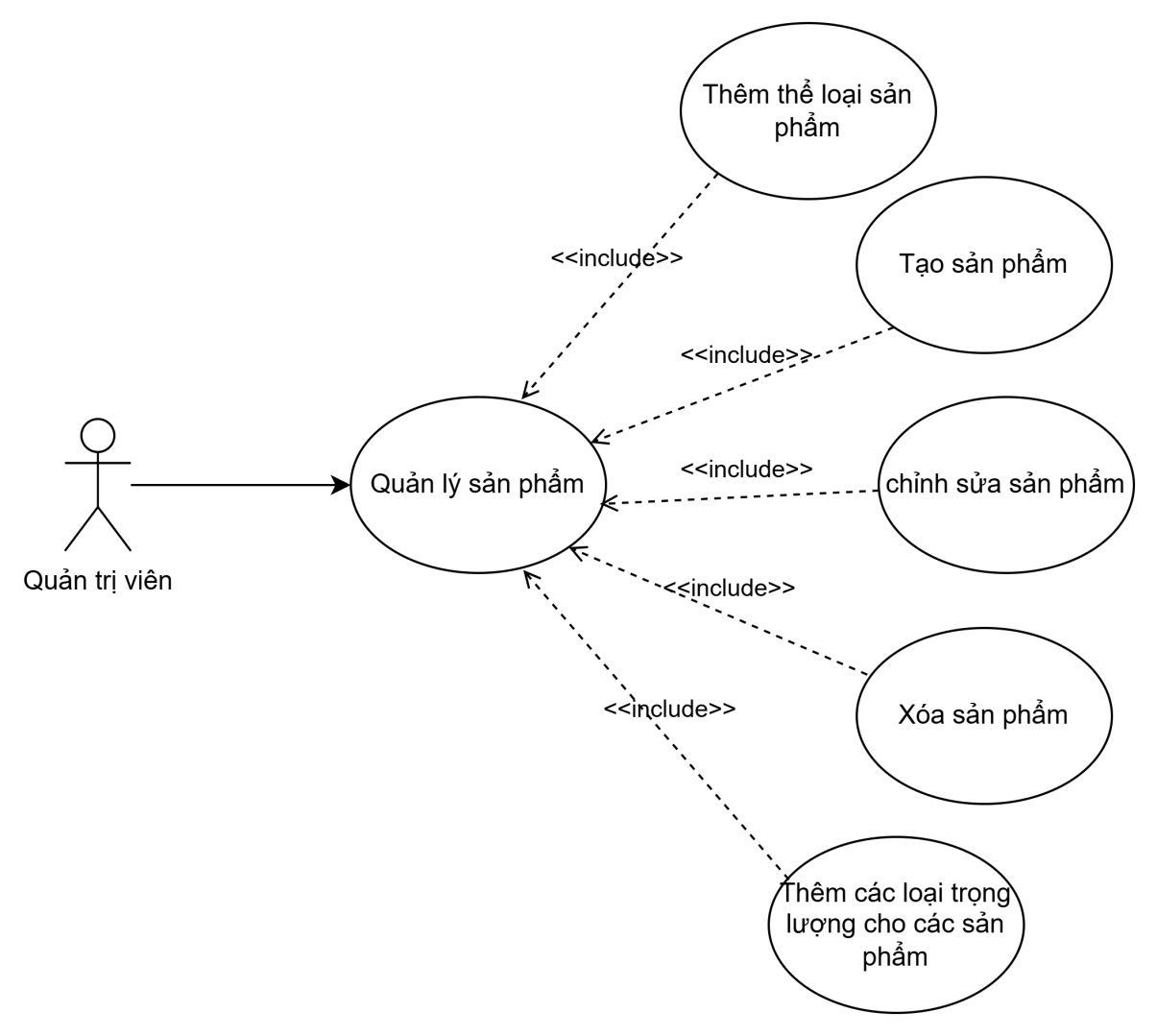
1. Lược đồ usecase xem sản phẩm



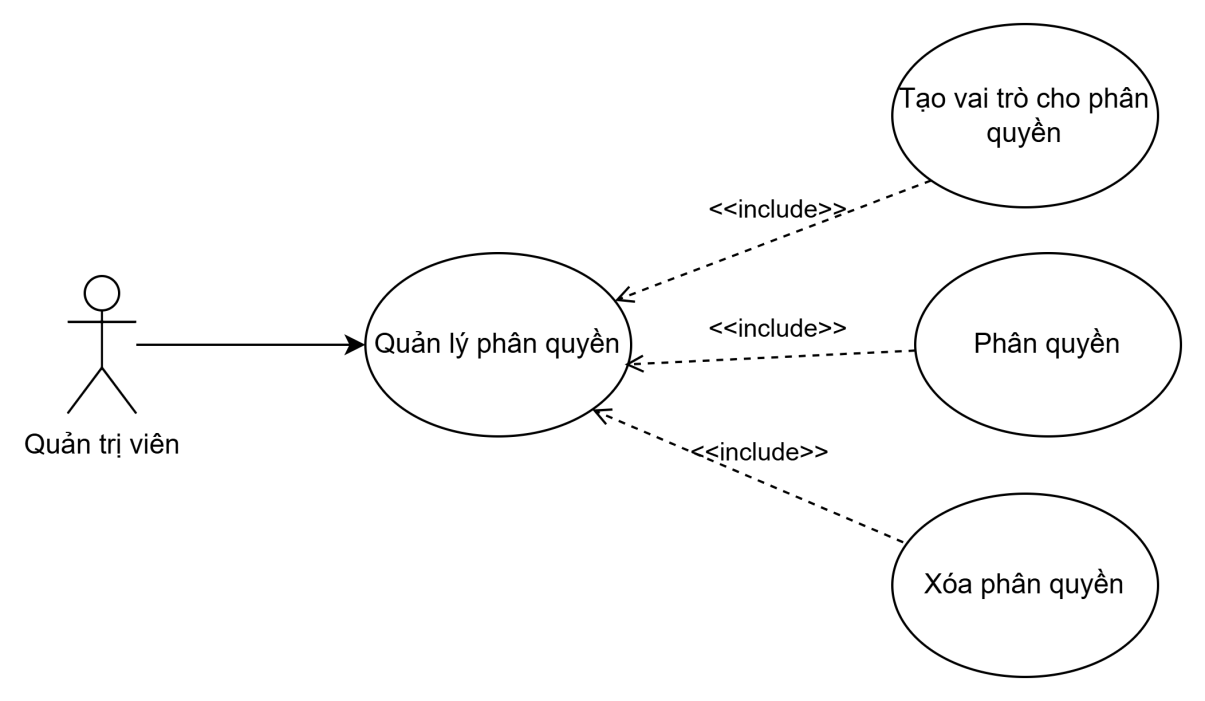
1. Lược đồ usecase quản lý người dùng



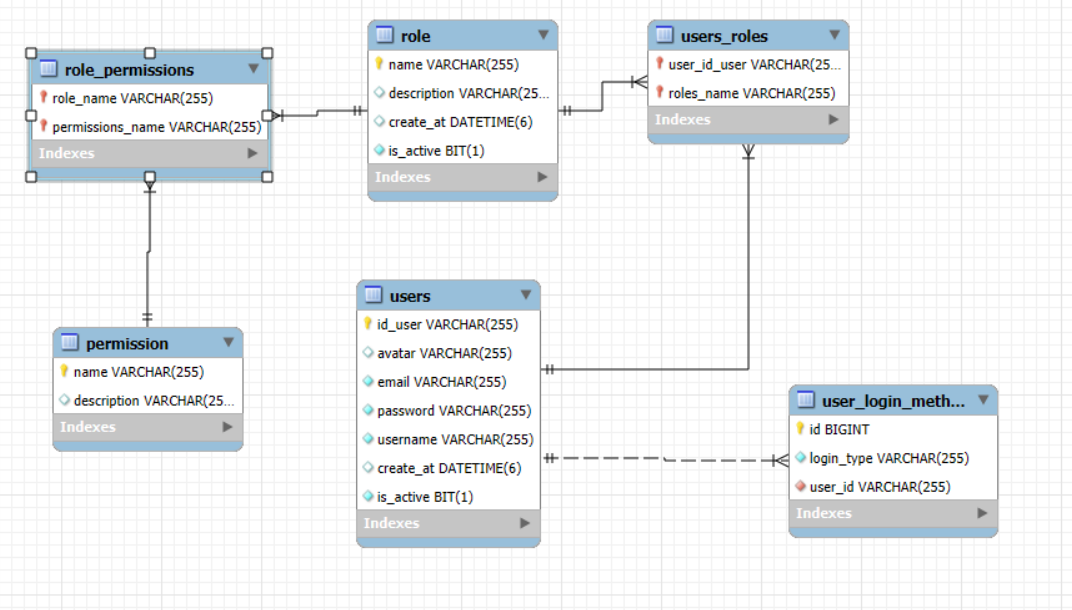
1. Lược đồ usecase quản lý đơn hàng



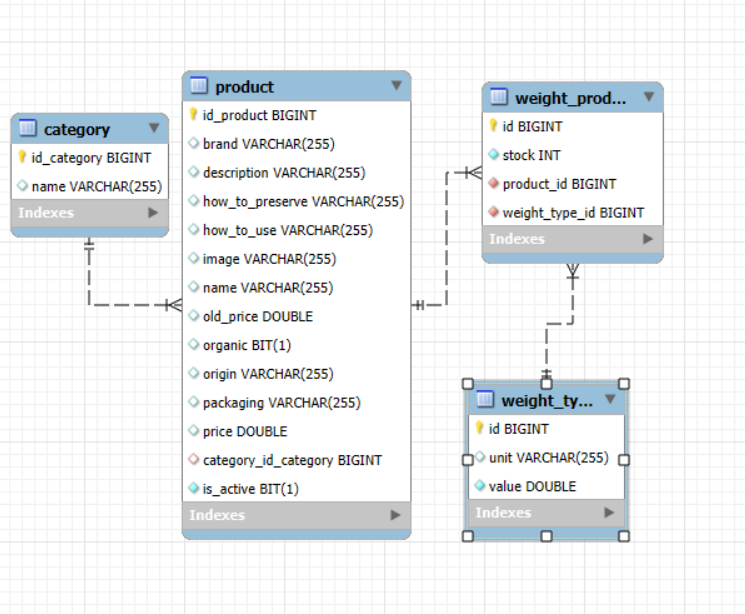
1. Lược đồ usecase quản lý sản phẩm



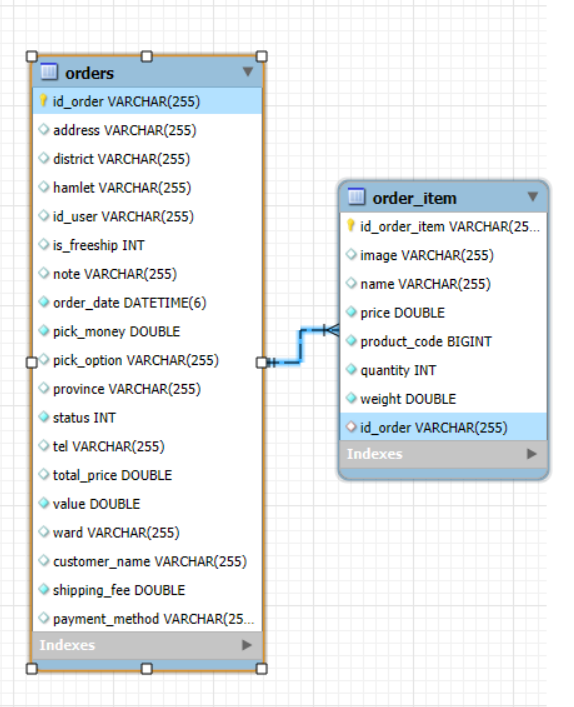
1. Lược đồ usecase quản lý phân quyền
2. Lược đồ quan hệ



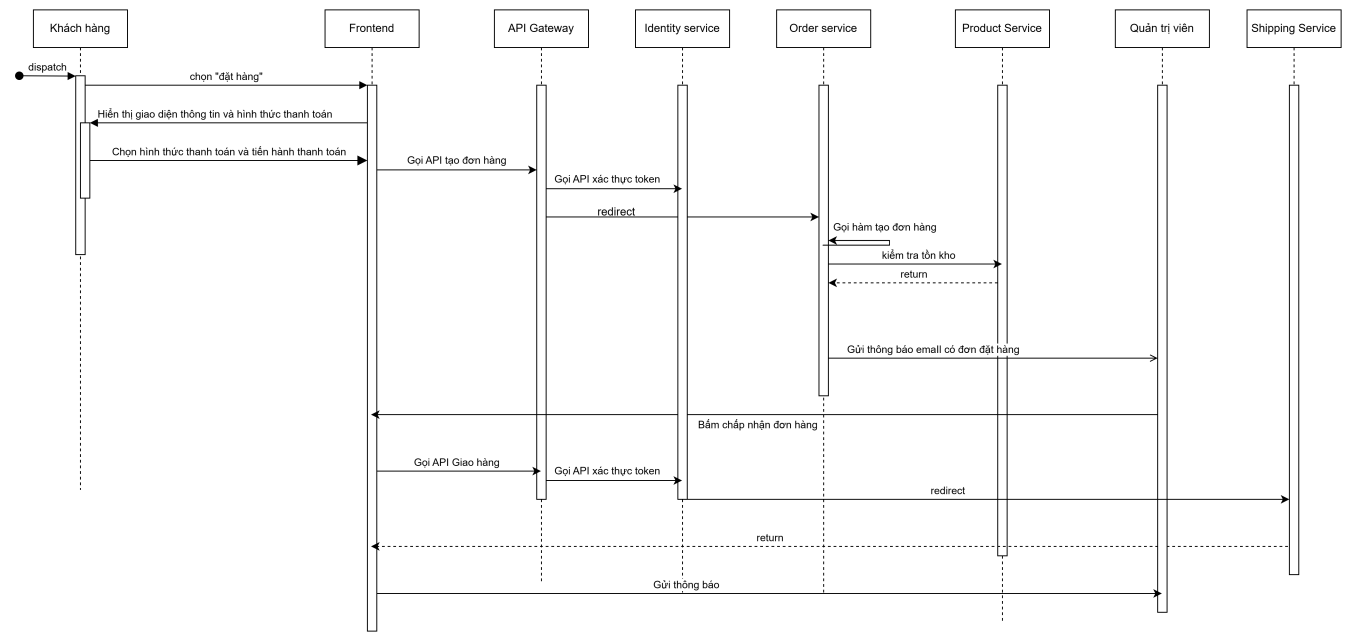
1. Lược đồ quan hệ user



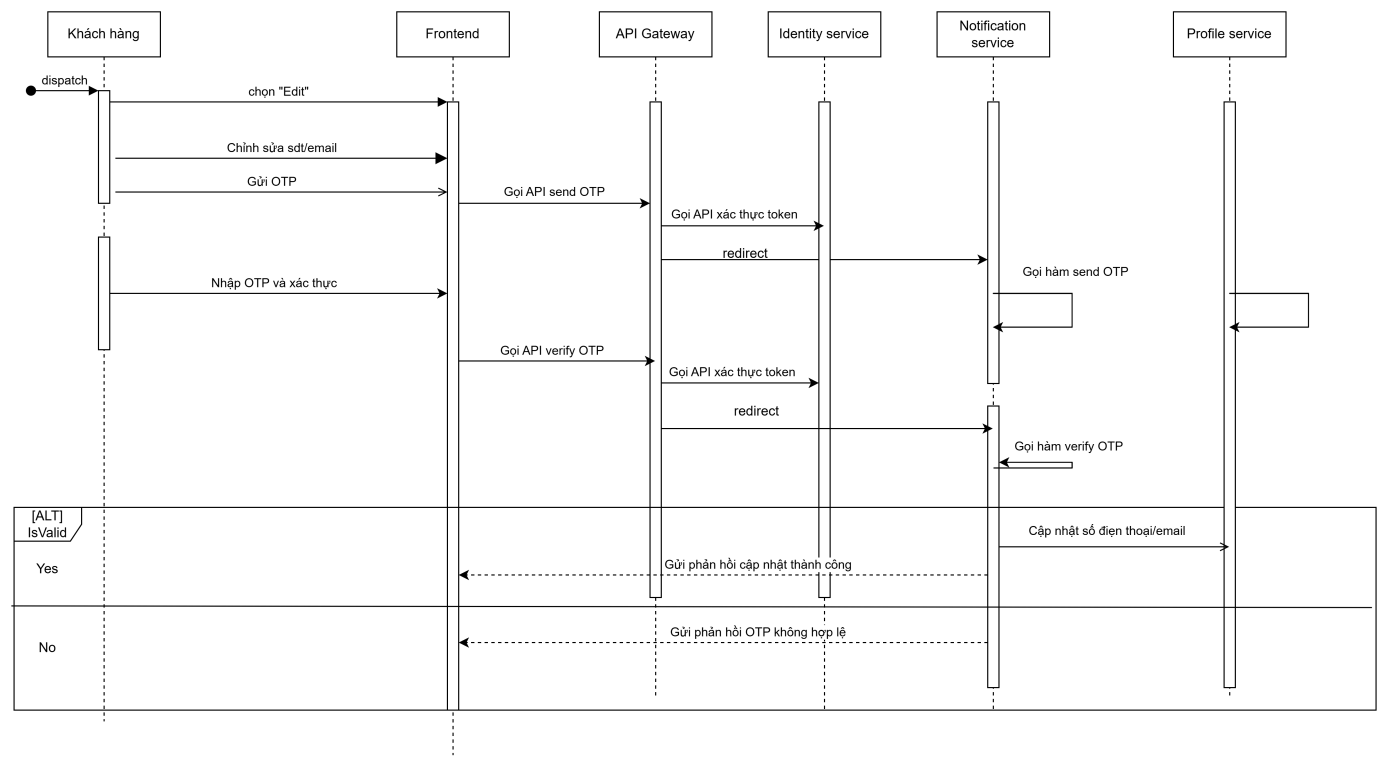
1. Lược đồ quan hệ sản phẩm



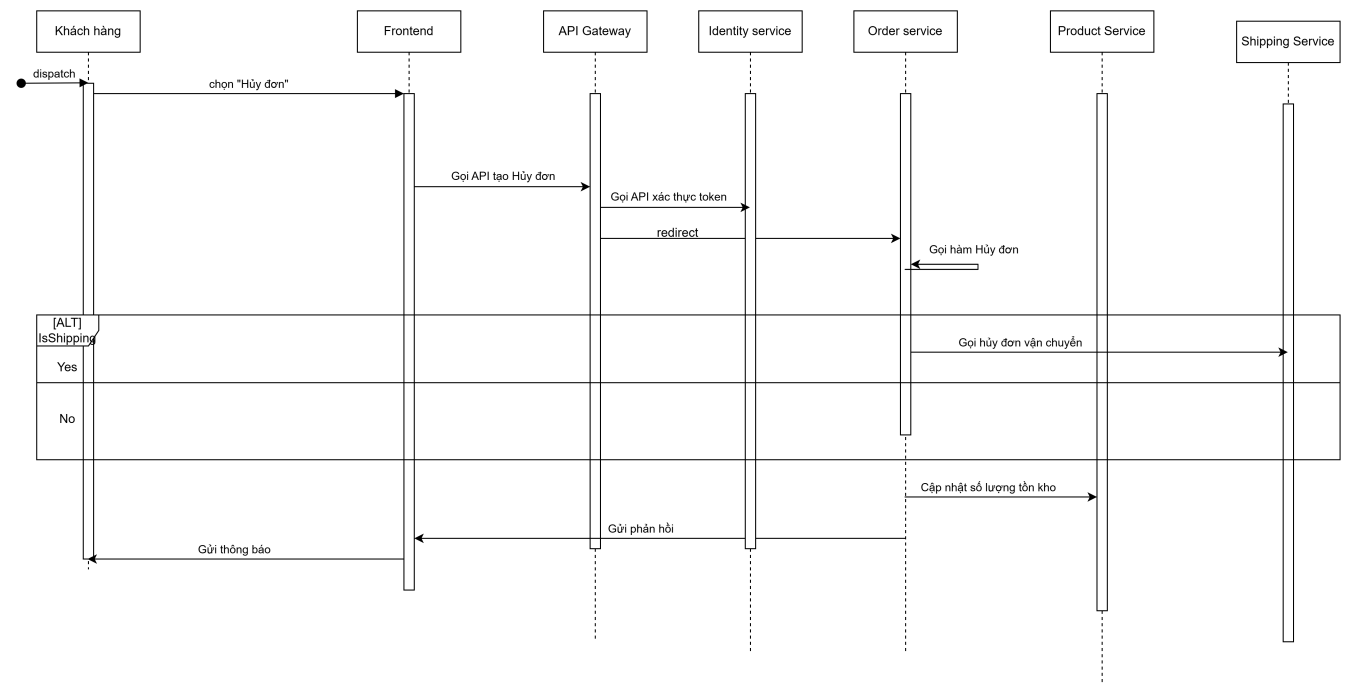
1. Lược đồ quan hệ đơn hàng
2. Lược đồ tuần tự



1. Lược đồ tuần tự luồng đặt hàng



1. Lược đồ tuần tự luồng chỉnh sửa email/sđt



1. Lược đồ tuần tự luồng hủy đặt hàng
2. Hệ thống Quản lý Trạng thái Đơn hàng

Trong các hệ thống thương mại điện tử, việc quản lý trạng thái đơn hàng là một yếu tố cốt lõi để đảm bảo quy trình kinh doanh diễn ra trơn tru và minh bạch. Hệ thống này không chỉ theo dõi trạng thái đơn hàng mà còn phải thực thi các luật chuyển đổi trạng thái một cách chặt chẽ, đồng thời đảm bảo tính bảo mật thông qua các cơ chế như mã hóa token.

1. Bộ nhãn trạng thái đơn hàng

Hệ thống sử dụng một tập hợp trạng thái đơn hàng được định nghĩa rõ ràng, bao gồm:

* PENDING\_CONFIRMATION: Đơn hàng chờ xác nhận từ phía người bán.
* WAITING\_FOR\_SHIPMENT: Đơn hàng đã xác nhận, chờ vận chuyển.
* SHIPPING: Đơn hàng đang được vận chuyển.
* DELIVERED: Đơn hàng đã giao đến khách hàng.
* RETURN\_APPROVED: Đơn hàng được phê duyệt trả hàng.
* RETURNED: Đơn hàng đã được trả lại.
* CANCELED: Đơn hàng bị hủy.

Các trạng thái này được lưu trữ dưới dạng mã trong cơ sở dữ liệu và ánh xạ sang một enum trong mã nguồn, giúp xử lý logic chuyển đổi trạng thái một cách nhất quán.

1. Luật chuyển đổi trạng thái:

Hệ thống sử dụng Finite State Machine (FSM) để quản lý các chuyển đổi trạng thái đơn hàng, đảm bảo chỉ những thay đổi hợp lệ được thực hiện. Bảng dưới đây mô tả các luật chuyển đổi trạng thái, bao gồm trạng thái hiện tại, trạng thái mục tiêu, và hành động liên quan.

|  |  |  |
| --- | --- | --- |
| Trạng thái hiện tại | Trạng thái mục tiêu | Hành động liên quan |
| PENDING\_CONFIRMATION | WAITING\_FOR\_SHIPMENT | Xác nhận đơn hàng, tạo đơn vận chuyển qua API bên GHTK. |
| PENDING\_CONFIRMATION | CANCELED | Hủy đơn hàng, khôi phục tồn kho. |
| WAITING\_FOR\_SHIPMENT | SHIPPING | Bắt đầu vận chuyển đơn hàng. |
| WAITING\_FOR\_SHIPMENT | CANCELED | Hủy đơn hàng, hủy đơn vận chuyển qua API bên thứ ba, khôi phục tồn kho. |
| SHIPPING | DELIVERED | Giao hàng thành công. |
| DELIVERED | RETURN\_APPROVED | Phê duyệt yêu cầu trả hàng từ khách hàng. |
| DELIVERED | CONFIRMED\_BY\_CUSTOMER | Khách hàng xác nhận đã nhận hàng. |

1. Mô tả trạng thái đơn hàng

Ngoài ra, để tương thích với các trạng thái đơn hàng của GHTK, hệ thống cần một luật phát sinh để chuyển đổi. Bảng dưới đây liệt kê các mã trạng thái giao hàng của GHTK, ánh xạ sang trạng thái đơn hàng trong hệ thống (OrderStatus).

|  |  |  |
| --- | --- | --- |
| Mã trạng thái GHTK | Trạng thái hệ thống (OrderStatus) | Mô tả trạng thái GHTK |
| 1 | PENDING\_CONFIRMATION | Đơn hàng vừa được tạo, chờ xác nhận từ người bán. |
| 2 | WAITING\_FOR\_SHIPMENT | Đơn hàng đã xác nhận, chuẩn bị giao cho vận chuyển. |
| 3 | WAITING\_FOR\_PICKUP | Đơn hàng sẵn sàng, chờ nhân viên GHTK đến lấy. |
| 5 | SHIPPING | Đơn hàng đang được vận chuyển đến khách hàng. |
| 6 | DELIVERED | Đơn hàng đã được giao thành công. |
| 8 | RETURN\_APPROVED | Đơn hàng được phê duyệt trả hàng. |
| 9 | RETURNED | Đơn hàng đã được trả lại kho hoặc người bán. |
| -2 | CANCELED | Đơn hàng bị hủy (do khách hàng hoặc người bán). |
| -1 | CANCELED | Đơn hàng không được chấp nhận (lỗi dữ liệu, địa chỉ sai, thiếu thông tin, v.v.). |
| 4 | CANCELED | Đơn hàng bị hủy trong quá trình lấy hàng. |
| 7 | CANCELED | Đơn hàng bị hủy trong quá trình giao hàng. |
| 10 | CANCELED | Đơn hàng bị hủy do lỗi vận chuyển hoặc không thể hoàn thành giao hàng. |
| (mặc định) | CANCELED | Mã trạng thái không xác định từ GHTK, coi như đơn hàng bị hủy. |

1. Chuyển đổi mã trạng thái từ GHTK sang bảng mã của hệ thống

Việc tích hợp quản lý đơn vận chuyển với GHTK thực sự rất dễ dàng áp dụng cho các doanh nghiệp không có bộ phận vận chuyển riêng. Đây là giải pháp lý tưởng giúp doanh nghiệp xử lý hiệu quả các nhu cầu vận chuyển mà không cần đầu tư xây dựng một đội ngũ vận chuyển nội bộ.

1. Hiện thực chương trình phần mềm
2. Cài đặt chương trình

Để triển khai hệ thống, cần cài đặt các công cụ và dịch vụ sau:

* Java Development Kit (JDK) phiên bản 17 trở lên để phát triển ứng dụng Spring Boot.
* Node.js phiên bản 18.x hoặc cao hơn để xây dựng ứng dụng ReactJS.
* Docker và Docker Compose để đóng gói và quản lý các container.
* Maven để quản lý các phụ thuộc của dự án Spring Boot.
* Các dịch vụ phụ trợ:
  + MySQL (phiên bản 8.0 hoặc cao hơn) để lưu trữ dữ liệu quan hệ.
  + Redis (phiên bản mới nhất) để quản lý bộ nhớ đệm và dữ liệu phiên.
  + Apache Kafka (phiên bản mới nhất) để xử lý hàng đợi tin nhắn và luồng dữ liệu.

1. Cấu hình và triển khai microservices với Spring Boot

Bước 1: Cấu hình maven trong pom.xml

Bước 2: Cấu hình các kết nối kafka, mysql, redis, kết nối từ api gateway tới các service... trong file application.yml.

Bước 3: Tạo và chạy file docker-compose.yml để khai báo các cài đặt mysql, kafka, redis.

Bước 4: Build project các microservices.

1. Cấu hình và triển khai reactjs

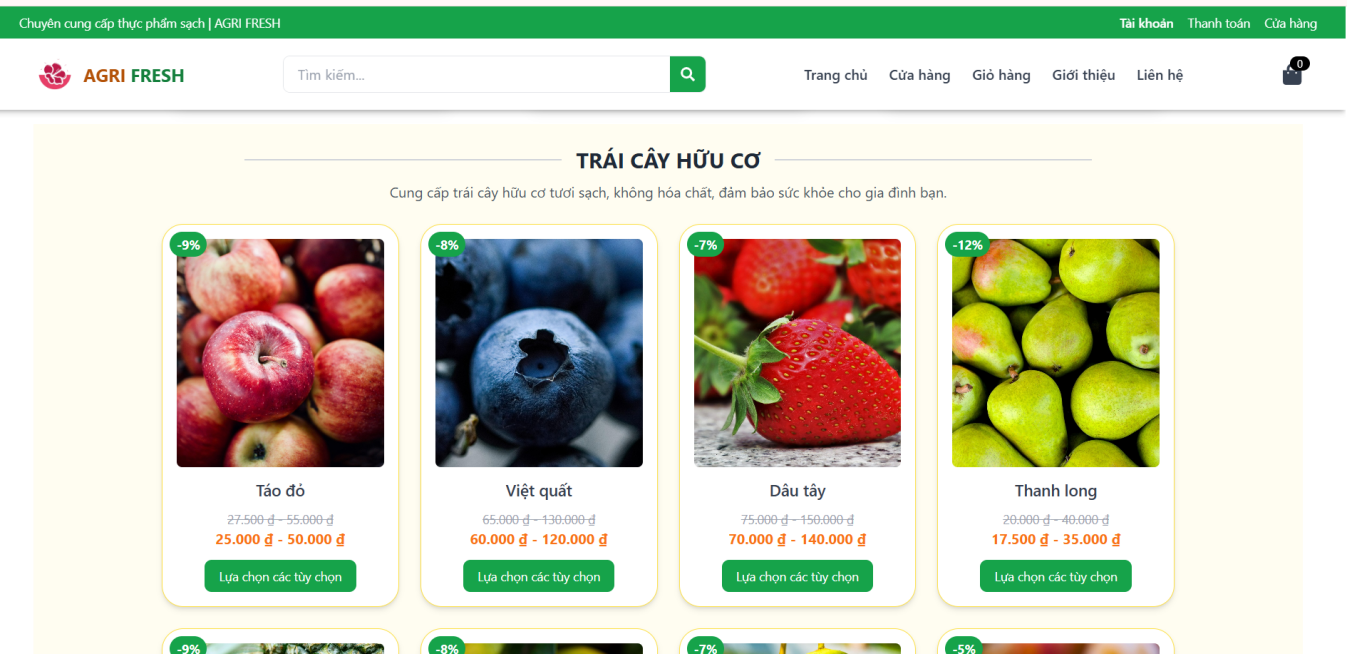
Bước 1: Dùng lệnh npx create-react-app để tạo project.

Bước 2: Cài đặt các dependency cần thiết như axios, react-router-dom, tailwind,...

Bước 3: Định nghĩa endpoint api trong file .env.

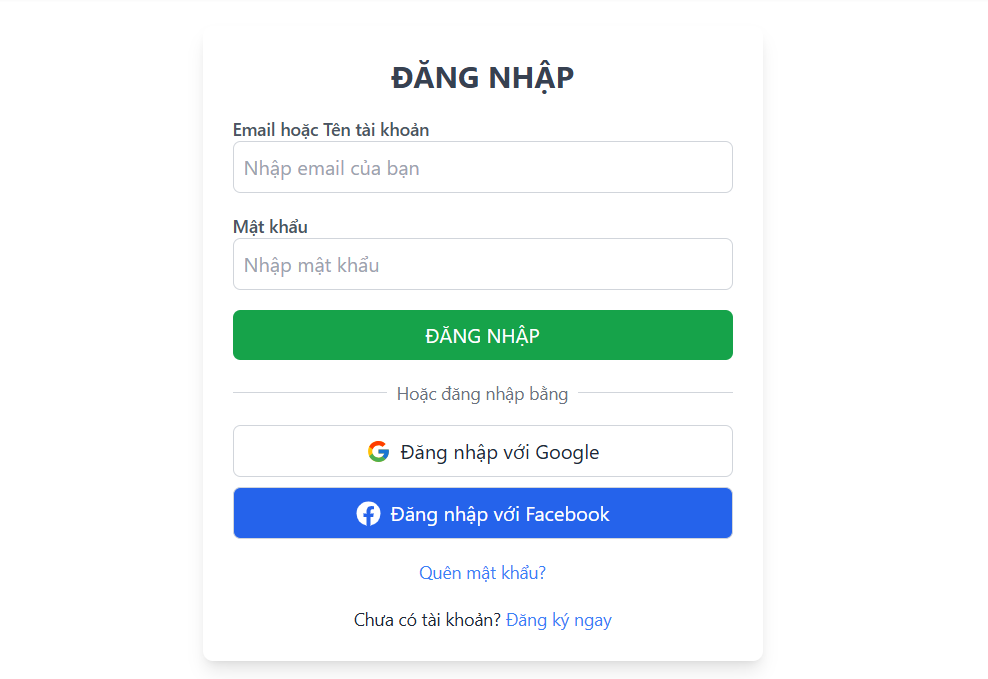
1. Giao diện chương trình
2. Giao diện trang chủ

Các thẻ sản phẩm hiển thị thông tin các tùy chọn gia gồm giá từ thấp nhất tới cao nhất tùy theo trọng lượng khách hàng muốn mua.



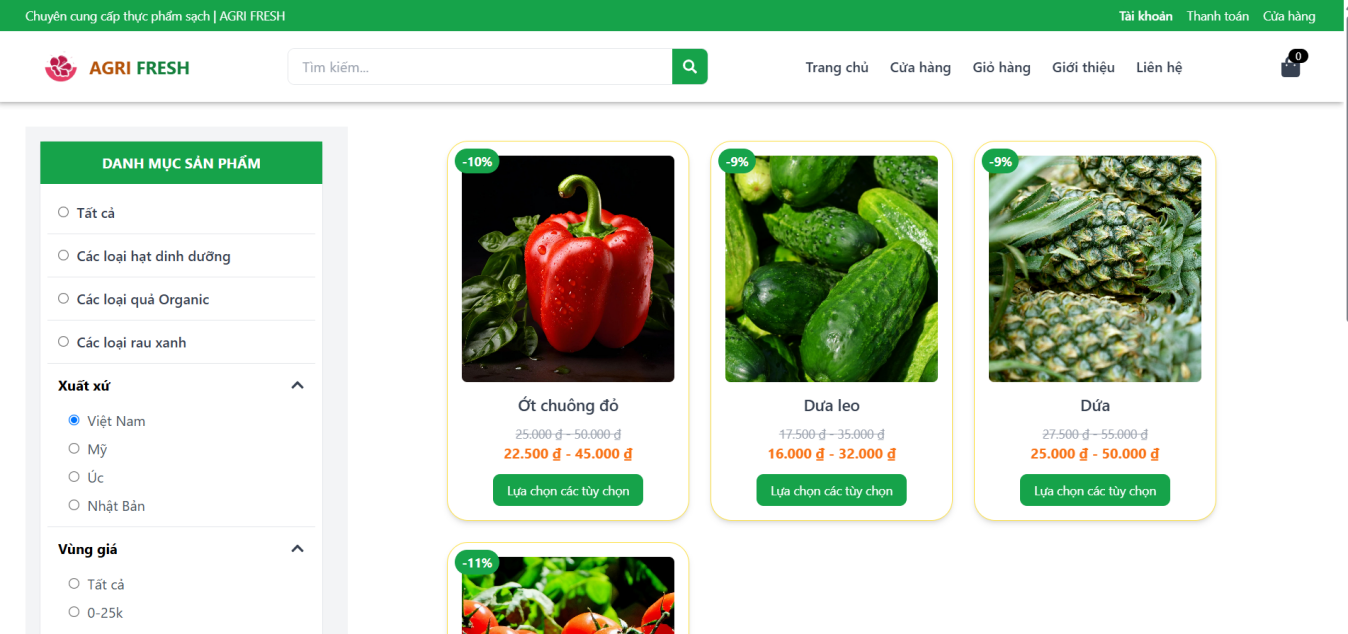
1. Giao diện trang chủ
2. Giao diện đăng nhập

Gồm có đăng nhập bằng tài khoản của hệ thống, hoặc đăng nhập bằng tài khoản social như facebook hoặc google.



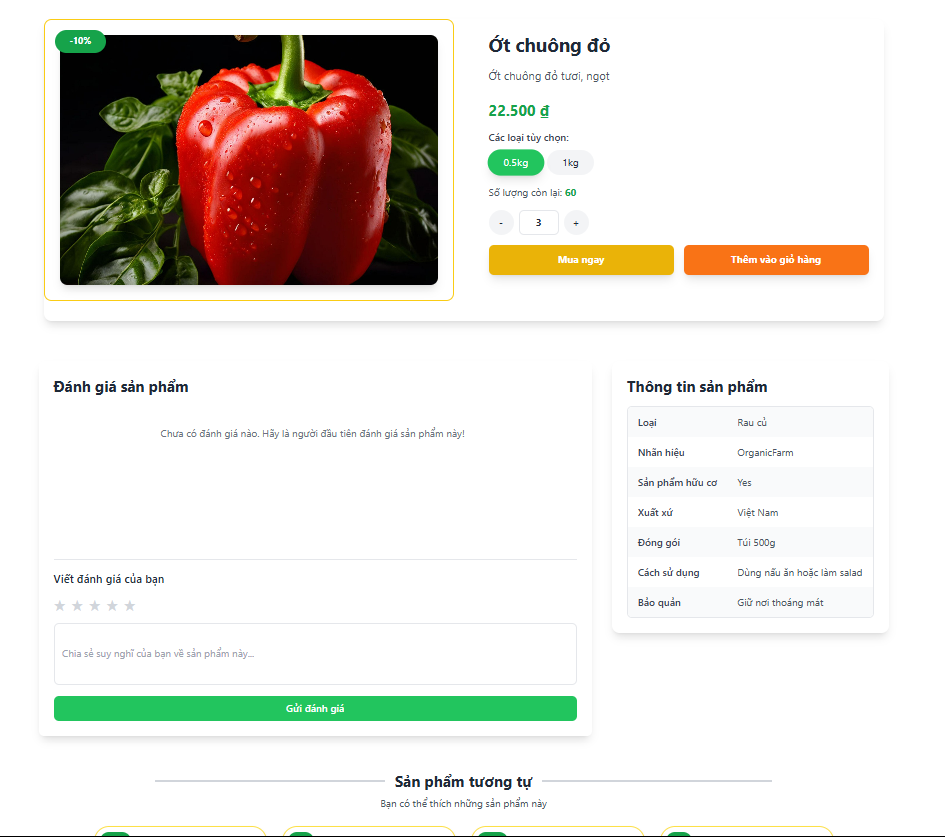
1. Giao diện đăng nhập
2. Giao diện cửa hàng

Tại đây khách hàng có thể tìm kiểm sản phẩm theo loại sản phẩm, xuất xử và giá cả.



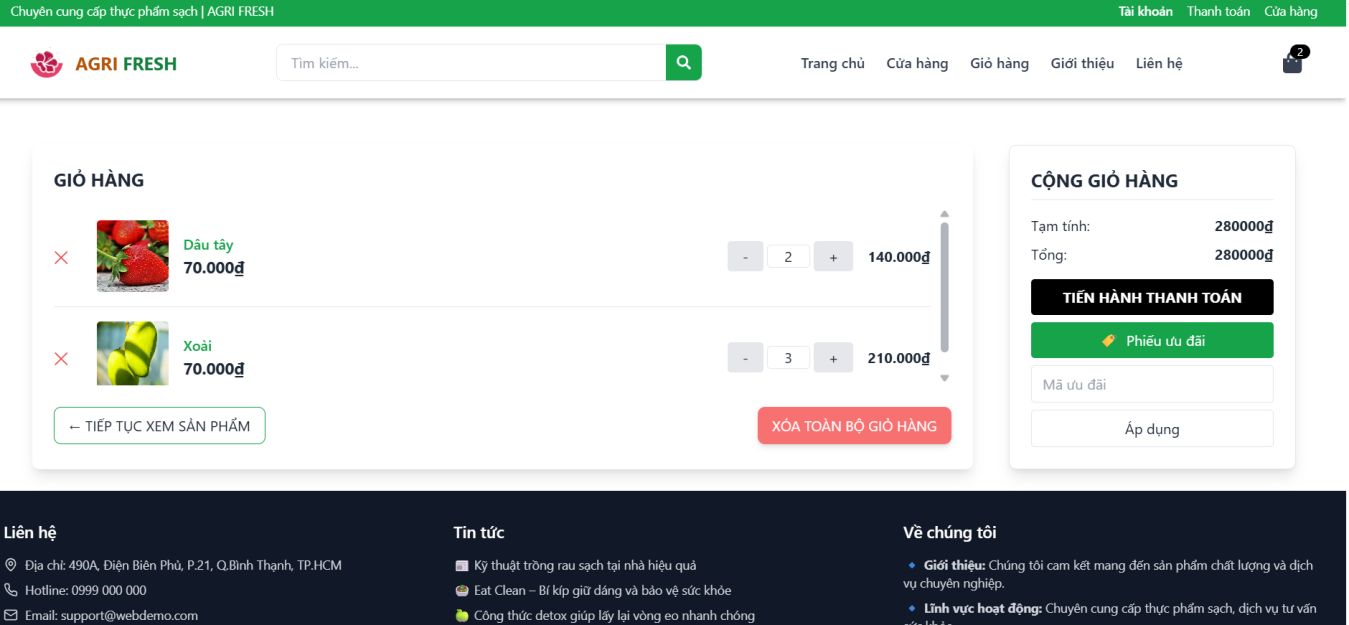
1. Giao diện cửa hàng
2. Giao diện chi tiết sản phẩm

Trang chi tiết sản phẩm sẽ thể hiện thông tin các trọng lượng tùy chọn và các số lượng còn lại, thông tin nhãn hiệu, suất xử, có phải sản phẩm hửu cơ không, cách bảo quản, và cách sử dụng. Tại đây người dùng có thể đánh giá, Bình luận sản phẩm



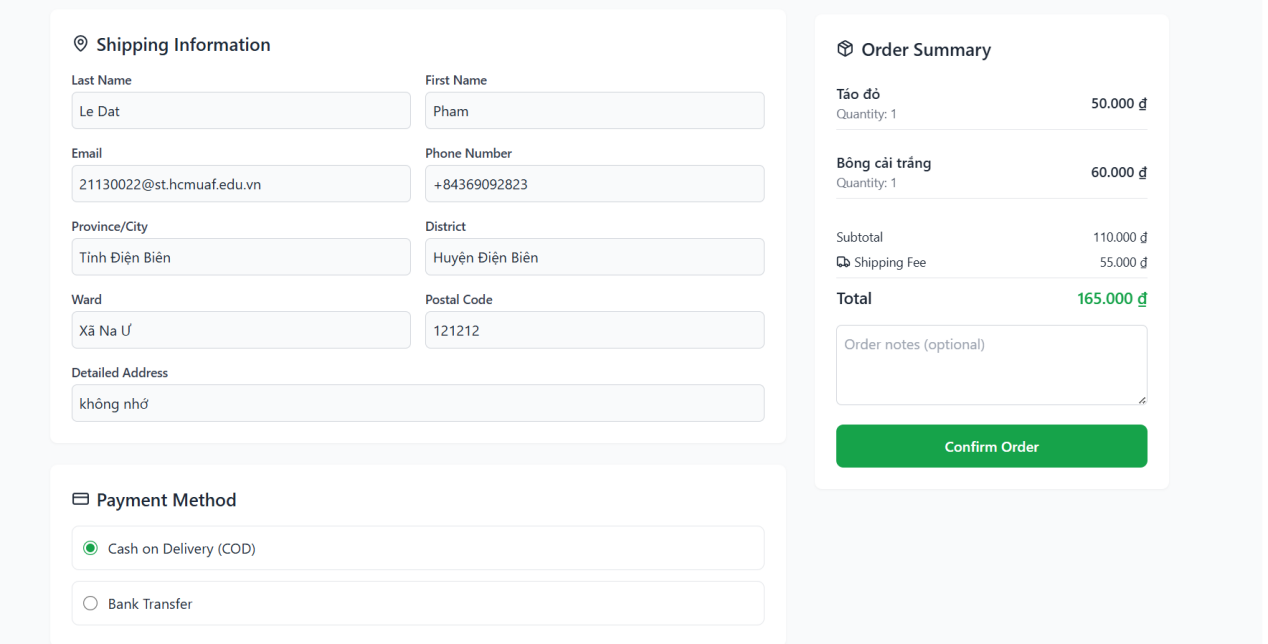
1. Giao diện chi tiết sản phẩm
2. Giao diện giỏ hàng

Trang giỏ hàng hiển thị các thông tin cơ bản của sản phẩm: hình ảnh, tên, giá, số lượng sản phẩm.



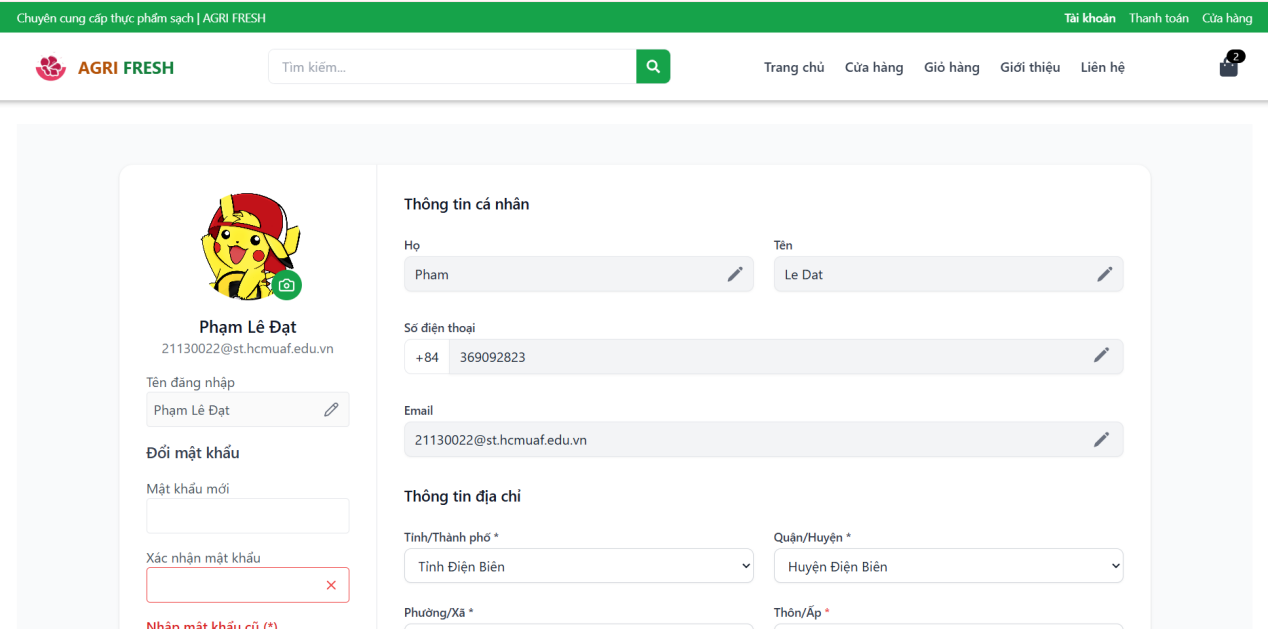
1. Giao diện giỏ hàng
2. Giao diện thanh toán

Giao diện thanh toán được chia thành hai khu vực chính, khu vực đầu tiên là thông tin vận chuyển với các trường nhập liệu như họ tên, email, số điện thoại, tỉnh/thành phố, quận/huyện, phường/xã, mã bưu chính, và địa chỉ chi tiết, được sắp xếp gọn gàng, khu vực thứ hai là tóm tắt đơn hàng hiển thị danh sách sản phẩm, số lượng, giá, phí vận chuyển, và tổng tiền, kèm lựa chọn phương thức thanh toán như COD hoặc chuyển khoản cùng nút xác nhận, bố cục này giúp khách hàng dễ dàng nhập thông tin và kiểm tra đơn hàng trước khi hoàn tất, tối ưu hóa trải nghiệm thanh toán.



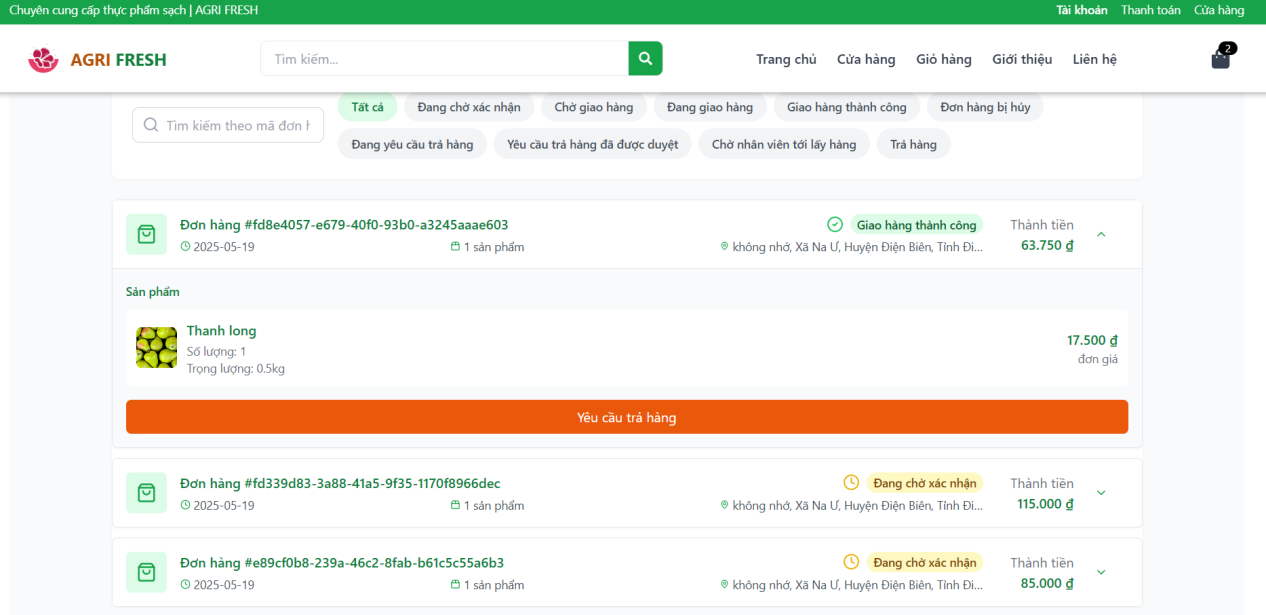
1. Giao diện thanh toán
2. Giao diện hồ sơ khách hàng

Giao diện hồ sơ thông tin hiển thị ảnh đại diện và thông tin cá nhân như họ tên, số điện thoại, email ở khu vực chính, tiếp theo là phần cập nhật thông tin bao gồm mật khẩu, địa chỉ với các trường chọn tỉnh/thành phố, quận/huyện, phường/xã, và xác nhận thay đổi, được sắp xếp gọn gàng với các nút hành động, giúp người dùng dễ dàng quản lý và chỉnh sửa thông tin cá nhân một cách nhanh chóng, đảm bảo bảo mật và tối ưu hóa trải nghiệm sử dụng.

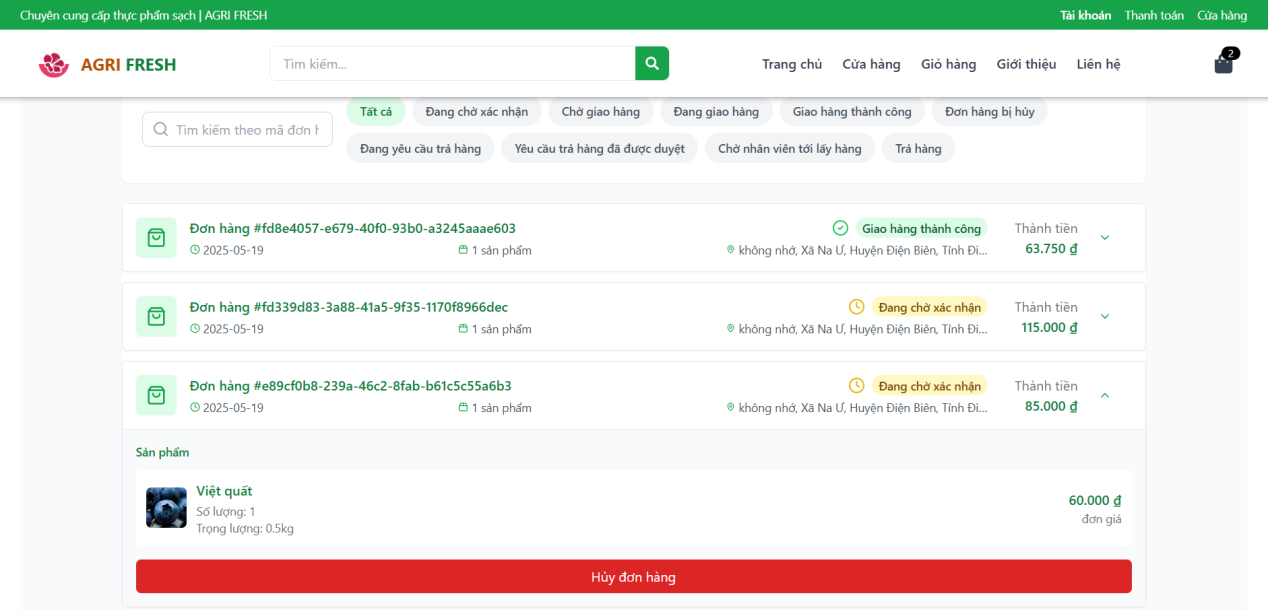


1. Giao diện hồ sơ thông tin
2. Giao diện lịch sử đơn hàng

Tại đây, khách hàng có thể thực hiện các chức năng xem thông tin, trạng thái của các đơn hàng đã đặt, hủy đơn hàng,...

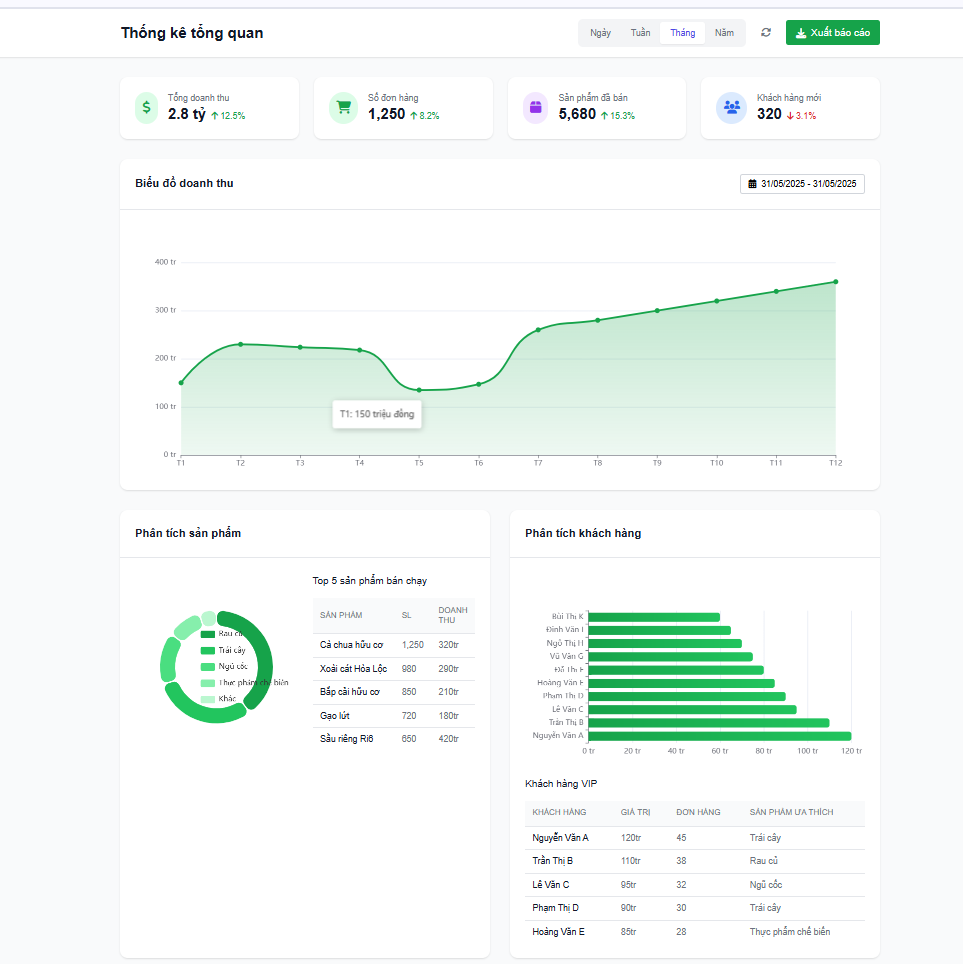


1. Giao diện lịch sử đơn hàng



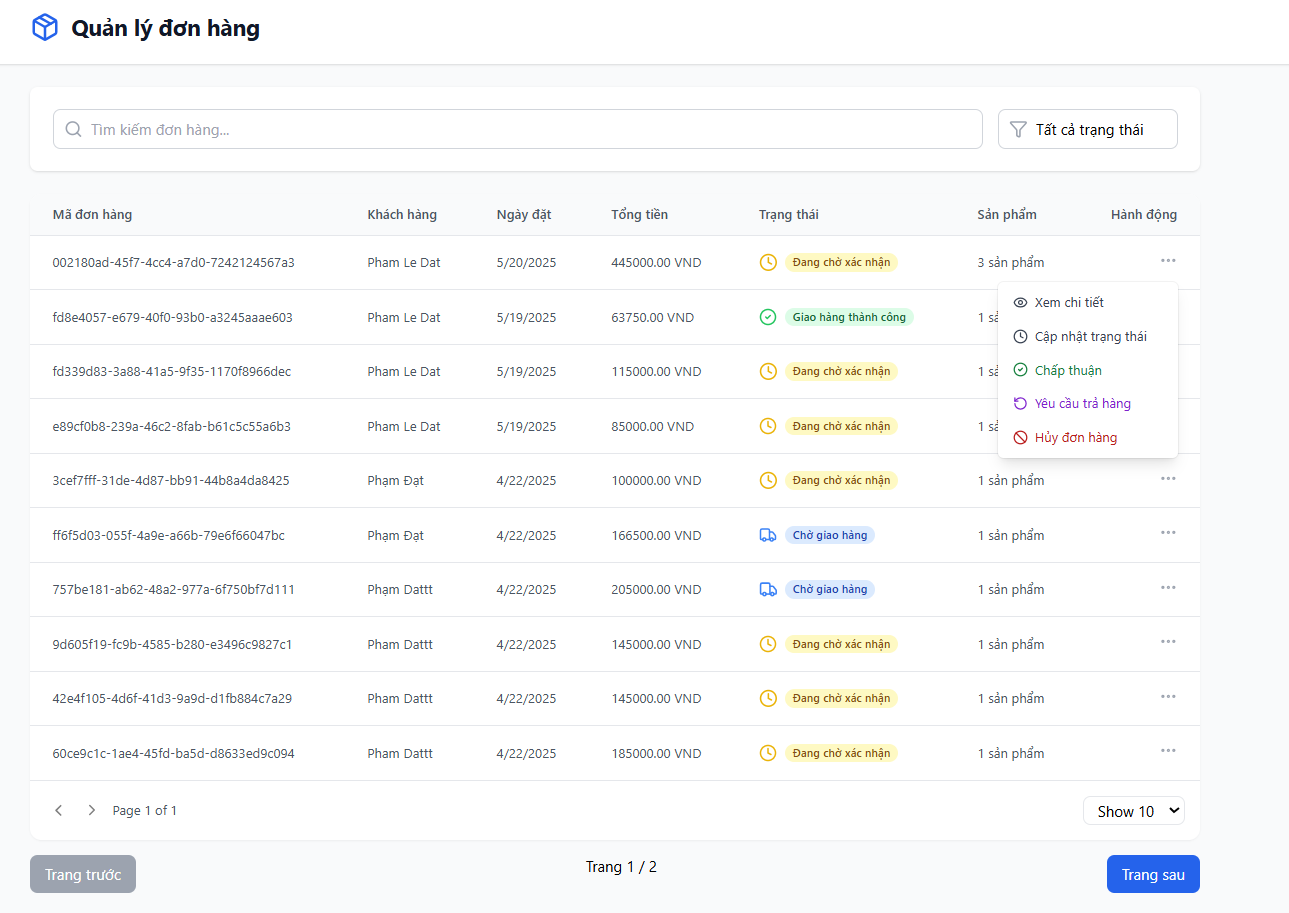
1. Giao diện lịch sử đơn hàng
2. Giao diện thống kê

Trang thống kê hiển thị các biểu đồ thể hiện doanh thu, bảng xếp hạng các loại sản phẩm, bảng xếp loại khách hàng. Quản trị viên có thể xuất thông bao doanh thu nều cần.



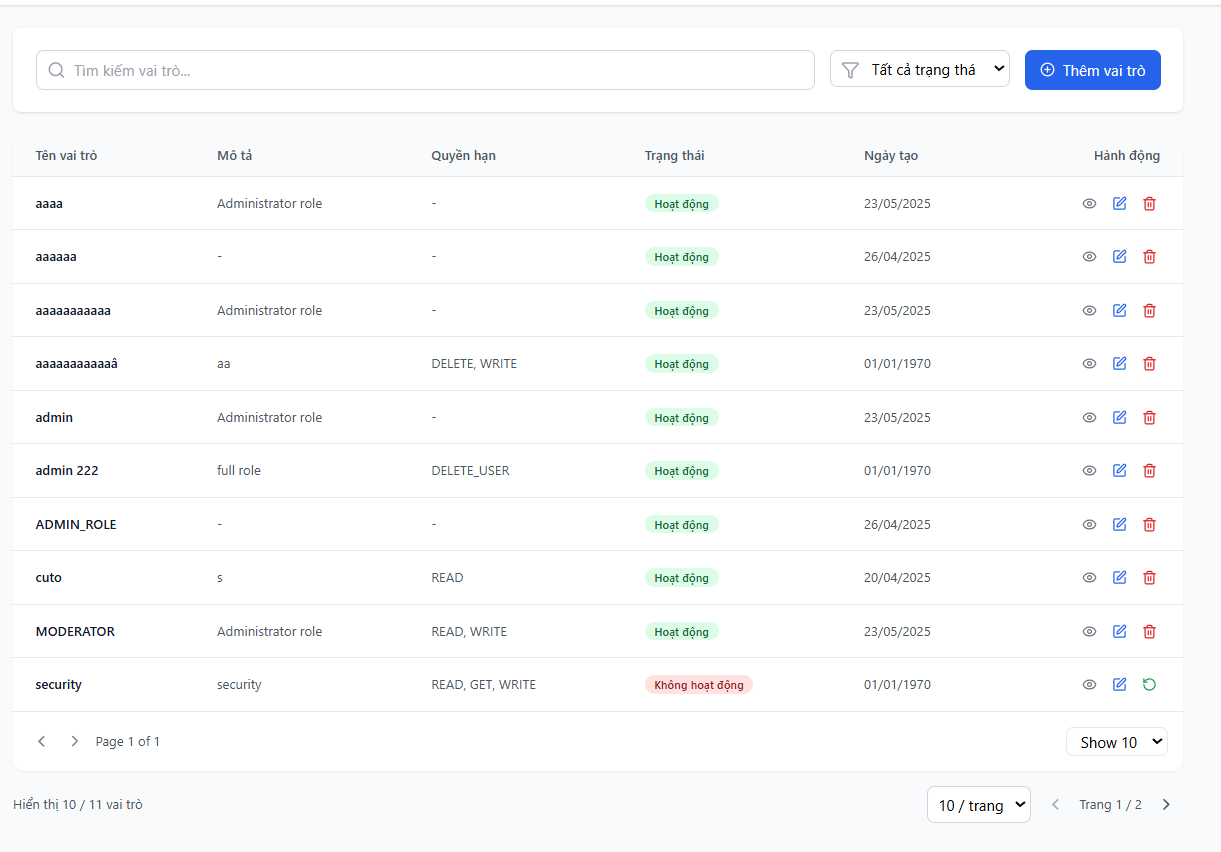
1. Giao diện thống kê
2. Giao diện quản lý đơn hàng

Giúp quản trị viên có thể theo dõi và cập nhật các trạng thái đơn hàng



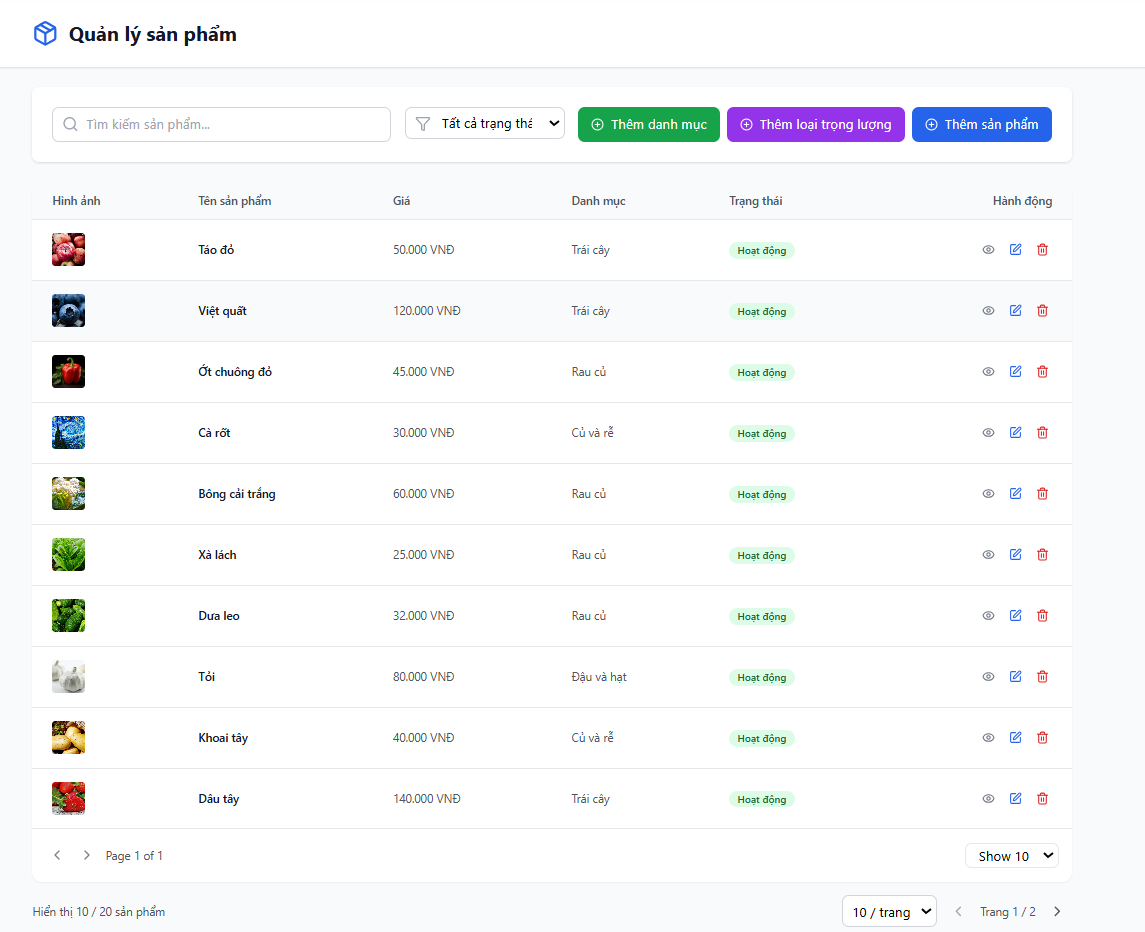
1. Giao diện quản lý đơn hàng
2. Giao diện quản lí phân quyền

Giúp quản trị viên gắm phân quyền, khóa và khôi phục phân quyền cho các tài khoản người dùng.

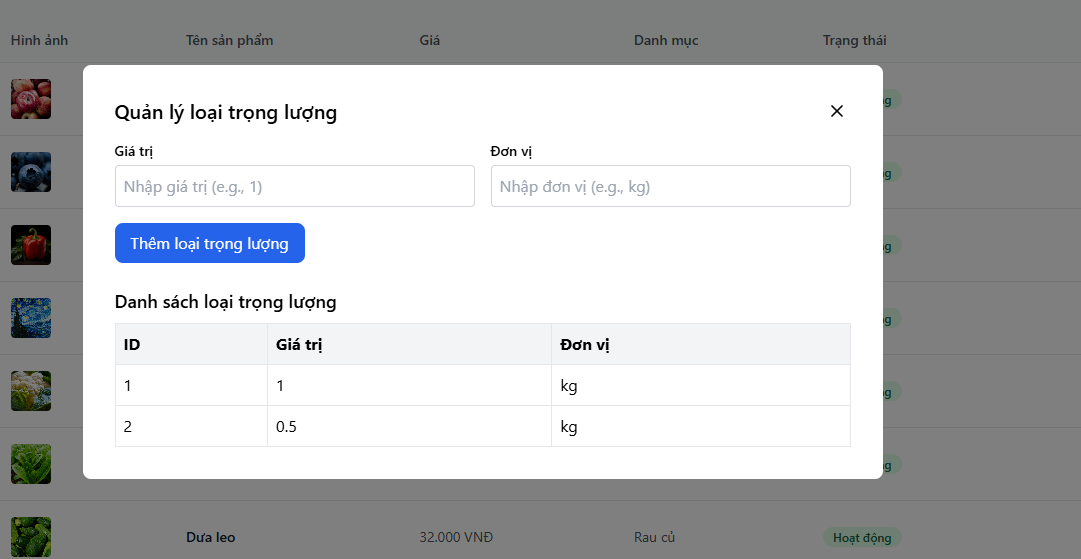


1. Giao diện quản lí phân quyền
2. Giao diện quản lý sản phẩm

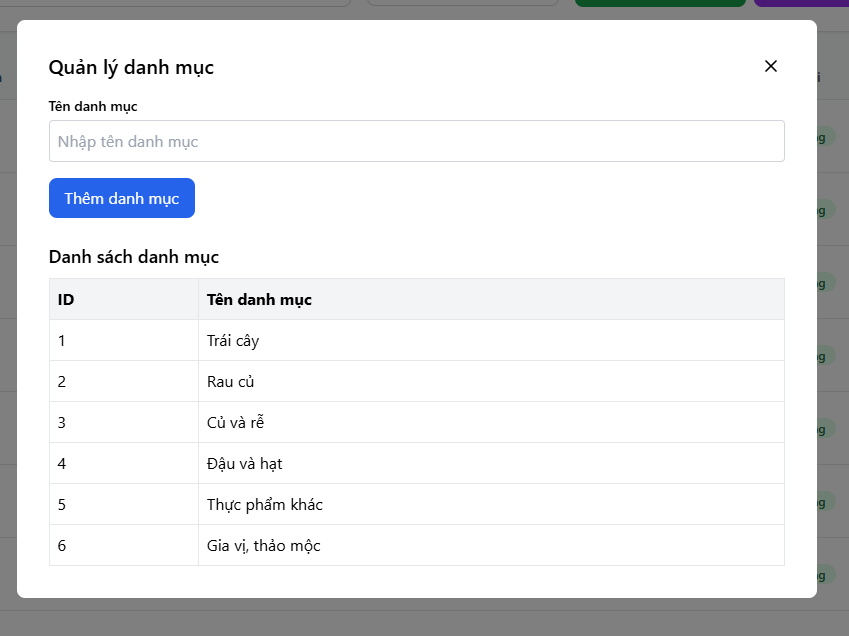
Quản trị viên có thể xem xét, chỉnh sửa, tạo, ẩn sản phẩm tại, thêm các loại sản phẩm, thêm các tùy chọn trọng lượng cần thiết tại trang này.



1. Giao diện quản lý sản phẩm

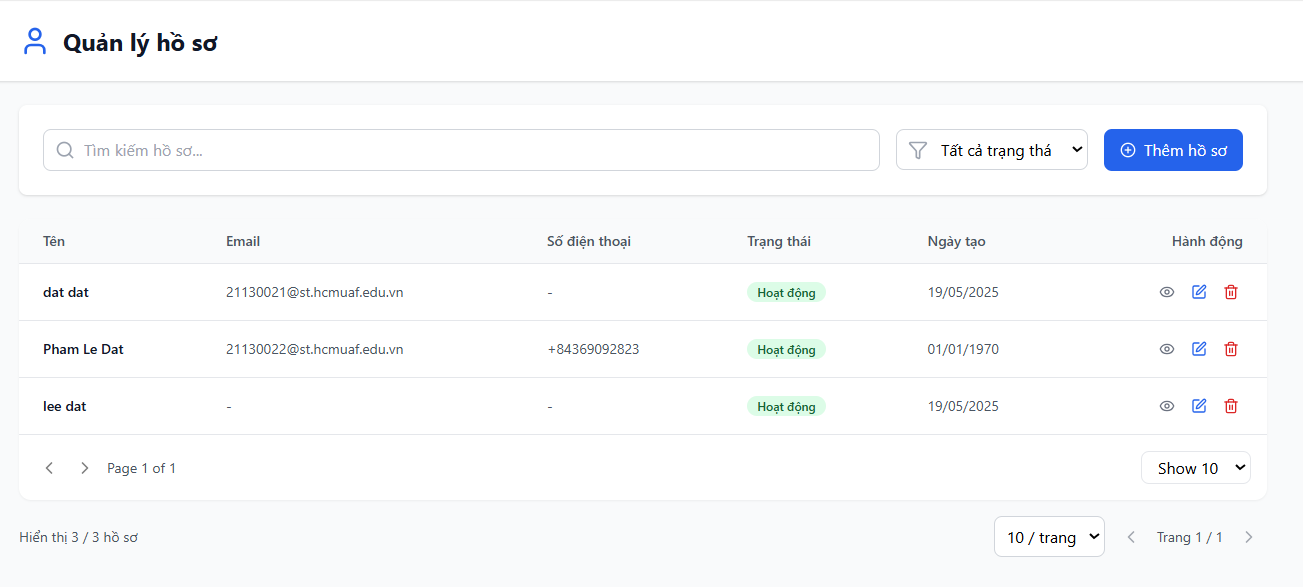


1. Giao diện thêm phân loại trọng lượng



1. Giao diện thêm loại sản phẩm
2. Giao diện quản lí hồ sơ

Thể hiện các thông tin cơ bản của hồ sơ, quản trị viên có thể thêm, tùy chỉnh thông tin khách hàng, ẩn và khôi phục các hồ sơ đó.



1. Giao diện quản lý hồ sơ
2. Đánh giá hiệu năng hệ thống

Tôi đã tiến hành kiểm thử các api gồm: đăng nhập, tiềm kiếm và lấy sản phảm bởi id loại sản phẩm, kết quả như sau:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Số mẫu dữ liệu trong bảng** | **Số request** | **Thời gian trung bình (ms)** | **Max (ms)** | **Tỷ lệ lỗi (%)** | **Throughput (req/s)** |
| 1000 | 10 | 106.0 | 150.0 | 0.0 | 10.03 |
| 1000 | 60 | 192.0 | 261.0 | 0.0 | 1.52 |
| 1000 | 100 | 654.0 | 1143.0 | 0.0 | 47.76 |
| 1000 | 200 | 1928.0 | 3443.0 | 0.0 | 44.98 |
| 1000 | 500 | 5960.0 | 10912.0 | 0.0 | 40.88 |
| 1000 | 1000 | 10977.0 | 21220.0 | 0.0 | 42.14 |

1. Kiểm thử truy cập đồng thời endpoint log-in

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Số mẫu dữ liệu trong bảng** | **Số request** | **Thời gian trung bình (ms)** | **Max (ms)** | **Tỷ lệ lỗi (%)** | **Throughput (req/s)** |
| 500 | 10 | 4.0 | 6.0 | 0.0 | 11.19 |
| 500 | 60 | 7.0 | 16.0 | 0.0 | 1.52 |
| 500 | 100 | 11.0 | 64.0 | 0.0 | 51.57 |
| 500 | 200 | 10.0 | 33.0 | 0.0 | 47.36 |
| 500 | 500 | 12.0 | 125.0 | 0.0 | 41.63 |
| 500 | 1000 | 19.0 | 388.0 | 0.0 | 42.75 |

1. Kiểm thử truy cập đồng thời endpoint log-in

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Số mẫu dữ liệu trong bảng** | **Số request** | **Thời gian trung bình (ms)** | **Max (ms)** | **Tỷ lệ lỗi (%)** | **Throughput (req/s)** |
| 500 | 10 | 5.0 | 7.0 | 0.0 | 11.15 |
| 500 | 60 | 7.0 | 14.0 | 0.0 | 1.52 |
| 500 | 100 | 11.0 | 74.0 | 0.0 | 51.31 |
| 500 | 200 | 10.0 | 38.0 | 0.0 | 47.16 |
| 500 | 500 | 13.0 | 175.0 | 0.0 | 41.60 |
| 500 | 1000 | 21.0 | 367.0 | 0.0 | 42.73 |

1. Kiểm thử truy cập đồng thời endpoint log-in

Dựa trên kết quả kiểm thử hiệu năng với số lượng người dùng đồng thời tăng dần từ 10 đến 1000, tôi chia hệ thống thành ba ngưỡng vận hành rõ ràng:

Ở ngưỡng dưới 100 người dùng, hệ thống hoạt động ổn định. Tất cả các API, bao gồm cả login, get products by category, và filter product, đều có thời gian phản hồi thấp (thường dưới 1 giây), throughput cao và không phát sinh lỗi. Đây là mức tải phù hợp để hệ thống vận hành chính thức mà không cần tối ưu thêm.

Khi số lượng người dùng tăng từ 100 đến 500, hệ thống bắt đầu có dấu hiệu chậm lại, đặc biệt là ở API login. Thời gian phản hồi trung bình của login tăng từ khoảng 650ms lên gần 6 giây, và thời gian tối đa có thể lên tới hơn 10 giây. Dù vậy, các API đọc vẫn duy trì hiệu năng tốt. Đây là ngưỡng cảnh báo – hệ thống vẫn hoạt động được nhưng cần theo dõi sát sao, đặc biệt là tài nguyên máy chủ và các cơ chế xác thực trong API login.

Ở ngưỡng trên 500 người dùng, đặc biệt là ở mức 1000, hệ thống có dấu hiệu quá tải rõ rệt. API login phản hồi rất chậm, với thời gian trung bình lên tới hơn 10 giây và tối đa tới hơn 21 giây. Mặc dù không xuất hiện lỗi logic, thời gian chờ cao như vậy sẽ ảnh hưởng nghiêm trọng đến trải nghiệm người dùng và tiềm ẩn nguy cơ timeout. Đây là ngưỡng mà hệ thống không nên vận hành nếu chưa có cải tiến về hiệu năng, đặc biệt cần tối ưu xử lý login, giảm truy vấn, hoặc bổ sung cơ chế cache/token.

Tóm lại, hệ thống hiện tại đáp ứng tốt tải dưới 100 người dùng, bắt đầu cần giám sát ở mức 100–500 người, và nên tránh vận hành trên 500 người dùng đồng thời nếu chưa tối ưu thêm.

1. KẾT QUẢ, KẾT LUẬN VÀ KIẾN NGHỊ
2. KẾT QUẢ

Sau một thời gian nhóm nghiên cứu và thực hiện đề tài “ỨNG DỤNG CÁC CÔNG NGHỆ REACT, SPRING BOOT, REDIS TRONG KIẾN TRÚC MICROSERVICE ĐỂ XÂY DỰNG WEBSITE BÁN NÔNG SẢN TRỰC TUYẾN”, nhóm em đã đạt được những kết quả như sau:

1. Kiến thức đã tìm hiểu được

Trong quá trình thực hiện đồ án, tôi đã tìm hiểu và nắm bắt được nhiều kiến thức quan trọng liên quan đến kiến trúc hệ thống và công nghệ hiện đại.

Về mặt kiến trúc, tôi đã tập trung nghiên cứu mô hình Microservices, cho phép xây dựng hệ thống bằng cách chia nhỏ thành các dịch vụ độc lập. Qua quá trình học tập và triển khai thực tế, tôi đã hiểu rõ cách tổ chức các service riêng biệt, giao tiếp với nhau thông qua FeignClient hoặc Kafka, đồng thời nắm được các nguyên tắc thiết kế nhằm đảm bảo tính độc lập, khả năng mở rộng và dễ bảo trì. Việc áp dụng Microservices giúp tôi xây dựng được một hệ thống linh hoạt, có thể phát triển theo từng chức năng riêng lẻ mà không ảnh hưởng đến tổng thể. Tôi cũng đã tìm hiểu và so sánh với kiến trúc Monolithic để thấy rõ điểm khác biệt: nếu như monolithic dễ triển khai ở quy mô nhỏ nhưng khó mở rộng, thì microservices hỗ trợ tốt hơn trong môi trường nhiều người dùng và hệ thống lớn, nhờ khả năng tách biệt và triển khai độc lập từng module.

Để hiện thực hóa kiến trúc này, tôi đã lựa chọn sử dụng Spring Boot, một framework mạnh mẽ của Java. Tôi đã học cách xây dựng các API RESTful, cấu hình các service độc lập, kết nối cơ sở dữ liệu, bảo mật hệ thống với Spring Security, và xử lý các luồng dữ liệu bằng Spring Data JPA. Qua đó, tôi hiểu rõ hơn về quy trình xây dựng backend hiện đại theo chuẩn Microservices.

Về phía giao diện người dùng, tôi sử dụng React để xây dựng frontend cho hệ thống. Tôi đã học cách thiết kế các component, quản lý state với các hook như useState, useEffect, điều hướng với React Router, và tích hợp API để hiển thị dữ liệu từ backend. Tôi đã phát triển đầy đủ các trang như: giao diện cho khách hàng mua hàng, tìm kiếm sản phẩm, và giao diện quản trị cho người bán. Việc sử dụng React giúp tôi xây dựng giao diện người dùng mượt mà, phản hồi nhanh và dễ mở rộng.

Bên cạnh đó, tôi cũng đã tìm hiểu và tích hợp Redis – một hệ thống lưu trữ dữ liệu dạng key-value trên bộ nhớ – vào ứng dụng. Redis giúp tôi cache dữ liệu, giảm tải truy vấn cơ sở dữ liệu và tối ưu hiệu năng, đặc biệt trong các chức năng như đăng nhập, truy xuất sản phẩm, lọc dữ liệu,... Việc áp dụng Redis giúp tôi hiểu thêm về các kỹ thuật tối ưu hóa hệ thống trong môi trường phân tán và thời gian thực.

1. Kết quả phát triển giao diện người dùng

Giao diện lịch sử đơn hàng được thiết kế với bố cục dạng bảng, hiển thị danh sách các đơn hàng với mã đơn, ngày đặt, số lượng sản phẩm, trạng thái như giao hàng thành công hoặc đang xử lý, địa chỉ giao hàng, và tổng thanh toán, kèm các nút hành động như yêu cầu trả hàng, hủy đơn hàng hoặc xem chi tiết, được sắp xếp gọn gàng, giúp khách hàng dễ dàng theo dõi và quản lý các đơn hàng đã đặt, nâng cao hiệu quả và tiện lợi trong trải nghiệm sử dụng.

1. ****Chương trình đã làm được****

Sau khi đã hoàn chỉnh ứng dụng, nhóm đã hoàn thiện được những điểm sau:

* Người dùng mua hàng: Hệ thống đảm bảo các tiêu chí tiêu chuẩn của một trang web bán hàng trực tuyến, bao gồm như đăng nhập, đăng ký, quên mật khẩu, thay đổi thông tin cá nhân, tạo và cập nhật địa chỉ giao hàng, tìm kiếm và lọc sản phẩm, đặt hàng (hỗ trợ sử dụng thanh toán VNPay, và GHTK để giao hàng), theo dõi và cập nhật trạng thái đơn hàng, đánh giá sản phẩm, hủy đơn hàng.
* Người quản trị viên: Quản lí sản phẩm, quản lí đơn hàng, quản lí người dùng, phân quyền, quản lí hồ sơ, xem, xuất báo cáo doanh thu, theo dõi được các sản phẩm đang bán chạy và các khách hàng tiềm năng.

1. Hiệu suất hệ thống

Sau khi xây dựng và hoàn thiện ứng dụng, tôi đã tiến hành kiểm thử hiệu năng hệ thống bằng công cụ Apache JMeter, tập trung vào các chức năng quan trọng như đăng nhập, lấy danh mục theo ID, và lọc sản phẩm – những API có tần suất truy cập cao và ảnh hưởng trực tiếp đến trải nghiệm người dùng.

Kết quả thử nghiệm cho thấy hệ thống vận hành ổn định và phản hồi nhanh trong khoảng từ 10 đến 100 người dùng đồng thời. Trong mức này, các API hoạt động mượt mà, không phát sinh lỗi, thời gian phản hồi trung bình duy trì ở mức thấp. Khi mở rộng quy mô lên 200 đến 500 người dùng, hệ thống bắt đầu ghi nhận hiện tượng trễ nhẹ tại một số API như đăng nhập và lọc sản phẩm; tuy nhiên, tốc độ phản hồi vẫn nằm trong ngưỡng chấp nhận được đối với một ứng dụng thương mại điện tử cỡ trung.

Khi số lượng người dùng đồng thời vượt qua 500, đặc biệt ở mức 1000, hệ thống bắt đầu xuất hiện dấu hiệu quá tải rõ rệt. Cụ thể, thời gian phản hồi tăng cao, có thời điểm vượt quá giới hạn timeout ở phía frontend, dẫn đến trải nghiệm người dùng bị ảnh hưởng. Đây là dấu hiệu cảnh báo về giới hạn chịu tải của hệ thống trong điều kiện hiện tại.

Từ kết quả kiểm thử, tôi có thể đưa ra ba ngưỡng đánh giá hiệu suất hệ thống:

* < 100 người dùng: Hệ thống vận hành ổn định, phản hồi tốt.
* 100 – 500 người dùng: Hệ thống có dấu hiệu tải tăng, cần bắt đầu theo dõi hiệu suất và tối ưu.
* > 500 người dùng: Hệ thống tiệm cận hoặc vượt quá ngưỡng chịu tải, cần mở rộng tài nguyên hoặc tối ưu kiến trúc để duy trì ổn định.

1. Những điểm hạn chế

Trong quá trình thực hiện đồ án, tôi nhận thấy bản thân vẫn còn tồn tại nhiều hạn chế cần khắc phục. Trước hết, trình độ lập trình còn hạn chế khiến việc triển khai một số chức năng phức tạp mất nhiều thời gian, đặc biệt là trong giai đoạn đầu tiếp cận với các công nghệ mới như Microservices, Spring Boot và React. Ngoài ra, kiến thức về mô hình kinh doanh trực tuyến còn nông, dẫn đến việc thiết kế hệ thống ban đầu chưa sát với thực tế và phải tốn nhiều thời gian để điều chỉnh.

Một khó khăn đáng kể trong quá trình phát triển là việc tích hợp API của Giao Hàng Tiết Kiệm (GHTK). Tài liệu hướng dẫn từ phía GHTK còn hạn chế, không đầy đủ và thiếu ví dụ minh họa cụ thể. Tôi đã chủ động tham gia các nhóm hỗ trợ cộng đồng trên Telegram để nhờ tư vấn, tuy nhiên vẫn gặp phải tình trạng một số API của GHTK không hoạt động trong môi trường thử nghiệm (sandbox). Điều này khiến tôi tốn rất nhiều thời gian để tìm kiếm giải pháp thay thế, kiểm tra thủ công hoặc mô phỏng các dữ liệu cần thiết để đảm bảo hệ thống vận hành mạch lạc.

Bên cạnh đó, thiếu kinh nghiệm trong việc phát triển một hệ thống hoàn chỉnh khiến tôi gặp khó khăn trong khâu phân chia module, tổ chức mã nguồn, tối ưu hiệu năng cũng như xử lý lỗi ở các tình huống thực tế. Đây là những điểm yếu mà tôi ghi nhận để tiếp tục học hỏi, rèn luyện và cải thiện trong các dự án sau này.

**TÀI LIỆU THAM KHẢO**

[1] Bai, R., Song, X., With the advancement and implementation of information technology, microservice architecture has gained popularity in current times, Proceedings of the 5th International Conference on Information Science, Parallel and Distributed Systems (ISPDS), Guangzhou, China, May 31–June 2, 2024.

[2] Nayim, N. N., Karmakar, A., Ahmed, M. R., Saifuddin, M., Kabir, M. H., Performance Evaluation of Monolithic and Microservice Architecture for an E-commerce Startup, Proceedings of the 26th International Conference on Computer and Information Technology (ICCIT), Cox's Bazar, Bangladesh, December 13–15, 2023.

[3] Carnell, J., Sánchez, I., Spring Microservices in Action, 2nd Edition, Manning Publications, Shelter Island, New York, 2021.

[4] Rajput, R., Spring Boot Microservices with Spring Cloud, Packt Publishing, Birmingham, 2020.

[5] Sharma, S., "Securing Microservices with Spring Security and JWT", Proceedings of the International Conference on Cloud Computing and Services, IEEE CloudCom, Singapore, December 10–12, 2022.

[6] Tikhonov, A., Solutions of Ill-Posed Problems in Microservice Architectures, Wiley, New York, 1997.

[7] Paradopop, B.K., "Fuzzy Sets and Relational Structures in Microservice Authentication", Proceedings of the First International Workshop on Distributed Systems, Thessaloniki, Greece, October 16–20, 1998.