|  |
| --- |
| **BỘ GIÁO DỤC VÀ ĐÀO TẠO**  **TRƯỜNG ĐẠI H ỌC NÔNG LÂM TP HCM**  **KHOA CÔNG NGHỆ THÔNG TIN**        **TIỂU LUẬN TỐT NGHIỆP**      **XÂY DỰNG NỀN TẢNG THƯƠNG MẠI ĐIỆN TỬ THIẾT BỊ ĐIỆN TỬ TÍCH HỢP TRÍ TUỆ NHÂN TẠO VÀ CHATBOT SỬ DỤNG SPRING BOOT, REACTJS VỚI HỆ THỐNG GỢI Ý SẢN PHẨM CÁ NHÂN HÓA**      **Ngành : Công nghệ thông tin**  **Niên khoá : 2022 – 2026**  **Lớp : DH22DTB**  **Sinh viên thực hiện : Từ Quang Chương** |

|  |
| --- |
| **BỘ GIÁO DỤC VÀ ĐÀO TẠO**  **TRƯỜNG ĐẠI HỌC NÔNG LÂM TP HCM KHOA CÔNG NGHỆ THÔNG TIN**        TIỂU LUẬN TỐT NGHIỆP  **XÂY DỰNG NỀN TẢNG THƯƠNG MẠI ĐIỆN TỬ THIẾT BỊ ĐIỆN TỬ TÍCH HỢP TRÍ TUỆ NHÂN TẠO VÀ CHATBOT SỬ DỤNG SPRING BOOT, REACTJS VỚI HỆ THỐNG GỢI Ý SẢN PHẨM CÁ NHÂN HÓA**      **CÁN BỘ HƯỚNG DẪN SINH VIÊN THỰC HIỆN**  TS. Nguyễn Thị Phương TrâmTừ Quang Chương (MSSV: 22130029)      TP.HỒ CHÍ MINH, tháng 06 năm 2024 |

# DANH SÁCH CHỮ VIẾT TẮT

|  |  |
| --- | --- |
| EVM | **E**thereum **V**irtual **M**achine |
|  | Máy ảo Ethereum |
| ETH | **E**ther |
|  | Đơn vị tiền tệ của Ethereum |
| PoW | **P**roof **o**f **W**ork |
|  | Bằng chứng công việc |
| PoS | **P**roof **o**f **S**take |
|  | Bằng chứng cổ phần |
| dApp | **D**ecentralized **App**lication |
|  | Ứng dụng phi tập trung |
| JWT | **J**son **W**eb **T**oken |
|  | Token web JSON |
| REST | **RE**presentational **S**tate **T**ransfer |
|  | Chuyển đổi trạng thái biểu diễn |
| API | **A**pplication **P**rogramming **I**nterface |
|  | Giao diện lập trình ứng dụng |
| CRUD **C**reate, **R**ead, **U**pdate, **D**elete  Tạo, Đọc, Cập nhật, Xóa  SDK **S**oftware **D**evelopment **K**it  Bộ công cụ phát triển phần mềm  QR **Q**uick **R**esponse  Phản hồi nhanh | |

# DANH MỤC HÌNH ẢNH

Hình 1. Các sáng chế ứng dụng blockchain trong các lĩnh vực ................................. 2

Hình 2. Cấu trúc của blockchain ................................................................................ 9

Hình 3. Phân loại blockchain theo quyền truy cập ................................................... 11

Hình 4. Cấu trúc MERN stack ................................................................................. 17

Hình 5. Cách thức hoạt động Restful API ................................................................ 21

Hình 6. Lược đồ activity cho usecase tạo cuộc bình chọn ....................................... 43

Hình 7. Lược đồ activity cho usecase thêm ứng viên .............................................. 44

Hình 8. Lược đồ activity usecase bình chọn. ........................................................... 45

Hình 9. Lược đồ activity cho usecase bình chọn bằng QR Code ............................. 46

Hình 10. Lược đồ activity cho usecase tham gia cuộc bình chọn. ........................... 47

Hình 11. Lược đồ tuần tự cho usecase tạo cuộc bình chọn. ..................................... 48

Hình 12. Lược đồ tuần tự cho usecase thêm ứng viên. ............................................ 49

Hình 13. Lược đồ tuần tự cho usecase bình chọn. ................................................... 49

Hình 14. Lược đồ tuận tự cho usecase bình chọn bằng QR Code. ........................... 50

Hình 15. Lược đồ tuần tự cho usecase tham gia cuộc bình chọn. ............................ 50

Hình 16. Lược đồ cơ sở dữ liệu của hệ thống. ......................................................... 64

Hình 17. Các giao được hợp đồng thông minh thực thi. .......................................... 67

Hình 18. Chi tiết một giao dịch trên blockchain. ..................................................... 68

Hình 19. Màn hình đăng nhập của hệ thống. ........................................................... 51

Hình 20. Màn hình đăng ký. ..................................................................................... 52

Hình 21. Màn hình trang chủ ................................................................................... 52

Hình 22. Màn hình chức năng đổi thông tin cá nhân. .............................................. 53

Hình 23. Màn hình chức năng đổi mật khẩu. ........................................................... 54

Hình 24. Màn hình trang chi tiết cuộc bình chọn. .................................................... 55

Hình 25. Giao diện của một ứng viên trong trang chi tiết cuộc bình chọn. ............. 56

Hình 26. QR code chứa đường dẫn của một ứng viên. ............................................ 56

Hình 27. Màn hình bình chọn bằng QR Code .......................................................... 57

Hình 28. Biểu mẫu đăng nhập và đăng kí trang bình chọn bằng QR Code ............. 58

Hình 29. Màn hình trang chi tiết cuộc bình chọn của chủ sở hữu ............................ 59 Hình 30. Hộp thoại thay đổi thông tin cuộc bình chọn. ........................................... 60

Hình 31. Màn hình tạo cuộc bình chọn. ................................................................... 60

Hình 32. Màn hình thêm ứng viên. .......................................................................... 61

Hình 33. Header với admin ...................................................................................... 62

Hình 34. Màn hình trang tổng quan. ........................................................................ 62

Hình 35. Màn hình danh sách người dùng của hệ thống. ......................................... 63

Hình 36. Màn hình danh sách các cuộc bình chọn. .................................................. 63

# DANH MỤC BẢNG

Bảng 1. Đặc tả usecase đăng ký. ............................................................................. 28

Bảng 2. Đặc tả usecase đăng nhập. ......................................................................... 29

Bảng 3. Đặc tả usecase cập nhật thông tin cá nhân ................................................. 30

Bảng 4. Đặc tả usecase đổi mật khẩu ...................................................................... 31

Bảng 5. Đặc tả usecase thay đổi thông tin cuộc bình chọn ..................................... 32

Bảng 6. Đặc tả use case xem các cuộc bình chọn đã tạo. ........................................ 32

Bảng 7: Đặc tả usecase tạo cuộc bình chọn. ............................................................ 33

Bảng 8: Đặc tả usecase thêm ứng viên .................................................................... 34

Bảng 9: Đặc tả usecase bình chọn. .......................................................................... 35

Bảng 10: Đặc tả usecase bình chọn bằng QR Code ................................................ 35

Bảng 11: Đặc tả usecase tham gia cuộc bình chọn. ................................................. 36

Bảng 12: Đặc tả usecase xem chi tiết cuộc bình chọn. ............................................ 37

Bảng 13: Đặc tả usecase cập nhật bình chọn. .......................................................... 38

Bảng 14: Đặc tả usecase truy xuất bình chọn bằng QR Code ................................. 39

Bảng 15: Đặc tả usecase tìm kiếm cuộc bình chọn. ................................................ 39

Bảng 16: Đặc tả usecase xem các cuộc bình chọn công khai. ................................. 40

Bảng 17: Đặc tả usecase tạo QRCode cho ứng viên. .............................................. 40

Bảng 18: Đặc tả usecase quản lý các cuộc bình chọn ............................................. 41

Bảng 19: Đặc tả usecase chặn và mở chặn tài khoản. ............................................. 41

Bảng 20. Các phương thức trong hợp đồng thông minh. ........................................ 66

# TÓM TẮT

Đề tài "Xây dựng nền tảng thương mại điện tử điện tử thiết bị điện tử tích hợp trí tuệ nhân tạo và chatbot sử dụng Spring Boot , ReactJS với hệ thống gợi ý sản phẩm cá nhân hóa" nghiên cứu và ứng dụng công nghệ trí tuệ nhân tạo để xây dựng một hệ thống thương mại điện tử hiện đại, thông minh và hiệu quả cho việc mua bán thiết bị điện tử.

Hệ thống sử dụng kiến trúc microservices, với Frontend được phát triển bằng React.js kết hợp TypeScript và Material-UI, Backend sử dụng Spring Boot kết hợp với Spring Cloud và MySQL. Các dịch vụ được quản lý và điều phối thông qua Eureka Service Discovery và Spring Cloud Gateway. Hệ thống tích hợp Redis để caching và Apache Kafka cho xử lý bất đồng bộ.

Hệ thống hỗ trợ chatbot trí tuệ nhân tạo sử dụng Gemini API, cho phép người dùng tương tác tự nhiên để tìm kiếm sản phẩm, nhận tư vấn và hỗ trợ mua hàng 24/7. Hệ thống gợi ý sản phẩm cá nhân hóa sử dụng kết hợp Collaborative Filtering (SVD model) và Content-Based Filtering, được triển khai thông qua Python FastAPI service với Scikit-learn, giúp gợi ý sản phẩm phù hợp dựa trên hành vi và sở thích của từng người dùng. Hệ thống sử dụng JWT (JSON Web Token) kết hợp OAuth2 (Google, Facebook) để xác thực người dùng và phân quyền cho hệ thống.

Tiểu luận đã phân tích các chức năng chính của hệ thống, bao gồm quản lý sản phẩm, quản lý đơn hàng, giỏ hàng, thanh toán (VNPay), vận chuyển (GHN), đánh giá sản phẩm với phân tích cảm xúc, quản lý người dùng và dashboard quản trị. Ngoài ra, tiểu luận còn đề cập đến việc triển khai các microservices, các công nghệ hỗ trợ như Spring Boot, Spring Cloud, React.js, Material-UI, MySQL, Redis, Apache Kafka, Python FastAPI, Scikit-learn, Gemini API, JWT, OAuth2 và các tích hợp thanh toán, vận chuyển.

MỤC LỤC

[DANH SÁCH CHỮ VIẾT TẮT 1](#_Toc217429956)

[DANH MỤC HÌNH ẢNH 2](#_Toc217429957)

[DANH MỤC BẢNG 4](#_Toc217429958)

[TÓM TẮT 5](#_Toc217429959)

[MỞ ĐẦU 1](#_Toc217429960)

[1. LÝ DO CHỌN ĐỀ TÀI 1](#_Toc217429961)

[2. MỤC TIÊU VÀ PHẠM VI NGHIÊN CỨU 5](#_Toc217429962)

[3. Ý NGHĨA KHOA HỌC VÀ THỰC TIỄN 6](#_Toc217429963)

[CHƯƠNG 1. TỔNG QUAN ĐỀ TÀI 7](#_Toc217429964)

[1. 1. Phân tích, đánh giá các công trình nghiên cứu liên quan 7](#_Toc217429965)

[1. 2. Những vấn đề còn tồn tại 8](#_Toc217429966)

[1. 3. Nội dung chính của tiểu luận 9](#_Toc217429967)

[CHƯƠNG 2. PHƯƠNG PHÁP VÀ NỘI DUNG NGHIÊN CỨU 11](#_Toc217429968)

[2. 1. Cơ sở lý thuyết 11](#_Toc217429969)

[2.1.1. Microservices 11](#_Toc217429970)

[2.1.1.2.5. Dễ bảo trì hơn 14](#_Toc217429971)

[2.1.1.3. So sánh kiến trúc Microservices và kiến trúc nguyên khối (Monolithic) 16](#_Toc217429972)

[2.1.1.4. Những khó khăn khi triển khai Microservices 19](#_Toc217429973)

[2.1.2. API Gateway, REST API và Spring Cloud Gateway 21](#_Toc217429974)

[2.1.2.1. API Gateway 21](#_Toc217429975)

[Ví dụ về hoạt động của API Gateway: 22](#_Toc217429976)

[2.1.2.1.2. Chức năng và vai trò của API Gateway 23](#_Toc217429977)

[2.1.2.1.2.2. Bảo mật API 23](#_Toc217429978)

[2.1.2.1.2.3. Quản lý lưu lượng và cân bằng tải 23](#_Toc217429979)

[2.1.2.1.2.4. Giám sát và ghi log 23](#_Toc217429980)

[2.1.2.1.3. Nhược điểm khi dùng API Gateway 24](#_Toc217429981)

[2.1.2.1.3.2. Độ trễ (latency) 24](#_Toc217429982)

[2.1.2.1.3.3. Điểm nghẽn cổ chai 24](#_Toc217429983)

[2.1.2.2. REST API 24](#_Toc217429984)

[2.1.2.2.2. Các phương thức HTTP 27](#_Toc217429985)

[2.2.6. JWT – Json web token 57](#_Toc217429986)

[CHƯƠNG 3. GIẢI PHÁP CHO BÀI TOÁN 83](#_Toc217429987)

[3.1. Phát biểu bài toán trong đề tài 83](#_Toc217429988)

[3.2. Giải pháp giải quyết bài toán: 86](#_Toc217429989)

[3.2.1. Kiến trúc tổng quan. 86](#_Toc217429990)

[3.2.2. Sơ đồ kiến trúc hệ thống 86](#_Toc217429991)

[3.2.3. Cơ chế bảo mật 89](#_Toc217429992)

[3. 3. HIỆN THỰC GIẢI PHÁP 89](#_Toc217429993)

[3.3.1. Mô tả chương trình 89](#_Toc217429994)

[3.3.2. Lược đồ activity. 116](#_Toc217429995)

[3.3.3. Lược đồ tuần tự. 121](#_Toc217429996)

[3.3.4. Các màn hình của hệ thống 123](#_Toc217429997)

[3.3.5. Lược đồ cơ sở dữ liệu 137](#_Toc217429998)

[3.3.6. Xây dựng các endpoint API 138](#_Toc217429999)

[3.3.8. Cơ chế bảo mật JWT 140](#_Toc217430000)

[CHƯƠNG 4. KẾT QUẢ, KẾT LUẬN VÀ KIẾN NGHỊ 142](#_Toc217430001)

[4.1. KẾT QUẢ 142](#_Toc217430002)

[4. 2. KẾT LUẬN 144](#_Toc217430003)

[4. 3. KIẾN NGHỊ 144](#_Toc217430004)

[*TÀI LIỆU THAM KHẢO* 146](#_Toc217430005)

# MỞ ĐẦU

## 1. LÝ DO CHỌN ĐỀ TÀI

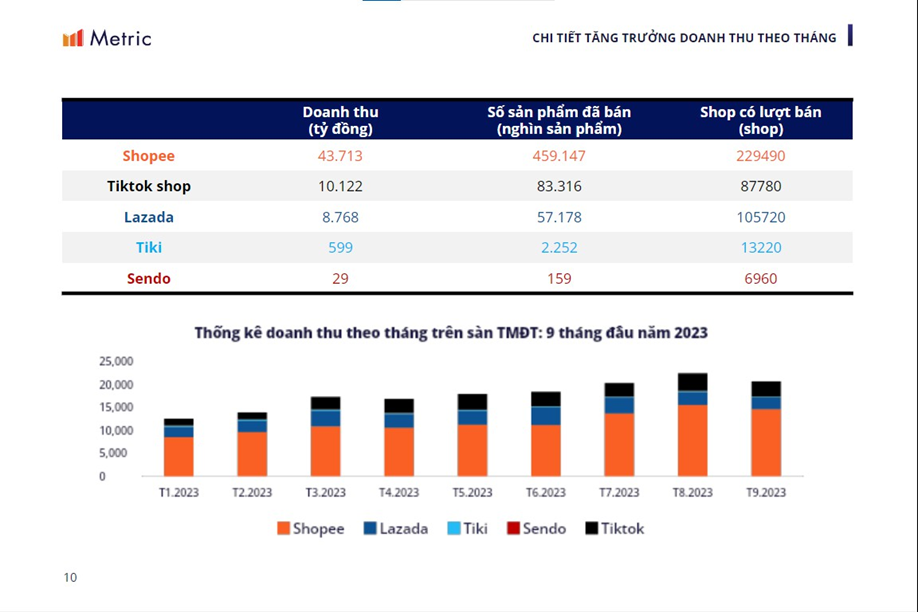
Thương mại điện tử đã và đang là xu hướng kinh doanh nằm trong top tăng trưởng trên thế giới, với quy mô và phạm vi trên gần như toàn bộ các lĩnh vực trong thương mại. Cuộc cách mạng về công nghệ thông tin đã làm thay đổi cuộc chơi về thương mại nói riêng cũng như các lĩnh vực khác nói chung.

Khoảng những năm 2000 – 2010, việc sở hữu một thiết bị kết nối mạng internet đã bắt đầu phổ biến, với những chiếc máy tính cá nhân có kết nối mạng internet, đến những chiếc laptop nhỏ gọn, tiện lợi. Qua đến giai đoạn từ 2010 đến nay, sự bùng nổ của các thiết bị di động cầm tay tiện lợi, và sự thay đổi chóng mặt của công nghệ internet giúp việc kết nối trở nên thuận tiện hơn, luôn sẵn sàng mọi lúc mọi nơi.

Với nền tảng công nghệ thông tin phát triển cộng với việc các thiết bị kết nối mạng trở nên phổ biến trong khoảng 2 thập kỉ vừa qua đã tác động tích cực lên ngành bán lẻ, giúp thay đổi thói quen của người tiêu dùng. Người tiêu dùng đã bắt đầu hình thành thói quen mua hàng trực tuyến trên các trang bán hàng điện tử hoặc các sàn thương mại điện tử. Nguyên nhân dẫn đến sự thay đổi này có thể được tóm gọn trong các ý sau

Thứ nhất, các trang thương mại điện tử đã xuất hiện rất nhiều, đa dạng sản phẩm, đa dạng thương hiệu và luôn có những ưu đãi hấp dẫn, kích thích người mua hàng chi tiêu. Thứ hai, khách hàng có thể mua hàng xuyên biên giới mà không cần phải đi đến tận nơi mua, chỉ cần click là đã có thể mua hàng, rất nhanh chóng và tiện lợi. Thứ ba, đối với các người bán, việc kinh doanh thương mại điện tử giúp bài toán chi phí của họ trở nên tối ưu, giảm các chi phí lớn như chi phí mặt bằng, chi phí nhân viên. Chỉ với một nguồn hàng ổn định, một kho lưu trữ và một hệ thống bán hàng hiệu quả, người bán đã có thể mang sản phẩm đến tận tay người tiêu dùng với chi phí thấp, giúp giá thành sản phẩm trở nên cạnh tranh hơn.

Hiện nay, không khó để tìm thấy một sàn thương mại điện tử trên internet. Shopee, Lazada hay Tiktok shop ngay chính tại Việt Nam chính là ví dụ điển hình cho sự phát triển của ngành thương mại điện tử tại Việt Nam. Theo Metric, Tổng doanh thu của toàn thị trường TMĐT trong quý III/2023 đạt 63 nghìn tỷ đồng và đạt mức tăng trưởng 54,42% so với cùng kỳ năm ngoái. Năm 2023, tổng doanh thu 9 tháng đầu tiên đã cao hơn cả năm 2022 khoảng 10 nghìn tỷ đồng, tương đương 7%. Trong đó, Tiktok Shop mang về 25 nghìn tỷ đồng. Nếu chỉ tính trên doanh thu 4 sàn Shopee, Lazada, Tiki, Sendo (TikTok Shop ra mắt vào cuối tháng 4/2022), tỷ lệ tăng trưởng so với cùng kỳ là 33%. Đây là các con số rất ấn tượng, chứng minh rằng thị trường thương mại điện tử vẫn đang tăng trưởng rất mạnh tại Việt Nam [1] . Điều này chứng minh cho xu hướng không thể phủ nhận – Dịch chuyển từ thương mại truyền thống sang thương mại điện tử.… (Hình 1).[1]



*Hình 1. Chi tiết tăng trưởng doanh thu theo tháng của các sàn thương mại điện tử tại Việt Nam – 9 tháng đầu năm 2023*

*(Nguồn:* [*https://advertisingvietnam.com/metric-cong-bo-bao-cao-thi-truong-san- tmdt-quy-iii2023-p23110*](https://advertisingvietnam.com/metric-cong-bo-bao-cao-thi-truong-san-tmdt-quy-iii2023-p23110)*)*

Dù hiện tại, thương mại điện tử vẫn đang phát triển và đã đạt được nhiều bước tiến lớn, vẫn còn tồn đọng rất nhiều vấn đề khi tăng trưởng quá nhanh. Đầu tiên, khi lượng khách hàng tăng cao, vấn đề đảm bảo khả năng sử dụng khi có một lượng lớn khách hàng truy cập vào cùng một thời điểm vẫn còn nhiều hạn chế. Bên cạnh đó, các yếu tố về xây dựng hệ thống dữ liệu, kiến trúc hệ thống làm sao để đảm bảo tốc độ thực thi, sự chính xác của dữ liệu và đưa ra các con số thống kê cũng là một thách thức lớn.

Để giải quyết những vấn đề đó, em đã bắt tay vào nghiên cứu giải pháp để làm sao xây dựng một sàn thương mại điện tử. Với yêu cầu là đảm bảo hệ thống vận hành trơn tru khi lượng truy cập tăng đột biến, mà vẫn mang lại trải nghiệm tuyệt vời dành cho khách hàng và người bán. Ngoài ra, khả năng mở rộng quy mô của sàn thương mại điện tử khi tăng trưởng và mở rộng cũng là tiêu chí hàng đầu để nhóm tìm hiểu các công nghệ có thể đáp ứng được điều này.

Sau quá trình nghiên cứu, em đã quyết định lựa chọn kiến trúc Microservice để áp dụng vào bài toán xây dựng sàn thương mại điện tử để đáp ứng với các yêu cầu trên. Kiến trúc Microservice có rất nhiều ưu điểm khi sử dụng cho hệ thống thương mại điện tử như

❖ **Kiểm soát lỗi và khả năng chịu lỗi**: Cho phép quản lý các lỗi phát sinh, bao gồm khả năng phát hiện và xử lý lỗi mà không làm ảnh hưởng đến toàn bộ hệ thống. Hệ thống có khả năng cách ly lỗi để duy trì sự ổn định và tiếp tục hoạt động.

❖ **Khả năng mở rộng và thu nhỏ dịch vụ**: Hỗ trợ việc mở rộng hoặc giảm quy mô của các dịch vụ mà không làm giảm hiệu suất của hệ thống. Điều này giúp hệ thống linh hoạt hơn trong việc đáp ứng nhu cầu thay đổi về tài nguyên.

❖ **Cập nhật phiên bản của service**: Cho phép cập nhật hoặc nâng cấp phiên bản của một dịch vụ để cải thiện hiệu suất hệ thống mà không làm gián đoạn hoạt động của các dịch vụ khác.

Kiến trúc Microservices có rất nhiều cải thiện về hiệu suất so với kiến trúc Monolith truyền thống. Theo bài báo “Designing microservice architectures for scalability and reliability in ecommerce” [6]

❖ Khả năng xử lý của một trang thương mại điện tử sử dụng Microservice đạt hiệu quả hơn 30% khi load script và 4% khi load phần tử hình ảnh.

❖ Khả năng chịu tải khi stress test với lần lượt 300, 500 và 1000 lần lặp kiểm tra dữ liệu với tối đa 25000 request đồng thời, tỷ lệ xuất hiện lỗi rất thấp (khoảng 3%)

❖ Các công nghệ API Gateway, Quản lý và đóng gói ứng dụng đã phát triển rất nhanh nhờ sự phổ biến của Microservices

❖ Từ đó, em quyết định chọn đề tài ứng dụng kiến trúc Microservices để xây dựng một sàn thương mại điện tử, tự động mở rộng hoặc thu nhỏ services đảm bảo hệ thống luôn luôn hoạt động ổn định.

## 2. MỤC TIÊU VÀ PHẠM VI NGHIÊN CỨU

Em tập trung vào mục tiêu tập trung vào việc ứng dụng kiến trúc Microservices và các công nghệ như Spring Boot, API Gateway và React để xây dựng một sàn thương mại điện tử có khả năng chịu tải cao, đồng thời có thể tự động mở rộng và thu nhỏ các service một cách hiệu quả. Đầu tiên, đề tài sẽ nghiên cứu sâu về kiến trúc Microservices, các khái niệm cơ bản, ưu điểm vượt trội như tính module hóa, khả năng mở rộng linh hoạt, khả năng phục hồi tốt cùng những thách thức đi kèm. Việc nắm vững kiến thức này sẽ tạo nền tảng vững chắc cho quá trình triển khai.

Về phía Front End, đề tài sử dụng ReactJs để hiện thực hóa giao diện và các chức năng giúp cho người dùng cuối như khách hàng, chủ cửa hàng và quản trị viên hệ thống có thể tương tác với hệ thống.

Về phần Back End, đề tài sử dụng Spring Boot, Mysql để xây dựng nên hệ thống RESTful API. Áp dụng RabbitMQ làm trung gian lưu trữ các message để các service có thể giao tiếp với nhau. Đồng thời để cải thiện tốc độ xử lý, nhóm em sử dụng các công nghệ hiện đại như Redis, Cron job và Background Job để tối ưu và tăng hiệu suất của hệ thống.

Front End và Back End giao tiếp với nhau thông qua Kong Api Gateway. Kong api gateway sẽ làm trung gian để trung chuyển các request từ Frontend đến đúng services.

## 3. Ý NGHĨA KHOA HỌC VÀ THỰC TIỄN

Đề tài "Xây dựng nền tảng thương mại điện tử điện tử thiết bị điện tử tích hợp trí tuệ nhân tạo và chatbot sử dụng Spring Boot , ReactJS với hệ thống gợi ý sản phẩm cá nhân hóa" mang ý nghĩa khoa học trong việc nghiên cứu kỹ thuật phát triển phần mềm theo hướng microservices và mang ý nghĩa thực tiễn khi áp dụng những nghiên cứu đó thành một sản phẩm hoàn chỉnh.

Việc nghiên cứu và triển khai kiến trúc microservice cho phép phân tách các dịch vụ độc lập, tăng cường khả năng quản lý, bảo trì và mở rộng hệ thống, đồng thời giảm thiểu sự phụ thuộc lẫn nhau giữa các thành phần.

Với việc phát triển một sàn thương mại điện tử thân thiện với người dùng, nhưng lại có khả năng mở rộng và chịu tải tốt, đề tài sẽ cung cấp một ví dụ cụ thể về cách triển khai một hệ thống microservices trên môi trường cloud và ứng dụng để xây dựng một sàn thương mại điện tử.

# CHƯƠNG 1. TỔNG QUAN ĐỀ TÀI

## 1. 1. Phân tích, đánh giá các công trình nghiên cứu liên quan

Trong “Designing microservice architectures for scalability and reliability in e- commerce” [6] của nhóm tác giả Asrowardi đã giới thiệu sự phát triển nhanh chóng của công nghệ thông tin và truyền thông (ICT) và tác động của nó đến thương mại điện tử (e-commerce). Với sự gia tăng nhanh chóng số lượng người sử dụng internet, đặc biệt là ở Indonesia, e-commerce trở thành một lĩnh vực tiềm năng với tốc độ tăng trưởng cao. Bài báo nhấn mạnh rằng các công ty e-commerce phải đối mặt với nhu cầu cung cấp các dịch vụ có thể tái sử dụng, triển khai nhanh chóng và mở rộng quy mô hiệu quả. Kiến trúc microservices được đề xuất như một giải pháp tối ưu cho những thách thức này. Bài báo đã cung cấp cái nhìn sâu sắc về cách thức triển khai và lợi ích của kiến trúc microservices trong lĩnh vực e-commerce. Với các kết quả thử nghiệm và phân tích chi tiết, bài báo đã chứng minh rằng microservices là một giải pháp hiệu quả cho các hệ thống e-commerce hiện đại, giúp chúng trở nên linh hoạt, dễ mở rộng và bền vững hơn. Các đề xuất và hướng nghiên cứu tương lai cung cấp nền tảng cho các nghiên cứu tiếp theo nhằm tối ưu hóa và nâng cao hiệu quả của microservices trong thực tiễn.

Trong “Microservices: architecture, container, and challenges” [8], nhóm tác giả đã đề cập đến kiến trúc Microservices trong những năm gần đây đã trở thành một kiến trúc của xu thế lập trình ứng dụng hiện đại. Càng ngày càng có nhiều ứng dụng cloud đã bắt đầu triển khai mô hình này cho hệ thống của mình. Trong công trình [8] đã đề cập đến Amazon, Netflix, The Guardian, Twitter, PayPal, SoundCloud, và các công ty khác đã bắt đầu sử dụng hệ thống phần mềm microservices trên nền tảng đám mây. Ở Trung Quốc, Tencent, Baidu, Jingdong, Taobao, 360 Search, v.v cũng đã bắt đầu chuyển đổi hệ thống phần mềm sang microservices. Tiêu biểu nhất là hệ thống Wechat với hơn 3000 microservices, được phân bố cho khoảng 20,000 máy; hoặc với Netflix, hệ thống online của họ sử dụng hơn 500 microservices, liên quan đến hơn 5 tỷ lượt tương tác mỗi ngày. Điều này cho thấy, các kiến trúc truyền thống như

Monolithic gặp phải các vấn đề như khó mở rộng ứng dụng trong tương lai. Cụ thể, để đáp ứng số lượng lớn user truy cập đồng thời, thì hệ thống phải nhân rộng lên để đáp ứng, bao gồm cả những chức năng không cần được nhân rộng. Điều này gây ra lãng phí tài nguyên, thời gian. Sự xuất hiện của công nghệ microservice sẽ giải quyết các vấn đề nêu trên gặp phải bởi các ứng dụng monolithic truyền thống, đối với các hệ thống yêu cầu tính mở rộng cao.

## 1. 2. Những vấn đề còn tồn tại

Các công trình trên chưa cung cấp đủ chi tiết về cách thức thực hiện các thử nghiệm. Cụ thể, không có thông tin rõ ràng về môi trường thử nghiệm, cấu hình hệ thống, và các công cụ cụ thể được sử dụng. Điều này khiến cho việc tái lập kết quả trở nên khó khăn, và các nghiên cứu khác cũng khó có thể so sánh trực tiếp với kết quả được trình bày trong bài báo. Việc thiếu chi tiết này làm giảm sự tin cậy và khả năng ứng dụng của các kết quả thử nghiệm. Thêm vào đó, bài báo chưa đề cập đủ rõ về chi phí và nguồn lực cần thiết để triển khai và duy trì hệ thống microservices. Mặc dù microservices mang lại nhiều lợi ích, việc chuyển đổi từ kiến trúc monolithic sang microservices đòi hỏi đầu rất lớn về tài chính và nhân lực. Người chịu trách nhiệm triển khai cần có cái nhìn rõ ràng về những thách thức này để có thể đánh giá khả năng áp dụng kiến trúc microservices một cách thực tế. Ngoài ra, bài báo thiếu các ví dụ thực tiễn hoặc case study cụ thể từ các công ty đã triển khai thành công kiến trúc microservices. Các ví dụ thực tiễn này có thể cung cấp cái nhìn rõ ràng hơn về cách áp dụng kiến trúc trong thực tế và những bài học kinh nghiệm quý báu từ các dự án cụ thể. Việc này không chỉ giúp minh họa rõ hơn các lợi ích mà còn cung cấp các chiến lược và phương pháp tiếp cận thực tế. Một thách thức lớn mà nhiều doanh nghiệp gặp phải là quá trình chuyển đổi từ kiến trúc monolithic sang microservices. Tuy nhiên, bài báo chưa cung cấp hướng dẫn chi tiết hoặc chiến lược để thực hiện quá trình chuyển đổi này một cách hiệu quả và ít rủi ro. Các doanh nghiệp cần biết cách lập kế hoạch, quản lý thay đổi và giải quyết các vấn đề phát sinh trong quá trình chuyển đổi để đạt được kết quả mong muốn. Đánh giá về hiệu năng của hệ thống microservices trong bài báo còn hạn chế. Mặc dù có một số đánh giá sơ bộ, nhưng chưa có phân tích sâu về các trường hợp sử dụng cụ thể, tính năng của từng microservice và tác động của các yếu tố khác nhau như tải hệ thống và độ phức tạp của dịch vụ đến hiệu năng tổng thể. Điều này làm giảm khả năng áp dụng kết quả nghiên cứu vào các tình huống thực tế đa dạng. Cuối cùng, bài báo chưa phân tích kỹ lưỡng về vấn đề giao tiếp giữa các microservices, đặc biệt là các thách thức liên quan đến độ trễ, độ tin cậy và chi phí giao tiếp mạng. Đây là một khía cạnh quan trọng cần được xem xét khi thiết kế và triển khai kiến trúc microservices, nhưng lại chưa được đề cập đầy đủ trong bài báo.

Từ những vấn đề trên, đề tài sẽ tiến hành nghiên cứu và áp dụng kiến trúc Microservices để giải quyết vấn để mở rộng trong tương lai.

## 1. 3. Nội dung chính của tiểu luận

Nghiên cứu này tập trung vào việc áp dụng kiến trúc Microservices để phát triển các hệ thống phần mềm hiện đại, đặc biệt là sàn thương mại điện tử. Mục tiêu chính của nghiên cứu là phân tích các khía cạnh quan trọng của kiến trúc Microservices, bao gồm khái niệm cơ bản, đặc điểm của hệ thống như giao tiếp giữa các dịch vụ để đảm bảo hoạt động trơn tru, và duy trì tính nhất quán trong một hệ thống phân tán. Ngoài ra, nghiên cứu cũng sẽ xem xét các vấn đề liên quan đến quản lý hiệu suất, độ tin cậy thông qua các tác vụ nền để tối ưu hóa hiệu quả hệ thống, cũng như so sánh với kiến trúc Monolithic để làm rõ những thuận lợi và thách thức khi triển khai Microservices, từ đó đánh giá khả năng áp dụng vào sàn thương mại điện tử.

Nghiên cứu sẽ tiếp tục làm rõ các thành phần chính trong hệ thống Microservices, như API Gateway, các dịch vụ, và hệ thống điều phối container nhằm tự động hóa triển khai. Sau khi hiểu rõ các khái niệm này, nghiên cứu sẽ phân tích các công nghệ hiện có tương ứng với từng thành phần, đánh giá ưu và nhược điểm của từng công nghệ, và đưa ra lựa chọn công nghệ phù hợp để xây dựng sàn thương mại điện tử dựa trên kiến trúc Microservices.

Cuối cùng, nghiên cứu sẽ phát triển một sàn thương mại điện tử dựa trên kiến trúc Microservices với các công nghệ đã được chọn, đảm bảo rằng giao diện người dùng thân thiện và đáp ứng được các yêu cầu về trải nghiệm người dùng. Sàn thương mại điện tử sẽ được thiết kế để thực hiện các nghiệp vụ cần thiết như đặt hàng, theo dõi trạng thái đơn hàng, thanh toán, và có khả năng xử lý lượng lớn khách hàng truy cập đồng thời.

# CHƯƠNG 2. PHƯƠNG PHÁP VÀ NỘI DUNG NGHIÊN CỨU

## 2. 1. Cơ sở lý thuyết

#### 2.1.1. Microservices

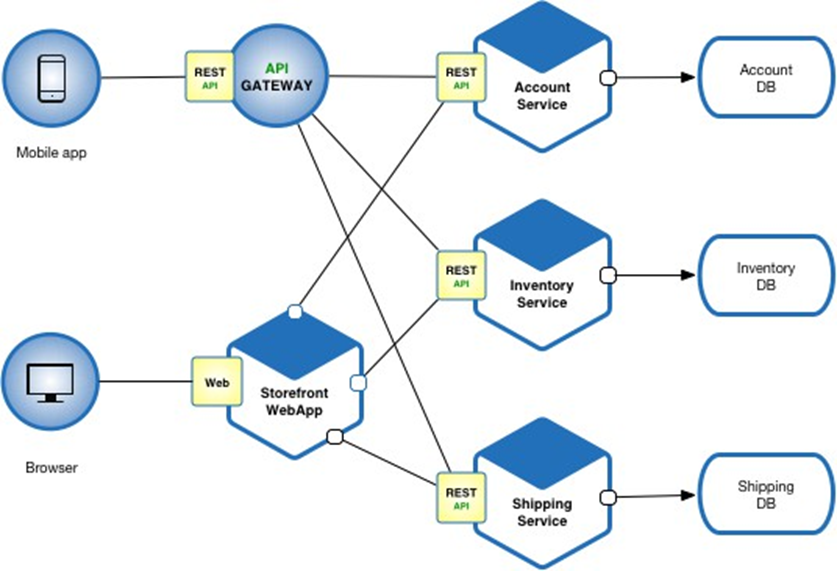
Việc lựa chọn kiến trúc để xây dựng nên sản phẩm là bước rất quan trọng. Với đặc thù yêu cầu của một sàn thương mại điện tử yêu cầu tính ổn định, cùng lúc đáp ứng được một lượng lớn các request mà không làm gián đoạn trải nghiệm mua hàng cũng như các thành phần của sàn (các service như xác thực / uỷ quyền, sản phẩm,…). Ngoài ra, hệ thống còn phải có khả năng mở rộng, giúp tăng quy mô cũng như khả năng chịu tải dành cho sàn. Để đáp ứng được những nhu cầu đó, Kiến trúc microservice chính là kiến trúc phù hợp để xây dựng một sàn thương mại điện tử. Nhóm quyết định áp dụng kiến trúc microservice để xây dựng các service riêng biệt.

**2.1.1.1. Giới thiệu về Microservices:**

Microservice là một kỹ thuật phát triển phần mềm, là một biến thể của kiến trúc hướng dịch vụ (SOA - xuất hiện trong những năm 1990). SOA tập trung vào việc chia ứng dụng thành các dịch vụ nhỏ và độc lập, nhằm tạo sự linh hoạt và tái sử dụng trong phát triển phần mềm. Tuy nhiên, SOA thường gặp khó khăn trong việc triển khai thực tế do sự phức tạp của các dịch vụ lớn và sự phụ thuộc mạnh mẽ giữa các dịch vụ.

Bắt đầu vào khoảng giữa những năm 2000, một số công ty công nghệ lớn như Amazon, Netflix và Twitter đã bắt đầu áp dụng kiến trúc microservice để tăng cường tính mở rộng, khả năng module và triển khai liên tục trong hệ thống của họ. Việc triển khai thành công của những công ty này đã thu hút sự quan tâm và lan rộng kiến trúc microservice sang các lĩnh vực phát triển phần mềm khác. Đến những năm 2010, kiến trúc microservice đã trở nên phổ biến và trở thành một trong những phong cách kiến trúc phổ biến trong công nghiệp phát triển phần mềm. Các công ty và tổ chức phát triển phần mềm khắp thế giới đã bắt đầu chuyển từ kiến trúc monolithic truyền thống sang kiến trúc microservice để tận dụng các lợi ích vượt trội của kiến trúc này trong việc xây dựng các ứng dụng phức tạp và linh hoạt. Cho đến hiện nay, kiến trúc microservice tiếp tục phát triển và tiến xa hơn trong việc tạo ra các hệ thống phần mềm phức tạp, đáp ứng nhanh chóng và hiệu quả yêu cầu của thị trường và người dùng.

Trong kiến trúc microservices, các logic ứng dụng được phân chia thành các phần riêng biệt, được gọi là các microservices. Thay vì phải hiện thực tất cả các logic ứng dụng trong một ứng dụng nguyên khối (monolithic application), mỗi microservice độc lập triển khai một phần cụ thể của ứng dụng. Các microservices có thể được viết bằng nhiều ngôn ngữ lập trình và được triển khai và mở rộng độc lập với nhau. Mỗi microservice hoạt động như một ứng dụng con độc lập, có khả năng giao tiếp với các microservices khác thông qua các giao thức giao tiếp như HTTP, RPC hoặc các hàng đợi tin nhắn. Việc phân chia ứng dụng thành các microservices độc lập giúp cải thiện tính module và tách biệt các chức năng, giúp ứng dụng dễ hiểu, phát triển, kiểm thử và duy trì



*Hình 2.1 Minh họa về kiến trúc Microservices*

*(Nguồn:* [*https://microservices.io/i/Microservice\_Architecture.png*](https://microservices.io/i/Microservice_Architecture.png)*)*

##### 2.1.1.2. 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.

##### 2.1.1.2.1. Tính mở rộng độc lập

Microservices cho phép từng dịch vụ mở rộng độc lập dựa trên nhu cầu. Điều này có nghĩa rằng trong những thời gian cao điểm như siêu sale giảm giá lớn của năm, chỉ các thành phần cần thiết như hệ thống đặt hàng mới cần mở rộng, tối ưu hóa phân bổ tài nguyên và giảm chi phí.

##### 2.1.1.2.2. Tùy chọn công nghệ linh hoạt

Microservices cho phép từng dịch vụ mở rộng độc lập dựa trên nhu cầu. Điều này có nghĩa rằng trong những thời gian cao điểm như siêu sale giảm giá lớn của năm, chỉ các thành phần cần thiết như hệ thống đặt hàng mới cần mở rộng, tối ưu hóa phân bổ tài nguyên và giảm chi phí.

##### 2.1.1.2.3. Cách ly và chịu lỗi

Bằng cách cô lập các dịch vụ, microservices đảm bảo rằng một lỗi ở một phần của hệ thống không làm sập toàn bộ nền tảng. Sự cô lập này cải thiện độ tin cậy hệ thống và giảm thời gian chết, đảm bảo trải nghiệm khách hàng liền mạch.

##### 2.1.1.2.4. Phát triển và triển khai nhanh hơn

Với microservices, các nhóm phát triển có thể làm việc trên các dịch vụ khác nhau đồng thời và triển khai các cập nhật mà không phụ thuộc vào nhau. Phát triển song song này làm tăng tốc việc phát hành tính năng mới và cho phép thử nghiệm nhanh chóng với các công nghệ mới.

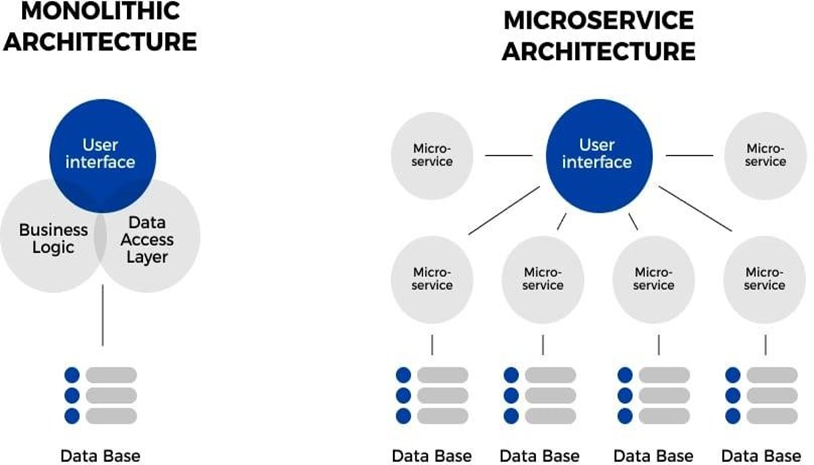
#### 2.1.1.2.5. Dễ bảo trì hơn

Các dịch vụ nhỏ hơn, phân tách ra làm cho việc bảo trì và cập nhật dễ dàng hơn so với kiến trúc monolithic. Tiếp cận mô-đun này đơn giản hóa việc xử lý sự cố và tăng cường tính linh hoạt để tích hợp các chức năng mới mà không làm gián đoạn toàn bộ hệ thống.

Ứng dụng kiến trúc Microservices trong thương mại điện tử đòi hỏi sự đầu tư kỹ lưỡng vào cơ sở hạ tầng công nghệ, đặc biệt là các công nghệ hỗ trợ tính năng mở rộng tự động và quản lý tài nguyên hiệu quả. Việc này bao gồm:

* **Load Balancing và API Gateway:** API Gateway đóng vai trò điều hướng các yêu cầu đến đúng dịch vụ mục tiêu, giúp quản lý lưu lượng lớn một cách hiệu quả. Cùng với đó, Load Balancer sử dụng các thuật toán để phân phối đều các yêu cầu giữa các dịch vụ, đảm bảo không dịch vụ nào bị quá tải, từ đó nâng cao độ ổn định và hiệu suất của hệ thống.
* **Message Queuing**: Hệ thống hàng đợi tin nhắn (message queuing) cho phép các dịch vụ giao tiếp một cách không đồng bộ, giảm thiểu sự phụ thuộc lẫn nhau và tăng cường khả năng chịu lỗi của hệ thống. Khi một dịch vụ gặp sự cố, các yêu cầu có thể được lưu trữ trong hàng đợi và xử lý sau, đảm bảo trải nghiệm của người dùng cuối không bị ảnh hưởng.
* **Container Orchestration (Điều phối Container)**: Container hóa mang lại lợi ích to lớn trong việc phát triển và triển khai dịch vụ bằng cách cung cấp môi trường đồng nhất và tách biệt giữa các giai đoạn phát triển. Điều này giúp giảm thiểu xung đột và các vấn đề tương thích. Kubernetes là nền tảng điều phối container hàng đầu, cho phép tự động hóa các tác vụ DevOps, mở rộng quy mô ứng dụng dễ dàng và quản lý toàn bộ vòng đời của các microservices. Với Kubernetes, các ứng dụng có thể tự động khởi động lại khi gặp sự cố, load balancing một cách hợp lý và tối ưu hóa việc sử dụng tài nguyên, từ đó đảm bảo hiệu suất và tính sẵn sàng cao cho hệ thống.

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



*Hình 2.2 So sánh giữa Monolithic và Microservices (Nguồn:* [*https://www.openlegacy.com/hs-*](https://www.openlegacy.com/hs-fs/hubfs/Picture1.webp?width=889&height=478&name=Picture1.webp)

[*fs/hubfs/Picture1.webp?width=889&height=478&name=Picture1.webp*](https://www.openlegacy.com/hs-fs/hubfs/Picture1.webp?width=889&height=478&name=Picture1.webp)*)*

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.

|  |  |  |
| --- | --- | --- |
|  | **Monolithic Architecture** | **Microservices Architecture** |
| **Định nghĩa** | Trong mô hình này, tất cả các thành phần của ứng dụng (bao gồm cả frontend, backend, và database layers) được phát triển và triển khai như một đơn thể duy nhất. Mô hình này thường đơn  giản hơn để phát triển và triển khai | Trong mô hình Microservices, ứng dụng được chia thành nhiều dịch vụ nhỏ, mỗi dịch vụ hoạt động độc lập và thực hiện một chức năng cụ thể của hệ thống. Các  Microservices này giao tiếp |

|  |  |  |
| --- | --- | --- |
|  | ban đầu, nhưng có thể gặp khó khăn khi quy mô tăng lên hoặc khi cần cập nhật và bảo trì. | với nhau thông qua API (Application Programming Interface) và thường được  triển khai một cách độc lập. |
| **Scalability và Maintenance** | Việc mở rộng quy mô (scaling) và bảo trì ứng dụng monolithic có thể trở nên phức tạp khi độ lớn của phần mềm tăng lên. Mọi thay đổi, dù nhỏ, đều yêu cầu xây dựng lại và triển khai lại toàn bộ ứng dụng, điều này có thể dẫn đến downtime đáng kể và tăng rủi ro trong quản lý version. | Các dịch vụ trong mô hình Microservices có thể được mở rộng độc lập, cho phép quản lý quy mô hiệu quả hơn. Bảo trì và cập nhật từng Microservice riêng biệt cũng dễ dàng hơn, giúp giảm thiểu rủi ro và downtime. Tuy nhiên, quản lý nhiều dịch vụ đồng thời đòi hỏi các công cụ  và kỹ năng quản lý phức tạp. |
| **Độ phức tạp và Cost of Development** | Phát triển ban đầu thường nhanh hơn và đơn giản hơn vì tất cả các thành phần của ứng dụng được tích hợp chặt chẽ. Tuy nhiên, chi phí và độ phức tạp có thể tăng lên đáng kể khi ứng dụng phát triển kích thước và tính năng. | Phát triển dựa trên Microservices đòi hỏi kỹ thuật cao hơn và ban đầu có thể tốn kém hơn do cần thiết lập một hạ tầng phức tạp để quản lý các dịch vụ độc lập. Tuy nhiên, trong dài hạn, mô hình này thường mang lại lợi ích về khả năng mở rộng và dễ  dàng bảo trì hơn. |
| **Thích hợp với từng loại ứng dụng** | Thường phù hợp với các dự án nhỏ đến trung bình, nơi mà độ phức tạp của sản phẩm không cao và không  yêu cầu mở rộng quy mô lớn. | Được ưu tiên cho các dự án lớn hơn, có yêu cầu cao về quy mô và tính linh hoạt, đặc |

|  |  |  |
| --- | --- | --- |
|  |  | biệt là trong môi trường phát  triển năng động và phức tạp. |

*Bảng 2. 1 So sánh kiến trúc Monolithic và Microservices*

#### 2.1.1.4. Những khó khăn khi triển khai Microservices

**2.1.1.4.1. 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.

**2.1.1.4.2. 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.

**2.1.1.4.3. 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.

**2.1.1.5. Kết luận về Microservices**

Microservices mang lại tính linh hoạt, khả năng phát triển và triển khai độc lập, cũng như tích hợp và khả năng tái sử dụng tốt hơn so với kiến trúc monolithic. Tuy nhiên, việc triển khai microservices có thể đòi hỏi quản lý và phát triển phức tạp hơn so với monolithic, do sự tách biệt và đa dạng của các dịch vụ. Trong đề tài này, hệ thống áp dụng kiến trúc Microservice để xây dựng hệ thống quản lý bán hàng, có khả năng mở rộng cao, đồng thời tích hợp nhiều công cụ quản lý hệ thống và công cụ DevOps giúp quản lý và giám sát các dịch vụ trong hệ thống một cách trực quan và hiệu quả, đồng thời cũng đảm bảo tính ổn định của hệ thống.

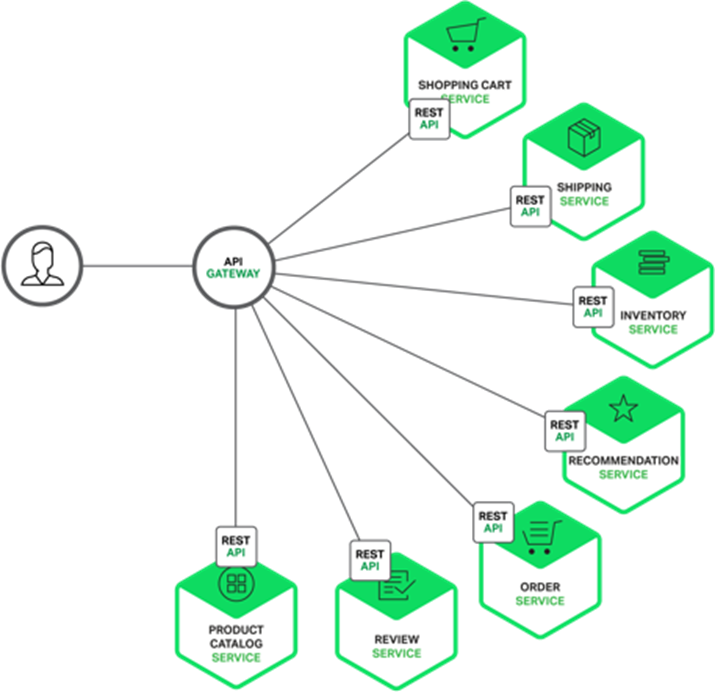
#### 2.1.2. API Gateway, REST API và Spring Cloud Gateway

API Gateway, REST API và Spring Cloud Gateway là các thành phần quan trọng trong hệ thống microservices của sàn thương mại điện tử. API Gateway đóng vai trò là “Người gác cổng” - cổng truy cập duy nhất cho tất cả các dịch vụ, quản lý và điều phối các yêu cầu từ phía người dùng đến các dịch vụ tương ứng. REST API cung cấp cách thức tiêu chuẩn để các dịch vụ giao tiếp với nhau thông qua HTTP, cho phép thực hiện các thao tác CRUD (Create, Read, Update, Delete) trên dữ liệu. Spring Cloud Gateway là một API Gateway mã nguồn mở thuộc hệ sinh thái Spring, hỗ trợ quản lý và điều phối các API, giúp tối ưu hóa luồng xử lý và tăng cường bảo mật cho hệ thống microservices của sàn thương mại điện tử.

#### 2.1.2.1. API Gateway

**2.1.2.1.1. Khái niệm**

API Gateway là một lớp trung gian giữa các client (như ứng dụng web hoặc di động) và các dịch vụ backend. Nó hoạt động như một reversed proxy, nhận tất cả các yêu cầu API từ client, định tuyến chúng đến các dịch vụ backend thích hợp và trả về phản hồi cho client. API Gateway giúp đơn giản hóa việc quản lý và bảo mật các API, cải thiện hiệu suất và khả năng mở rộng của hệ thống.



*Hình 2.3 Mô tả về hệ thống API Gateway*

### Ví dụ về hoạt động của API Gateway:

* Khi một ứng dụng di động gửi yêu cầu để lấy thông tin người dùng, yêu cầu này sẽ đi qua API Gateway.
* API Gateway sau đó chuyển yêu cầu đến route User service.
* API Gateway sẽ đợi phản hồi từ User service, đến khi nào có phản hồi thì chuyển kết quả đến nơi request được gửi đến

#### 2.1.2.1.2. Chức năng và vai trò của API Gateway

**2.1.2.1.2.1. Monitoring (quản lý) và routing (định tuyến) các yêu cầu API**

API Gateway định tuyến các yêu cầu từ client đến các dịch vụ backend dựa trên URL, phương thức HTTP, và các tham số khác. Ví dụ, một yêu cầu GET đến /api/products có thể được định tuyến đến Product service, trong khi một yêu cầu POST đến /api/orders có thể được chuyển đến Order service

#### 2.1.2.1.2.2. Bảo mật API

API Gateway cung cấp các cơ chế bảo mật như xác thực (authentication), ủy quyền (authorization), và mã hóa (encryption). Nó tích hợp với các hệ thống xác thực như OAuth2, JWT, và API Key để đảm bảo chỉ có các yêu cầu hợp lệ mới được xử lý. Ví dụ, API Gateway có thể yêu cầu mỗi yêu cầu kèm theo một token xác thực, sau đó kiểm tra quyền truy cập của token này để đảm bảo nó có quyền thực hiện hành động được yêu cầu.

#### 2.1.2.1.2.3. Quản lý lưu lượng và cân bằng tải

API Gateway giúp quản lý lưu lượng truy cập, bảo vệ các dịch vụ backend khỏi bị quá tải thông qua các cơ chế giới hạn tốc độ (rate limiting) và làm mịn lưu lượng (traffic shaping). Ví dụ, nó có thể giới hạn một client cụ thể chỉ được gửi tối đa 100 yêu cầu mỗi phút. API Gateway cũng cung cấp các giải pháp cân bằng tải để phân phối các yêu cầu đến nhiều instance của dịch vụ backend, đảm bảo rằng không có instance nào bị quá tải và hệ thống luôn hoạt động ổn định.

#### 2.1.2.1.2.4. Giám sát và ghi log

API Gateway ghi lại các thông tin về request và response, giúp giám sát để theo dõi hiệu suất và sức khỏe của các dịch vụ. API Gateway cũng cung cấp các công cụ phân tích log để phát hiện và xử lý lỗi phát sinh trong quá trình hoạt động. Ví dụ, API Gateway có thể ghi lại thông tin chi tiết về mỗi yêu cầu như thời gian phản hồi, status code, và các lỗi xảy ra, giúp các nhà phát triển và quản trị viên dễ dàng theo dõi và tối ưu hóa hệ thống.

#### 2.1.2.1.3. Nhược điểm khi dùng API Gateway

**2.1.2.1.3.1** **Độ phức tạp và chi phí triển khai**

Việc triển khai và cấu hình một API Gateway có thể phức tạp và tốn kém, đặc biệt là đối với các tổ chức mới bắt đầu.

Khi triển khai một hệ thống API Gateway, ta cần phải đầu tư vào hạ tầng, phần mềm và nhân lực để triển khai và duy trì API Gateway. Điều này bao gồm việc cài đặt, cấu hình và tích hợp với các dịch vụ hiện có, cũng như đào tạo về các công cụ được sử dụng trong hệ thống và quy trình triển khai.

### 2.1.2.1.3.2. Độ trễ (latency)

API Gateway có thể làm tăng độ trễ của các yêu cầu API do phải xử lý nhiều bước trung gian trước khi chuyển tiếp yêu cầu đến dịch vụ backend.

Các bước như xác thực, ủy quyền, logging và chuyển đổi dữ liệu có thể thêm thời gian xử lý cho mỗi yêu cầu, dẫn đến tăng độ trễ. Điều này đặc biệt quan trọng đối với các ứng dụng yêu cầu thời gian phản hồi nhanh

### 2.1.2.1.3.3. Điểm nghẽn cổ chai

API Gateway có thể trở thành điểm nghẽn duy nhất trong hệ thống, nếu nó gặp sự cố hoặc bị tấn công, toàn bộ hệ thống có thể bị ảnh hưởng.

Để giảm thiểu rủi ro này, cần phải thiết kế hệ thống API Gateway với khả năng chịu lỗi cao (high availability) và phân tán (distributed). Điều này bao gồm sử dụng nhiều instance của API Gateway, các cơ chế failover và các dịch vụ backup

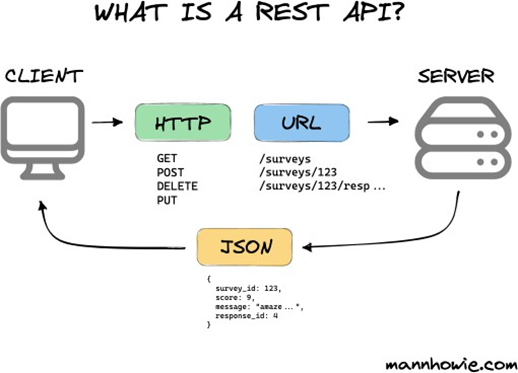
**2.1.2.1.3.4. Phụ thuộc vào hosting cung cấp dịch vụ API Gateway**

Nếu sử dụng một API Gateway thương mại hoặc dịch vụ đám mây, tổ chức có thể trở nên phụ thuộc vào nhà cung cấp dịch vụ. Sự phụ thuộc này có thể dẫn đến rủi ro về chi phí gia tăng, giới hạn tính năng hoặc khó khăn trong việc chuyển đổi sang giải pháp khác. Khi triển khai hệ thống API Gateway cần đánh giá cẩn thận và chọn lựa các nhà cung cấp có uy tín và hỗ trợ tốt, đảm bảo vận hành mượt mà và hạn chế phát sinh sự cố.

#### 2.1.2.2. REST API

**2.1.2.2.1. Khái niệm và lịch sử hình thành**

REST (Representational State Transfer) là một phong cách kiến trúc được sử dụng để thiết kế các hệ thống mạng, cụ thể là các dịch vụ web. RESTful API là một loại API tuân theo các nguyên tắc của kiến trúc REST, sử dụng các phương thức HTTP để thực hiện các hoạt động CRUD (Create, Read, Update, Delete) trên các tài nguyên web.



*Hình 2.4 Mô tả cơ bản về cách RESTful API hoạt động (Nguồn:*

[*https://images.ctfassets.net/vwq10xzbe6iz/5sBH4Agl614xM7exeLsTo7/9e84dce0173*](https://images.ctfassets.net/vwq10xzbe6iz/5sBH4Agl614xM7exeLsTo7/9e84dce01735f155911e611c42c9793f/rest-api.png)

[*5f155911e611c42c9793f/rest-api.png*](https://images.ctfassets.net/vwq10xzbe6iz/5sBH4Agl614xM7exeLsTo7/9e84dce01735f155911e611c42c9793f/rest-api.png)*)*

REST tuân theo các nguyên tắc kiến trúc cụ thể để tăng tính đơn giản và hiệu quả của các dịch vụ web. Một API tuân thủ các nguyên tắc REST được gọi là RESTful API. Các nguyên tắc cơ bản của REST bao gồm:

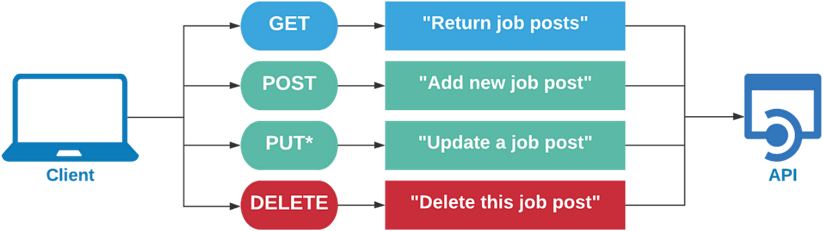
* **Stateless:** Mỗi yêu cầu từ client đến server phải chứa tất cả thông tin cần thiết để server xử lý yêu cầu đó. Server không lưu trữ trạng thái của client giữa các yêu cầu.
* **Client-Server:** Kiến trúc REST phân chia rõ ràng giữa client và server, cho phép chúng phát triển và mở rộng độc lập.
* **Uniform Interface:** Sử dụng giao diện thống nhất để cải thiện tính tương tác giữa các hệ thống khác nhau.
* **Layered System:** REST hỗ trợ hệ thống phân tầng, nơi các tầng trung gian có thể được thêm vào để cải thiện tính linh hoạt và khả năng mở rộng.
* **Cacheable:** Các phản hồi từ server có thể được lưu trữ lại (caching) để cải thiện hiệu suất.
* **Code on Demand (tùy chọn):** Server có thể gửi mã thực thi đến client khi cần thiết.

Lịch sử của REST API bắt đầu từ cuối thập kỷ 1990 và đầu 2000. Vào giai đoạn 1994-1995, Các nền tảng web đầu tiên bắt đầu hình thành, với các giao thức đơn giản như CGI (Common Gateway Interface) được sử dụng để xử lý yêu cầu HTTP. Các ứng dụng web lúc này còn đơn giản và chủ yếu là các trang HTML tĩnh. Năm 1996, Roy Fielding, một trong những người sáng lập của HTTP và tác giả chính của HTTP/1.1, bắt đầu nghiên cứu các nguyên tắc kiến trúc để xây dựng các hệ thống phân tán. Mục tiêu của ông là tạo ra một kiến trúc có thể mở rộng, linh hoạt và dễ bảo trì. Đến năm 2000 Roy Fielding trình bày luận án tiến sĩ của mình tại Đại học California, Irvine, trong đó ông giới thiệu khái niệm REST. Luận án của Fielding đặt nền móng lý thuyết cho REST và giải thích cách mà các hệ thống web có thể được xây dựng theo nguyên tắc này. Trong giai đoạn những năm 2000, REST nhanh chóng được các nhà phát triển web đón nhận do tính đơn giản và hiệu quả của nó. Các công ty như Amazon, eBay, và Yahoo bắt đầu triển khai REST API để cung cấp dịch vụ của họ qua web. Đến giai đoạn những năm 2010, REST API trở thành tiêu chuẩn de facto (thực tế) cho việc xây dựng các dịch vụ web và microservices. Các công cụ và framework như Django, Rails, Spring, và Flask bắt đầu hỗ trợ REST API một cách toàn diện, giúp việc triển khai và quản lý trở nên dễ dàng hơn. Hiện nay, REST API tiếp tục được sử dụng rộng rãi trong hầu hết các ứng dụng web và di động. Nó được tích hợp trong các nền tảng lớn như Google, Facebook, Twitter, và GitHub. Các công nghệ mới như GraphQL và gRPC cũng xuất hiện để giải quyết một số hạn chế của REST, nhưng REST vẫn giữ vai trò quan trọng và không thể thiếu trong việc xây dựng các dịch vụ web hiện đại.

REST API đã phát triển từ một khái niệm lý thuyết thành một tiêu chuẩn phổ biến, giúp đơn giản hóa việc xây dựng, triển khai và quản lý các dịch vụ web trên toàn thế giới. Với lịch sử phát triển lâu dài và cộng đồng người dùng lớn, REST API sẽ tiếp tục đóng vai trò quan trọng trong tương lai của công nghệ web.

#### 2.1.2.2.2. Các phương thức HTTP

* **GET**: Truy xuất thông tin từ server mà không làm thay đổi trạng thái của tài nguyên. Ví dụ: Yêu cầu GET đến /api/products trả về danh sách tất cả các sản phẩm.
* **POST**: Gửi dữ liệu đến server để tạo mới một tài nguyên. Ví dụ: Yêu cầu POST đến /api/products với dữ liệu sản phẩm mới trong phần thân yêu cầu sẽ tạo một sản phẩm mới.
* **PUT**: Cập nhật một tài nguyên hiện có trên server với dữ liệu mới. Ví dụ: Yêu cầu PUT đến /api/products/1 với dữ liệu cập nhật trong phần thân yêu cầu sẽ cập nhật thông tin của sản phẩm có ID là 1.
* **DELETE**: Xóa một tài nguyên trên server. Ví dụ: Yêu cầu DELETE đến /api/products/1 sẽ xóa sản phẩm có ID là 1.
* **PATCH**: Cập nhật từng phần của một tài nguyên hiện có. Ví dụ: Yêu cầu PATCH đến /api/products/1 với dữ liệu cập nhật từng phần trong phần thân yêu cầu sẽ cập nhật một hoặc vài thuộc tính của sản phẩm có ID là 1.



*Hình 2.5 Mô tả cơ bản về cách RESTful API hoạt động (Nguồn:*

*https://miro.medium.com/v2/resize:fit:1400/1\*UXTnlhEg4aua\_6eQv3HHSg.png)*

Một số các Status Code (mã trạng thái) nhận được khi gửi một yêu cầu API:[10]

* 200 OK – Trả về thành công cho những phương thức GET, PUT, PATCH hoặc DELETE.
* 201 Created – Trả về khi một Resource vừa được tạo thành công.
* 204 No Content – Trả về khi Resource xoá thành công.
* 304 Not Modified – Client có thể sử dụng dữ liệu cache.
* 400 Bad Request – Request không hợp lệ - 401 Unauthorized – Request cần có auth.
* 403 Forbidden – bị từ chối.
* 404 Not Found – Không tìm thấy resource từ URI
* 405 Method Not Allowed – Phương thức không cho phép với user hiện tại.
* 410 Gone – Resource không còn tồn tại, Version cũ đã không còn hỗ trợ.
* 415 Unsupported Media Type – Không hỗ trợ kiểu Resource này.
* 422 Unprocessable Entity – Dữ liệu không được xác thực.
* 429 Too Many Requests – Request bị từ chối do bị giới hạn.

Trong tiểu luận này, Restful API được sử dụng để Frontend gửi yêu cầu đến Backend, Backend xử lý yêu cầu và trả về phản hồi cho Frontend.

**2.1.2.2.3. Thiết kế của RESTful API**

Khi thiết kế một RESTful API, việc tuân thủ các nguyên tắc thiết kế rất quan trọng. Mặc dù các ràng buộc và quy ước trên các nhà phát triển không cần tuân thủ. Tuy nhiên nếu làm “đúng”, chúng sẽ mang lại rất nhiều lợi ích. Một trong những nguyên tắc cơ bản là stateless, nghĩa là mỗi yêu cầu từ client đến server phải chứa tất cả thông tin cần thiết để server hiểu và xử lý yêu cầu đó. Server không lưu trữ trạng thái của client giữa các yêu cầu, việc tách biệt giúp tăng tính linh hoạt của client cũng như khả năng mở rộng của server mà vẫn đảm bảo được giao tiếp. Kiến trúc client- server tách biệt rõ ràng giữa client và server, mỗi server sẽ có nhiều service khác nhau, tách biệt và đảm bảo tính linh hoạt và khả năng mở rộng về sau.

**2.1.2.2.3.1. Uniform interface**

Là ràng buộc quan trọng nhất của hệ thống REST. Để đảm bảo ràng buộc này, hệ thống tập trung vào việc xử lý các tài nguyên (resource). Mỗi một resource sẽ được xác định (định danh) bằng một URI (Uniform Resource Identifier) riêng biệt**.** đảm bảo sử dụng một giao diện thống nhất cho tất cả các resource, giúp cải thiện khả năng giao tiếp giữa client và server.

**2.1.2.2.3.2. Layered system**

Là nguyên tắc cho phép hệ thống có thể có nhiều tầng trung gian để tăng tính linh hoạt và khả năng mở rộng. Ví dụ, ta có thể thêm các tầng bảo mật hoặc caching mà không làm ảnh hưởng đến client hoặc server. **Cacheable** yêu cầu các phản hồi từ server phải có khả năng được lưu cache để tối ưu hóa hiệu suất. Điều này giúp giảm tải cho server và tăng tốc độ phản hồi cho client.

**2.1.2.2.3.3. Code on demand (tùy chọn)**

Cho phép server gửi mã thực thi đến client khi cần thiết, giúp mở rộng chức năng của client một cách linh hoạt.

Về cách thức định dạng URL và tài nguyên, các URL nên rõ ràng và có nghĩa, đại diện cho tài nguyên mà chúng truy cập. Ví dụ, /api/products cho sản phẩm. Tên resource nên được đặt nhất quán, sử dụng các từ ngữ chuẩn. Ví dụ, /api/users cho người dùng và /api/orders cho tài nguyên đơn hàng. Sử dụng tham số URL để lọc, phân trang, và sắp xếp kết quả, chẳng hạn như /api/products?page=2&sort=price\_desc để lấy danh sách sản phẩm ở trang 2 và sắp xếp theo giá giảm dần.

Các phương pháp xác thực và bảo mật cũng rất quan trọng trong REST API. Sử dụng API Key cho mỗi yêu cầu để xác thực, ví dụ: GET

/api/products?api\_key=your\_api\_key. OAuth2 có thể được sử dụng để ủy quyền cho các ứng dụng bên thứ ba truy cập vào tài nguyên của người dùng mà không cần chia sẻ thông tin xác thực của người dùng. JWT (JSON Web Token) được sử dụng để truyền tải thông tin xác thực giữa client và server, mỗi yêu cầu API sẽ kèm theo một JWT trong phần tiêu đề, ví dụ: Authorization: Bearer your\_jwt\_token.

Trong REST API, các tiêu đề HTTP (headers) đóng vai trò quan trọng trong việc truyền tải thông tin bổ sung. Một số tiêu đề phổ biến bao gồm Content-Type để xác định định dạng của dữ liệu trong yêu cầu hoặc phản hồi (ví dụ: application/json), Authorization để gửi thông tin xác thực (ví dụ: Bearer your\_jwt\_token), và Accept để xác định loại định dạng mà client có thể xử lý (ví dụ: application/json).

**2.1.2.2.4. Ưu điểm và nhược điểm khi sử dụng RESTful API**

**2.1.2.2.4.1. Ưu điểm**

* Đơn giản và dễ hiểu: REST API sử dụng các phương thức HTTP chuẩn và định dạng dữ liệu phổ biến như JSON, giúp việc tiếp cận trở nên đơn giản hơn, dễ đọc và dễ sử dụng.
* Tương thích với nhiều ngôn ngữ lập trình: REST API có thể xây dựng trên nhiều nền tảng, nhiều ngôn ngữ lập trình khác nhau, như Javascript (Nodejs),…
* Khả năng mở rộng: REST API có thể dễ dàng mở rộng và tích hợp với các hệ thống khác nhờ vào tính linh hoạt và sự độc lập giữa client và server.
* Hiệu suất cao: Bởi vì các yêu cầu REST không lưu trạng thái, chúng có thể được cache, giúp cải thiện hiệu suất hệ thống.
* Khả năng phân tán: REST hỗ trợ tốt cho các hệ thống phân tán, cho phép các dịch vụ được triển khai trên nhiều máy chủ khác nhau.

**2.1.2.2.4.2. Nhược điểm**

* Chỉ hoạt động trên các giao thức HTTP
* **Bảo mật:** REST có thể phù hợp với các URL public, nhưng không phải là một lựa chọn tốt nếu cần truyền dữ liệu nhạy cảm giữa client và server.
* **Quản lý trạng thái phức tạp:** Trong một số trường hợp, việc quản lý trạng thái của ứng dụng trên client có thể trở nên phức tạp do nguyên tắc stateless của REST.

**2.1.2.2.5. 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** | **REST** | **SOAP** |
| **Giao thức** | HTTP, HTTPS | HTTP, HTTPS, SMTP, TCP |
| **Định dạng dữ liệu** | JSON, XML, YAML, v.v. | XML |
| **Khả năng mở rộng** | Dễ dàng mở rộng | Khó khăn hơn do tính chặt chẽ của giao thức |
| **Caching** | Tận dụng rất tốt caching | Bỏ qua caching |
| **Bảo mật** | Tích hợp bảo mật HTTP, OAuth, JWT | WS-Security, SSL, HTTPS |
| **Khả năng xử lý trạng thái** | Stateless | Có thể stateful hoặc stateless |
| **Khả năng tương thích** | Tương thích với nhiều ngôn ngữ lập trình | Hạn chế hơn do yêu cầu SOAP libraries |

|  |  |  |
| --- | --- | --- |
| **Khả năng giao tiếp** | Đơn giản, dễ hiểu | Phức tạp, yêu cầu kiến thức chuyên sâu về XML và SOAP |
| **Tiêu chuẩn** | Không tuân theo tiêu chuẩn cụ thể | Tuân theo các tiêu chuẩn WSDL, XSD |
| **Sử dụng phổ biến** | Web APIs, Microservices, Mobile applications | Enterprise applications, Financial services |

*Bảng 2.2 So sánh giữa SOAP và REST API*

**2.1.2.3. Spring Cloud Gateway**

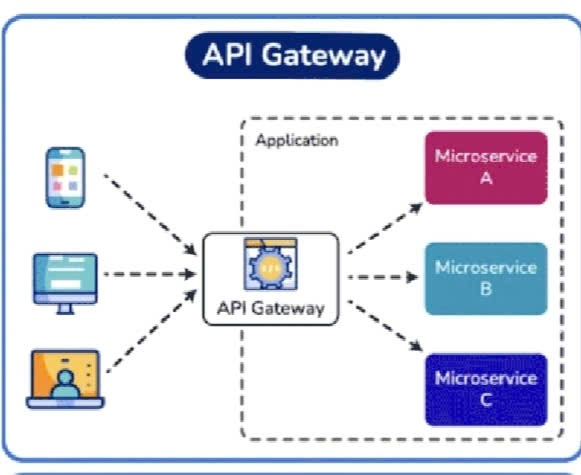
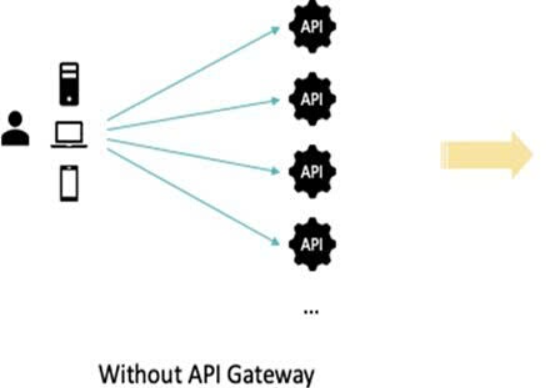
**2.1.2.3.1 Giới thiệu về Spring Cloud Gateway**

Spring Cloud Gateway là một API Gateway mã nguồn mở thuộc hệ sinh thái Spring Cloud, đóng vai trò là phần mềm trung gian (middleware) trong kiến trúc microservices. Công cụ này được thiết kế nhằm cung cấp một cách tiếp cận linh hoạt và hiệu quả để định tuyến, quản lý và bảo mật các API trong hệ thống.

Spring Cloud Gateway được xây dựng trên nền tảng Spring Boot và Project Reactor, hỗ trợ mô hình non-blocking (reactive), giúp hệ thống xử lý lượng lớn request với hiệu năng cao. Nhờ khả năng tích hợp chặt chẽ với các thành phần khác của Spring như Spring Security, Spring Cloud Config và Spring Cloud Netflix, Spring Cloud Gateway đặc biệt phù hợp với các hệ thống microservices phát triển bằng Java.

Spring Cloud Gateway cung cấp các chức năng quản lý API quan trọng như xác thực, ủy quyền, giới hạn tốc độ truy cập (rate limiting), cân bằng tải, ghi log và giám sát hệ thống. Gateway có thể được triển khai linh hoạt trên nhiều môi trường khác nhau như Docker, Kubernetes hoặc các máy chủ vật lý và máy chủ ảo.

Trong kiến trúc hệ thống, Spring Cloud Gateway hoạt động như điểm truy cập trung tâm (entry point) cho mọi yêu cầu từ client. Mọi request từ phía client đều được gửi đến Gateway trước, sau đó Gateway sẽ thực hiện các bộ lọc (filters) cần thiết như kiểm tra xác thực, phân quyền và logging, rồi mới chuyển tiếp request đến các service backend tương ứng. Nhờ đó, Spring Cloud Gateway giúp giảm tải logic xử lý chung cho các microservices và nâng cao tính bảo mật, mở rộng cho toàn bộ hệ thống.



Hình 2.6 Mô tả cách client tương tác với API khi có và không có API Gateway

**2.1.2.3.2. Kiến trúc và thành phần của Spring Cloud Gateway**

Các thành phần chính của Spring Cloud Gateway bao gồm:

* Routes: là thành phần định nghĩa quy tắc định tuyến request từ client đến các service backend. Mỗi route bao gồm điều kiện (predicate) và hành động xử lý tương ứng. Một Gateway có thể chứa nhiều route khác nhau.
* Predicate: là điều kiện để xác định một request có được ánh xạ tới route hay không, ví dụ như dựa trên URL, HTTP method, header hoặc tham số của request.
* Filter: là các bộ lọc được thực thi trước hoặc sau khi request được chuyển đến service backend. Filter được sử dụng để thực hiện các chức năng như xác thực JWT, ghi log, chỉnh sửa request/response và giới hạn tốc độ truy cập.
* Gateway Handler: chịu trách nhiệm xử lý request sau khi route được xác định và áp dụng các filter tương ứng, sau đó chuyển tiếp request đến service đích.
* Spring Security Integration: cho phép tích hợp cơ chế bảo mật như JWT, OAuth2 để kiểm soát quyền truy cập vào các API.
* Actuator & Monitoring: hỗ trợ giám sát, theo dõi trạng thái và hiệu năng của Gateway thông qua các endpoint quản trị.

Spring Cloud Gateway được cấu hình chủ yếu thông qua file cấu hình hoặc Java-based configuration, giúp dễ dàng mở rộng và bảo trì. Nhờ kiến trúc linh hoạt và khả năng tích hợp sâu với hệ sinh thái Spring, Gateway đóng vai trò quan trọng trong việc đảm bảo tính nhất quán, bảo mật và khả năng mở rộng của hệ thống microservices

**2.2 Các công nghệ hỗ trợ**

**2.2.1 Spring Boot**

**2.2.1.1 Spring Boot là gì ?**

Spring Boot là một framework mã nguồn mở được xây dựng trên nền tảng Spring Framework, được thiết kế để đơn giản hóa việc phát triển các ứng dụng Java enterprise. Spring Boot cung cấp cấu hình tự động (auto-configuration), giảm thiểu việc cấu hình thủ công và cho phép các nhà phát triển tập trung vào logic nghiệp vụ thay vì cấu hình infrastructure.

Spring Boot được phát triển bởi Pivotal Software (hiện tại là VMware) và được xây dựng dựa trên nguyên tắc "convention over configuration", giúp giảm thiểu boilerplate code và tăng tốc độ phát triển ứng dụng. Framework này tích hợp sẵn các web server như Tomcat, Jetty, cho phép chạy ứng dụng như một standalone JAR file mà không cần triển khai trên application server riêng biệt.



*Hình 2.19: Logo của Spring Boot*

**2.2.1.2. Ưu điểm Spring Boot**

Spring Boot mang lại nhiều ưu điểm vượt trội cho việc phát triển ứng dụng Java:

* Auto-configuration: Spring Boot tự động cấu hình các bean dựa trên các dependencies có trong classpath, giảm thiểu việc viết code cấu hình. Điều này giúp developers không cần phải cấu hình thủ công các thành phần như DataSource, JPA, Security, v.v.
* Starter Dependencies: Cung cấp các dependency packages được đóng gói sẵn (như spring-boot-starter-web, spring-boot-starter-data-jpa, spring-boot-starter-security) để tích hợp nhanh các tính năng. Mỗi starter bao gồm tất cả các dependencies cần thiết cho một tính năng cụ thể.
* Embedded Server: Tích hợp sẵn các web server như Tomcat, Jetty, cho phép chạy ứng dụng như một standalone JAR file. Điều này giúp đơn giản hóa quá trình deployment và testing.
* Production-ready Features: Cung cấp các tính năng sẵn sàng cho production như health checks, metrics, externalized configuration thông qua Actuator. Các tính năng này giúp monitoring và quản lý ứng dụng trong môi trường production.
* Giảm thiểu boilerplate code: Spring Boot giúp giảm đáng kể lượng code cấu hình cần thiết, cho phép developers tập trung vào business logic.
* Cộng đồng lớn và tài liệu phong phú: Spring Boot có cộng đồng phát triển rất lớn và tài liệu chi tiết, giúp dễ dàng tìm kiếm giải pháp cho các vấn đề gặp phải.

**2.2.1.3 Lợi ích của Spring Boot trong xây dựng ứng dụng kiến trúc Microservice**

Spring Boot cung cấp nhiều tính năng vượt trội giúp dễ dàng triển khai kiến trúc microservice:

**2.2.1.3.1. Modularity (Tính Module)**

Spring Boot cho phép chia ứng dụng thành các module nhỏ độc lập, mỗi module có thể được đóng gói thành một microservice riêng biệt. Mỗi service có thể có database riêng, dependencies riêng, và được deploy độc lập. Điều này rất quan trọng trong một hệ thống microservice với nhiều dịch vụ độc lập.

**2.2.1.3.2. Dependency Injection (DI)**

Spring Framework sử dụng dependency injection để quản lý sự phụ thuộc giữa các thành phần của ứng dụng, giúp giảm sự rắc rối trong việc quản lý các module và dịch vụ. Điều này làm cho việc thêm, thay đổi hoặc loại bỏ các thành phần trở nên dễ dàng hơn, đặc biệt là trong một môi trường microservice có sự phức tạp cao.

**2.2.1.3.3. Auto-configuration**

Spring Boot tự động cấu hình các thành phần dựa trên dependencies có trong classpath, giúp giảm thiểu việc cấu hình thủ công. Điều này đặc biệt hữu ích khi xây dựng nhiều microservices, vì mỗi service có thể được cấu hình tự động dựa trên các dependencies của nó.

**2.2.1.3.4. Scalability (Khả năng Mở Rộng)**

Spring Boot hỗ trợ mô hình mở rộng dễ dàng, cho phép mở rộng các dịch vụ một cách độc lập mà không ảnh hưởng đến các phần khác của hệ thống. Mỗi microservice có thể được scale riêng biệt dựa trên nhu cầu của nó.

**2.2.1.3.5. Flexibility (Tính linh hoạt)**

Spring Boot cung cấp một loạt các tính năng và công nghệ để tùy chỉnh và mở rộng ứng dụng theo cách phù hợp nhất với nhu cầu cụ thể của dự án microservice. Từ REST APIs đến WebSockets, từ SQL databases đến NoSQL, Spring Boot hỗ trợ nhiều công nghệ khác nhau.

**2.2.1.3.6. Ease of Testing (Dễ Kiểm Thử)**

Nhờ vào cấu trúc modularity và dependency injection, việc testing các service trong một ứng dụng microservice được thực hiện dễ dàng hơn. Có thể tách biệt và test từng dịch vụ một cách độc lập mà không cần phải triển khai toàn bộ hệ thống. Spring Boot cung cấp các công cụ testing như @SpringBootTest, MockMvc, và TestContainers.

**2.2.1.4. Tại sao sử dụng Spring Boot trong project**

Trong project nền tảng thương mại điện tử với kiến trúc microservices, Spring Boot được chọn làm framework chính vì các lý do sau:

1. Phù hợp với kiến trúc Microservices:

- Spring Boot cho phép xây dựng 17 microservices độc lập, mỗi service có thể được build thành một standalone JAR file và deploy độc lập. Điều này phù hợp với yêu cầu của kiến trúc microservices, nơi mỗi service cần độc lập về code, database, và deployment.

2. Giảm thiểu thời gian phát triển:

- Với auto-configuration và starter dependencies, việc setup một microservice mới trở nên rất nhanh chóng. Thay vì phải cấu hình thủ công DataSource, JPA, Security, v.v., Spring Boot tự động cấu hình dựa trên dependencies có trong classpath. Điều này giúp giảm đáng kể thời gian phát triển từ setup ban đầu đến production.

3. Tích hợp tốt với các công nghệ khác:

- Spring Boot tích hợp tốt với MySQL thông qua Spring Data JPA và Hibernate, với Redis thông qua Spring Data Redis, và với các công nghệ khác trong hệ thống. Điều này giúp đơn giản hóa việc tích hợp các thành phần của hệ thống.

4. Production-ready:

- Spring Boot Actuator cung cấp các endpoints để monitor health, metrics, và các thông tin khác của ứng dụng. Điều này rất quan trọng trong môi trường production, đặc biệt là với hệ thống có nhiều microservices.

5. Cộng đồng và tài liệu:

- Spring Boot có cộng đồng rất lớn và tài liệu phong phú, giúp dễ dàng tìm kiếm giải pháp cho các vấn đề gặp phải. Điều này đặc biệt quan trọng khi phát triển một hệ thống phức tạp với nhiều microservices.

6. Dễ test và maintain:

- Spring Boot cung cấp các công cụ testing mạnh mẽ, cho phép test từng service độc lập. Điều này giúp đảm bảo chất lượng code và dễ dàng maintain hệ thống.

**2.2.1.5 Ứng dụng trong tiểu luận**

Trong tiểu luận này, Spring Boot được sử dụng làm framework chính để xây dựng 17 microservices độc lập, mỗi service có database riêng và có thể deploy độc lập. Mỗi microservice được xây dựng như một Spring Boot application độc lập với các dependencies riêng, có thể được build thành một standalone JAR file và deploy trên các server khác nhau.

Các microservices sử dụng Spring Boot REST Controllers để expose REST APIs, Spring Data JPA để tương tác với MySQL database, và Spring Security để xử lý authentication và authorization. Spring Boot Actuator được sử dụng để monitor health và metrics của các services.

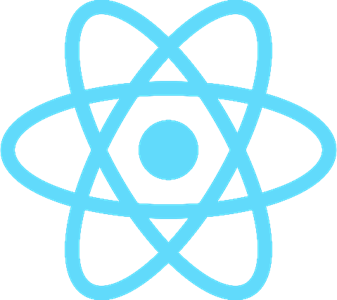
Điều này giúp hệ thống có khả năng mở rộng cao, dễ bảo trì và phát triển. Mỗi service có thể được phát triển, test, và deploy độc lập, giúp tăng tốc độ phát triển và giảm thiểu rủi ro khi thay đổi code.

**2.2.2 React.js và TypeScript**

**2.2.2.1. React.js là gì**

React.js là một thư viện JavaScript mã nguồn mở được phát triển bởi Facebook (Meta), được sử dụng để xây dựng giao diện người dùng (UI) cho các ứng dụng web. React sử dụng khái niệm component-based architecture, cho phép xây dựng các giao diện phức tạp từ các component đơn giản và có thể tái sử dụng.

React được ra mắt lần đầu vào năm 2013 và nhanh chóng trở thành một trong những công cụ phổ biến nhất trong lĩnh vực phát triển web nhờ vào tính hiệu quả và dễ sử dụng. Một trong những đặc điểm nổi bật của React là cơ chế Virtual DOM, cho phép tối ưu hóa quá trình cập nhật giao diện bằng cách chỉ thay đổi những phần cần thiết thay vì toàn bộ trang.



*Hình 2.19: Logo của React Framework*

**2.2.2.2. TypeScript là gì**

TypeScript là một superset của JavaScript được phát triển bởi Microsoft, thêm static type checking vào JavaScript. TypeScript được biên dịch (compile) thành JavaScript thuần, có nghĩa là code TypeScript có thể chạy trên bất kỳ môi trường nào hỗ trợ JavaScript.

TypeScript cung cấp các tính năng như type annotations, interfaces, enums, và generics, giúp phát hiện lỗi tại compile-time thay vì runtime, giảm thiểu bugs trong production. TypeScript cũng cung cấp better IDE support với autocomplete, refactoring tools tốt hơn nhờ type information, và tự động document code thông qua types.

**2.2.2.3. Ưu điểm của React.js**

React.js mang lại nhiều ưu điểm vượt trội cho việc phát triển giao diện người dùng:

- \*\*Component-based Architecture\*\*: Giao diện được chia nhỏ thành các component độc lập, mỗi component quản lý state và UI riêng của nó. Điều này giúp code dễ đọc, dễ maintain, và có thể tái sử dụng.

- \*\*Virtual DOM\*\*: React sử dụng Virtual DOM để tối ưu hóa việc render, chỉ cập nhật những phần thay đổi thực sự trong DOM thực tế. Điều này giúp tăng hiệu suất ứng dụng đáng kể.

- \*\*Unidirectional Data Flow\*\*: Dữ liệu chảy một chiều từ parent component xuống child component, giúp dễ dàng debug và maintain. Kiến trúc này đảm bảo tính dự đoán và dễ quản lý state.

- \*\*JSX Syntax\*\*: Cho phép viết HTML-like syntax trong JavaScript, làm code dễ đọc và maintain hơn. JSX được biên dịch thành JavaScript thuần bởi Babel.

- \*\*Ecosystem phong phú\*\*: React có cộng đồng phát triển rất lớn với nhiều thư viện và tools hỗ trợ như React Router, Redux, Material-UI, v.v.

- \*\*Code Splitting\*\*: Hỗ trợ code splitting để tối ưu hóa thời gian tải trang bằng cách chỉ load các component cần thiết.

**2.2.2.4. Ưu điểm của TypeScript**

TypeScript mang lại nhiều lợi ích khi phát triển ứng dụng:

- \*\*Type Safety\*\*: Phát hiện lỗi tại compile-time thay vì runtime, giảm thiểu bugs trong production. TypeScript sẽ cảnh báo về các lỗi type mismatch, undefined variables, và các lỗi logic khác.

- \*\*Better IDE Support\*\*: Autocomplete, refactoring tools tốt hơn nhờ type information. IDE có thể hiểu rõ hơn về cấu trúc code và cung cấp suggestions chính xác hơn.

- \*\*Code Documentation\*\*: Types tự động document code, giúp developers hiểu rõ hơn về API và cách sử dụng. Types serve as inline documentation.

- \*\*Refactoring an toàn\*\*: Với type information, việc refactor code trở nên an toàn hơn vì TypeScript sẽ cảnh báo về các breaking changes.

- \*\*Improved Developer Experience\*\*: TypeScript giúp developers viết code nhanh hơn và ít lỗi hơn nhờ IntelliSense và type checking.

**2.2.2.5. Tại sao sử dụng React.js và TypeScript trong project**

Trong project nền tảng thương mại điện tử, React.js và TypeScript được chọn làm công nghệ frontend vì các lý do sau:

1. Component-based Architecture phù hợp với UI phức tạp:

- Nền tảng thương mại điện tử có nhiều trang và component phức tạp như ProductList, ProductDetail, ShoppingCart, Checkout, OrderHistory, AdminDashboard, v.v. React.js cho phép chia nhỏ UI thành các component có thể tái sử dụng, giúp code dễ quản lý và maintain. Ví dụ: ProductCard component có thể được sử dụng ở nhiều nơi như trang chủ, trang danh mục, trang tìm kiếm.

2. Hiệu suất cao với Virtual DOM:

- Với số lượng sản phẩm lớn và nhiều tương tác của người dùng (thêm vào giỏ hàng, filter, sort, pagination), React Virtual DOM giúp tối ưu hóa việc render, chỉ cập nhật những phần thay đổi thực sự. Điều này đảm bảo trải nghiệm người dùng mượt mà ngay cả khi có nhiều dữ liệu.

3. Type Safety với TypeScript:

- Khi giao tiếp với 17 microservices backend, việc đảm bảo type safety là rất quan trọng. TypeScript giúp phát hiện lỗi tại compile-time, đảm bảo frontend code sử dụng đúng cấu trúc dữ liệu từ backend APIs. Ví dụ: khi gọi API từ Product Service, TypeScript sẽ đảm bảo rằng response có đúng cấu trúc Product interface đã định nghĩa.

4. Ecosystem phong phú:

- React có ecosystem rất lớn với nhiều thư viện hỗ trợ như React Router cho routing, Axios cho HTTP requests, Material-UI cho UI components. Điều này giúp tăng tốc độ phát triển và giảm thiểu code cần viết.

5. Dễ tích hợp với Material-UI:

- Material-UI được xây dựng trên React, cho phép tích hợp dễ dàng để tạo ra giao diện đẹp và professional. Điều này đặc biệt quan trọng cho một nền tảng thương mại điện tử cần giao diện thu hút và dễ sử dụng.

6. Developer Experience:

- TypeScript cung cấp IntelliSense và autocomplete tốt, giúp developers viết code nhanh hơn và ít lỗi hơn. Điều này đặc biệt quan trọng khi làm việc với nhiều APIs từ các microservices khác nhau.

7. Code Splitting và Lazy Loading:

- Với ứng dụng có nhiều trang và tính năng, React code splitting giúp tối ưu hóa thời gian tải trang bằng cách chỉ load các component cần thiết. Điều này cải thiện trải nghiệm người dùng, đặc biệt là trên mobile.

**2.2.2.6. Ứng dụng trong tiểu luận**

Trong tiểu luận này, React.js 18 kết hợp với TypeScript được sử dụng để xây dựng frontend cho nền tảng thương mại điện tử. Các component được tổ chức theo cấu trúc modular, chia thành các thư mục như `components`, `pages`, `services`, `types`, `hooks`, v.v.

Material-UI (MUI) được sử dụng để tạo giao diện đẹp và responsive, cung cấp các component UI sẵn có như Button, TextField, Card, Dialog, DataGrid. TypeScript giúp đảm bảo type safety khi giao tiếp với backend APIs thông qua Axios, định nghĩa interfaces cho các API requests và responses, giảm thiểu lỗi và cải thiện developer experience.

React Router được sử dụng để quản lý routing và navigation trong ứng dụng single-page application (SPA), cho phép điều hướng giữa các trang mà không cần reload trang. React Hooks như useState, useEffect, useContext được sử dụng để quản lý state và side effects trong functional components.

Việc sử dụng React và TypeScript giúp tạo ra một frontend application mạnh mẽ, dễ bảo trì, và có hiệu suất cao, đáp ứng được các yêu cầu của một nền tảng thương mại điện tử hiện đại.

**2.2.3 MySQL**

**2.2.3.1. MySQL là gì ?**

MySQL là một hệ quản trị cơ sở dữ liệu quan hệ (RDBMS - Relational Database Management System) mã nguồn mở, được sử dụng rộng rãi trong các ứng dụng web. MySQL được phát triển bởi Oracle Corporation và sử dụng mô hình client-server, hỗ trợ SQL (Structured Query Language) để quản lý và truy vấn dữ liệu.

MySQL là một trong những database phổ biến nhất trên thế giới, được sử dụng bởi nhiều ứng dụng web lớn như Facebook, Twitter, YouTube, và nhiều hệ thống enterprise khác. MySQL hỗ trợ đầy đủ các tính năng của một RDBMS hiện đại như transactions, foreign keys, indexes, stored procedures, triggers, và views.



*Hình 2.19: Logo của MySQL*

**2.2.3.2. Ưu điểm của MySQL**

MySQL mang lại nhiều ưu điểm vượt trội cho việc lưu trữ và quản lý dữ liệu:

- Hiệu suất cao và ổn định: MySQL được tối ưu hóa cho hiệu suất cao, phù hợp cho các ứng dụng production với lượng truy vấn lớn. MySQL sử dụng các kỹ thuật như query caching, connection pooling, và indexing để tối ưu hóa hiệu suất.

- ACID Properties: Đảm bảo tính nhất quán dữ liệu thông qua Atomicity (tính nguyên tử), Consistency (tính nhất quán), Isolation (tính cô lập), và Durability (tính bền vững). Điều này đảm bảo tính toàn vẹn dữ liệu trong các thao tác phức tạp.

- Relational Model: Dữ liệu được tổ chức thành các bảng (tables) với các mối quan hệ giữa chúng thông qua foreign keys. Mô hình này giúp giảm thiểu data redundancy và đảm bảo data integrity.

- Transaction Support: Hỗ trợ transactions để đảm bảo tính toàn vẹn dữ liệu trong các thao tác phức tạp. Transactions cho phép nhóm nhiều thao tác thành một đơn vị công việc, đảm bảo tất cả hoặc không có gì được thực hiện.

- Indexing: Sử dụng indexes để tăng tốc độ truy vấn dữ liệu. MySQL hỗ trợ nhiều loại indexes như B-tree, Hash, Full-text, và Spatial indexes.

- Dễ sử dụng: MySQL có cú pháp SQL đơn giản và dễ học, phù hợp cho cả developers mới bắt đầu và experienced developers.

- Cộng đồng hỗ trợ lớn: MySQL có cộng đồng phát triển rất lớn với nhiều tài liệu, tutorials, và forums hỗ trợ.

- Tích hợp tốt: Tích hợp tốt với các framework như Spring Boot thông qua JPA/Hibernate, cho phép developers làm việc với database một cách dễ dàng thông qua ORM.

- Hỗ trợ đầy đủ các tính năng RDBMS: MySQL hỗ trợ đầy đủ các tính năng như transactions, constraints, triggers, stored procedures, views, và functions.

**2.2.3.3. Cách thức hoạt động của MySQL**

MySQL hoạt động dựa trên mô hình client-server:

- Relational Model: Dữ liệu được tổ chức thành các bảng (tables) với các mối quan hệ giữa chúng thông qua foreign keys. Mỗi bảng có các cột (columns) định nghĩa kiểu dữ liệu và các ràng buộc (constraints).

- ACID Properties: Đảm bảo tính nhất quán dữ liệu thông qua:

- Atomicity: Mỗi transaction là một đơn vị công việc không thể chia nhỏ, tất cả hoặc không có gì được thực hiện.

- Consistency: Database luôn ở trạng thái nhất quán, không vi phạm các ràng buộc.

- Isolation: Các transactions đồng thời không ảnh hưởng lẫn nhau.

- Durability: Dữ liệu đã commit được lưu trữ vĩnh viễn, không bị mất khi có sự cố.

- Indexing: Sử dụng indexes để tăng tốc độ truy vấn dữ liệu. Indexes là cấu trúc dữ liệu giúp MySQL tìm kiếm dữ liệu nhanh hơn mà không cần scan toàn bộ bảng.

- Transaction Support: Hỗ trợ transactions để đảm bảo tính toàn vẹn dữ liệu trong các thao tác phức tạp. Transactions cho phép nhóm nhiều SQL statements thành một đơn vị công việc.

- Connection Management: MySQL quản lý các kết nối từ clients, sử dụng connection pooling để tối ưu hóa hiệu suất.

**2.2.3.4. Tại sao sử dụng MySQL trong project**

Trong project nền tảng thương mại điện tử với kiến trúc microservices, MySQL được chọn làm database chính vì các lý do sau:

1. Phù hợp với Database per Service Pattern:

- Trong kiến trúc microservices, mỗi service cần có database riêng để đảm bảo tính độc lập. MySQL cho phép dễ dàng tạo và quản lý nhiều databases (product\_db, order\_db, user\_db, v.v.), mỗi database chỉ chứa dữ liệu liên quan đến service đó. Điều này giúp mỗi service có thể scale và maintain độc lập.

2. Đảm bảo Data Consistency:

- Trong thương mại điện tử, tính nhất quán dữ liệu là rất quan trọng. ACID properties của MySQL đảm bảo tính nhất quán dữ liệu trong các thao tác phức tạp như tạo đơn hàng, thanh toán, cập nhật tồn kho. Ví dụ: khi tạo đơn hàng, cần đảm bảo rằng tồn kho được trừ đúng, đơn hàng được tạo thành công, và thanh toán được xử lý đúng - tất cả phải thành công hoặc tất cả phải rollback.

3. Tích hợp tốt với Spring Boot:

- MySQL tích hợp tốt với Spring Boot thông qua Spring Data JPA và Hibernate. JPA/Hibernate tự động tạo schema thông qua `ddl-auto: update`, map entities với tables, và xử lý các thao tác CRUD. Điều này giúp giảm thiểu code cần viết và đơn giản hóa việc quản lý database.

4. Hiệu suất cao:

- MySQL được tối ưu hóa cho hiệu suất cao với query optimization, indexing, và connection pooling. Điều này đặc biệt quan trọng cho các service có lượng truy vấn lớn như Product Service (tìm kiếm sản phẩm), Order Service (xem lịch sử đơn hàng), v.v.

5. Hỗ trợ đầy đủ các tính năng RDBMS:

- MySQL hỗ trợ đầy đủ các tính năng như transactions, foreign keys, constraints, triggers, stored procedures. Điều này giúp đảm bảo tính toàn vẹn dữ liệu và thực hiện các business rules phức tạp ở database level.

6. Cộng đồng và tài liệu:

- MySQL có cộng đồng rất lớn và tài liệu phong phú, giúp dễ dàng tìm kiếm giải pháp cho các vấn đề gặp phải. Điều này đặc biệt quan trọng khi phát triển một hệ thống phức tạp.

7. Chi phí:

- MySQL là mã nguồn mở và miễn phí, phù hợp cho các dự án học thuật và startup. Điều này giúp giảm chi phí phát triển và vận hành hệ thống.

**2.2.3.5. Ứng dụng trong tiểu luận**

Trong tiểu luận này, MySQL 8.0 được sử dụng làm cơ sở dữ liệu chính cho tất cả các microservices. Mỗi service có database riêng (database per service pattern) để đảm bảo tính độc lập và có thể scale riêng biệt.

Ví dụ: Product Service có database `product\_db`, Order Service có database `order\_db`, User Service có database `user\_db`, v.v. Mỗi database chỉ chứa dữ liệu liên quan đến service đó, đảm bảo tính độc lập về dữ liệu.

Spring Data JPA và Hibernate được sử dụng làm ORM (Object-Relational Mapping) để ánh xạ các entity Java với các bảng trong database. Hibernate tự động tạo schema thông qua `ddl-auto: update`, giúp đơn giản hóa việc quản lý database schema. Các entity được định nghĩa với các annotations như `@Entity`, `@Table`, `@Column`, `@OneToMany`, `@ManyToOne`, v.v.

Điều này giúp đảm bảo tính nhất quán dữ liệu trong mỗi service, đơn giản hóa việc quản lý database, và cho phép mỗi service scale độc lập dựa trên nhu cầu của nó.

**2.2.4. Redis**

**2.2.4.1. Redis là gì**

Redis (Remote Dictionary Server) là một in-memory data structure store mã nguồn mở, được sử dụng như một database, cache, và message broker. Redis lưu trữ dữ liệu trong RAM, cho phép truy cập và thao tác dữ liệu với tốc độ cực kỳ nhanh (microseconds).

Redis được phát triển bởi Salvatore Sanfilippo và được sử dụng rộng rãi trong các ứng dụng web hiện đại để cải thiện hiệu suất và giảm tải cho database. Redis hỗ trợ nhiều kiểu dữ liệu như strings, hashes, lists, sets, sorted sets, và streams, làm cho nó phù hợp cho nhiều use cases khác nhau.



*Hình 2.19: Logo của Redis*

**2.2.4.2. Ưu điểm của Redis**

Redis mang lại nhiều ưu điểm vượt trội cho việc lưu trữ và quản lý dữ liệu:

- Tốc độ cực kỳ nhan\*: Nhờ in-memory storage, Redis có thể đọc/ghi dữ liệu với tốc độ rất cao (microseconds), nhanh hơn rất nhiều so với disk-based databases.

- Hỗ trợ nhiều kiểu dữ liệu: Redis hỗ trợ nhiều kiểu dữ liệu như strings, hashes, lists, sets, sorted sets, streams, và bitmaps, phù hợp cho nhiều use cases khác nhau.

- Atomic operations: Tất cả các operations trong Redis đều là atomic, đảm bảo tính nhất quán dữ liệu ngay cả trong môi trường concurrent.

- Persistence Options: Redis có thể lưu trữ dữ liệu vào disk thông qua RDB snapshots hoặc AOF (Append-Only File), đảm bảo dữ liệu không bị mất khi server restart.

- Pub/Sub: Hỗ trợ publish/subscribe pattern cho messaging, cho phép các services giao tiếp bất đồng bộ.

- Linh hoạt trong sử dụng: Redis có thể được sử dụng như cache, session store, message broker, hoặc primary database tùy theo nhu cầu.

- Scalability: Redis hỗ trợ replication và clustering để scale horizontally.

**2.2.4.3. Cách thức hoạt động của Redis**

Redis hoạt động dựa trên các nguyên tắc sau:

- In-Memory Storage: Dữ liệu được lưu trữ trong RAM, cho phép đọc/ghi với tốc độ rất cao (microseconds). Điều này làm cho Redis phù hợp cho các use cases cần tốc độ cao như caching, session management, real-time analytics.

- Data Structures: Redis hỗ trợ nhiều kiểu dữ liệu:

- Strings: Lưu trữ strings, integers, hoặc floats.

- Hashes: Lưu trữ các cặp key-value, phù hợp cho objects.

- Lists: Danh sách các strings được sắp xếp theo thứ tự insert.

- Sets: Tập hợp các strings không có thứ tự, không trùng lặp.

- Sorted Sets: Sets với score để sắp xếp, phù hợp cho leaderboards.

- Streams: Log-like data structure cho event streaming.

- Persistence Options: Redis cung cấp hai phương thức persistence:

- RDB (Redis Database): Tạo snapshot của database vào disk theo định kỳ.

- AOF (Append-Only File): Ghi lại tất cả các write operations vào file log.

- Pub/Sub: Hỗ trợ publish/subscribe pattern cho messaging, cho phép các services giao tiếp bất đồng bộ thông qua channels.

- TTL (Time To Live): Redis hỗ trợ TTL cho các keys, tự động xóa keys sau một khoảng thời gian nhất định.

**2.2.4.4. Tại sao sử dụng Redis trong project**

Trong project nền tảng thương mại điện tử với kiến trúc microservices, Redis được chọn vì các lý do sau:

1. Tối ưu hiệu suất với Caching:

- Với số lượng người dùng và sản phẩm lớn, việc truy vấn database liên tục sẽ gây tải cao và chậm. Redis được sử dụng như caching layer để cache các dữ liệu thường xuyên được truy cập như thông tin sản phẩm, danh mục, warranty requests. Điều này giúp giảm tải cho database và tăng tốc độ phản hồi của hệ thống đáng kể.

2. Lưu trữ Shopping Cart:

- Cart Service sử dụng Redis làm primary storage cho giỏ hàng của người dùng với TTL 24 giờ. Vì giỏ hàng được truy cập và cập nhật thường xuyên (thêm/xóa sản phẩm, cập nhật số lượng), việc lưu trữ trong Redis với tốc độ đọc/ghi cao (microseconds) đảm bảo trải nghiệm người dùng mượt mà. Nếu lưu trong MySQL, mỗi lần thêm/xóa sản phẩm sẽ phải query database, gây chậm và tải cao.

3. Session và Token Management:

- Auth Service sử dụng Redis để lưu trữ session data và token blacklist. Khi user logout, token được thêm vào blacklist trong Redis với TTL bằng expiration time của token. Khi validate token, hệ thống kiểm tra xem token có trong blacklist không - việc này cần tốc độ cao và Redis phù hợp cho use case này.

4. User Behavior Tracking:

- Recommendation Service sử dụng Redis để track user behavior (sản phẩm đã xem, đã mua, ratings) để phân tích và tạo recommendations. Dữ liệu này được cập nhật thường xuyên và cần tốc độ cao, Redis phù hợp cho use case này.

5. Giảm tải cho Database:

- Với 17 microservices, mỗi service có database riêng, việc giảm tải cho database là rất quan trọng. Redis giúp giảm số lượng queries đến database bằng cách cache các dữ liệu thường xuyên được truy cập, giúp database tập trung vào các thao tác quan trọng hơn.

6. Tốc độ cực nhanh:

- Redis lưu trữ dữ liệu trong RAM, cho phép đọc/ghi với tốc độ cực nhanh (microseconds). Điều này đặc biệt quan trọng cho các use cases cần tốc độ cao như shopping cart, session management, real-time tracking.

7. Hỗ trợ TTL:

- Redis hỗ trợ TTL (Time To Live) cho các keys, tự động xóa keys sau một khoảng thời gian nhất định. Điều này hữu ích cho shopping cart (TTL 24 giờ), session data, và token blacklist.

**2.2.4.5. Ứng dụng trong tiểu luận**

Trong tiểu luận này, Redis được sử dụng trong nhiều microservices với các mục đích khác nhau:

- Cart Service: Redis được sử dụng làm primary storage cho giỏ hàng của người dùng với TTL 24 giờ. Mỗi user có một key riêng trong Redis chứa thông tin giỏ hàng của họ. Việc sử dụng Redis làm primary storage giúp tăng tốc độ truy cập và giảm tải cho database.

- Auth Service: Redis được sử dụng để lưu trữ session data và token blacklist. Khi user logout, token được thêm vào blacklist trong Redis với TTL bằng với expiration time của token. Khi validate token, hệ thống kiểm tra xem token có trong blacklist không.

- Warranty Service: Redis được sử dụng để cache warranty requests, giảm số lượng queries đến database khi user xem lại các warranty requests đã tạo.

- Recommendation Service: Redis được sử dụng để track user behavior và cache recommendation results. User behavior data được lưu trữ trong Redis để phân tích và tạo recommendations.

Việc sử dụng Redis giúp giảm tải cho database, tăng tốc độ phản hồi của hệ thống, và cải thiện trải nghiệm người dùng. Cart Service sử dụng Redis làm primary storage với TTL 24 giờ, trong khi các service khác sử dụng Redis như một cache layer với fallback về database khi Redis không available.

**2.2.5. Eureka Service Discovery**

**2.2.5.1. Eureka là gì ?**

Eureka là một service registry và discovery tool được phát triển bởi Netflix, được tích hợp vào Spring Cloud. Eureka cho phép các microservices tự động đăng ký và khám phá lẫn nhau mà không cần cấu hình tĩnh IP/port.

Eureka hoạt động theo mô hình client-server, trong đó Eureka Server đóng vai trò là service registry, và các microservices đóng vai trò là Eureka Clients. Khi một service khởi động, nó tự động đăng ký với Eureka Server, và các services khác có thể query Eureka Server để tìm các instances của service đó.

**2.2.5.2. Ưu điểm của Eureka**

Eureka mang lại nhiều ưu điểm vượt trội cho việc quản lý microservices:

- Tự động Service Discovery: Các services tự động đăng ký và khám phá lẫn nhau mà không cần cấu hình tĩnh IP/port. Điều này giúp giảm thiểu công việc cấu hình và quản lý.

- High Availability: Eureka hỗ trợ high availability với multiple Eureka servers trong một cluster. Các Eureka servers có thể replicate với nhau để đảm bảo tính sẵn sàng cao.

- Health Monitoring: Eureka Server định kỳ kiểm tra health của các registered services và loại bỏ các unhealthy instances. Điều này đảm bảo chỉ các services đang hoạt động được sử dụng.

- Load Balancing: Khi có nhiều instances của cùng một service, Eureka cung cấp danh sách để client có thể load balance. Spring Cloud LoadBalancer sử dụng thông tin từ Eureka để thực hiện load balancing.

- Tích hợp tốt với Spring Cloud: Eureka tích hợp tốt với Spring Cloud và Spring Cloud Gateway, cho phép Gateway tự động discover và route requests đến các services.

- Self-preservation Mode: Eureka có cơ chế self-preservation để bảo vệ registry khỏi network partitions, tránh loại bỏ tất cả instances khi có vấn đề về mạng.

**2.2.5.3. Cách thức hoạt động của Eureka**

Eureka hoạt động dựa trên các cơ chế sau:

- Service Registration: Khi một service khởi động, nó tự động đăng ký với Eureka Server, cung cấp thông tin như service name, host, port, health status, và metadata. Service gửi heartbeat đến Eureka Server định kỳ để báo hiệu rằng nó vẫn đang hoạt động.

- Service Discovery: Các services khác có thể query Eureka Server để tìm các instances của một service cụ thể. Eureka Server trả về danh sách các instances đang available, và client có thể chọn một instance để gọi.

- Health Checks: Eureka Server định kỳ kiểm tra health của các registered services thông qua heartbeat. Nếu một service không gửi heartbeat trong một khoảng thời gian nhất định, Eureka Server sẽ đánh dấu nó là unhealthy và loại bỏ khỏi registry.

- Load Balancing: Khi có nhiều instances của cùng một service, Eureka cung cấp danh sách để client có thể load balance. Spring Cloud LoadBalancer sử dụng thông tin từ Eureka để thực hiện round-robin hoặc các thuật toán load balancing khác.

- Replication: Trong một Eureka cluster, các Eureka servers replicate registry với nhau để đảm bảo tính sẵn sàng cao. Mỗi Eureka server có thể đóng vai trò là peer của các servers khác.

**2.2.5.4. Tại sao sử dụng Eureka trong project**

Trong project nền tảng thương mại điện tử với 17 microservices, Eureka được chọn làm service discovery vì các lý do sau:

1. Quản lý 17 Microservices:

- Với 17 microservices độc lập, việc quản lý địa chỉ IP/port của từng service là rất phức tạp và dễ lỗi. Eureka cho phép các services tự động đăng ký và discover lẫn nhau thông qua service name (ví dụ: `product-service`, `order-service`), không cần biết địa chỉ IP/port cụ thể. Điều này giúp đơn giản hóa việc quản lý và cấu hình hệ thống.

2. Tự động Load Balancing:

- Khi một service cần scale (ví dụ: Product Service cần xử lý nhiều requests hơn), chỉ cần start thêm instances mới. Eureka tự động đăng ký các instances mới, và Gateway/Client tự động load balance giữa tất cả instances. Điều này giúp phân phối tải một cách tự động và hiệu quả mà không cần cấu hình thủ công.

3. Fault Tolerance:

- Eureka tự động loại bỏ các unhealthy instances khỏi registry thông qua health checks. Nếu một service instance bị crash hoặc không phản hồi, Eureka sẽ tự động loại bỏ nó và chỉ route requests đến các instances còn hoạt động. Điều này giúp tăng tính resilient của hệ thống và đảm bảo tính sẵn sàng cao.

4. Tích hợp với API Gateway:

- Spring Cloud Gateway sử dụng Eureka để discover các services và route requests. Gateway có thể sử dụng service name (ví dụ: `lb://product-service`) thay vì IP/port cụ thể, giúp đơn giản hóa cấu hình routing. Khi có instances mới được thêm vào hoặc instances cũ bị loại bỏ, Gateway tự động cập nhật mà không cần restart.

5. Dynamic Service Discovery:

- Trong môi trường development hoặc cloud, IP addresses có thể thay đổi. Eureka cho phép các services tự động discover lẫn nhau mà không cần cấu hình tĩnh, giúp hệ thống linh hoạt và dễ dàng deploy trên các môi trường khác nhau.

6. Health Monitoring:

- Eureka cung cấp dashboard để monitor health và status của tất cả services trong hệ thống. Điều này giúp dễ dàng theo dõi và quản lý hệ thống, đặc biệt quan trọng với hệ thống có nhiều microservices.

7. Giảm thiểu cấu hình:

- Không cần cấu hình tĩnh IP/port cho từng service, giảm thiểu công việc cấu hình và nguy cơ lỗi. Mỗi service chỉ cần cấu hình Eureka Server URL, và Eureka sẽ tự động quản lý phần còn lại.

**2.2.5.5. Ứng dụng trong tiểu luận**

Trong tiểu luận này, Eureka Server được triển khai ở port 8761, đóng vai trò là service registry cho toàn bộ hệ thống. Tất cả 17 microservices đều đăng ký với Eureka khi khởi động bằng cách sử dụng `@EnableEurekaClient` annotation và cấu hình Eureka Server URL trong `application.yml`.

Mỗi service cung cấp thông tin như service name (ví dụ: `product-service`, `order-service`), host, port, và health status khi đăng ký với Eureka. Eureka Server hiển thị danh sách tất cả các registered services trên dashboard tại `http://localhost:8761`.

Spring Cloud Gateway sử dụng Eureka để discover các services và thực hiện load balancing. Gateway được cấu hình với `spring.cloud.gateway.discovery.locator.enabled=true` và sử dụng `lb://service-name` prefix để route requests đến các services thông qua Eureka. Ví dụ: `lb://product-service` sẽ tự động load balance giữa tất cả instances của Product Service.

Điều này cho phép hệ thống tự động phát hiện các service instances mới, loại bỏ các instances không còn hoạt động, và phân phối tải một cách tự động mà không cần cấu hình thủ công. Khi một service instance mới được start, nó tự động đăng ký với Eureka và bắt đầu nhận requests. Khi một service instance bị stop, Eureka tự động loại bỏ nó khỏi registry sau một khoảng thời gian không nhận được heartbeat.

#### 2.2.6. JWT – Json web token

**2.2.6.1. JWT là gì ?**

JSON Web Token (JWT) là một chuẩn mở (RFC 7519) được sử dụng để tạo ra và xác thực các token an toàn giữa hai bên bằng cách sử dụng JSON. JWT được sử dụng rộng rãi để xác thực và phân quyền trong các ứng dụng web và API.

JWT bao gồm ba phần chính được ngăn cách bởi dấu chấm (`.`): Header, Payload, và Signature. Header chứa loại token và thuật toán mã hóa được sử dụng. Payload chứa các claims (thông tin về người dùng hoặc các dữ liệu mở rộng khác). Signature được tạo ra từ Header, Payload và một chìa khóa bí mật, đảm bảo tính toàn vẹn của token và xác thực nguồn gốc của nó.

**2.2.6.2. Ưu điểm của JWT**

JWT có các ưu điểm sau:

- Phù hợp cho mô hình RESTful: JWT thích hợp để được sử dụng trong các ứng dụng RESTful vì nó có thể được gửi như một chuỗi trong header của HTTP request hoặc trong một cookie. JWT không yêu cầu server lưu trữ session state, phù hợp với kiến trúc stateless của REST.

- Khả năng mở rộng: JWT có thể chứa bất kỳ dữ liệu JSON nào, làm cho nó linh hoạt cho các trường hợp sử dụng đa dạng. Developers có thể thêm bất kỳ claims nào vào payload để phục vụ nhu cầu của ứng dụng.

- An toàn: Với chữ ký kỹ thuật số, JWT đảm bảo rằng nó không bị sửa đổi trong quá trình chuyển tiếp và chỉ có thể được xác thực bằng chìa khóa bí mật. Signature được tạo ra từ Header, Payload, và secret key, đảm bảo tính toàn vẹn của token.

- Stateless: Server không cần lưu trữ session, giảm tải cho server và dễ dàng scale. Mỗi request chứa đầy đủ thông tin cần thiết để xác thực và phân quyền, không cần query database hoặc session store.

- Self-contained: Token chứa đầy đủ thông tin cần thiết, không cần query database mỗi lần validate. Điều này giúp giảm tải cho database và tăng tốc độ xử lý requests.

- Cross-domain: Có thể sử dụng JWT giữa các domain khác nhau, phù hợp cho các ứng dụng microservice hoặc distributed systems.

**2.2.6.3. Cách thức hoạt động của JWT**

JWT hoạt động dựa trên các cơ chế sau:

- Cấu trúc JWT: JWT bao gồm ba phần được ngăn cách bởi dấu chấm (`.`):

- Header: Chứa loại token (JWT) và thuật toán mã hóa được sử dụng (ví dụ: HS256, RS256). Header được encode bằng Base64URL.

- Payload: Chứa các claims (thông tin về người dùng như userId, role, email) và các thông tin metadata như expiration time (exp), issued at (iat), issuer (iss), subject (sub). Payload được encode bằng Base64URL.

- Signature: Được tạo ra từ Header, Payload và một secret key (hoặc private key trong trường hợp RS256), đảm bảo tính toàn vẹn của token và xác thực nguồn gốc. Signature được tạo bằng cách: `HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)`

- Token Generation: Khi người dùng đăng nhập thành công, server tạo JWT token chứa thông tin người dùng (claims) và gửi về client. Token thường có expiration time để đảm bảo bảo mật.

- Token Validation: Khi client gửi request kèm JWT token trong header `Authorization: Bearer <token>`, server validate token bằng cách:

- Kiểm tra signature để đảm bảo token không bị sửa đổi. Server tạo lại signature từ Header và Payload với secret key, và so sánh với Signature trong token.

- Kiểm tra expiration time (exp) để đảm bảo token còn hiệu lực.

- Kiểm tra issued at (iat) và not before (nbf) nếu có.

- Extract claims để lấy thông tin người dùng và phân quyền.

**2.2.6.4. Tại sao sử dụng JWT trong project**

Trong project nền tảng thương mại điện tử với kiến trúc microservices, JWT được chọn làm phương thức xác thực vì các lý do sau:

1. Stateless Authentication phù hợp với Microservices:

- Với 17 microservices độc lập, việc sử dụng session-based authentication sẽ rất phức tạp vì cần shared session store và session replication. JWT cho phép xác thực stateless - mỗi request chứa đầy đủ thông tin cần thiết để xác thực và phân quyền, không cần lưu trữ session trên server. Điều này phù hợp với kiến trúc microservices nơi mỗi service độc lập.

2. Giảm tải cho Auth Service:

- Nếu sử dụng session-based authentication, mỗi request đến bất kỳ service nào đều phải gọi Auth Service để validate session, gây tải cao cho Auth Service. Với JWT, Auth Service chỉ cần tạo token một lần khi user đăng nhập, và các services khác tự validate token mà không cần gọi Auth Service. Điều này giúp giảm tải cho Auth Service và tăng tốc độ xử lý requests.

3. Authorization độc lập:

- JWT token chứa thông tin về role và permissions của user (ví dụ: USER, ADMIN). Mỗi service có thể extract claims từ JWT token để lấy userId và role, và thực hiện authorization độc lập mà không cần gọi Auth Service. Ví dụ: Order Service có thể kiểm tra xem user có phải là owner của order không, Product Service có thể kiểm tra xem user có quyền ADMIN để cập nhật sản phẩm không.

4. Tích hợp với API Gateway:

- Spring Cloud Gateway có thể validate JWT token một lần trước khi forward requests đến các backend services. Nếu token không hợp lệ, Gateway sẽ trả về 401 Unauthorized ngay lập tức mà không cần forward đến backend. Điều này giúp bảo vệ các backend services và giảm tải cho chúng.

5. Scalability:

- Vì JWT là stateless, hệ thống có thể scale dễ dàng mà không cần lo lắng về session replication hoặc shared session store. Mỗi service instance có thể validate JWT token độc lập mà không cần chia sẻ state với các instances khác.

6. Cross-domain Support:

- JWT có thể được sử dụng giữa các domain khác nhau, phù hợp cho các ứng dụng có frontend và backend ở các domain khác nhau hoặc có nhiều frontend applications.

7. Security:

- JWT được ký bằng secret key, đảm bảo token không bị giả mạo hoặc sửa đổi. Token có expiration time để đảm bảo bảo mật - nếu token bị lộ, nó sẽ hết hạn sau một khoảng thời gian nhất định.

**2.2.6.5. Ứng dụng trong tiểu luận**

Trong tiểu luận này, JWT được sử dụng để xác thực người dùng trong toàn bộ hệ thống. Auth Service tạo JWT token khi người dùng đăng nhập thành công, token này chứa thông tin như userId, email, và role (USER, ADMIN, etc.). Token có expiration time (ví dụ: 24 giờ) để đảm bảo bảo mật.

Spring Cloud Gateway validate JWT token trước khi forward requests đến các backend services. Gateway sử dụng JWT filter để extract token từ `Authorization: Bearer <token>` header, validate signature và expiration time, và extract claims để lấy thông tin user. Nếu token không hợp lệ hoặc đã hết hạn, Gateway sẽ trả về 401 Unauthorized.

Các services sử dụng JWT để xác định người dùng hiện tại và thực hiện phân quyền dựa trên role. Mỗi service có thể extract claims từ JWT token để lấy userId và role, và thực hiện authorization dựa trên role. Ví dụ: chỉ ADMIN mới có thể truy cập các endpoints quản lý, trong khi USER chỉ có thể truy cập các endpoints thông thường.

Token được lưu trữ ở client (localStorage hoặc sessionStorage) và được gửi kèm trong mỗi request thông qua Authorization header. Frontend sử dụng Axios interceptor để tự động thêm JWT token vào mỗi request.

JWT giúp đảm bảo tính bảo mật và phân quyền trong hệ thống microservices, cho phép mỗi service thực hiện authentication và authorization độc lập mà không cần gọi Auth Service mỗi lần.

**2.2.7. OAuth2**

**2.2.7.1. OAuth2 là gì ?**

OAuth2 là một framework authorization được sử dụng để cho phép các ứng dụng truy cập tài nguyên của người dùng từ các nhà cung cấp dịch vụ (như Google, Facebook) mà không cần chia sẻ mật khẩu. OAuth2 cung cấp các flow khác nhau để phù hợp với các loại ứng dụng khác nhau.

OAuth2 được phát triển như một chuẩn mở (RFC 6749) để giải quyết vấn đề authorization trong các ứng dụng web và mobile. OAuth2 cho phép người dùng cấp quyền truy cập tài nguyên của họ cho các ứng dụng bên thứ ba mà không cần chia sẻ credentials.

**2.2.7.2. Ưu điểm của OAuth2**

OAuth2 mang lại nhiều ưu điểm vượt trội cho việc xác thực và authorization:

- Người dùng không cần tạo tài khoản mới: Người dùng có thể sử dụng tài khoản Google/Facebook hiện có để đăng nhập, giảm friction trong quá trình đăng ký/đăng nhập.

- Tăng tính bảo mật: Không cần lưu trữ mật khẩu của người dùng, giảm rủi ro về bảo mật. OAuth2 providers (Google, Facebook) chịu trách nhiệm quản lý credentials và xác thực người dùng.

- Cải thiện trải nghiệm người dùng: Single sign-on (SSO) cho phép người dùng đăng nhập một lần và sử dụng nhiều ứng dụng mà không cần đăng nhập lại.

- Tuân thủ các chuẩn bảo mật: OAuth2 là một chuẩn được công nhận rộng rãi, được sử dụng bởi nhiều nhà cung cấp dịch vụ lớn như Google, Facebook, Microsoft, v.v.

- Linh hoạt: OAuth2 cung cấp nhiều flow khác nhau (Authorization Code, Implicit, Client Credentials, Resource Owner Password Credentials) để phù hợp với các loại ứng dụng khác nhau.

- Phân quyền chi tiết: OAuth2 cho phép người dùng cấp quyền truy cập cụ thể cho từng ứng dụng, không phải toàn bộ tài khoản.

**2.2.7.3. Cách thức hoạt động của OAuth2**

OAuth2 hoạt động dựa trên Authorization Code Flow (flow phổ biến nhất cho web applications):

- Authorization Flow:

1. Ứng dụng redirect người dùng đến nhà cung cấp OAuth2 (Google/Facebook) với client\_id, redirect\_uri, scope, và state.

2. Người dùng đăng nhập vào tài khoản của họ trên provider và cấp quyền cho ứng dụng.

3. Nhà cung cấp redirect người dùng về redirect\_uri với authorization code.

4. Ứng dụng đổi authorization code lấy access token bằng cách gửi POST request đến token endpoint với client\_id, client\_secret, và authorization code.

5. Ứng dụng sử dụng access token để truy cập thông tin người dùng từ provider API (ví dụ: Google UserInfo API, Facebook Graph API).

- Provider Integration:

- Google OAuth2: Sử dụng Google Identity Platform để xác thực. Google cung cấp OAuth2 endpoints và UserInfo API để lấy thông tin người dùng.

- Facebook OAuth2: Sử dụng Facebook Login để xác thực. Facebook cung cấp OAuth2 endpoints và Graph API để lấy thông tin người dùng.

**2.2.7.4. Tại sao sử dụng OAuth2 trong project**

Trong project nền tảng thương mại điện tử, OAuth2 được tích hợp để cho phép người dùng đăng nhập bằng tài khoản Google hoặc Facebook vì các lý do sau:

1. Giảm Friction trong quá trình đăng ký/đăng nhập:

- Trong thương mại điện tử, việc yêu cầu người dùng tạo tài khoản mới với email và mật khẩu có thể tạo ra friction và giảm tỷ lệ chuyển đổi. OAuth2 cho phép người dùng đăng nhập bằng tài khoản Google/Facebook hiện có chỉ với một vài cú click, giúp giảm thời gian đăng ký/đăng nhập và tăng tỷ lệ chuyển đổi.

2. Tăng tính bảo mật:

- Việc lưu trữ và quản lý mật khẩu của người dùng là một trách nhiệm lớn và rủi ro về bảo mật. OAuth2 giúp giảm rủi ro này bằng cách không cần lưu trữ mật khẩu - OAuth2 providers (Google, Facebook) chịu trách nhiệm quản lý credentials và xác thực người dùng. Điều này đặc biệt quan trọng cho một nền tảng thương mại điện tử xử lý thông tin thanh toán và đơn hàng.

3. Cải thiện trải nghiệm người dùng:

- OAuth2 cung cấp single sign-on (SSO) - người dùng đã đăng nhập vào Google/Facebook có thể đăng nhập vào website ngay lập tức mà không cần nhập lại thông tin. Điều này tạo ra trải nghiệm mượt mà và thuận tiện cho người dùng.

4. Tăng tỷ lệ chuyển đổi:

- Với OAuth2, quá trình đăng ký/đăng nhập trở nên nhanh chóng và dễ dàng hơn, giúp giảm tỷ lệ bỏ giỏ hàng và tăng tỷ lệ chuyển đổi. Người dùng có thể nhanh chóng đăng nhập và tiếp tục mua sắm mà không bị gián đoạn bởi quá trình đăng ký phức tạp.

5. Tích hợp với JWT:

- Sau khi xác thực thành công với OAuth2 provider, hệ thống tạo JWT token chứa thông tin user (userId, email, role) để sử dụng trong các request tiếp theo. Điều này kết hợp ưu điểm của OAuth2 (dễ đăng nhập) và JWT (stateless authentication trong microservices).

6. Phù hợp với thị trường Việt Nam:

- Google và Facebook là hai nền tảng phổ biến nhất tại Việt Nam, hầu hết người dùng đều có tài khoản. Việc tích hợp OAuth2 với Google và Facebook giúp tăng khả năng người dùng sử dụng tính năng đăng nhập nhanh.

7. Giảm chi phí quản lý:

- Không cần xây dựng và maintain hệ thống quản lý mật khẩu, reset password, email verification, v.v. OAuth2 providers xử lý tất cả các tính năng này.

**2.2.7.5. Ứng dụng trong tiểu luận**

Trong tiểu luận này, OAuth2 được tích hợp để cho phép người dùng đăng nhập bằng tài khoản Google hoặc Facebook. Auth Service xử lý OAuth2 flow:

1. Frontend redirect người dùng đến OAuth2 provider (Google/Facebook) với client\_id và redirect\_uri.

2. Người dùng đăng nhập và cấp quyền cho ứng dụng trên provider.

3. Provider redirect về redirect\_uri với authorization code.

4. Frontend gửi authorization code đến Auth Service.

5. Auth Service đổi authorization code lấy access token từ provider.

6. Auth Service sử dụng access token để lấy thông tin người dùng từ provider API (Google UserInfo API hoặc Facebook Graph API).

7. Auth Service kiểm tra xem user đã tồn tại trong database chưa. Nếu chưa, tạo user mới; nếu có, cập nhật thông tin.

8. Auth Service tạo JWT token chứa thông tin user (userId, email, role) và trả về cho frontend.

9. Frontend lưu JWT token và sử dụng trong các request tiếp theo.

OAuth2 giúp giảm friction trong quá trình đăng ký/đăng nhập, tăng tỷ lệ chuyển đổi và cải thiện trải nghiệm người dùng. Người dùng không cần nhớ thêm một mật khẩu mới, và quá trình đăng nhập trở nên nhanh chóng và thuận tiện hơn.

**2.2.8. Gemini API**

**2.2.8.1. Gemini API là gì ?**

Gemini API là một API trí tuệ nhân tạo được phát triển bởi Google, cung cấp khả năng xử lý ngôn ngữ tự nhiên (NLP), hiểu ngữ cảnh, và tạo ra phản hồi thông minh. Gemini API hỗ trợ nhiều loại tác vụ AI như chat, text generation, và content understanding.

Gemini API sử dụng các mô hình ngôn ngữ lớn (LLM) được train trên lượng dữ liệu khổng lồ, cho phép hiểu và xử lý ngôn ngữ tự nhiên một cách chính xác và tự nhiên. API có thể duy trì ngữ cảnh trong cuộc hội thoại, hiểu các tham chiếu và câu hỏi tiếp theo, và tạo ra phản hồi phù hợp với ngữ cảnh.



*Hình 2.19: Logo của Gemini*

**2.2.8.2. Ưu điểm của Gemini API**

Gemini API mang lại nhiều ưu điểm vượt trội:

- Khả năng hiểu ngôn ngữ tự nhiên tốt: Gemini API có khả năng hiểu ngôn ngữ tự nhiên rất tốt, đặc biệt là tiếng Việt, giúp tạo ra trải nghiệm tương tác tự nhiên cho người dùng Việt Nam.

- Response time nhanh: API có response time nhanh, đảm bảo trải nghiệm người dùng mượt mà và không bị gián đoạn.

- Chất lượng phản hồi cao: Gemini API tạo ra phản hồi chất lượng cao, phù hợp với ngữ cảnh và yêu cầu của người dùng.

- Dễ dàng tích hợp: Gemini API dễ dàng tích hợp thông qua REST API, cho phép developers tích hợp AI capabilities vào ứng dụng một cách nhanh chóng.

- Hỗ trợ nhiều use cases: API hỗ trợ nhiều use cases từ chatbot đến content generation, phù hợp cho nhiều loại ứng dụng khác nhau.

- Context Awareness: API có thể duy trì ngữ cảnh trong cuộc hội thoại, hiểu các tham chiếu và câu hỏi tiếp theo, tạo ra trải nghiệm tương tác tự nhiên hơn.

**2.2.8.3. Cách thức hoạt động của Gemini API**

Gemini API hoạt động dựa trên các cơ chế sau:

- Natural Language Processing: Gemini API sử dụng các mô hình ngôn ngữ lớn (LLM) để hiểu và xử lý ngôn ngữ tự nhiên. LLM được train trên lượng dữ liệu khổng lồ, cho phép hiểu ngữ nghĩa, ngữ cảnh, và ý định của người dùng.

- Context Awareness: API có thể duy trì ngữ cảnh trong cuộc hội thoại, hiểu các tham chiếu và câu hỏi tiếp theo. Điều này cho phép tạo ra các cuộc hội thoại tự nhiên và liên tục.

- Content Generation: Tạo ra nội dung văn bản dựa trên prompt và context được cung cấp. API có thể tạo ra các phản hồi phù hợp với ngữ cảnh và yêu cầu của người dùng.

- Multi-turn Conversation: Hỗ trợ các cuộc hội thoại nhiều lượt với khả năng nhớ context. API có thể theo dõi lịch sử hội thoại và sử dụng thông tin này để tạo ra phản hồi phù hợp.

**2.2.8.4. Tại sao sử dụng Gemini API trong project**

Trong project nền tảng thương mại điện tử, Gemini API được chọn để tích hợp AI chatbot vì các lý do sau:

1. Không cần train và deploy mô hình AI:

- Việc train và deploy một mô hình AI từ đầu là rất phức tạp, tốn kém về thời gian và tài nguyên. Gemini API cho phép tích hợp AI capabilities vào hệ thống một cách dễ dàng thông qua REST API, không cần phải train và deploy các mô hình AI phức tạp. Điều này giúp giảm đáng kể thời gian và chi phí phát triển.

2. Hỗ trợ tiếng Việt tốt:

- Gemini API có khả năng hiểu và xử lý tiếng Việt rất tốt, đặc biệt quan trọng cho một nền tảng thương mại điện tử tại Việt Nam. Chatbot có thể hiểu câu hỏi của người dùng Việt Nam và tạo ra phản hồi tự nhiên, phù hợp với ngữ cảnh và văn hóa Việt Nam.

3. Cải thiện dịch vụ khách hàng:

- Chatbot sử dụng Gemini API có thể hỗ trợ khách hàng 24/7, trả lời câu hỏi về sản phẩm, chính sách, đơn hàng, v.v. mà không cần đội ngũ customer support luôn online. Điều này giúp giảm chi phí vận hành và cải thiện trải nghiệm người dùng.

4. Tư vấn sản phẩm thông minh:

- Chatbot có thể phân tích câu hỏi của người dùng, hiểu ý định (ví dụ: "Tôi cần tìm điện thoại giá rẻ"), tìm kiếm sản phẩm phù hợp trong database, và tạo ra phản hồi tự nhiên với thông tin sản phẩm. Điều này tạo ra trải nghiệm tư vấn sản phẩm tự nhiên và thông minh, giúp tăng tỷ lệ chuyển đổi.

5. Context Awareness:

- Gemini API có thể duy trì ngữ cảnh trong cuộc hội thoại, hiểu các tham chiếu và câu hỏi tiếp theo. Ví dụ: người dùng hỏi "Giá bao nhiêu?", chatbot có thể hiểu rằng đang nói về sản phẩm được đề cập trước đó. Điều này tạo ra trải nghiệm tương tác tự nhiên hơn.

6. Dễ tích hợp:

- Gemini API dễ dàng tích hợp thông qua REST API, cho phép AI Service gọi API và nhận phản hồi một cách đơn giản. Không cần cấu hình phức tạp hoặc infrastructure đặc biệt.

7. Response time nhanh:

- Gemini API có response time nhanh, đảm bảo chatbot phản hồi người dùng một cách nhanh chóng và mượt mà, không làm gián đoạn trải nghiệm người dùng.

**2.2.8.5. Ứng dụng trong tiểu luận**

Trong tiểu luận này, Gemini API được sử dụng trong AI Service để cung cấp chatbot trí tuệ nhân tạo cho nền tảng thương mại điện tử. Chatbot sử dụng Gemini API để hiểu câu hỏi của người dùng, tìm kiếm sản phẩm trong database, và tạo ra phản hồi tự nhiên và hữu ích.

Khi người dùng gửi câu hỏi, AI Service gửi prompt đến Gemini API kèm theo context về sản phẩm và chính sách của website. Gemini API phân tích câu hỏi, hiểu ý định của người dùng, và tạo ra phản hồi phù hợp. Chatbot có thể tư vấn sản phẩm, trả lời câu hỏi về chính sách, và hỗ trợ người dùng 24/7.

Gemini API giúp tạo ra trải nghiệm tương tác tự nhiên và thông minh, cải thiện dịch vụ khách hàng và tăng tỷ lệ chuyển đổi. Chatbot có thể hiểu ngôn ngữ tự nhiên của người dùng Việt Nam và tạo ra phản hồi phù hợp, tạo ra trải nghiệm tương tác gần như với con người.

**2.2.9. Python FastAPI và Scikit-learn**

**2.2.9.1. FastAPI là gì ?**

FastAPI là một framework web hiện đại, nhanh (high-performance) được xây dựng cho Python để xây dựng APIs. FastAPI dựa trên các tiêu chuẩn Python hiện đại như type hints và async/await, được phát triển bởi Sebastián Ramírez.

FastAPI được thiết kế để có hiệu suất cao tương đương với Node.js và Go, đồng thời cung cấp tính năng tự động generate API documentation (Swagger/OpenAPI) và type safety thông qua type hints.

**2.2.9.2. Scikit-learn là gì ?**

Scikit-learn là một thư viện machine learning mã nguồn mở cho Python, cung cấp các công cụ đơn giản và hiệu quả cho data mining và data analysis. Scikit-learn được xây dựng trên NumPy, SciPy, và matplotlib, và cung cấp một API nhất quán cho các thuật toán machine learning.

**2.2.9.3. Ưu điểm của FastAPI**

FastAPI mang lại nhiều ưu điểm vượt trội:

- Hiệu suất cao: FastAPI có hiệu suất cao tương đương với Node.js và Go nhờ sử dụng Starlette và Pydantic.

- Tự động documentation: FastAPI tự động generate API documentation (Swagger/OpenAPI) dựa trên code, giúp developers không cần viết documentation thủ công.

- Type safety: Sử dụng type hints để tự động validate request/response, giúp phát hiện lỗi tại development time.

- Async/await support: Hỗ trợ async/await cho hiệu suất cao khi xử lý nhiều requests đồng thời.

- Dễ sử dụng: FastAPI có syntax đơn giản và dễ học, phù hợp cho cả developers mới và experienced.

**2.2.9.4. Ưu điểm của Scikit-learn**

Scikit-learn mang lại nhiều ưu điểm:

- Dễ sử dụng: Scikit-learn có API nhất quán và dễ sử dụng, phù hợp cho cả beginners và experts.

- Nhiều thuật toán ML: Cung cấp nhiều thuật toán machine learning như classification, regression, clustering, dimensionality reduction, v.v.

- Cộng đồng lớn: Scikit-learn có cộng đồng rất lớn với nhiều tài liệu, tutorials, và examples.

- Tích hợp tốt: Tích hợp tốt với các thư viện Python khác như NumPy, Pandas, matplotlib.

**2.2.9.5. Tại sao sử dụng FastAPI và Scikit-learn trong project**

Trong project nền tảng thương mại điện tử, FastAPI và Scikit-learn được chọn để xây dựng ML Recommendation Service vì các lý do sau:

1. Python phù hợp cho Machine Learning:

- Python có ecosystem rất phong phú cho machine learning với nhiều thư viện như Scikit-learn, TensorFlow, PyTorch, Pandas, NumPy. Việc sử dụng Python cho ML service cho phép tận dụng các thư viện này một cách dễ dàng.

2. Tách biệt ML Service:

- Việc tách ML Recommendation Service riêng (port 8000) cho phép scale độc lập với các Java services. ML service có thể được scale riêng dựa trên nhu cầu tính toán (training models, predictions), không ảnh hưởng đến các services khác.

3. FastAPI hiệu suất cao:

- FastAPI có hiệu suất cao, phù hợp cho việc xử lý nhiều requests prediction đồng thời. Điều này đặc biệt quan trọng khi có nhiều người dùng yêu cầu recommendations cùng lúc.

4. Scikit-learn cho Collaborative Filtering:

- Scikit-learn cung cấp thuật toán SVD (Singular Value Decomposition) cho Collaborative Filtering, phù hợp cho việc gợi ý sản phẩm dựa trên ratings và reviews của người dùng. Thuật toán này đã được chứng minh hiệu quả trong các hệ thống recommendation.

5. Dễ tích hợp với Java Services:

- FastAPI expose REST APIs, cho phép Recommendation Service (Java) gọi ML service một cách dễ dàng thông qua HTTP requests. Không cần phức tạp về protocol hoặc serialization.

6. Tự động documentation:

- FastAPI tự động generate Swagger documentation, giúp developers dễ dàng test và sử dụng ML APIs.

7. Type safety:

- FastAPI sử dụng type hints để validate request/response, đảm bảo dữ liệu được truyền đúng format giữa Java service và Python service.

**2.2.9.6. Ứng dụng trong tiểu luận**

Trong tiểu luận này, Python FastAPI được sử dụng để xây dựng ML Recommendation Service (port 8000), cung cấp các API để gợi ý sản phẩm dựa trên AI. Service sử dụng Scikit-learn để train và sử dụng mô hình Collaborative Filtering (SVD) dựa trên dữ liệu reviews và ratings của người dùng.

Recommendation Service (Java) gửi request đến ML Recommendation Service với userId, và ML service predict ratings cho các sản phẩm dựa trên mô hình đã train, sau đó trả về danh sách sản phẩm được gợi ý. Việc tách ML service riêng cho phép scale độc lập và sử dụng các thư viện Python phong phú cho machine learning mà không cần tích hợp vào Java services.

**2.2.10. VNPay**

**2.2.10.1. VNPay là gì ?**

VNPay là một cổng thanh toán điện tử phổ biến tại Việt Nam, cung cấp các giải pháp thanh toán trực tuyến cho các doanh nghiệp. VNPay hỗ trợ nhiều phương thức thanh toán như thẻ ATM nội địa, thẻ quốc tế, ví điện tử, và các ngân hàng địa phương tại Việt Nam.

**2.2.10.2. Ưu điểm của VNPay**

VNPay mang lại nhiều ưu điểm:

- Phù hợp với thị trường Việt Nam: VNPay hỗ trợ nhiều ngân hàng địa phương tại Việt Nam, phù hợp với thói quen thanh toán của người dùng Việt Nam.

- Hỗ trợ nhiều phương thức thanh toán: VNPay hỗ trợ thẻ ATM nội địa, thẻ quốc tế, ví điện tử, và các phương thức thanh toán khác.

- Tích hợp dễ dàng: VNPay cung cấp API và SDK để tích hợp dễ dàng vào các ứng dụng web và mobile.

- Bảo mật cao: VNPay sử dụng chữ ký số để đảm bảo tính toàn vẹn của các giao dịch.

**2.2.10.3. Tại sao sử dụng VNPay trong project**

Trong project nền tảng thương mại điện tử, cả VNPay được tích hợp vì các lý do sau:

1. Phục vụ cả khách hàng Việt Nam và quốc tế:

- VNPay được sử dụng cho khách hàng Việt Nam với các ngân hàng nội địa. Điều này giúp mở rộng phạm vi khách hàng và tăng khả năng thanh toán thành công.

2. Tăng tỷ lệ thanh toán thành công:

- Việc cung cấp nhiều phương thức thanh toán giúp người dùng có nhiều lựa chọn, tăng khả năng họ tìm thấy phương thức phù hợp và hoàn thành thanh toán. Điều này đặc biệt quan trọng trong thương mại điện tử, nơi mỗi giao dịch bị hủy đều là mất doanh thu.

3. Phù hợp với thị trường Việt Nam:

- VNPay là cổng thanh toán phổ biến nhất tại Việt Nam, hỗ trợ nhiều ngân hàng địa phương. Hầu hết người dùng Việt Nam đều quen thuộc với VNPay, giúp tăng độ tin cậy và tỷ lệ sử dụng.

4. Bảo mật và tuân thủ:

- VNPay tuân thủ các tiêu chuẩn bảo mật cao, đảm bảo an toàn cho thông tin thanh toán của người dùng. Điều này đặc biệt quan trọng cho một nền tảng thương mại điện tử xử lý thông tin thanh toán nhạy cảm.

5. Tích hợp dễ dàng:

- VNPay cung cấp API và SDK rõ ràng, dễ tích hợp vào Payment Service. Payment Service có thể xử lý cả hai payment gateway một cách thống nhất.

6. Webhook và Callback:

- VNPay hỗ trợ callback/webhook để thông báo về kết quả thanh toán, cho phép Payment Service xử lý bất đồng bộ và cập nhật trạng thái đơn hàng một cách đáng tin cậy.

**2.2.10.4. Ứng dụng trong tiểu luận**

Trong tiểu luận này, Payment Service tích hợp cả VNPay để cung cấp nhiều phương thức thanh toán cho người dùng. Khi người dùng checkout, họ có thể chọn thanh toán bằng VNPay.

VNPay được sử dụng cho khách hàng Việt Nam với các ngân hàng nội địa. Payment Service tạo payment URL với thông tin đơn hàng và signature, redirect người dùng đến VNPay payment page. Sau khi thanh toán, VNPay gọi callback URL với kết quả, và Payment Service verify signature và cập nhật trạng thái đơn hàng trong Order Service.

Việc tích hợp nhiều payment gateway giúp tăng khả năng thanh toán thành công và cải thiện trải nghiệm người dùng, đặc biệt quan trọng cho một nền tảng thương mại.

**2.2.11. GHN Shipping API**

**2.2.11.1. GHN Shipping API là gì ?**

GHN (Giao Hàng Nhanh) là một dịch vụ vận chuyển hàng hóa tại Việt Nam, cung cấp API để tích hợp tính toán phí vận chuyển và tạo đơn hàng vận chuyển. GHN là một trong những đơn vị vận chuyển lớn nhất tại Việt Nam, phục vụ trên toàn quốc.

GHN API cho phép các nền tảng thương mại điện tử tích hợp tính toán phí vận chuyển, tạo đơn hàng vận chuyển, và theo dõi trạng thái vận chuyển một cách tự động.

2.2.11.2. Ưu điểm của GHN Shipping API

GHN Shipping API mang lại nhiều ưu điểm:

- Tích hợp dễ dàng: GHN cung cấp REST API rõ ràng, dễ tích hợp vào các ứng dụng web và mobile.

- Tính toán phí vận chuyển chính xác: GHN API tính toán phí vận chuyển dựa trên địa chỉ gửi/nhận, trọng lượng, và các yếu tố khác, đảm bảo tính chính xác và real-time.

- Hỗ trợ theo dõi đơn hàng: GHN API cung cấp thông tin tracking để theo dõi trạng thái vận chuyển của đơn hàng, giúp người dùng biết được đơn hàng đang ở đâu.

- Master Data: GHN API cung cấp danh sách tỉnh/thành, quận/huyện, phường/xã để hiển thị trong form nhập địa chỉ, giúp đảm bảo tính chính xác của địa chỉ.

- Phù hợp với thị trường Việt Nam: GHN là dịch vụ vận chuyển phổ biến tại Việt Nam, phục vụ trên toàn quốc với mạng lưới rộng lớn.

**2.2.11.3. Tại sao sử dụng GHN Shipping API trong project**

Trong project nền tảng thương mại điện tử, GHN Shipping API được tích hợp vì các lý do sau:

1. Tính toán phí vận chuyển chính xác:

- Khi người dùng checkout, họ cần biết phí vận chuyển để quyết định có tiếp tục mua hàng không. GHN API tính toán phí vận chuyển real-time dựa trên địa chỉ gửi/nhận và trọng lượng, đảm bảo tính chính xác và minh bạch. Điều này giúp tăng độ tin cậy và giảm tỷ lệ hủy đơn hàng do phí vận chuyển không mong đợi.

2. Tự động hóa quy trình vận chuyển:

- Sau khi thanh toán thành công, hệ thống tự động tạo shipping order trên GHN và cập nhật tracking number vào đơn hàng. Điều này giúp tự động hóa quy trình vận chuyển, giảm công việc thủ công và tăng hiệu quả.

3. Cải thiện trải nghiệm người dùng:

- Người dùng có thể theo dõi trạng thái vận chuyển trực tiếp trên website thông qua tracking number. Điều này giúp người dùng biết được đơn hàng đang ở đâu, khi nào sẽ được giao, cải thiện trải nghiệm và giảm số lượng câu hỏi về đơn hàng.

4. Master Data cho form địa chỉ:

- GHN API cung cấp danh sách tỉnh/thành, quận/huyện, phường/xã để hiển thị trong form nhập địa chỉ giao hàng. Điều này giúp đảm bảo tính chính xác của địa chỉ và giảm lỗi do người dùng nhập sai địa chỉ.

5. Phù hợp với thị trường Việt Nam:

- GHN là dịch vụ vận chuyển phổ biến tại Việt Nam, phục vụ trên toàn quốc với mạng lưới rộng lớn. Việc tích hợp GHN API giúp đảm bảo khả năng giao hàng đến mọi nơi tại Việt Nam.

6. Tích hợp dễ dàng:

- GHN cung cấp REST API rõ ràng, dễ tích hợp vào Order Service. Order Service có thể gọi GHN API để tính phí vận chuyển, tạo đơn hàng, và theo dõi trạng thái một cách dễ dàng.

7. Giảm chi phí vận hành:

- Việc tự động hóa quy trình vận chuyển giúp giảm công việc thủ công, giảm chi phí vận hành và tăng hiệu quả. Nhân viên không cần phải tạo đơn hàng vận chuyển thủ công, hệ thống tự động xử lý.

**2.2.11.4. Ứng dụng trong tiểu luận**

Trong tiểu luận này, Order Service tích hợp GHN API để tính toán phí vận chuyển khi người dùng checkout và tạo đơn hàng vận chuyển sau khi thanh toán thành công.

Khi người dùng checkout, Order Service gọi GHN API để tính phí vận chuyển dựa trên địa chỉ gửi/nhận và trọng lượng đơn hàng. Phí vận chuyển được hiển thị cho người dùng trước khi họ quyết định thanh toán.

GHN API cũng được sử dụng để lấy danh sách địa chỉ (tỉnh/thành, quận/huyện) để hiển thị trong form nhập địa chỉ giao hàng. Điều này giúp đảm bảo tính chính xác của địa chỉ và cải thiện trải nghiệm người dùng.

Sau khi thanh toán thành công, Order Service tự động tạo shipping order trên GHN và cập nhật tracking number vào đơn hàng. Người dùng có thể theo dõi trạng thái vận chuyển trực tiếp trên website thông qua tracking number, biết được đơn hàng đang ở đâu và khi nào sẽ được giao.

**2.2.12. Material-UI (MUI)**

**2.2.12.1. Material-UI là gì ?**

Material-UI (MUI) là một thư viện React component được xây dựng dựa trên Material Design guidelines của Google. MUI cung cấp một bộ component UI đẹp, responsive, và có thể tùy biến, giúp developers xây dựng giao diện người dùng một cách nhanh chóng và chuyên nghiệp.

MUI được phát triển bởi cộng đồng và là một trong những thư viện UI component phổ biến nhất cho React, với hàng triệu lượt tải xuống mỗi tuần.

2.2.12.2. Ưu điểm của Material-UI

Material-UI mang lại nhiều ưu điểm vượt trội:

- Component đẹp và professional: MUI cung cấp các component đẹp và professional out-of-the-box, tuân thủ Material Design guidelines, giúp tạo ra giao diện hiện đại và thu hút.

- Responsive tự động: Các component tự động responsive trên các kích thước màn hình khác nhau, phù hợp với cả mobile và desktop mà không cần viết CSS responsive thủ công.

- Dễ tùy biến: MUI hỗ trợ theming để tùy biến màu sắc, typography, spacing, giúp tạo ra giao diện phù hợp với brand identity của ứng dụng.

- Component Library phong phú: MUI cung cấp nhiều component sẵn có như Button, TextField, Card, Dialog, DataGrid, Table, Form, v.v., giúp giảm thiểu code cần viết.

- Accessibility: MUI tuân thủ các tiêu chuẩn accessibility (WCAG), đảm bảo ứng dụng có thể sử dụng được bởi mọi người, kể cả người khuyết tật.

- Cộng đồng lớn: MUI có cộng đồng rất lớn với nhiều tài liệu, tutorials, và examples, giúp dễ dàng tìm kiếm giải pháp cho các vấn đề gặp phải.

**2.2.12.3. Tại sao sử dụng Material-UI trong project**

Trong project nền tảng thương mại điện tử, Material-UI được chọn làm thư viện UI component vì các lý do sau:

1. Tăng tốc độ phát triển:

- Với nhiều component sẵn có như Button, TextField, Card, Dialog, DataGrid, developers không cần viết CSS từ đầu, giúp tăng tốc độ phát triển frontend đáng kể. Thay vì phải code từng component, developers chỉ cần import và sử dụng.

2. Giao diện đẹp và professional:

- MUI tuân thủ Material Design guidelines, giúp tạo ra giao diện đẹp, hiện đại, và professional. Điều này đặc biệt quan trọng cho một nền tảng thương mại điện tử cần giao diện thu hút và dễ sử dụng để tăng tỷ lệ chuyển đổi.

3. Responsive tự động:

- Với số lượng người dùng mobile ngày càng tăng, việc có giao diện responsive là rất quan trọng. MUI components tự động responsive, giúp tiết kiệm thời gian và đảm bảo giao diện hoạt động tốt trên mọi thiết bị.

4. DataGrid cho Admin Dashboard:

- MUI DataGrid cung cấp các tính năng mạnh mẽ như pagination, sorting, filtering, export, giúp xây dựng Admin Dashboard để quản lý sản phẩm, đơn hàng, người dùng một cách dễ dàng. Thay vì phải code từ đầu, developers chỉ cần sử dụng DataGrid component.

5. Tính nhất quán:

- MUI giúp đảm bảo tính nhất quán trong giao diện - tất cả các component đều tuân thủ cùng một design system, giúp giao diện trông chuyên nghiệp và nhất quán.

6. Tích hợp tốt với React và TypeScript:

- MUI được xây dựng trên React và hỗ trợ TypeScript, phù hợp với stack công nghệ của project. MUI components có type definitions đầy đủ, giúp phát hiện lỗi tại compile-time.

7. Dễ maintain:

- Với MUI, việc maintain và update giao diện trở nên dễ dàng hơn. Khi cần thay đổi style hoặc thêm component mới, developers chỉ cần sử dụng MUI components và theming, không cần viết CSS phức tạp.

8. Cộng đồng và tài liệu:

- MUI có cộng đồng rất lớn và tài liệu phong phú, giúp dễ dàng tìm kiếm giải pháp cho các vấn đề gặp phải. Điều này đặc biệt quan trọng khi phát triển một ứng dụng phức tạp với nhiều component.

**2.2.12.4. Ứng dụng trong tiểu luận**

Trong tiểu luận này, Material-UI được sử dụng để xây dựng giao diện frontend cho nền tảng thương mại điện tử. Các component như ProductCard, ShoppingCart, OrderForm được xây dựng dựa trên MUI components như Card, Button, TextField, Dialog.

MUI DataGrid được sử dụng trong Admin Dashboard để hiển thị danh sách sản phẩm, đơn hàng, người dùng với các tính năng như pagination, sorting, filtering, export. Điều này giúp xây dựng Admin Dashboard một cách nhanh chóng và chuyên nghiệp mà không cần code từ đầu.

MUI theming được sử dụng để tùy biến màu sắc, typography, spacing phù hợp với brand identity của nền tảng. Tất cả các component đều tự động responsive, đảm bảo giao diện hoạt động tốt trên cả mobile và desktop.

MUI giúp tạo ra giao diện đẹp, nhất quán, và responsive mà không cần viết CSS từ đầu, tăng tốc độ phát triển frontend và giảm thời gian maintain.

# CHƯƠNG 3. GIẢI PHÁP CHO BÀI TOÁN

### 3.1. Phát biểu bài toán trong đề tài

Từ những khảo sát và đánh giá các nền tảng thương mại điện tử hiện nay trên thị trường cho thấy có nhiều vấn đề và hạn chế cần được giải quyết:

- Vấn đề về khả năng mở rộng: Các hệ thống thương mại điện tử truyền thống thường được xây dựng theo kiến trúc monolithic, khiến việc mở rộng hệ thống trở nên khó khăn và tốn kém. Khi một phần của hệ thống cần scale, phải scale toàn bộ hệ thống, dẫn đến lãng phí tài nguyên và chi phí cao.

- Vấn đề về hiệu suất: Với số lượng người dùng và sản phẩm lớn, các hệ thống truyền thống thường gặp vấn đề về hiệu suất, đặc biệt là khi truy vấn database và xử lý các tác vụ phức tạp. Việc không có caching layer hiệu quả dẫn đến thời gian phản hồi chậm và trải nghiệm người dùng kém.

- Vấn đề về bảo mật và xác thực: Nhiều nền tảng yêu cầu người dùng tạo tài khoản mới với email và mật khẩu, tạo ra friction trong quá trình đăng ký/đăng nhập và tăng rủi ro về bảo mật khi phải lưu trữ và quản lý mật khẩu của người dùng.

- Vấn đề về thanh toán: Các nền tảng thường chỉ hỗ trợ một hoặc một số ít phương thức thanh toán, không đáp ứng được nhu cầu đa dạng của người dùng. Điều này đặc biệt quan trọng tại Việt Nam, nơi người dùng có nhiều lựa chọn về ngân hàng và phương thức thanh toán.

- Vấn đề về dịch vụ khách hàng: Hầu hết các nền tảng chỉ cung cấp dịch vụ khách hàng trong giờ hành chính, không đáp ứng được nhu cầu hỗ trợ 24/7 của người dùng. Việc không có chatbot thông minh dẫn đến thời gian chờ đợi lâu và trải nghiệm người dùng kém.

- Vấn đề về gợi ý sản phẩm: Nhiều nền tảng không có hệ thống gợi ý sản phẩm thông minh, khiến người dùng khó tìm được sản phẩm phù hợp với nhu cầu của mình, dẫn đến tỷ lệ chuyển đổi thấp.

- Vấn đề về quản lý đơn hàng và vận chuyển: Việc tính toán phí vận chuyển và quản lý đơn hàng vận chuyển thường được thực hiện thủ công, tốn thời gian và dễ xảy ra lỗi. Người dùng không thể theo dõi trạng thái vận chuyển một cách trực quan và dễ dàng.

Để cải thiện những vấn đề trên, tiểu luận này xây dựng một nền tảng thương mại điện tử hiện đại với kiến trúc microservices, tích hợp trí tuệ nhân tạo và các công nghệ tiên tiến. Hệ thống dành cho các đối tượng như:

- Các doanh nghiệp bán lẻ: Cho phép họ xây dựng cửa hàng trực tuyến với đầy đủ tính năng quản lý sản phẩm, đơn hàng, khách hàng, và báo cáo doanh thu một cách hiệu quả và chuyên nghiệp.

- Các nhà bán hàng cá nhân: Giúp họ dễ dàng tạo và quản lý cửa hàng trực tuyến, tiếp cận khách hàng trên toàn quốc với các công cụ marketing và bán hàng tích hợp.

- Người tiêu dùng: Cung cấp trải nghiệm mua sắm trực tuyến thuận tiện, an toàn, và thông minh với nhiều phương thức thanh toán, giao hàng nhanh chóng, và hệ thống gợi ý sản phẩm cá nhân hóa.

- Các tổ chức và doanh nghiệp: Cho phép họ mua sắm hàng loạt với các tính năng quản lý đơn hàng, báo giá, và thanh toán phù hợp với nhu cầu doanh nghiệp.

Để đảm bảo được việc phục vụ cho các đối tượng trên, hệ thống sẽ có các tính năng như:

- Quản lý sản phẩm: Cho phép người bán tạo, cập nhật, và quản lý danh mục sản phẩm với đầy đủ thông tin như hình ảnh, mô tả, giá cả, tồn kho, và các thuộc tính khác.

- Quản lý đơn hàng: Hệ thống tự động xử lý đơn hàng từ khi đặt hàng đến khi giao hàng, bao gồm tính toán phí vận chuyển, tạo đơn hàng vận chuyển, và theo dõi trạng thái vận chuyển.

- Thanh toán đa dạng: Tích hợp nhiều phương thức thanh toán như VNPay cho khách hàng Việt Nam, đảm bảo người dùng có thể thanh toán một cách thuận tiện và an toàn.

- Xác thực và phân quyền: Hỗ trợ đăng nhập bằng tài khoản Google/Facebook thông qua OAuth2, và sử dụng JWT để xác thực và phân quyền trong toàn bộ hệ thống, đảm bảo tính bảo mật và trải nghiệm người dùng tốt.

- Giỏ hàng thông minh: Lưu trữ giỏ hàng trong Redis với tốc độ cao, cho phép người dùng thêm/xóa sản phẩm một cách mượt mà và giữ giỏ hàng trong 24 giờ.

- Hệ thống gợi ý sản phẩm: Sử dụng machine learning (Collaborative Filtering) để phân tích hành vi người dùng và gợi ý sản phẩm phù hợp, giúp tăng tỷ lệ chuyển đổi và cải thiện trải nghiệm mua sắm.

- Chatbot trí tuệ nhân tạo: Tích hợp Gemini API để cung cấp chatbot thông minh hỗ trợ khách hàng 24/7, trả lời câu hỏi về sản phẩm, chính sách, và đơn hàng một cách tự nhiên và chính xác.

- Quản lý người dùng: Cho phép người dùng đăng ký, đăng nhập, quản lý thông tin cá nhân, xem lịch sử đơn hàng, và theo dõi trạng thái đơn hàng.

- Hệ thống đánh giá và phản hồi: Cho phép người dùng đánh giá và review sản phẩm, giúp người dùng khác đưa ra quyết định mua hàng và cải thiện chất lượng dịch vụ.

- Quản lý khuyến mãi: Hệ thống voucher và khuyến mãi cho phép người bán tạo các chương trình khuyến mãi và người dùng áp dụng voucher khi thanh toán.

- Bảo hành và hỗ trợ: Hệ thống bảo hành cho phép người dùng tạo yêu cầu bảo hành và theo dõi trạng thái xử lý.

- Admin Dashboard: Cung cấp giao diện quản trị để quản lý sản phẩm, đơn hàng, người dùng, và các báo cáo thống kê một cách trực quan và dễ dàng.

- Kiến trúc microservices: Hệ thống được xây dựng với kiến trúc microservices, cho phép mở rộng từng service độc lập dựa trên nhu cầu, đảm bảo hiệu suất cao và khả năng mở rộng linh hoạt.

Hệ thống được xây dựng với mục tiêu tạo ra một nền tảng thương mại điện tử hiện đại, an toàn, hiệu quả, và có khả năng mở rộng cao, đáp ứng được nhu cầu của cả người bán và người mua trong thời đại số hóa hiện nay.

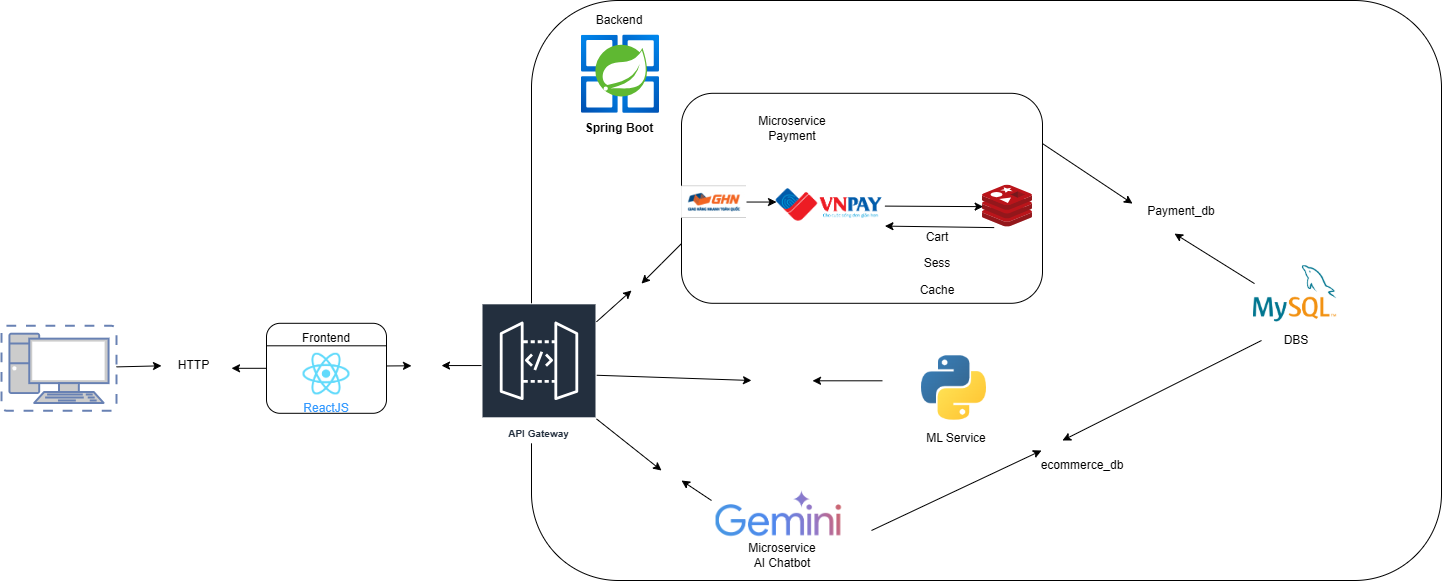
### 3.2. Giải pháp giải quyết bài toán:

#### 3.2.1. Kiến trúc tổng quan.

Giải pháp được thiết kế theo mô hình kiến trúc client-server, trong đó:

* Client: Giao diện người dùng được xây dựng bằng ReactJS, cung cấp một trải nghiệm người dùng mượt mà và tương tác cao.
* Server: Xử lý logic nghiệp vụ, quản lý dữ liệu và cung cấp API RESTful, được triển khai bằng Spring Boot và Spring Cloud có khả tăng tương tác với hợp đồng thông minh trên mạng và lưu trữ dữ liệu bằng MySQL.

#### 3.2.2. Sơ đồ kiến trúc hệ thống



Hình 3.1: Sơ đồ kiến trúc hệ thống

Ý nghĩa của các thành phần:

* Frontend: Dùng ReactJS kết hợp với TypeScript và Material-UI để xây dựng giao diện người dùng cho nền tảng thương mại điện tử. Frontend chịu trách nhiệm nhận các tương tác từ người dùng (xem sản phẩm, thêm vào giỏ hàng, thanh toán, quản lý đơn hàng), hiển thị dữ liệu từ backend, và cung cấp trải nghiệm người dùng mượt mà và trực quan. Frontend giao tiếp với backend thông qua REST API qua API Gateway, đảm bảo tính bảo mật và nhất quán trong giao tiếp.
* API Gateway: Dùng ReactJS kết hợp với TypeScript và Material-UI để xây dựng giao diện người dùng cho nền tảng thương mại điện tử. Frontend chịu trách nhiệm nhận các tương tác từ người dùng (xem sản phẩm, thêm vào giỏ hàng, thanh toán, quản lý đơn hàng), hiển thị dữ liệu từ backend, và cung cấp trải nghiệm người dùng mượt mà và trực quan. Frontend giao tiếp với backend thông qua REST API qua API Gateway, đảm bảo tính bảo mật và nhất quán trong giao tiếp.
* Backend: Dùng Spring Boot để xây dựng 17 microservices độc lập, mỗi service chịu trách nhiệm cho một domain cụ thể (Auth, User, Product, Order, Cart, Payment, Review, Voucher, Warranty, Recommendation, AI, Notification, Admin, Config, Search, Image, Analytics). Mỗi microservice xử lý các yêu cầu từ API Gateway, thực hiện business logic, tương tác với database riêng của nó, và giao tiếp với các services khác hoặc external APIs khi cần. Các services sử dụng Eureka Service Discovery để tự động đăng ký và khám phá lẫn nhau, cho phép hệ thống scale và maintain dễ dàng.
* Database: Dùng hệ quản trị cơ sở dữ liệu MySQL để lưu trữ dữ liệu persistent cho tất cả các microservices. Mỗi service có database riêng (database per service pattern) để đảm bảo tính độc lập và có thể scale riêng biệt. MySQL lưu trữ các thông tin như người dùng, sản phẩm, đơn hàng, thanh toán, đánh giá, khuyến mãi, bảo hành, và các dữ liệu nghiệp vụ khác. MySQL đảm bảo tính nhất quán dữ liệu thông qua ACID properties, đặc biệt quan trọng cho các thao tác phức tạp như tạo đơn hàng và thanh toán.
* Redis:Dùng Redis (in-memory data store) để lưu trữ dữ liệu tạm thời với tốc độ cực nhanh. Redis được sử dụng để lưu giỏ hàng của người dùng (Cart Service sử dụng Redis làm primary storage với TTL 24 giờ), lưu trữ session data và token blacklist trong Auth Service, cache dữ liệu sản phẩm và đơn hàng để giảm tải cho database, và track user behavior trong Recommendation Service. Redis giúp tăng tốc độ phản hồi của hệ thống và cải thiện trải nghiệm người dùng.
* AI service: Dùng Gemini API để cung cấp chatbot trí tuệ nhân tạo cho nền tảng thương mại điện tử. AI Service gọi Gemini API để hiểu câu hỏi của người dùng, tìm kiếm sản phẩm trong database, và tạo ra phản hồi tự nhiên và hữu ích. Chatbot có thể tư vấn sản phẩm, trả lời câu hỏi về chính sách, và hỗ trợ người dùng 24/7. Gemini API giúp tạo ra trải nghiệm tương tác tự nhiên và thông minh, cải thiện dịch vụ khách hàng.
* ML Service: Dùng Python FastAPI và Scikit-learn để xây dựng ML Recommendation Service, cung cấp các API để gợi ý sản phẩm dựa trên AI. ML Service sử dụng thuật toán Collaborative Filtering (SVD - Singular Value Decomposition) dựa trên dữ liệu reviews và ratings của người dùng để predict ratings và gợi ý sản phẩm phù hợp. Recommendation Service (Java) gọi ML Service để lấy gợi ý, và ML Service xử lý và trả về danh sách sản phẩm được gợi ý. Việc tách ML service riêng cho phép scale độc lập và sử dụng các thư viện Python phong phú cho machine learning.
* Shipping API: Dùng GHN (Giao Hàng Nhanh) API để tích hợp tính toán phí vận chuyển và tạo đơn hàng vận chuyển. Order Service tích hợp GHN API để tính phí vận chuyển khi người dùng checkout, lấy danh sách địa chỉ (tỉnh/thành, quận/huyện) để hiển thị trong form, tạo đơn hàng vận chuyển sau khi thanh toán thành công, và theo dõi trạng thái vận chuyển. GHN API giúp tự động hóa quy trình vận chuyển và cải thiện trải nghiệm người dùng.
* Payment Gateways: Dùng VNPay để xử lý thanh toán trực tuyến. Payment Service tích hợp cả hai payment gateway để cung cấp nhiều phương thức thanh toán cho người dùng. VNPay được sử dụng cho khách hàng Việt Nam với các ngân hàng nội địa. Payment Service tạo payment requests, verify callbacks/webhooks từ các payment gateway, và cập nhật trạng thái đơn hàng trong Order Service sau khi thanh toán thành công.

#### 3.2.3. Cơ chế bảo mật

Hệ thống sử dụng chế bảo mật JWT:

* Xác thực và phân quyền: Hệ thống sử dụng JWT để xác thực người dùng khi đăng nhập và phân quyền cho các hoạt động khác. JWT giúp đảm bảo rằng chỉ những người dùng được phép mới có thể truy cập vào các chức năng và dịch vụ cụ thể.
* Tính bảo mật và linh hoạt**:** Việc sử dụng JWT giúp hệ thống tránh được các vấn đề bảo mật như việc lưu trữ trạng thái phiên. Các thông tin về người dùng và quyền hạn được mã hóa trong JWT và xác thực bởi máy chủ, đảm bảo tính toàn vẹn và bảo mật của dữ liệu.
* Quản lý phiên làm việc và hiệu suất: JWT giúp giảm tải cho máy chủ bằng việc cho phép người dùng giữ phiên đăng nhập trong khoảng thời gian nhất định, đồng thời cải thiện trải nghiệm người dùng bởi tính tiện lợi trong quản lý phiên làm việc.

## 3. 3. HIỆN THỰC GIẢI PHÁP

#### 3.3.1. Mô tả chương trình

Hệ thống thương mại điện tử được xây dựng nhằm mục tiêu cung cấp giải pháp mua sắm trực tuyến hiện đại, an toàn và hiệu quả, khắc phục những hạn chế của các nền tảng thương mại điện tử truyền thống. Hệ thống hỗ trợ người dùng duyệt sản phẩm, quản lý giỏ hàng, đặt hàng và thanh toán, đồng thời tích hợp trí tuệ nhân tạo để cung cấp chatbot hỗ trợ khách hàng 24/7 và hệ thống gợi ý sản phẩm thông minh.

Hệ thống được phát triển theo kiến trúc microservices, với Frontend sử dụng React.js kết hợp TypeScript và Material-UI để xây dựng giao diện người dùng hiện đại và thân thiện. Backend được xây dựng dựa trên Spring Boot với 17 microservices độc lập, mỗi service chịu trách nhiệm cho một domain cụ thể. Hệ thống sử dụng MySQL để lưu trữ dữ liệu persistent với pattern "database per service" (mỗi service có database riêng), và Redis để caching dữ liệu thường xuyên truy cập như giỏ hàng, session, và cache sản phẩm.

Hệ thống tích hợp Spring Cloud Gateway làm API Gateway để quản lý routing, authentication, và load balancing. Eureka Service Discovery được sử dụng để tự động đăng ký và khám phá các microservices, cho phép hệ thống scale linh hoạt. Hệ thống sử dụng JWT (JSON Web Token) để xác thực và phân quyền, đồng thời hỗ trợ OAuth2 để đăng nhập bằng tài khoản Google và Facebook, giảm friction trong quá trình đăng ký/đăng nhập và tăng cường tính bảo mật.

Hệ thống tích hợp nhiều dịch vụ bên ngoài để cung cấp trải nghiệm mua sắm hoàn chỉnh: VNPay và Stripe cho thanh toán trực tuyến, GHN API cho tính toán phí vận chuyển và theo dõi đơn hàng, Gemini API cho chatbot trí tuệ nhân tạo, và Python FastAPI với Scikit-learn cho hệ thống gợi ý sản phẩm sử dụng machine learning.

Các chức năng chính của hệ thống:

1. Quản lý sản phẩm:

* Duyệt danh sách sản phẩm với filter và tìm kiếm.
* Xem chi tiết sản phẩm với hình ảnh, mô tả, đánh giá
* Quản lý sản phẩm (Admin): Tạo, cập nhật, xóa sản phẩm
* Quản lý danh mục (Category) và thương hiệu (Brand)
* Quản lý tồn kho (Inventory).

2. Quản lý giỏ hàng:

* Thêm/xóa sản phẩm vào giỏ hàng.
* Cập nhật số lượng sản phẩm trong giỏ hàng.
* Lưu trữ giỏ hàng trong Redis với TTL 24 giờ.
* Xem giỏ hàng và tính tổng tiền.

3. Quản lý đơn hàng:

* Tạo đơn hàng từ giỏ hàng.
* Tính toán phí vận chuyển tự động qua GHN API.
* Theo dõi trạng thái đơn hàng (Đang xử lý, Đã xác nhận, Đang giao hàng, Đã giao hàng, Đã hủy).
* Xem lịch sử đơn hàng.
* Quản lý đơn hàng (Admin): Xem danh sách, cập nhật trạng thái.

4. Thanh toán:

* Tích hợp VNPay cho thanh toán nội địa Việt Nam.
* Xử lý callback/webhook từ payment gateway.
* Cập nhật trạng thái đơn hàng sau khi thanh toán thành công.

5. Xác thực và phân quyền:

* Đăng ký tài khoản mới.
* Đăng nhập với email và mật khẩu.
* Đăng nhập bằng Google OAuth2.
* Đăng nhập bằng Facebook OAuth2.
* Quản lý thông tin cá nhân.
* Reset mật khẩu.
* Phân quyền theo vai trò (User, Admin, Employee).

6. Chatbot trí tuệ nhân tạo:

* Chatbot hỗ trợ khách hàng 24/7
* Trả lời câu hỏi về sản phẩm, chính sách, đơn hàng
* Gợi ý sản phẩm thông minh dựa trên câu hỏi
* Tìm kiếm sản phẩm trong database
* Lưu lịch sử chat
* Public API không cần đăng nhập

7. Hệ thống gợi ý sản phẩm (Machine Learning):

* Gợi ý sản phẩm cá nhân hóa dựa trên hành vi người dùng
* Sử dụng Collaborative Filtering (SVD) và Content-Based Filtering
* Phân tích dữ liệu reviews, ratings, và purchases.
* Gợi ý sản phẩm trên trang chủ và trang chi tiết sản phẩm.

8. Đánh giá và phản hồi:

* Đánh giá sản phẩm với rating (1-5 sao)
* Viết review chi tiết về sản phẩm.
* Xem đánh giá của người dùng khác.
* Phân tích cảm xúc đánh giá.

9. Quản lý khuyến mãi:

* Tạo và quản lý voucher (Admin).
* Áp dụng voucher khi thanh toán.
* Tính toán giảm giá tự động.
* Quản lý chương trình khuyến mãi.

10. Quản lý bảo hành:

* Tạo yêu cầu bảo hành.
* Theo dõi trạng thái xử lý bảo hành.
* Quản lý bảo hành (Admin).

11. Quản lý người dùng:

* Xem và cập nhật thông tin cá nhân.
* Xem lịch sử đơn hàng.
* Quản lý địa chỉ giao hàng.
* Quản lý danh sách yêu thích (Favorites/Wishlist).
* Quản lý người dùng (Admin): Xem danh sách, khóa/mở khóa tài khoản.

12. Thông báo:

* Gửi email thông báo đơn hàng.
* Gửi email xác nhận thanh toán.
* Gửi email cập nhật trạng thái đơn hàng.
* Thông báo khuyến mãi và sản phẩm mới.

13. Admin Dashboard:

* Quản lý sản phẩm, đơn hàng, người dùng.
* Xem báo cáo doanh thu và thống kê.
* Quản lý inventory và stock.
* Quản lý voucher và khuyến mãi.
* Phân tích dữ liệu và xuất báo cáo.

Frontend được viết bằng React.js với TypeScript để đảm bảo type safety và code quality. Hệ thống sử dụng Material-UI (MUI) để xây dựng giao diện người dùng hiện đại và responsive. Frontend được build bằng Vite để tối ưu tốc độ phát triển và build time.

Frontend giao tiếp với Backend thông qua REST API qua API Gateway (port 8080). Tất cả các requests đều được gửi đến API Gateway, sau đó Gateway sẽ route đến các microservices tương ứng. Frontend sử dụng Axios để thực hiện HTTP requests và quản lý authentication tokens (JWT).

Các trang chính của Frontend:

- Trang chủ: Hiển thị sản phẩm nổi bật, sản phẩm mới, và gợi ý sản phẩm

- Trang sản phẩm: Danh sách sản phẩm với filter và search

- Trang chi tiết sản phẩm: Thông tin chi tiết, đánh giá, và gợi ý sản phẩm tương tự

- Trang giỏ hàng: Quản lý giỏ hàng và checkout

- Trang đơn hàng: Xem lịch sử và theo dõi đơn hàng

- Trang đăng nhập/đăng ký: Xác thực người dùng

- Trang profile: Quản lý thông tin cá nhân

- Trang chatbot: Giao diện chat với AI chatbot

- Admin Dashboard: Giao diện quản trị cho admin

Backend:

Backend được viết bằng Java và sử dụng Spring Boot để xây dựng các microservices. Hệ thống bao gồm 17 microservices độc lập, mỗi service chịu trách nhiệm cho một domain cụ thể:

1. Auth Service (Port 8081): Xử lý authentication, JWT token generation/validation, và OAuth2 integration

2. User Service (Port 8082): Quản lý thông tin người dùng, profile, và địa chỉ

3. Product Service (Port 8083): Quản lý sản phẩm, danh mục, và thương hiệu

4. Cart Service (Port 8084): Quản lý giỏ hàng, sử dụng Redis làm primary storage

5. Order Service (Port 8085): Xử lý đơn hàng và tích hợp GHN API cho shipping

6. Payment Service (Port 8086): Xử lý thanh toán với VNPay

7. Review Service (Port 8087): Quản lý đánh giá và reviews sản phẩm

8. AI Service (Port 8088): Chatbot trí tuệ nhân tạo tích hợp Gemini API

9. Category Service (Port 8089): Quản lý danh mục sản phẩm

10. Brand Service (Port 8090): Quản lý thương hiệu

11. Inventory Service (Port 8091): Quản lý tồn kho và stock

12. Voucher Service (Port 8092): Quản lý voucher và khuyến mãi

13. Warranty Service (Port 8093): Quản lý bảo hành

14. Notification Service (Port 8094): Gửi email và SMS thông báo

15. Favorites Service (Port 8095): Quản lý danh sách yêu thích

16. Recommendation Service (Port 8096): Gợi ý sản phẩm, gọi ML Service

17. Admin Service (Port 8097): Dashboard và báo cáo cho admin

Ngoài ra, hệ thống còn có 3 infrastructure services:

- Discovery Server (Port 8761): Eureka Service Registry

- Config Server (Port 8888): Centralized configuration

- API Gateway (Port 8080): Spring Cloud Gateway

Mỗi microservice sử dụng MySQL để lưu trữ dữ liệu persistent với database riêng (database per service pattern), đảm bảo tính độc lập và có thể scale riêng biệt. Các services sử dụng Redis để cache dữ liệu thường xuyên truy cập và lưu trữ session data.

Backend tương tác với các dịch vụ bên ngoài thông qua REST API:

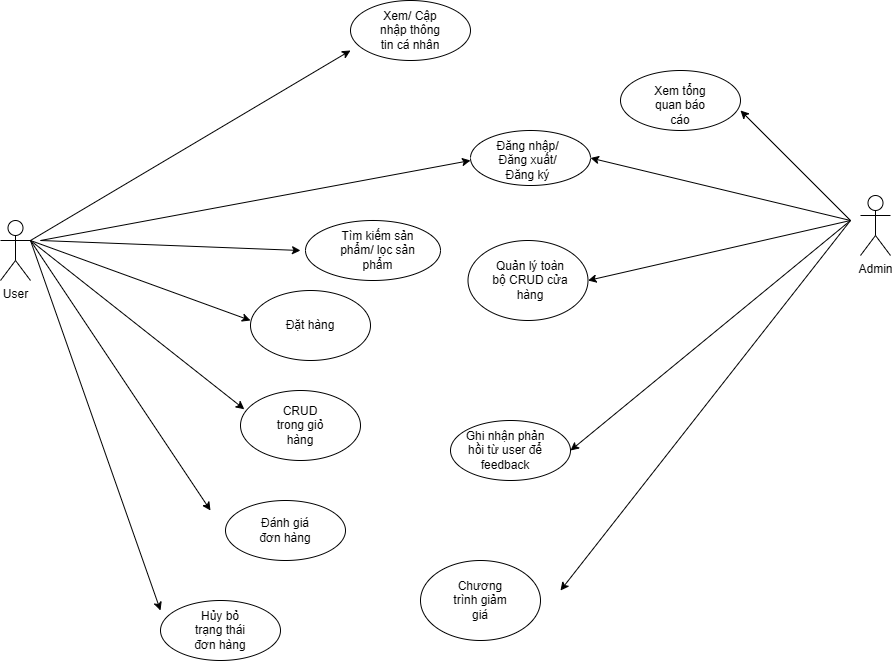
- VNPay API: Tạo payment URL và xử lý callback

- GHN API: Tính toán phí vận chuyển và tạo đơn hàng vận chuyển

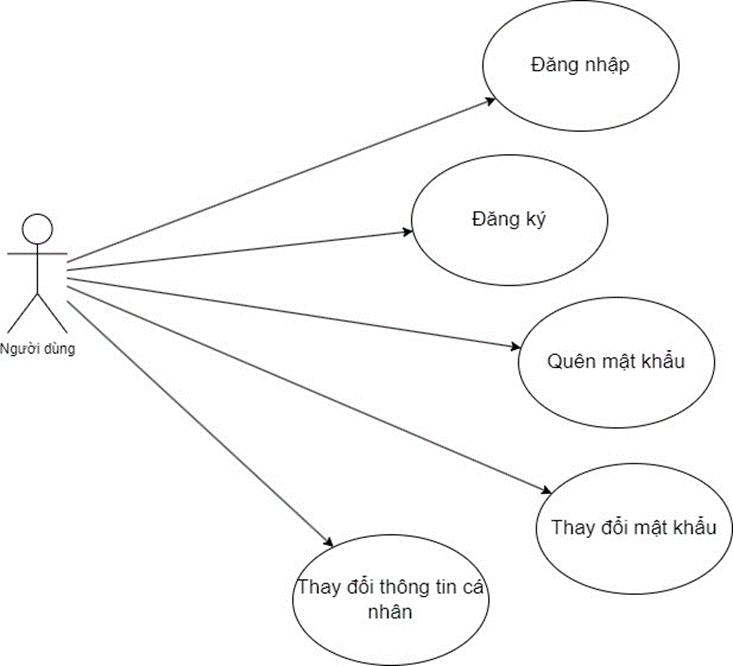
- Gemini API: Xử lý chat requests và generate responses

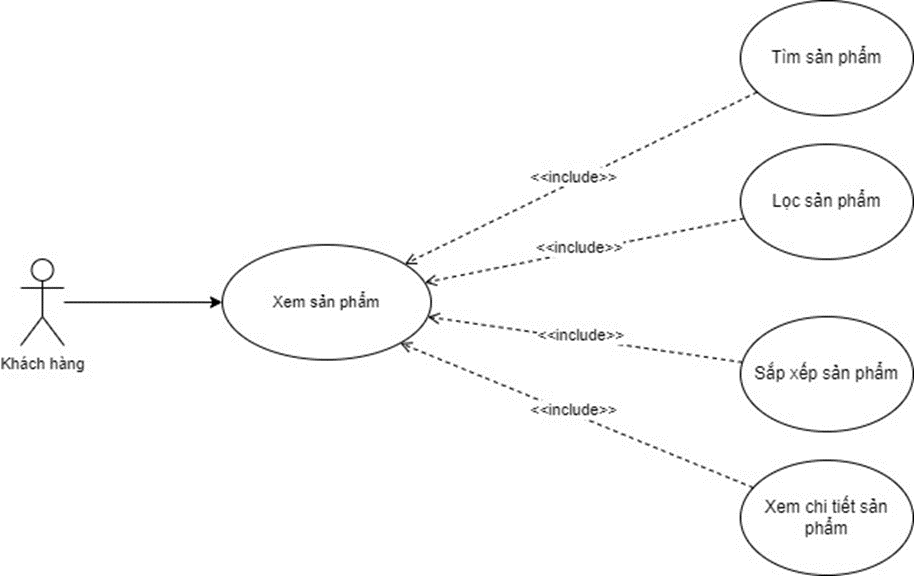
- ML Service (Python FastAPI, Port 8000): Gọi API để lấy gợi ý sản phẩm

Tất cả các services đều expose REST API endpoints và được quản lý bởi API Gateway. API Gateway xử lý authentication (JWT validation), authorization (role-based), routing, load balancing, và CORS. Các services tự động đăng ký với Eureka Service Discovery khi khởi động, cho phép API Gateway và các services khác tìm và giao tiếp với nhau một cách tự động.

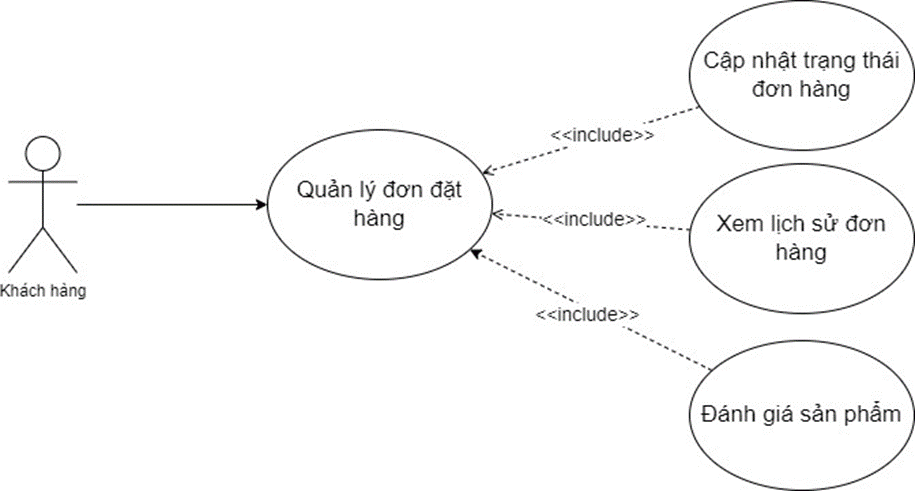


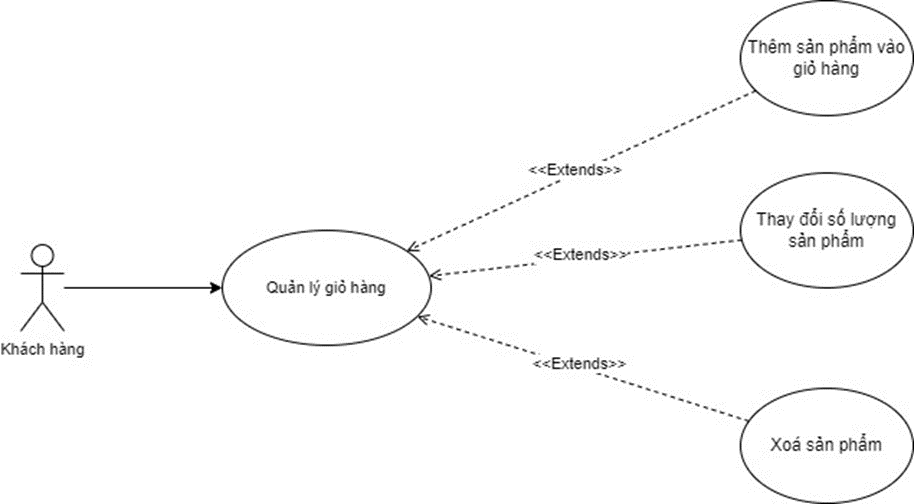
*Hình 6. Sơ đồ usecase của hệ thống.*

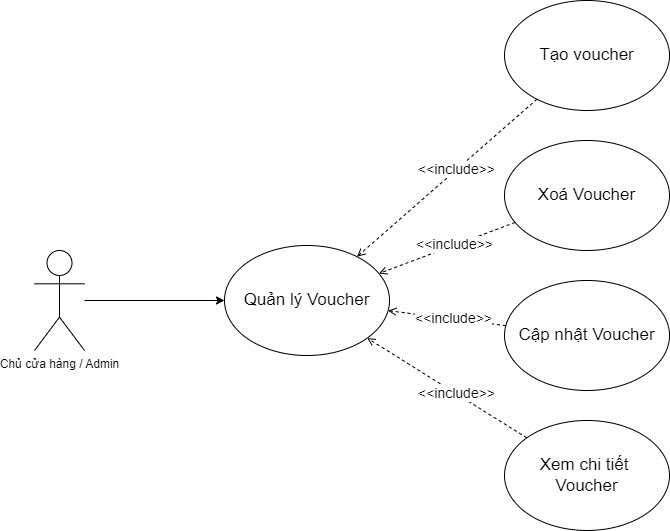
*Hình 3.15 Lược đồ use case tổng quát của hệ thống*

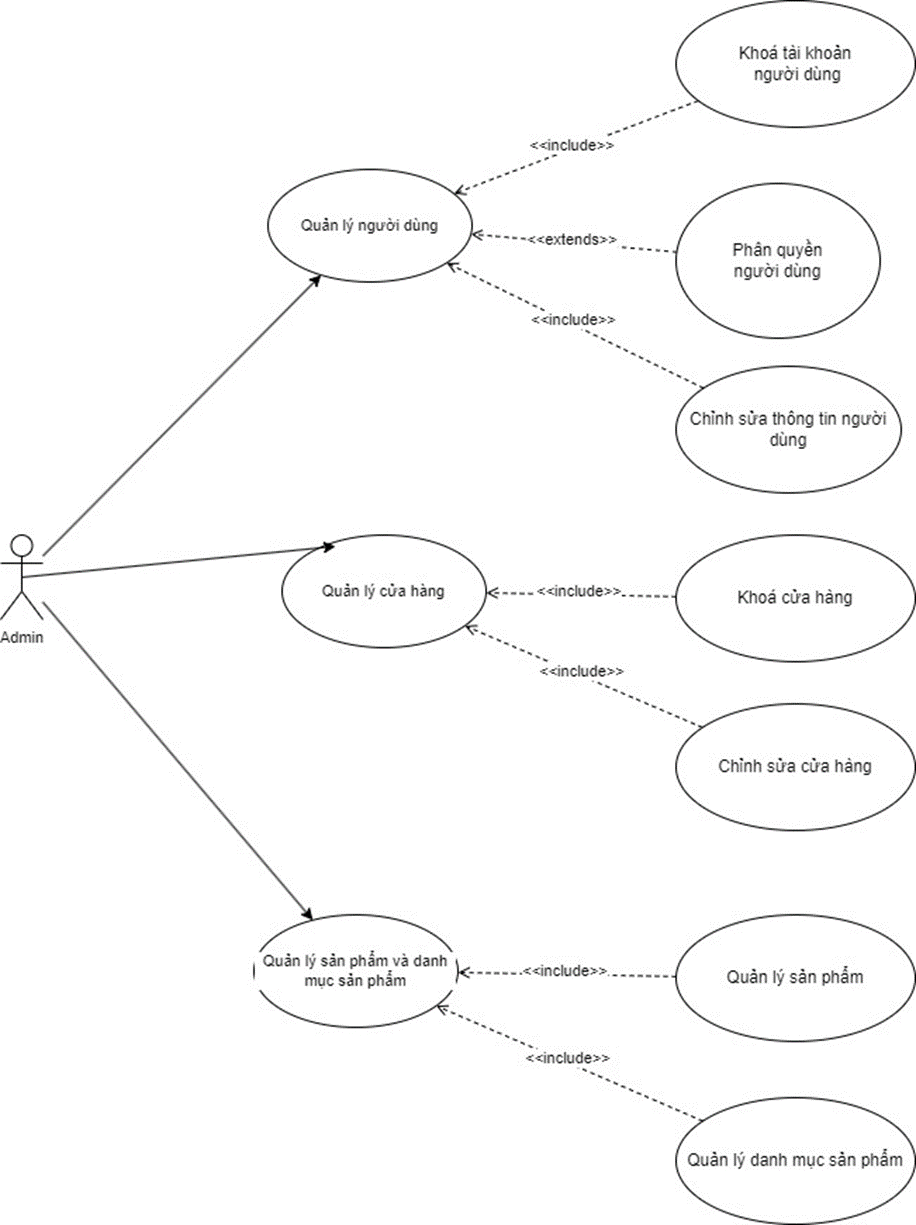
**

*Hình 3.16 Lược đồ use case Xem sản phẩm*

*Hình 3.17 Lược đồ use case quản lý đơn đặt hàng của khách hàng*

*Hình 3.18 Lược đồ use case quản lý giỏ hàng*

*Hình 3.21 Lược đồ use case quản lý khuyến mãi của cửa hàng*

*Hình 3.24 Lược đồ use case quản lý hệ thống sàn thương mại điện tử*

**Bảng 1. Đặc tả usecase đăng ký.**

|  |  |
| --- | --- |
| Tên usecase | Đăng kí |
| Tác nhân | Người dùng |
| Mô tả | Cho phép người dùng tạo một tài khoản với hệ thống. |
| Tiền điều kiện | Người dùng có email hợp lệ. |
| Hậu điều kiện | Thông tin của người dùng được lưu vào cơ sở dữ liệu.  Mật khẩu của người dùng được mã hóa trong cơ sở dữ liệu.  Hệ thống tự động đăng nhập khi đăng kí thành công. |
| Luồng sự kiện chính | 1. Người dùng truy cập vào trang đăng nhập. 2. Người dùng nhập họ tên, email, password. 3. Hệ thống kiểm tra thông tin đăng ký. 4. Nếu hợp lệ. Hệ thống thông báo đăng kí thành công điều hướng sang trang chủ. |
| Luồng sự kiện thay thế | 4a. Nếu không hợp lệ. Hệ thống thông báo và yêu cầu nhập lại |

**Bảng 2. Đặc tả usecase đăng nhập.**

|  |  |
| --- | --- |
| Tên usecase | Đăng nhập |
| Tác nhân | Người dùng, admin |
| Mô tả | Cho phép người dùng đăng nhập vào hệ thống. |
| Tiền điều kiện | Người dùng đã đăng kí tài khoản trước đó.  Tài khoản không bị chặn. |
| Luồng sự kiện chính | 1. Người dùng truy cập vào trang đăng nhập của hệ thống. 2. Người dùng nhập thông tin đăng nhập gồm có email và password. 3. Hệ thống kiểm tra thông tin đăng nhập. 4. Nếu hợp lệ. Hệ thống điều hướng sang trang chủ. |
| Luồng sự kiện thay thế | 4a. Nếu không hợp lệ. Hệ thống thông báo lỗi và yêu cầu nhập lại thông tin đăng nhập. |

**Bảng 3. Đặc tả usecase cập nhật thông tin cá nhân**

|  |  |
| --- | --- |
| Tên usecase | Cập nhật thông tin cá nhân |
| Tác nhân | Người dùng |
| Mô tả | Cho phép người dùng xem và cập nhật thông như họ và tên email, ảnh đại diện. |
| Tiền điều kiện | Người dùng đã đăng nhập.  Thông tin của người dùng được cập nhật trong cơ sở dữ liệu. |
| Luồng sự kiện chính | 1. Người dùng nhấn nút có biểu tượng “account” ở Header. 2. Hệ thống hiển thị ngắn kéo các hiển thị thông tin cá nhân hiện tại của người dùng. 3. Người dùng nhấn biểu tượng “edit” để cập nhật ảnh đại diện. 4. Hệ thống hiển thị các file trong thiết bị 5. Người dùng chọn file và nhấn OK. 6. Hệ thống hiển thị hình ảnh. 7. Người dùng nhập email và họ và tên và nhấn “Cập nhật”. 8. Hệ thống kiểm tra thông tin cập nhật. 9. Nếu người dùng có thay đổi email. Hệ thống yêu cầu nhập mật khẩu. 10. Nếu đúng mật khẩu, hệ thống thông báo cập nhật thành công. |
|  | 9. Nếu hợp lệ. Hệ thống hiển thị thông báo câp nhật thành công và gửi mail cho đến tài khoản email cũ của tài khoản. |
| Luồng sự kiện thay thế | 9a. Nếu hệ người không thay đổi thông tin nào. Hệ thống thông báo “Vui lòng thay đổi thông tin”.  9b. Nếu người dùng không thay đổi email. Hệ thống thông báo cập nhật thông tin thành công.  10a. Nếu người dùng nhập sai mật khẩu. Hệ thống thông báo sai mật khẩu. |

**Bảng 4. Đặc tả usecase đổi mật khẩu**

|  |  |
| --- | --- |
| Tên usecase | Đổi mật khẩu |
| Tác nhân | Người dùng |
| Mô tả | Cho phép người xem và cập nhật thông như họ và tên email, ảnh đại diện. |
| Tiền điều kiện | Người dùng đã đăng nhập.  Người dùng đã mở “ngăn kéo” cập nhập thông tin cá nhân. |
| Luồng sự kiện chính | 1. Người dùng nhấn “Đổi mật khẩu”. 2. Hệ thống mở “ngăn kéo” đổi mật khẩu. 3. Người dùng nhập mật khẩu cũ và mật khẩu mới và nhấn “Thay đổi”. 4. Hệ thống gửi yêu cầu kiểm tra mật khẩu. 5. Nếu mật khẩu đúng. Hệ thống thông báo thay đổi mật khẩu thành công. |
| Luồng sự kiện thay thế | 5a. Nếu mật khẩu sai, hệ thống thống thông báo mật khẩu cũ không chính xác.  5b. Nếu người dùng chưa nhập vào mật khẩu cũ hoặc mật khẩu mới, hệ thống thông báo “Mật khẩu cũ và mới không được để trống”. |

**Bảng 5. Đặc tả usecase thay đổi thông tin cuộc bình chọn**

|  |  |
| --- | --- |
| Tên usecase | Thay đổi thông tin cuộc bình chọn |
| Tác nhân | Người dùng |
| Mô tả | Cho phép chủ sở hữu thay đổi thông tin của cuộc bình như tiêu đề, ảnh và mô tả. |
| Tiền điều kiện | Người dùng đã đăng nhập.  Người dùng là chủ sở hữu của cuộc bình chọn.  Người dùng truy cập vào trang chi tiết cuộc bình chọn. |
| Luồng sự kiện chính | 1. Người dùng nhấn “Cập nhật thông tin”. 2. Hệ thống hiển thị hộp thoại có các input nhập ảnh, tiêu đề và mô tả, ngày bắt đầu, ngày kết thúc. 3. Người dùng nhập các thông tin và nhấn “Cập nhật”. (không cập nhật hình ảnh) 4. Hệ thống gửi yêu cầu cập nhật thông tin. 5. Hệ thống cập nhật thông tin xuống cơ sở dữ liệu và thông báo “Cập nhật thông tin thành công” |
| Luồng sự kiện thay thế | 3a. Nếu người dùng chọn nhập liên kết ảnh.  3a1. Hệ thống hiển thị input nhập liên kết.  3a2. Người dùng nhập liên kết.  3a3. Hệ thống hiển thị hình ảnh.  3b. Nếu người dùng chọn tải ảnh lên.  3b1. Hệ thống hiển thị các File của thiết bị.  3b2. Người dùng chọn File và nhấn OK.  3b3. Hệ thống tải ảnh lên và hiển thị ảnh. |

**Bảng 6. Đặc tả use case xem các cuộc bình chọn đã tạo.**

|  |  |
| --- | --- |
| Tên usecase | Xem các cuộc bình chọn đã tạo |
| Tác nhân | Người dùng |
| Mô tả | Cho phép xem lại danh sách các cuộc bình chọn mà mình đã tạo. |
| Tiền điều kiện | Người dùng đã đăng nhập. |
| Luồng sự kiện chính | 1. Người dùng nhấn biểu tượng “account’ ở header.  2.Hệ thống hiển thị ngăn kéo Thông tin cá nhân.   1. Người dùng nhấn “Cuộc bình chọn của bạn” 2. Hệ thống điều hướng tới trang “Cuộc bình chọn của bạn”. 3. Hệ thống gửi yêu cầu lấy các cuộc bình chọn của người dùng đang đăng nhập. 4. Hệ thống hiển thị danh sách các cuộc bình chọn mà người dùng đã tạo. 5. Người dùng nhấn vào cuộc bình chọn bất kỳ để đến trang xem chi tiết. |

**Bảng 7: Đặc tả usecase tạo cuộc bình chọn.**

|  |  |
| --- | --- |
| Tên usecase | Tạo cuộc bình chọn |
| Tác nhân | Người dùng |
| Mô tả | Cho phép người dùng tạo một cuộc bình chọn mới. |
| Tiền điều kiện | Người dùng đã đăng nhập  Đang ở trang tạo cuộc bình chọn |
| Hậu điều kiện | Thông tin cuộc bình chọn được lưu.  Hệ thống chuyển bước thêm ứng viên. |
| Luồng sự kiện chính | 1. Người dùng nhập tiêu đề, mô tả, ngày bắt đầu, ngày kết thúc. 2. Người dùng chọn cách thức nhập ảnh (tải ảnh lên hoặc nhập liên kết). 3. Hệ thống hiển thị hình ảnh. 4. Người dùng chọn hình thức cuộc bình chọn (công khai/riêng tư). 5. Người dùng nhấn hoàn tất. 6. Hệ thống gửi yêu cầu tạo cuộc bình chọn. 7. Hệ thống điều hưởng sang bước thêm ứng viên |
| Luồng sự kiện thay thế | 2a. Người dùng chọn nhập liên kết  2a1. Hệ thống hiển thị input nhập liên kết.  2a2. Người dùng nhập liên kết ảnh.  2b. Nếu chọn tải ảnh lên:  2b1. Hệ thống hiển thị giao diện chọn file.  2b2. Người dùng chọn file ảnh và nhấn OK.  2b3. Hệ thống tải ảnh lên firebase.  4a. Nếu chọn riêng tư:  4a1. Hệ thống hiển thị input nhập mật khẩu.  4a2. Người dùng nhập mật khẩu.  4b. Người dùng chọn công khai |

**Bảng 8: Đặc tả usecase thêm ứng viên**

|  |  |
| --- | --- |
| Tên usecase | Thêm ứng viên |
| Tác nhân | Người dùng |
| Mô tả | Cho phép người dùng thêm các ứng viên vào cuộc bình chọn do minh sở hữu, ngay sau bước tạo cuộc bình chọn hoặc sau đó. Có 2 hình thức thêm ứng viên: thêm thủ công từng ứng viên và nhập từ file excel để thêm nhiều ứng viên cùng lúc. |
| Tiền điều kiện | Người dùng đã đăng nhập.  Cuộc bình chọn thuộc sở hữu của người dùng hiện tại.  Đang ở trang thêm ứng viên. |
| Luồng sự kiện chính | 1. Người dùng truy cập vào trang thêm ứng viên. 2. Thêm ứng viên. 3. 1.Người dùng nhấn vào nút “Thêm” (thủ công):   2.1.1. Hệ thống hiển thị hộp thoại có Form thông tin.   * 1. 2.Người dùng nhập thông tin ứng viên.   2.1.3. Người dùng nhấn “Thêm”.   * 1. Người dùng nhấn “Nhập từ File”   2.2.1. Hệ thống hiển thị giao diện chọn file. |
|  | 2.2.2. Người dùng chọn file ảnh và nhấn OK.  2.2.3. Hệ thống xử lý File.   1. Hệ thống gửi yêu cầu thêm ứng viên. 2. Hệ thống cập nhật danh sách ứng viên. |
| Luồng sự kiện thay thế | 2.1.2a Nếu người chọn nhập liên kết ảnh  2.1.2a1. Hệ thống hiển thị input nhập liên kết.  2.1.2a2. Người dùng nhập liên kết ảnh.  2.1.2b. Nếu người dùng chọn tải ảnh lên.  2.1.2b1. Hệ thống hiển thị các file của thiết bị.  2.1.2b2. Người dùng chọn file và nhấn OK.  2.1.2b3. Hệ thống tải file ảnh lên firebase và nhận về liên kết |

**Bảng 9: Đặc tả usecase bình chọn.**

|  |  |
| --- | --- |
| Tên usecase | Bình chọn |
| Tác nhân | Người dùng |
| Mô tả | Cho phép người dùng bình chọn cho ứng viên mà bình yêu thích trong một cuộc bình chọn |
| Tiền điều kiện | Người dùng đã đăng nhập.  Đang đã truy cập thành công vào trang chi tiết cuộc bình chọn. |
| Luồng sự kiện chính | 1. Người dùng chọn ứng viên và nhấn “Bình chọn” 2. Hệ thống gọi API thêm bình chọn. 3. Hệ thống gửi yêu cầu thêm bình chọn đến backend. 4. Hệ thống gửi yêu cầu bình chọn đến SmartContract. 5. SmartContract kiểm tra bình chọn hợp lệ hay không. 6. Nếu hợp lệ, hệ thống thông báo bình chọn thành công. |
| Luồng sự kiện thay thế | 6a. Nếu không hợp lệ thì thông báo không thành công. |

**Bảng 10: Đặc tả usecase bình chọn bằng QR Code**

|  |  |
| --- | --- |
| Tên usecase | Bình chọn bằng QR Code |
| Tác nhân | Người dùng |
| Mô tả | Cho phép người bình chọn nhanh một ứng viên bằng QR code được tạo ra trước đó từ trang chi tiết cuộc bình chọn. |
| Tiền điều kiện | Người dùng đã đăng nhập.  QR Code phải hợp lệ. |
| Luồng sự kiện chính | 1. Người dùng quét QR Code 2. Smartphone nhận diện liên kết trong QR Code. 3. Người dùng nhấn mở liên kết 4. SmartPhone truy cập vào trang Bình chọn 5. Hệ thống gửi yêu cầu lấy thông tin ứng viên 6. Hệ thống truy vấn thông tin ứng viên từ MongoDB 7. Hệ thống hiển thị thông tin ứng viên 7. Người dùng nhấn bình chọn. 8. Hệ thống gọi API bình chọn. 9. Hệ thống gửi yêu cầu thêm bình chọn 10. Hệ thống tạo một yêu cầu bình chọn đến SmartContract 11. SmartContract kiểm tra thông tin bình chọn 12. Nếu hợp lệ, Hệ thống ghi nhận bình chọn 13. Hệ thống thông báo bình chọn thành công |
| Luồng sự kiện thay thế | 13a. Hệ thống thông báo không hợp lệ. |

**Bảng 11: Đặc tả usecase tham gia cuộc bình chọn.**

|  |  |
| --- | --- |
| Tên usecase | Tham gia cuộc bình chọn |
| Tác nhân | Người dùng |
| Mô tả | Cho phép người tham gia một cuộc bình chọn khi biết được ID và mật khẩu của cuộc bình chọn. |
| Tiền điều kiện | Người dùng đã đăng nhập.  Cuộc bình chọn tồn tại. |
| Hậu điều kiện | Thông tin người dùng tham gia vào cuộc bình chọn được cập nhật ở MongoDB. |
| Luồng sự kiện chính | 1. Người dùng nhập ID cuộc bình chọn vào thành tìm kiếm 2. Hệ thống gửi yêu cầu lấy thông tin cuộc bình chọn 3. Hệ thống hiển thị hộp thoại thông tin cuộc bình chọn   4a. Nếu như là riêng tư, người dùng nhập mật khẩu và nhấn “Tham gia”  5. Hệ thống gửi yêu cầu kiểm tra mật khẩu  6a. Nếu mật khẩu đúng, hệ thống điều hướng sang trang chi tiết cuộc bình chọn |
| Luồng sự kiện thay thế | 4b. Nếu công khai. Người dùng nhấn “Tham gia”  6b. Nếu mật khẩu sai. Hệ thống thông báo và yêu cầu nhập lại. |

**Bảng 12: Đặc tả usecase xem chi tiết cuộc bình chọn.**

|  |  |
| --- | --- |
| Tên usecase | Xem chi tiết cuộc bình chọn. |
| Tác nhân | Người dùng |
| Mô tả | Cho phép người dùng truy cập vào thông tin chi tiết của một cuộc bình chọn, bao gồm tiêu đề, mô tả, ngày bắt đầu, ngày kết thúc, hình ảnh, danh sách ứng viên và kết quả bỏ phiếu. Hệ thống sẽ kiểm tra quyền truy cập dựa trên loại hình cuộc bình chọn (công khai/riêng tư) và trạng thái của cuộc bình chọn (đang diễn ra/đã kết thúc). |
| Tiền điều kiện | Người dùng đã đăng nhập  Người dùng đã tham gia cuộc bình chọn nếu là cuộc bình chọn riêng tư. |
| Luồng sự kiện chính | 1. Người dùng truy cập vào trang chi tiết cuộc bình chọn. 2. Hệ thống gửi yêu cầu lấy thông tin cuộc bình chọn (bao gồm ID của cuộc bình chọn) và danh sách ứng viên đến Backend 3. Backend truy xuất thông tin cuộc bình chọn và các ứng viên từ MongoDB. 4. Hệ thống gửi yêu cầu lấy các bình chọn từ blockchain đến Backend. |
|  | 1. Backend gửi yêu cầu lấy các bình chọn từ Smart Contract trên blockchain. 2. Hệ thống hiển thị các xử lý dữ liệu từ mongoDB và blockchain. 3. Hệ thống hiển thị thông tin và số phiếu của các ứng viên. |
| Luồng sự kiện thay thế | 3.1. Nếu là riêng tư và chưa tham gia  3.1.1. Hệ thống yêu cầu nhập mật khẩu  3.1.2. Người dùng nhập mật khẩu  3.1.3. Hệ thống kiểm tra mật khẩu  3.1.4. Nếu mật khẩu đúng, thực hiện bước 4.  3.2. Nếu cuộc bình chọn đã kết thúc.  3.2.1. Hệ thống thông báo đã kết thúc hiển thị kết quả cuộc bình chọn. |
| Trường hợp ngoại lệ | 3.1.4a. Nếu mật khẩu sai thì thông báo và yêu cầu nhập lại |

**Bảng 13: Đặc tả usecase cập nhật bình chọn.**

|  |  |
| --- | --- |
| Tên usecase | Cập nhật bình chọn |
| Tác nhân | Người dùng |
| Mô tả | Cho phép người dùng bình chọn ứng viên trong một cuộc bình chọn. |
| Tiền điều kiện | Người dùng đang ở trang chi tiết cuộc bình chọn.  Người dùng đã bình chọn rồi và muốn thay đổi bình chọn. |
| Luồng sự kiện chính | 1. Người dùng truy cập trang thông tin cuộc bình chọn. 2. Người dùng chọn một ứng viên muốn bình chọn lại. 3. Hệ thống gửi yêu cầu bình chọn lại đến Backend. 4. Backend xử lý yêu cầu và tương tác với Smart Contract trên Blockchain. 5. Smart Contract cập nhật lại bình chọn và ghi nhận thông tin cập nhật vào Blockchain 6. Frontend hiển thị thông báo cho người dùng. |

**Bảng 14: Đặc tả usecase truy xuất bình chọn bằng QR Code**

|  |  |
| --- | --- |
| Tên usecase | Truy xuất bình chọn bằng QR Code |
| Tác nhân | Người dùng |
| Mô tả | Cho phép người dùng xem lại đã bình chọn cho ứng viên nào và và đã cập nhật thành ứng viên nào, xem số lần bình chọn và cập nhật bình chọn của mình ở cuộc bình chọn hiện tại. |
| Tiền điều kiện | Người dùng đang ở trang chi tiết cuộc bình chọn.  Người dùng có điện thoại thông minh có máy ảnh. |
| Luồng sự kiện chính | 1. Người dùng truy cập trang thông tin cuộc bình chọn. 2. Người dùng quét QRcode bằng điện thoại. 3. Hệ điều hành của điện thoại sẽ nhận diện mã và hiển thị các lịch sử cập nhật bình chọn của bạn. |
| Luồng sự kiện thay thế | 3a. Nếu người dùng chưa bình chọn cho cuộc bình chọn hiện thì nội dung của QR Code là: “Bạn chưa bình chọn” |

**Bảng 15: Đặc tả usecase tìm kiếm cuộc bình chọn.**

|  |  |
| --- | --- |
| Tên usecase | Tìm kiếm cuộc bình chọn |
| Tác nhân | Người dùng |
| Mô tả | Cho phép người dùng tìm kiếm cuộc bình chọn bằng tiêu đề của cuộc bình chọn với hiệu ứng thời gian thực. |
| Tiền điều kiện | Người dùng đã đăng nhập vào hệ thống.  Người dùng ở trang các cuộc bình chọn công khai, trang các cuộc bình chọn của bạn. |
| Luồng sự kiện chính | 1. Người dùng nhập tiêu đề của cuộc bình chọn vào thanh tìm kiếm. 2. Hệ thống lọc các cuộc bình chọn theo tiêu đề chứ nội dung mà người dùng nhập. 3. Hệ thống hiển thị các cuộc bình chọn có tiêu đề chưa nội dung người dùng nhập. |
| Luồng sự kiện thay thế | 3a. Nếu không có cuộc bình chọn trùng khớp, hệ thống hiển thị  “Cuộc bình chọn không tồn tại.” |

**Bảng 16: Đặc tả usecase xem các cuộc bình chọn công khai.**

|  |  |
| --- | --- |
| Tên usecase | Xem các cuộc bình chọn công khai. |
| Tác nhân | Người dùng |
| Mô tả | Cho phép người dùng xem tất cả các cuộc bình chọn công khai trên hệ thống. |
| Tiền điều kiện | Người dùng đã đăng nhập. |
| Luồng sự kiện chính | 1, Người dùng truy cập vào trang chủ của hệ thống.   1. Người dùng vào nút xem tất cả ở phần các cuộc bình công khai. 2. Hệ thống chuyển sang các cuộc bình chọn công khai. 3. Hệ thống gửi yêu cầu lấy các cuộc bình chọn công khai. 4. Hệ thống hiển thị tất cả các cuộc bình chọn công khai của hệ thống. |

**Bảng 17: Đặc tả usecase tạo QRCode cho ứng viên.**

|  |  |
| --- | --- |
| Tên usecase | Tạo QR Code cho ứng viên |
| Tác nhân | Người dùng |
| Mô tả | Cho phép người tạo một QRCode chứa đường dẫn đến trang bình chọn bằng QR Code. |
| Tiền điều kiện | Người dùng đã đăng nhập.  Cuộc bình chọn trong thời gian bình chọn. |
| Luồng sự kiện chính | 1. Người dùng truy cập vào trang chi tiết cuộc bình chọn. 2. Người nhấn chọn một ứng viên và nhấn “QR Code”. 3. Hệ thống lấy thông tin của ứng viên. 4. Hệ thống tạo một QR code từ thông tin của ứng viên. 5. Hệ thống hiển thị một hộp thoại chứ QR code của ứng viên mà người dùng chọn. |

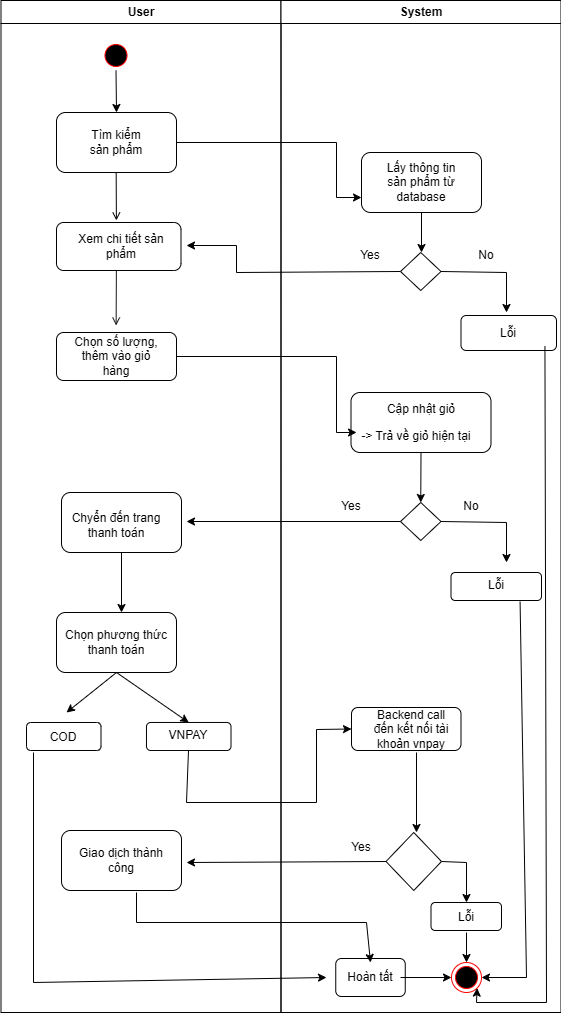
**Bảng 18: Đặc tả usecase quản lý các cuộc bình chọn**

|  |  |
| --- | --- |
| Tên usecase | Quản lý các cuộc bình chọn. |
| Tác nhân | Admin |
| Mô tả | Cho phép người quản trị hệ thống xem được các danh sách cuộc bình chọn mà người dùng đã tạo. |
| Tiền điều kiện | Đã đăng nhập bằng tài khoản admin |
| Luồng sự kiện chính | 1. Admin truy cập trang danh sách các cuộc bình chọn. 2. Hệ thống xác thực người dùng dùng. 3. Nếu là admin, hệ thống gửi yêu cầu lấy tất cả các cuộc bình chọn của hệ thống. 4. Hệ thống hiển thị danh sách tất các ứng viên của hệ thống. |
| Luồng sự kiện thay thế | 3a. Nếu người dùng không phải là admin, hệ thống điều hướng người dùng sang trang chủ. |

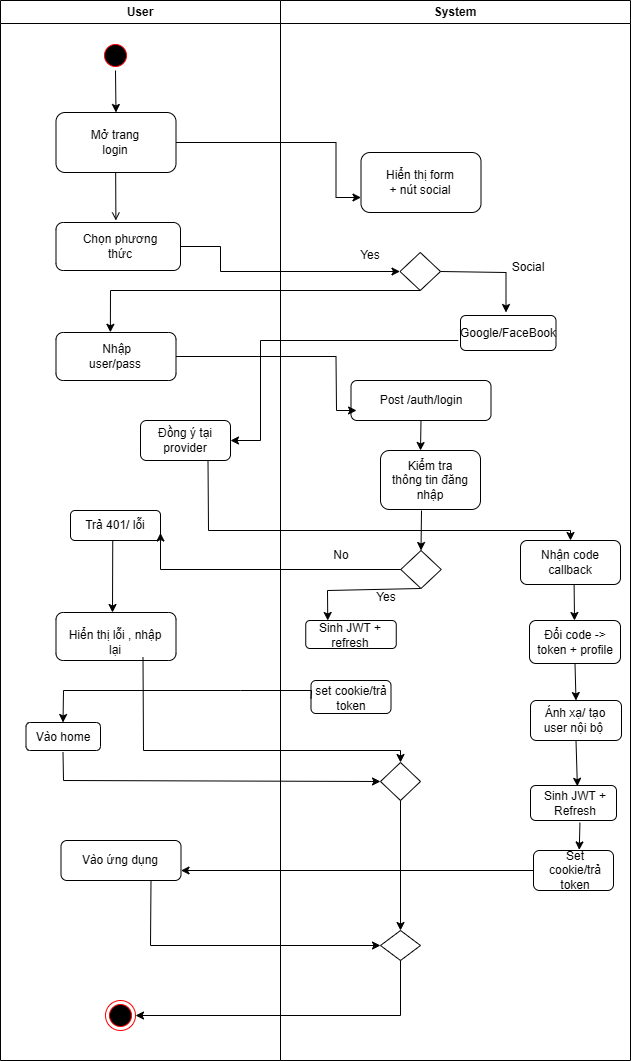
**Bảng 19: Đặc tả usecase chặn và mở chặn tài khoản.**

|  |  |
| --- | --- |
| Tên usecase | Chặn và mở chặn người |
| Tác nhân | Admin |
| Mô tả | Cho phép Admin chặn và mở tài khoản người dùng. |
| Tiền điều kiện | Đăng nhập bằng tài khoản admin. |
| Hậu điều kiện | Tài khoản được cập nhật lại trạng thái trong cơ sở dữ liệu. |
| Luồng sự kiện chính | 1, Admin truy cập vào trang quản lí người dùng.   1. Hệ thống xác thực danh tính người dùng. 2. Nếu là admin, hệ thống gửi yêu cầu lấy danh sách các người dùng hiện tại có trên hệ thống 3. Hệ thống hiển thị danh sách tất cả các tài khoản bào gồm ID, họ và tên, Email. 4. Người dùng chọn tài khoản và nhấn Chặn. 5. Hệ thống gửi yêu cầu chặn tài khoản. 6. Hệ thống thông báo chặn thành công. |
| Luồng sự kiện thay thế | 4a. Nếu người dùng chọn “Tài khoản bị chặn”  4a1. Hệ thống hiển thị các tài khoản đã bị chặn. 4a2. Người dùng chọn tài khoản và nhấn “Mở chặn” 4a3. Hệ thống thông báo mở chặn thành công. |

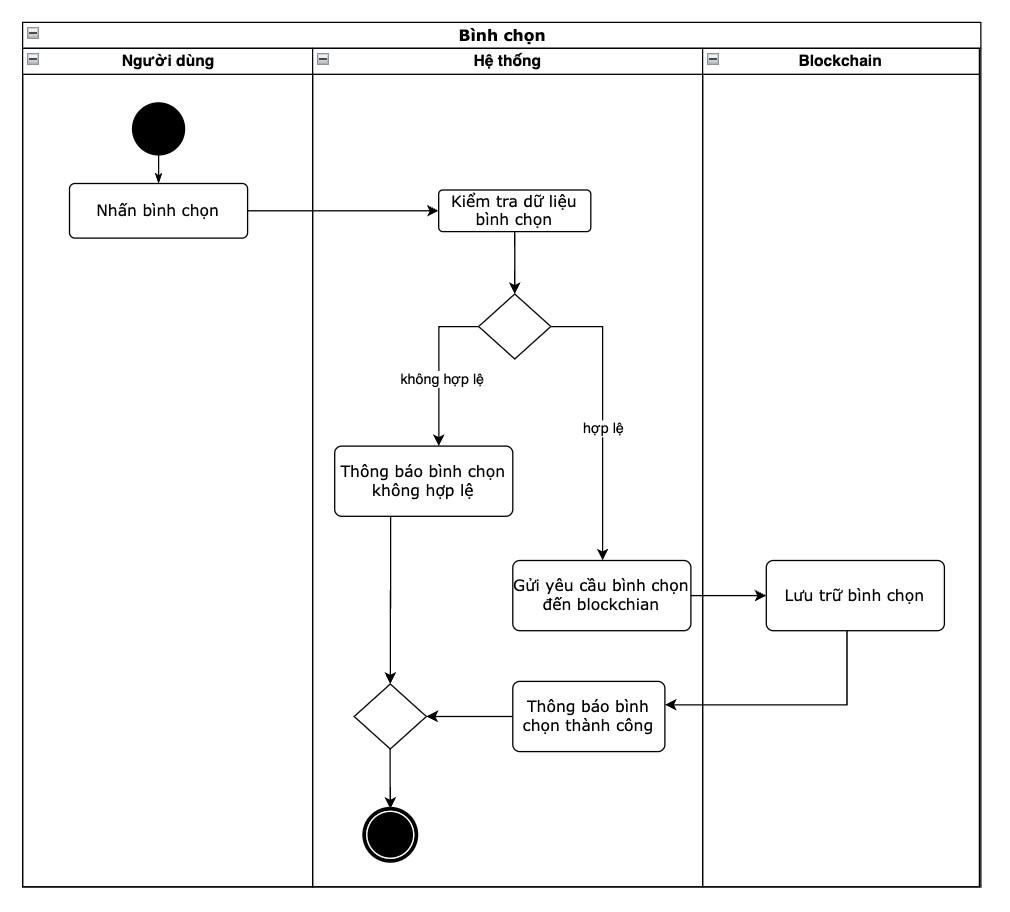
#### 3.3.2. Lược đồ activity.



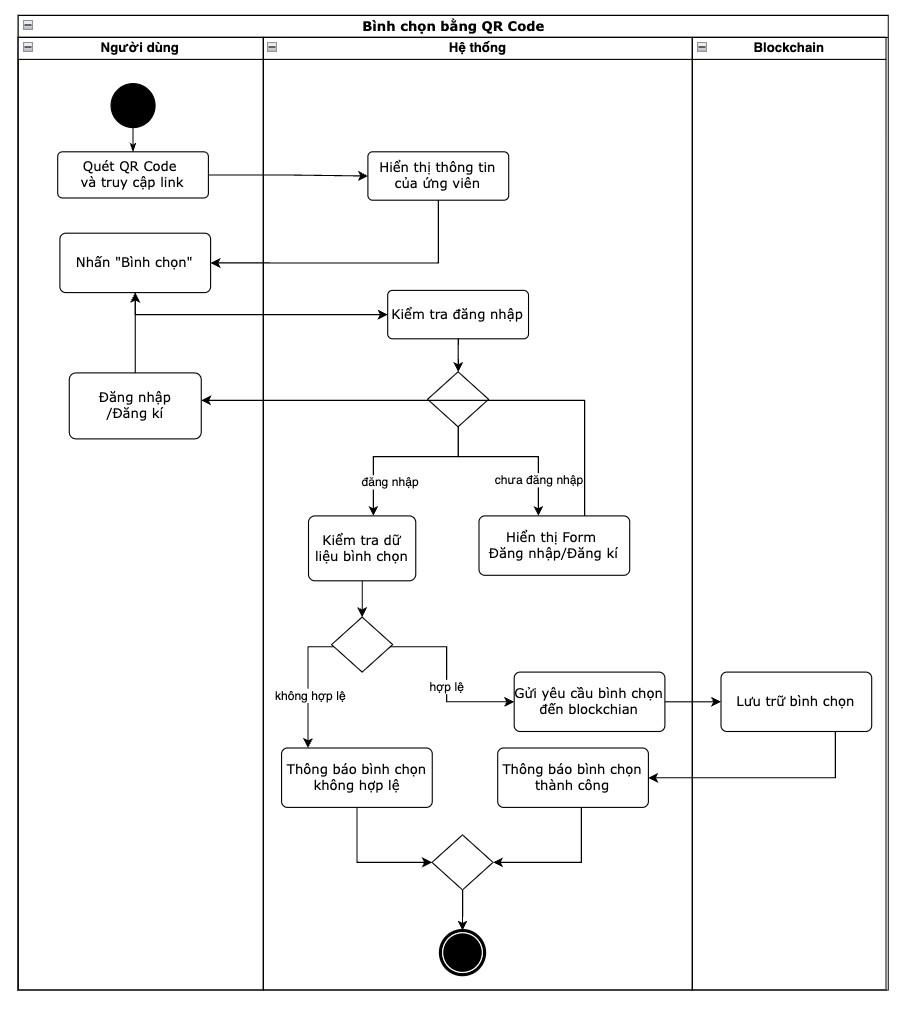
*Hình 7. Lược đồ activity cho usecase mua hàng*



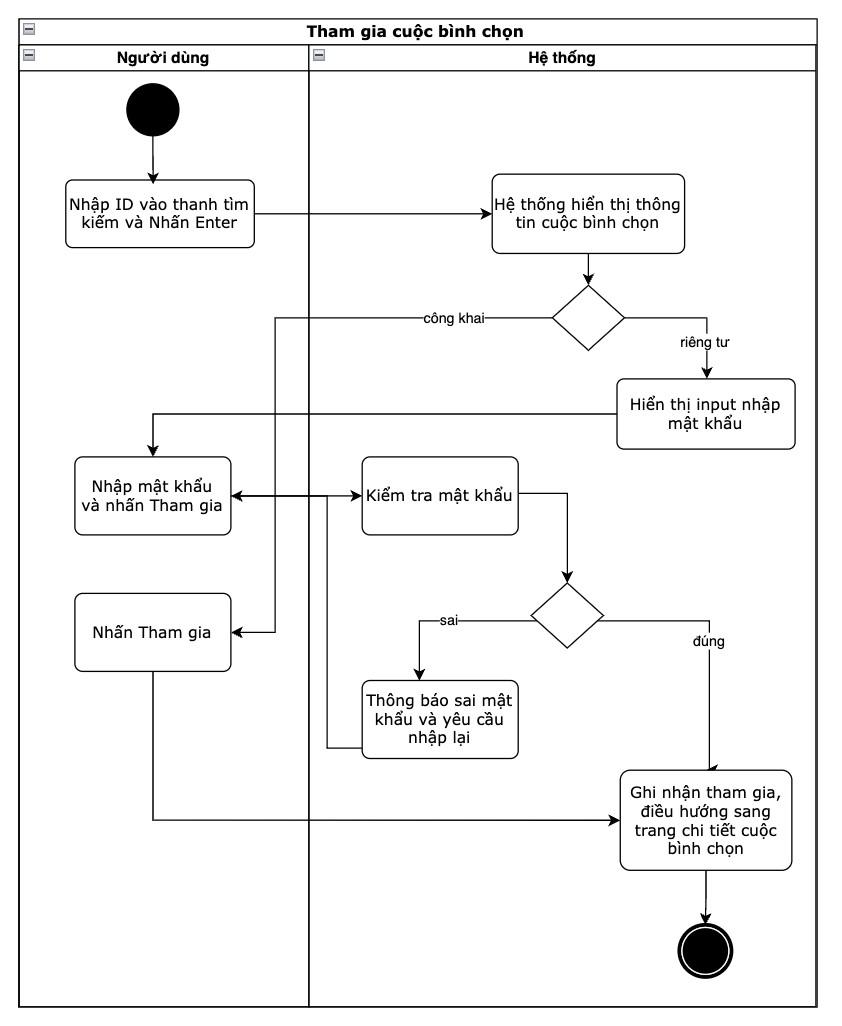
*Hình 8. Lược đồ activity cho usecase login*



*Hình 9. Lược đồ activity usecase bình chọn.*

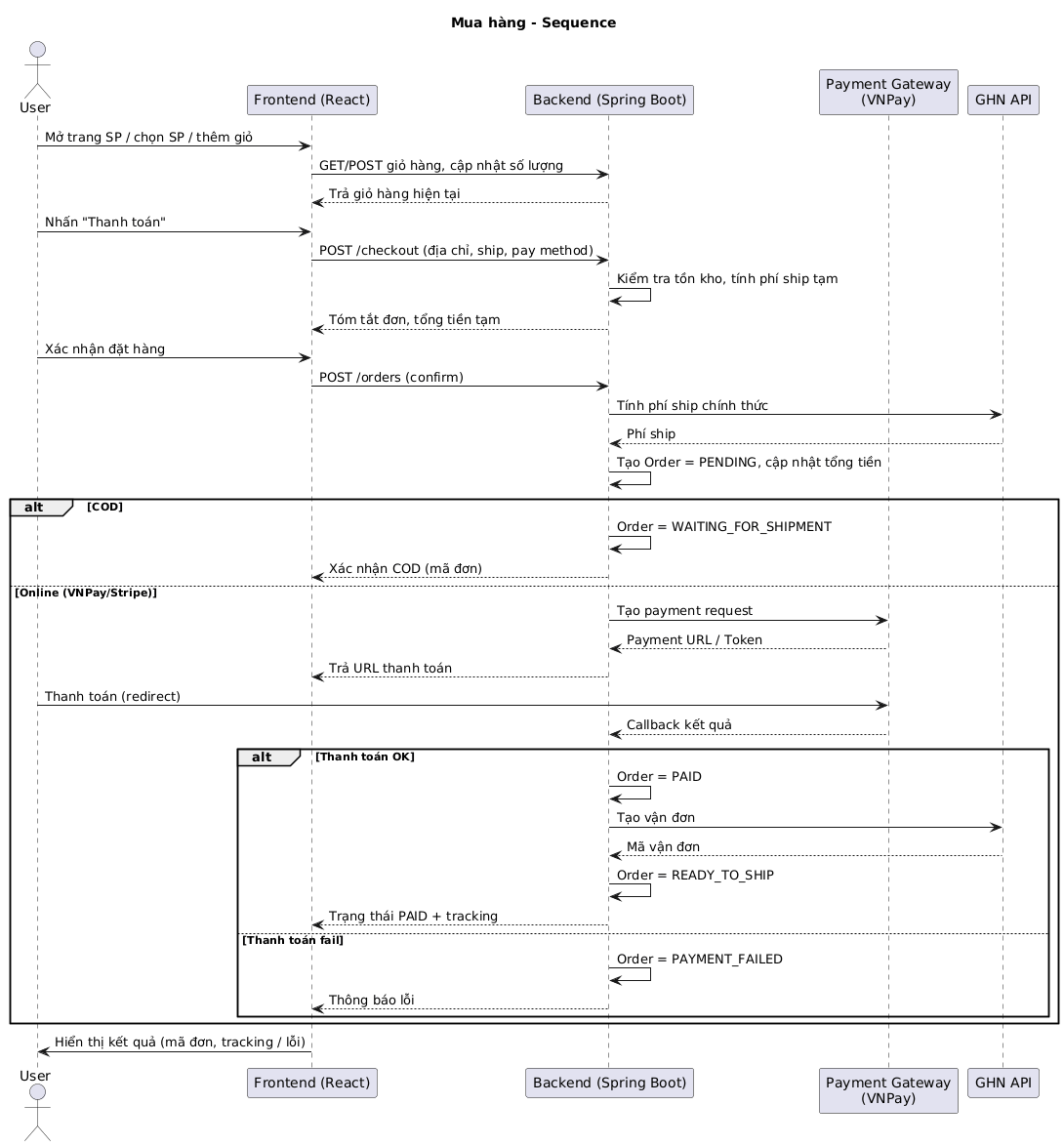


*Hình 10. Lược đồ activity cho usecase bình chọn bằng QR Code*

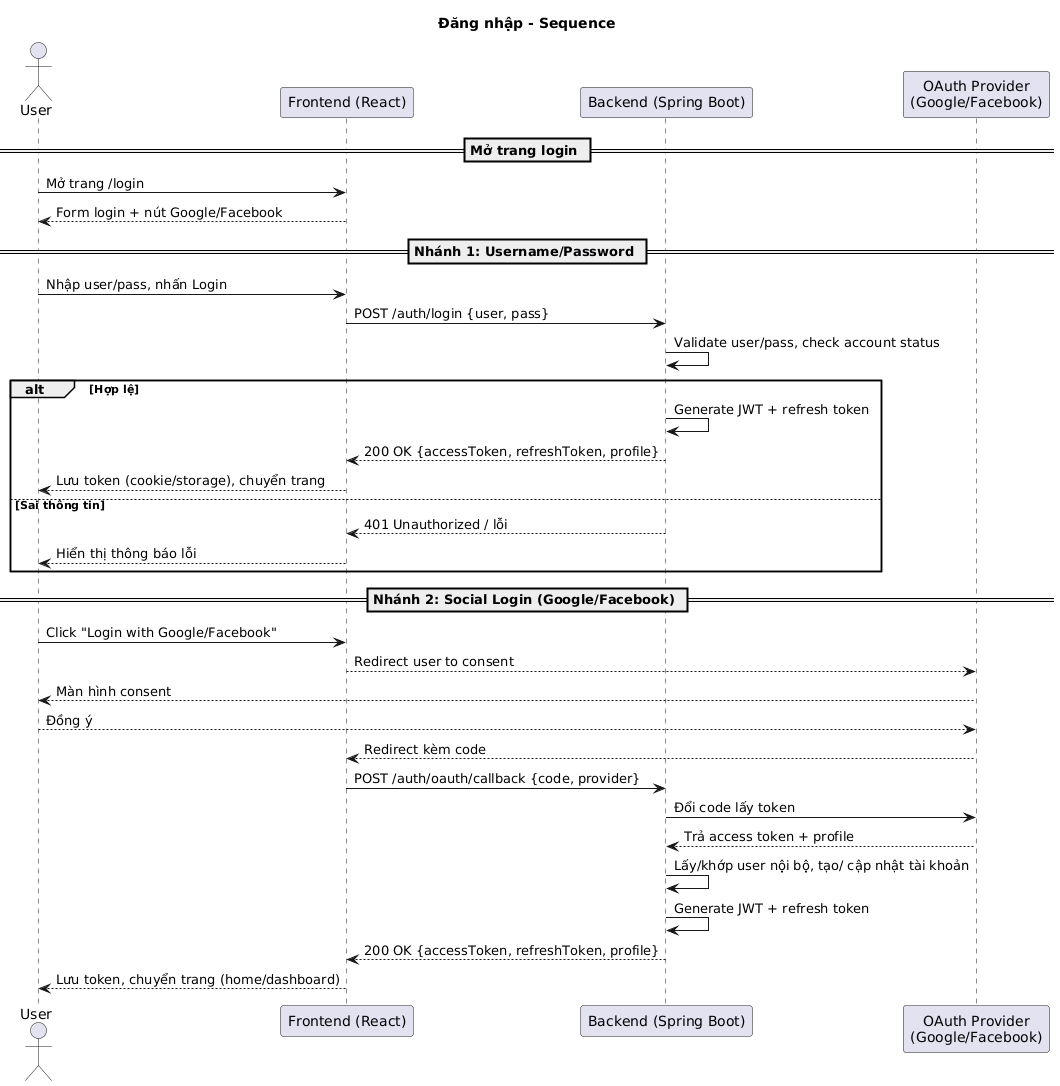


*Hình 11. Lược đồ activity cho usecase tham gia cuộc bình chọn.*

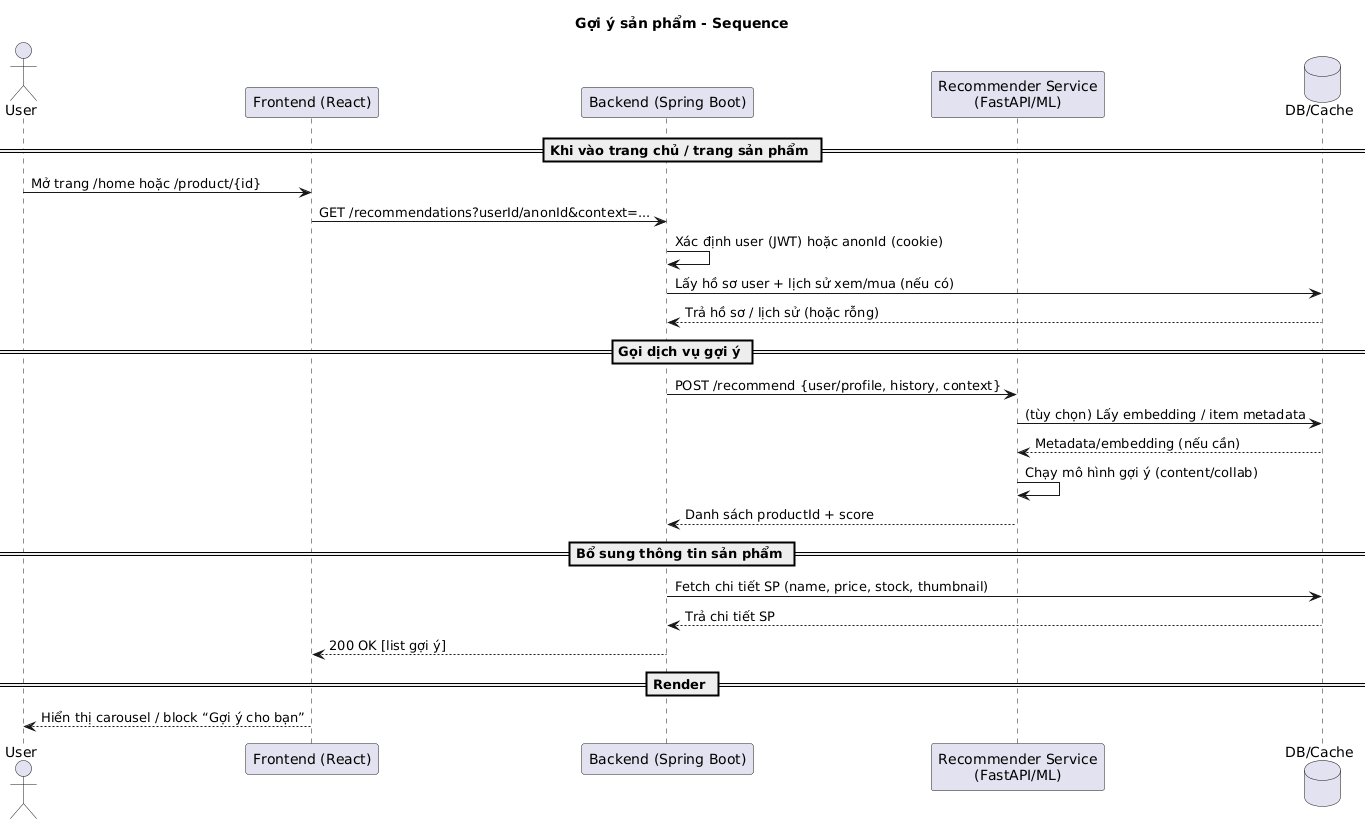
#### 3.3.3. Lược đồ tuần tự.



*Hình 12. Lược đồ tuần tự cho usecase mua hàng.*



*Hình 13. Lược đồ tuần tự cho usecase thêm ứng viên.*



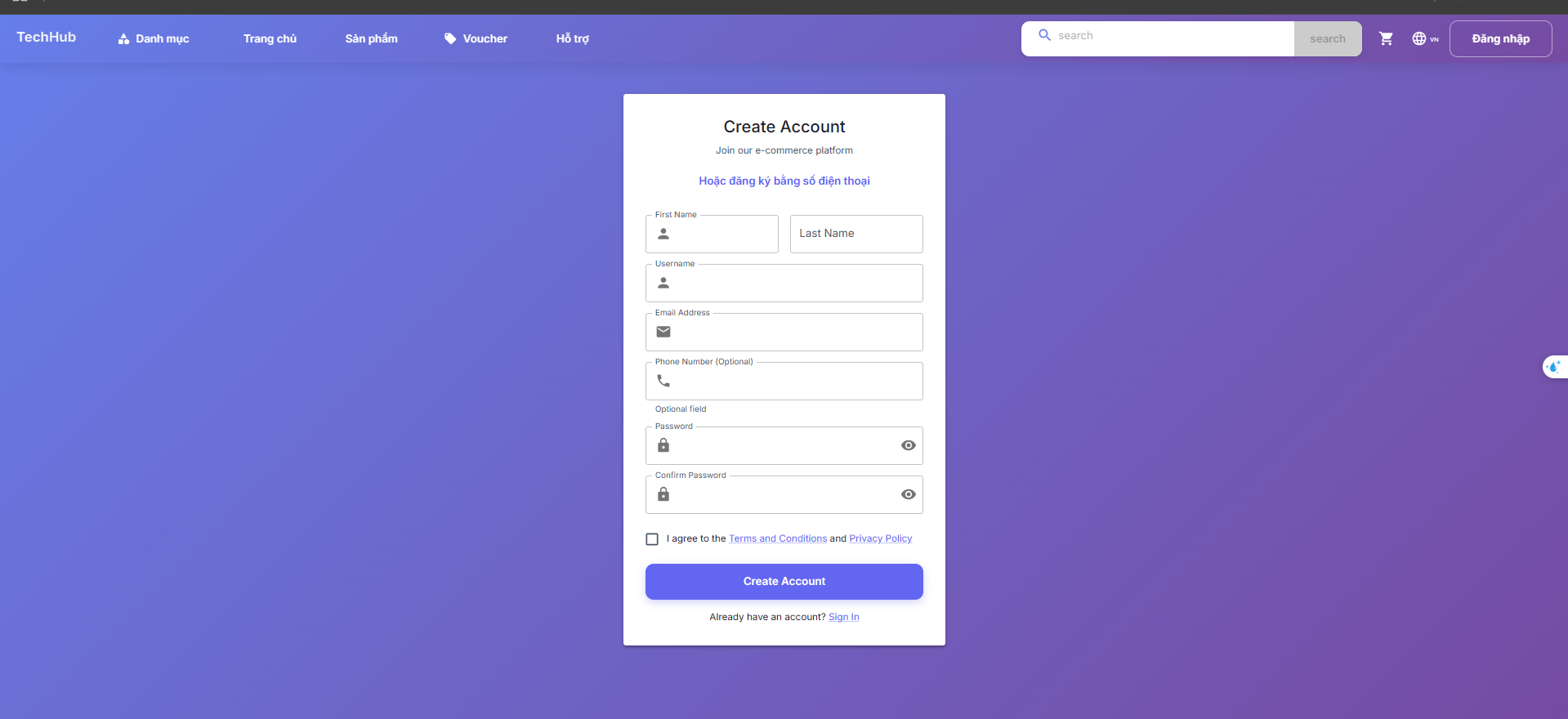
*Hình 14. Lược đồ tuần tự cho usecase bình chọn.*

#### 3.3.4. Các màn hình của hệ thống

Hệ thống được triển khai với Frontend React + TypeScript (MUI), Backend Spring Boot (Java 17) dùng MySQL làm cơ sở dữ liệu và Redis cho cache/session, tích hợp VNPay/Stripe cho thanh toán và GHN cho vận chuyển. Xác thực người dùng dựa trên JWT, hỗ trợ đăng nhập Google/Facebook. Tiểu luận đã hoàn thiện nghiên cứu và xây dựng luồng mua hàng trực tuyến: duyệt sản phẩm, giỏ hàng, đặt hàng, thanh toán (COD hoặc online), tạo vận đơn, cùng các tính năng gợi ý sản phẩm và chatbot hỗ trợ, đảm bảo khả năng mở rộng và tích hợp với hạ tầng Spring Cloud (Gateway, Eureka, Config).

Tiểu luận đã hoàn thành việc nghiên cứu và xây dựng một hệ thống mua sắm, tích hợp các chức năng chính sau:

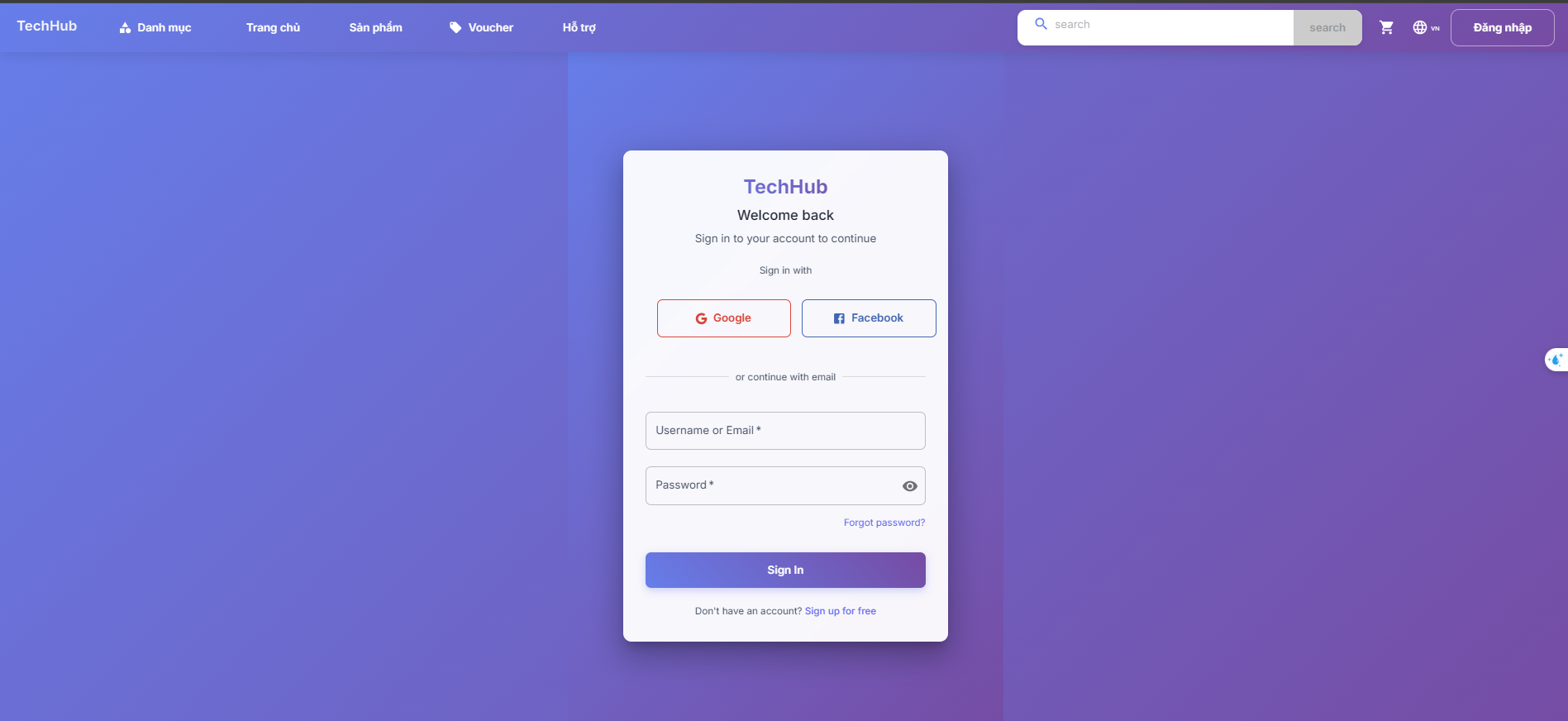
Giao diện trang đăng kí:



*Hình 17. Màn hình đăng nhập của hệ thống.*

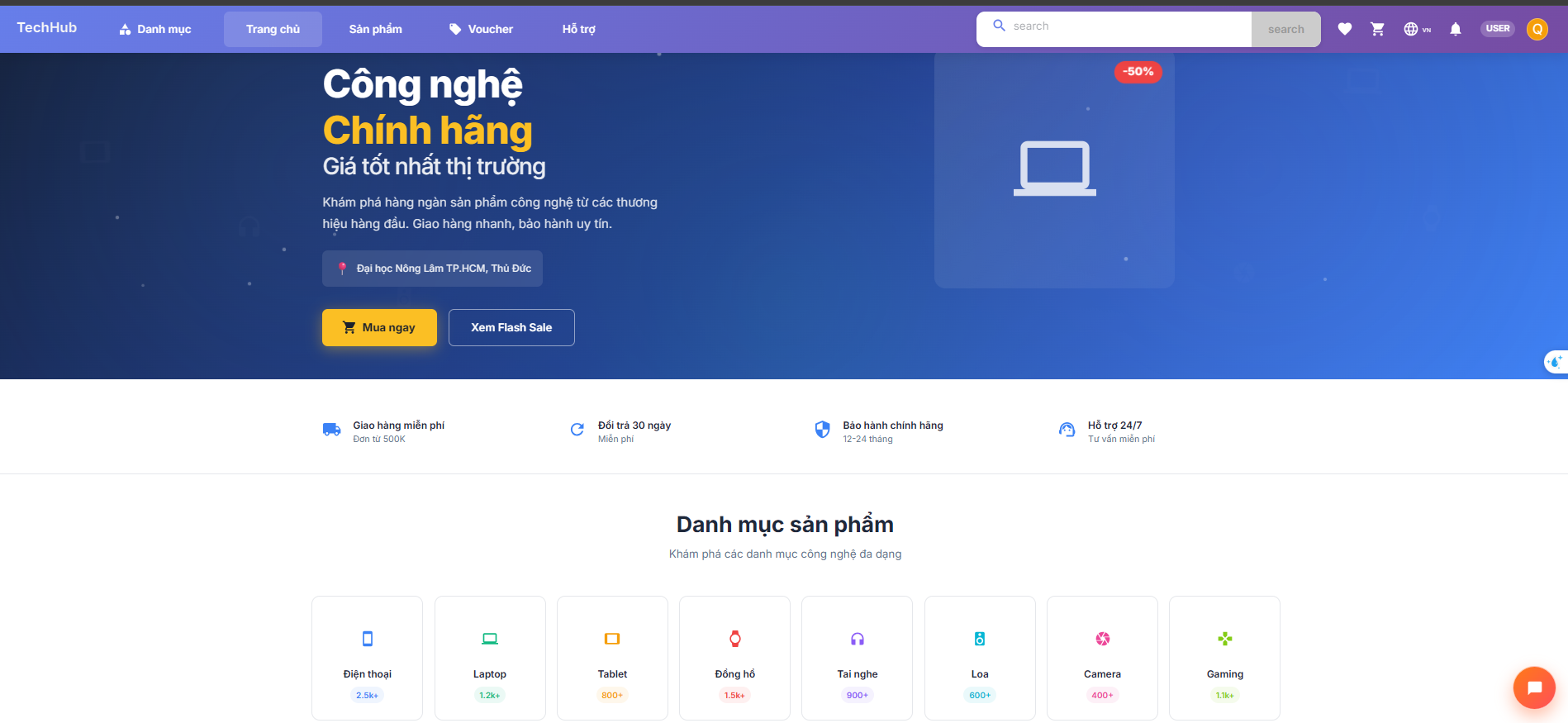
* Người dùng nhập thông tin như : Họ và Tên, Email và Password để đăng kí tài khoản.
* Sau khi đăng ký xong hệ thống tự động đăng nhập vào hệ thống.

Màn hình đăng nhập:



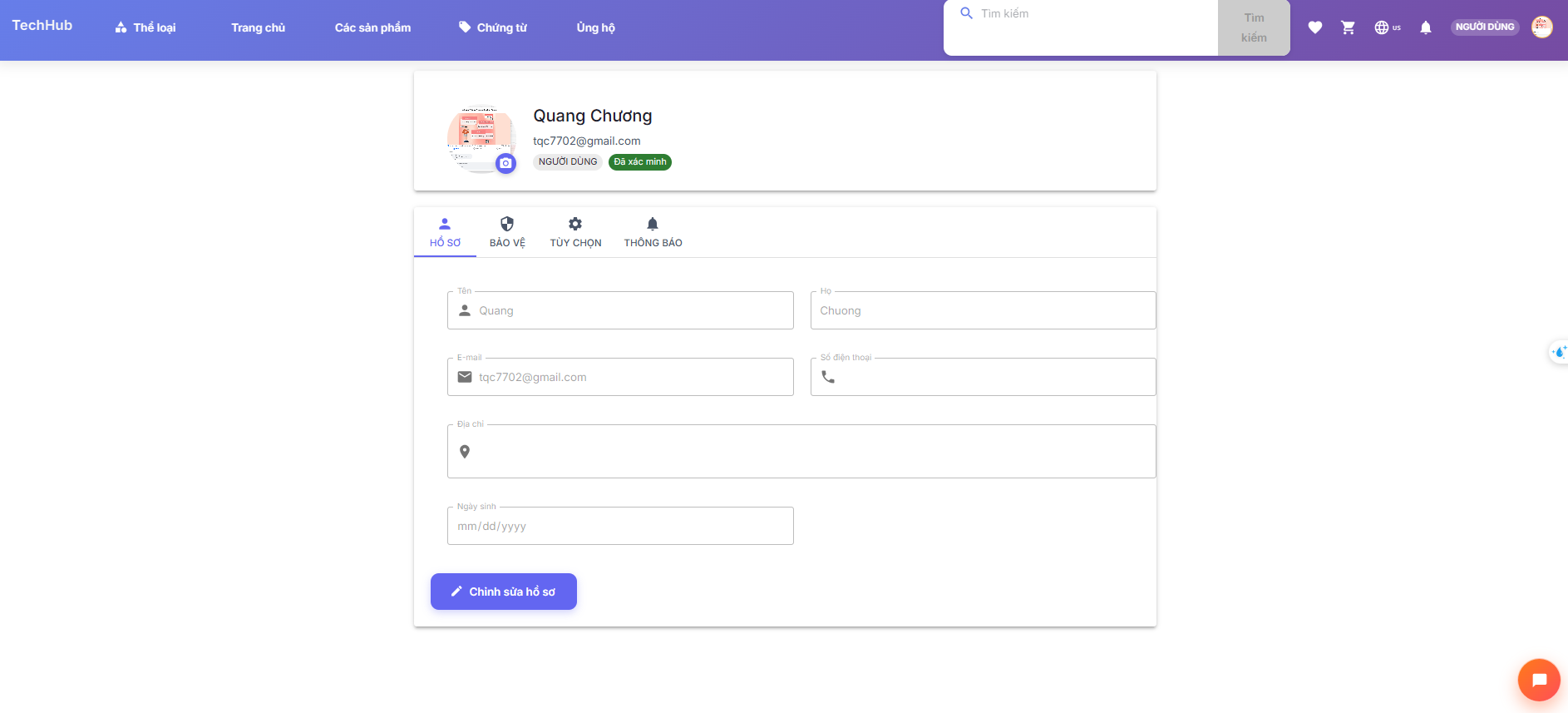
*Hình 18. Màn hình đăng ký.*

* Người dùng sẽ nhập Email và Password để đăng nhập vào hệ thống.
* Nếu đăng nhập thành công hệ thống sẽ điều hướng sang trang chủ.



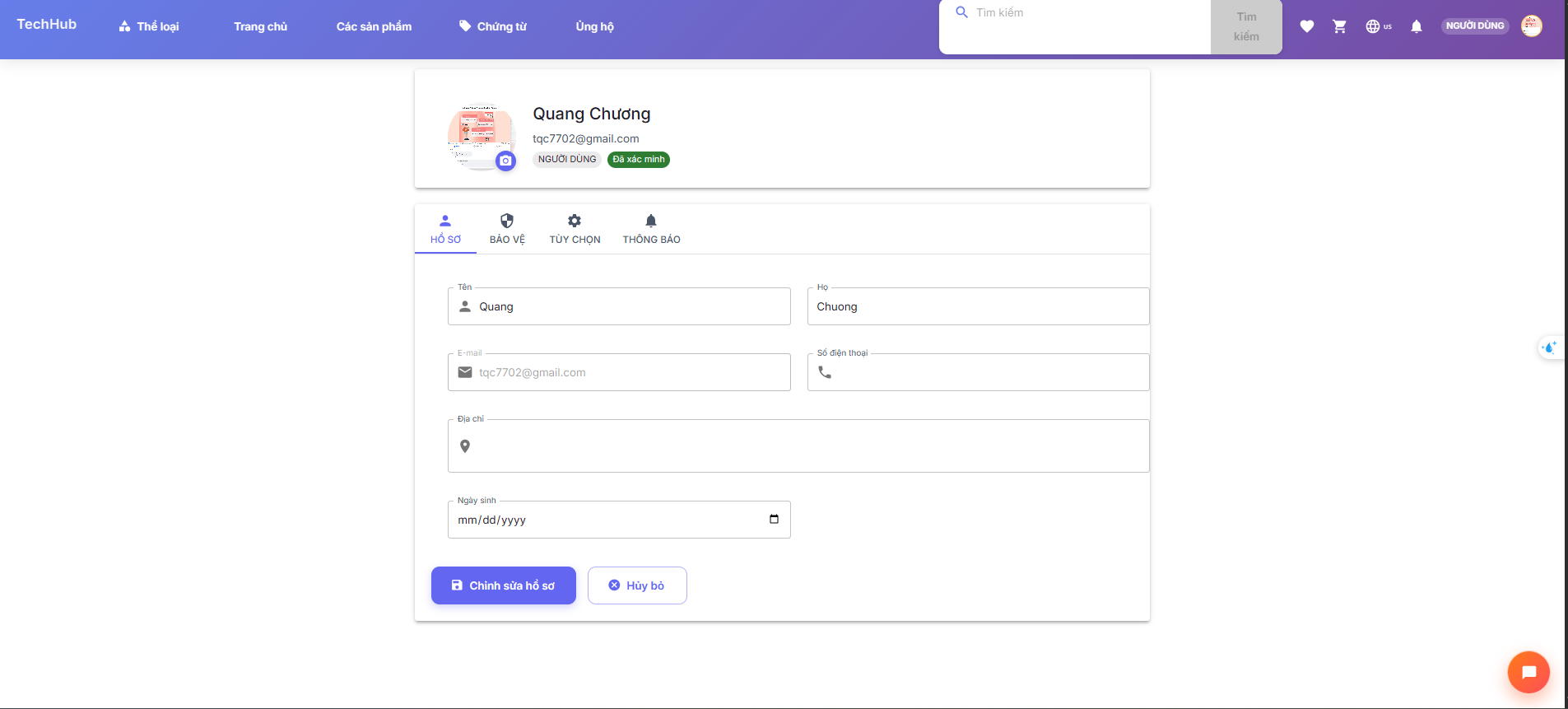
*Hình 19. Màn hình trang chủ* Trang chủ gồm có:

* Thanh tìm kiếm: giúp người dùng nhanh chóng tìm sản phẩm (tên/ID/SKU) hoặc truy cập trực tiếp trang chi tiết khi biết mã sản phẩm.
* Danh sách đề xuất/công khai: hiển thị các sản phẩm nổi bật/khuyến mãi/bán chạy để người dùng truy cập nhanh.
* Header: gồm nút tài khoản (đăng nhập/đăng ký, hồ sơ, đơn hàng) và nút tạo/gửi yêu cầu (ví dụ: yêu cầu tư vấn, báo giá), kèm các nút giỏ hàng và ngôn ngữ nếu cần.



*Hình 20. Màn hình chức năng đổi thông tin cá nhân.*

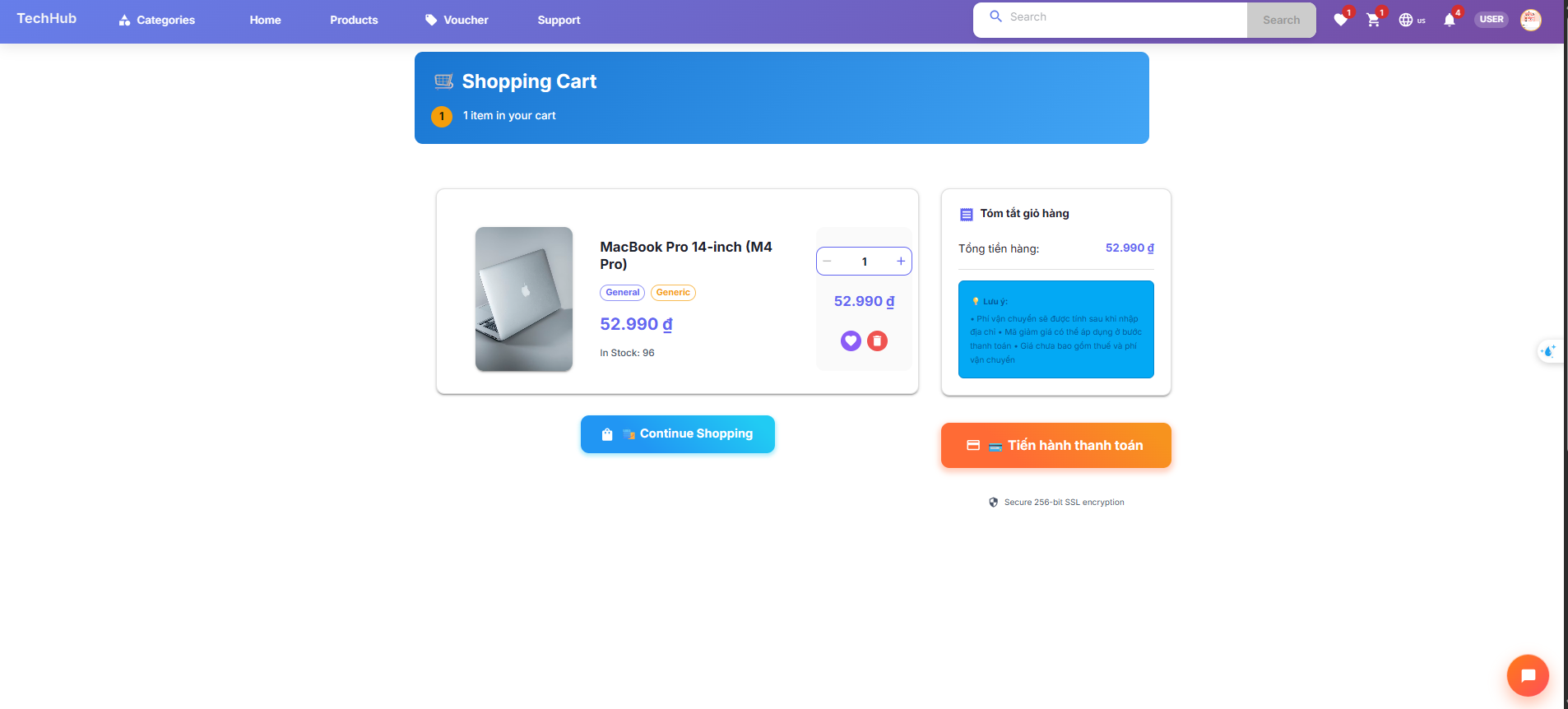
* Nhấn vào Tài khoản để mở ngăn kéo thông tin cá nhân. Tại đây người dùng có thể cập lại thông tin cá nhân như Email, Họ và tên, xem các cuộc bình chọn, đổi mật khẩu.
* Khi chọn đổi mật khẩu ngăn kéo đổi mật khẩu được mở ra.



*Hình 21. Màn hình chức năng đổi mật khẩu.*

* Người dùng cần nhập mật khẩu cũ và mật khẩu mới để thực hiện đổi mật khẩu.

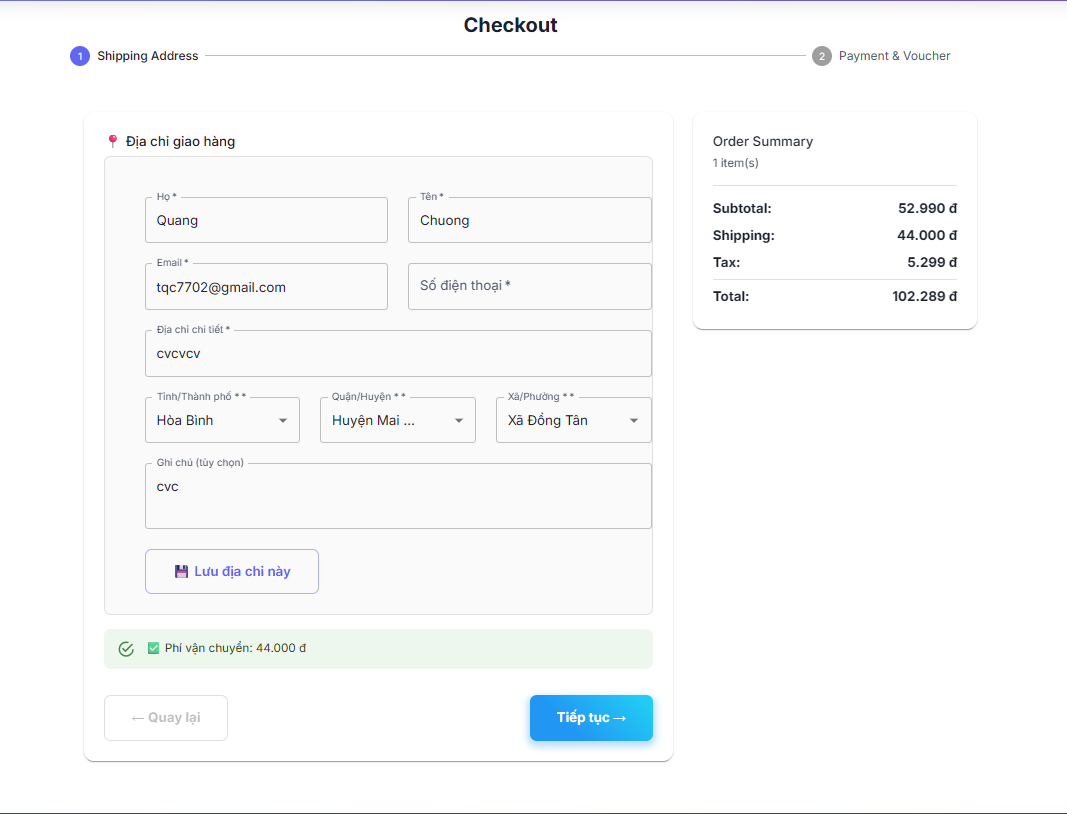
Màn hình chi tiết giỏ hàng:



*Hình 22. Màn hình trang chi tiết giỏ hàng.*

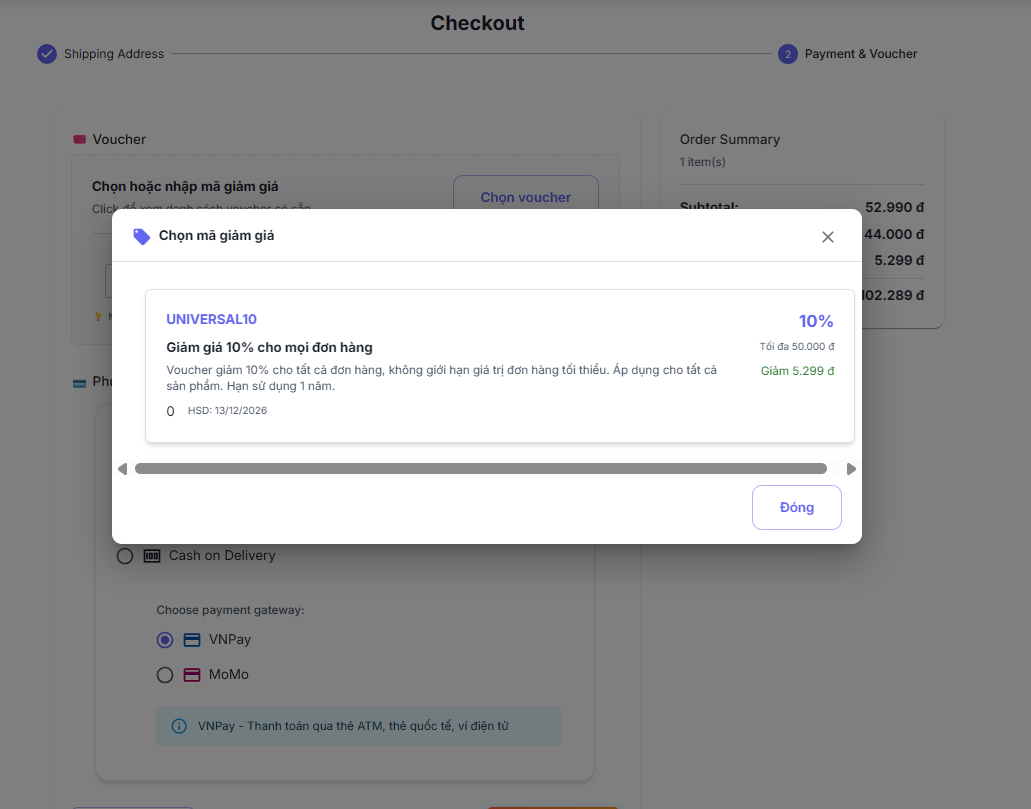
Cho phép người dùng xem đầy đủ thông tin các sản phẩm trong giỏ hàng, bao gồm ảnh, tên sản phẩm, mô tả ngắn, đơn giá, số lượng, thành tiền và mã sản phẩm/ID. Người dùng có thể thực hiện các chức năng sau:

* Điều chỉnh giỏ hàng: tăng/giảm số lượng, xóa sản phẩm khỏi giỏ, chọn/bỏ chọn sản phẩm để thanh toán.
* Xem tổng quan chi phí: hiển thị tổng tiền hàng, phí vận chuyển tạm tính (nếu có), giảm giá (voucher/mã khuyến mãi nếu hệ thống hỗ trợ) và tổng thanh toán dự kiến.



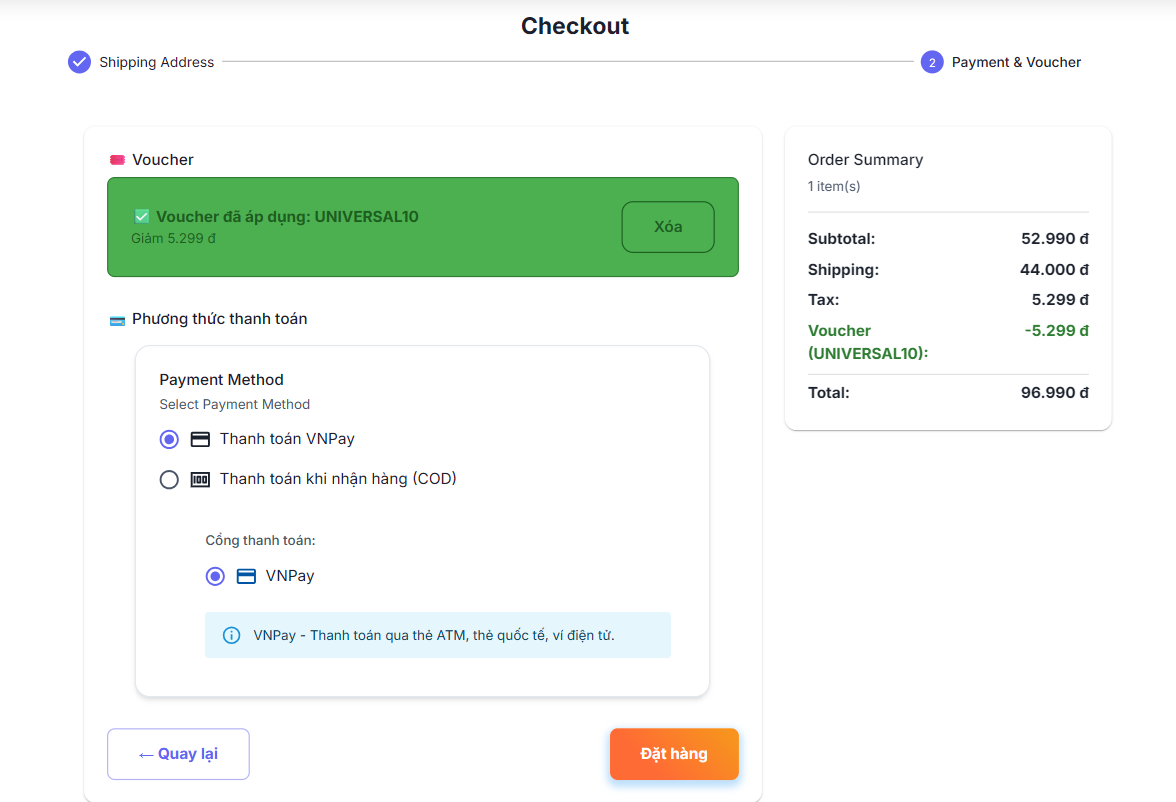
*Hình 23. Giao diện của trang khi điền thông tin để ra giá tiền giao hàng.*

Khi nhấn điền thông tin các tỉnh/huyện để giao hàng thì hệ thống tin tính ra giá tiền giao hàng.



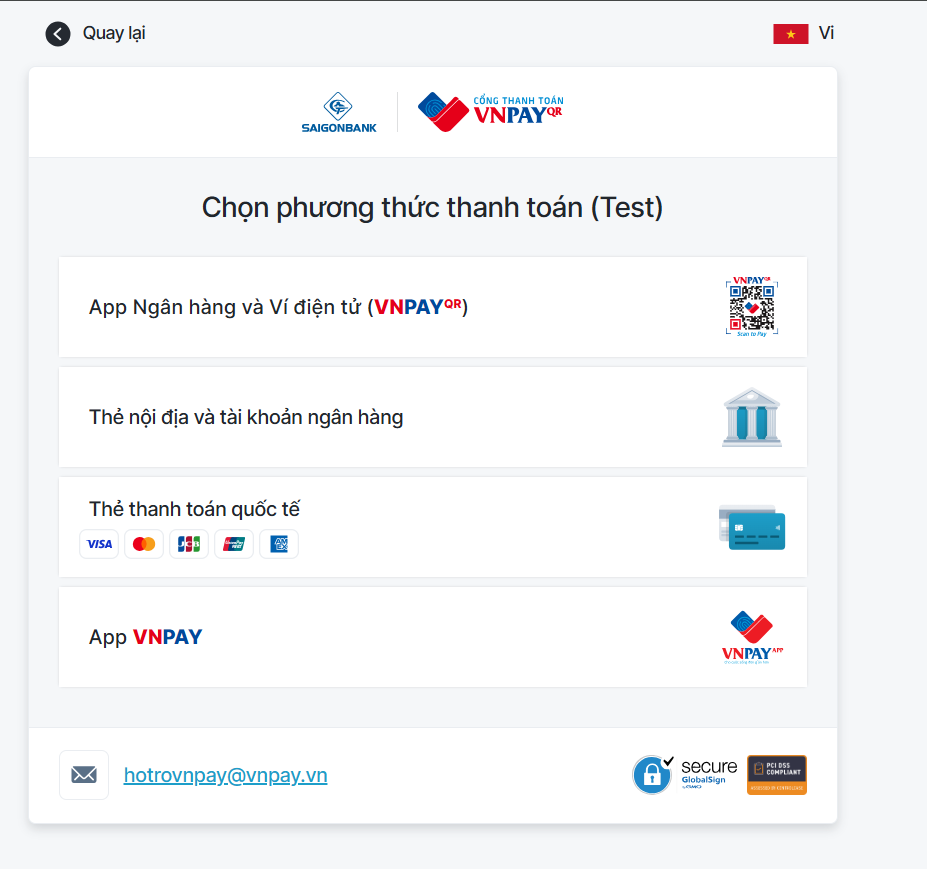
*Hình 24.Chọn voucher phù hợp với đơn hàng.*

* Người dùng có thể chọn các phương thức thanh toán phù hợp với cá nhân.



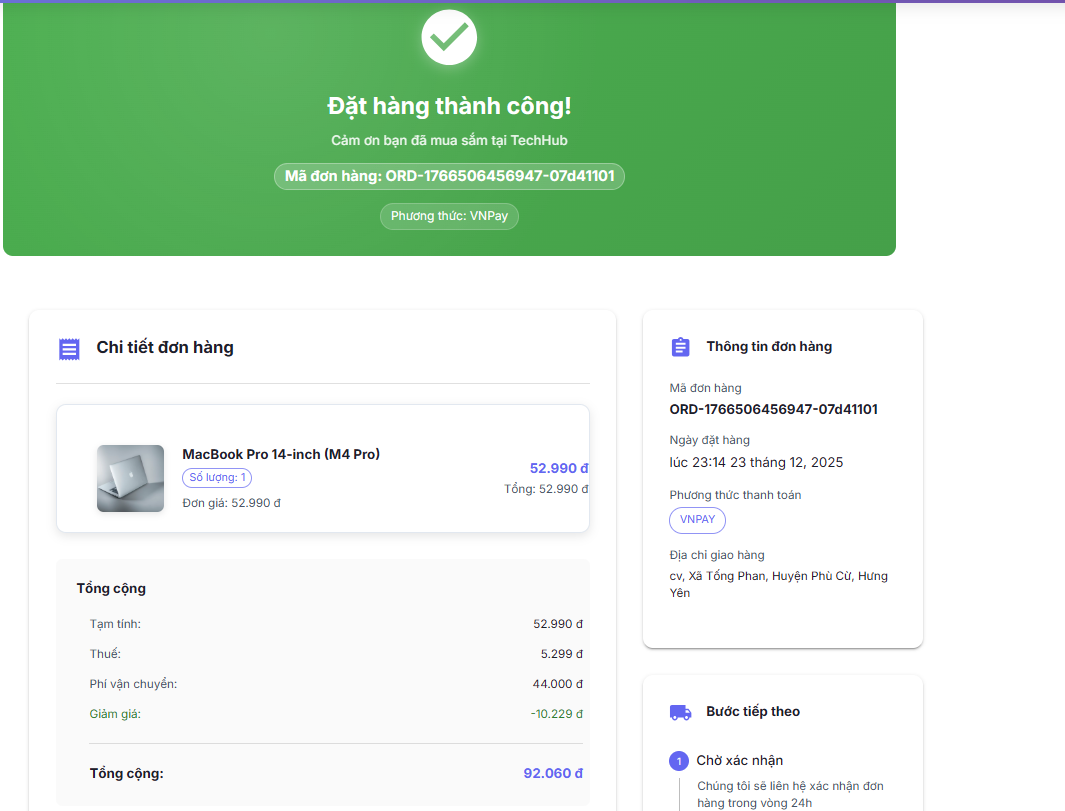
*Hình 25. Hình các phương thức thanh toán.*

* Khi nhấn “đặt hàng”, thì hệ thống sẽ tiến hàng bước thanh toán theo phương thức thanh toán mà khách hàng chọn.



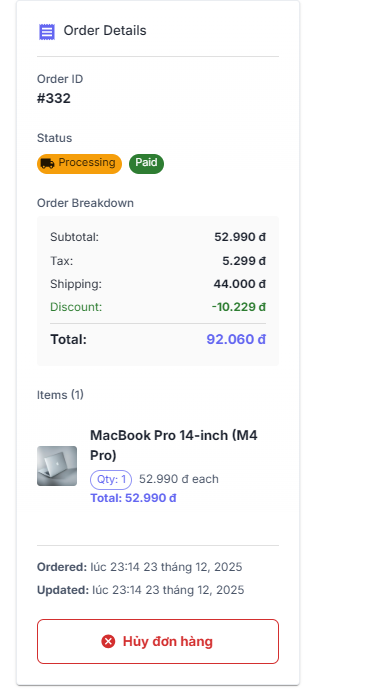
*Hình 26. Khi khách hàng mà chọn vnpay.*

* Nếu khách hàng thanh toán hoàn toán bằng tài khoản vnpay và đáp ứng đủ các giá tiền thành công thì hệ thống sẽ thống báo là đặt hàng thành công.



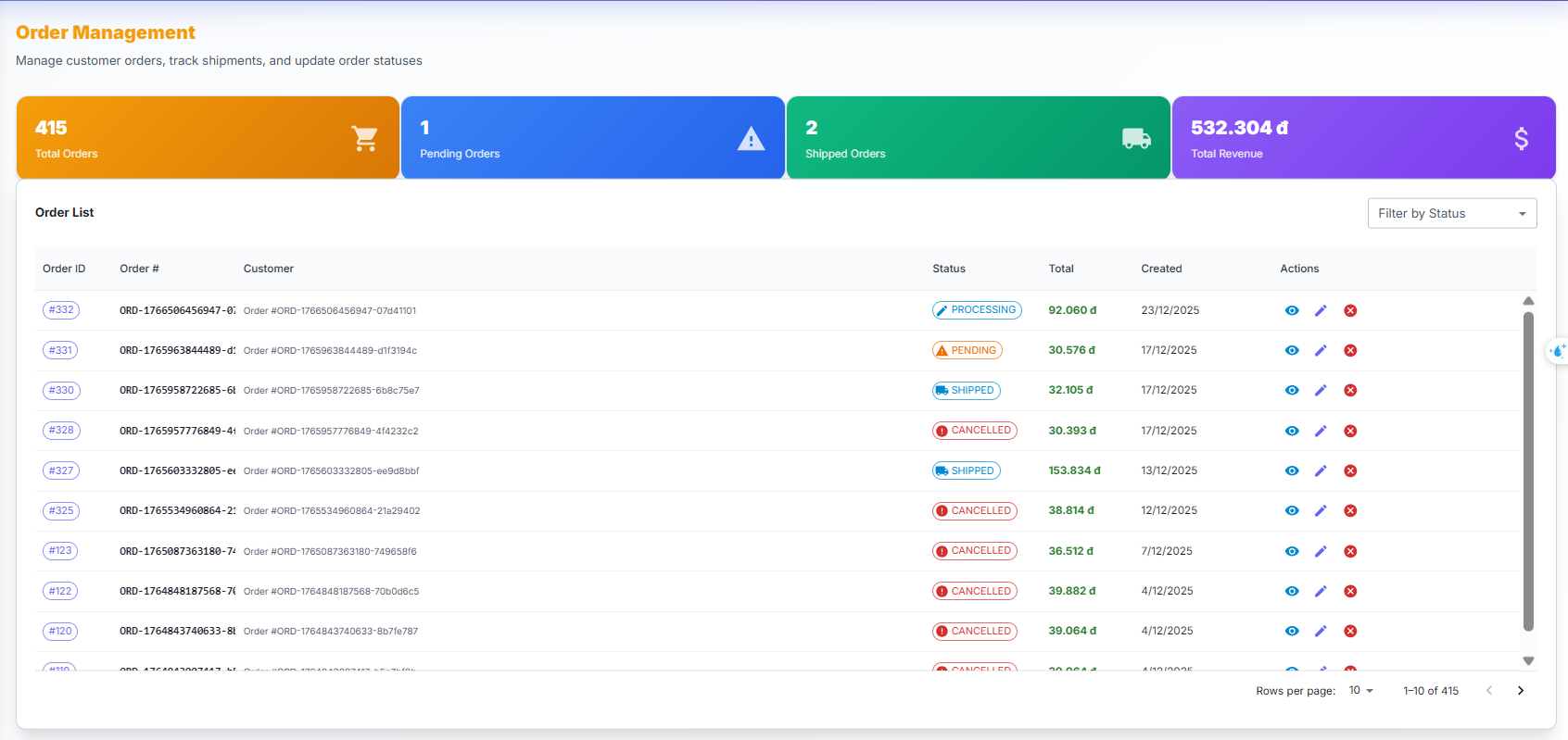
*Hình 27. Màn hình trang chi tiết đặt hàng thành công.*

* Chủ sở hữu nhấn vào “xem chi tiết đơn hàng” để xem lại đúng thông tin đơn hàng mình chọn chưa.



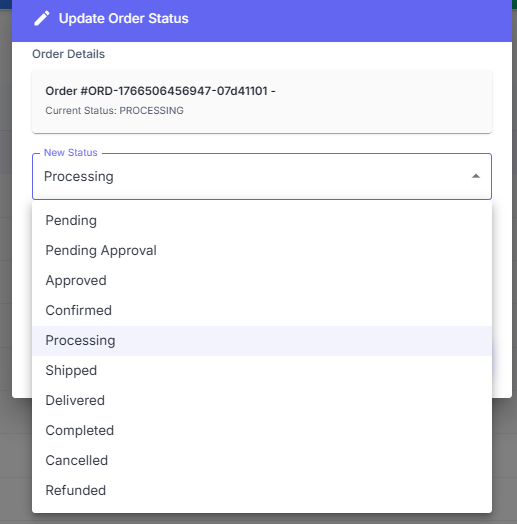
*Hình 28.Hình lịch sử đơn hàng.*

Màn hình admin duyệt đơn hàng:



*Hình 29. Màn hình admin quản lý các đơn hàng được tạo.*

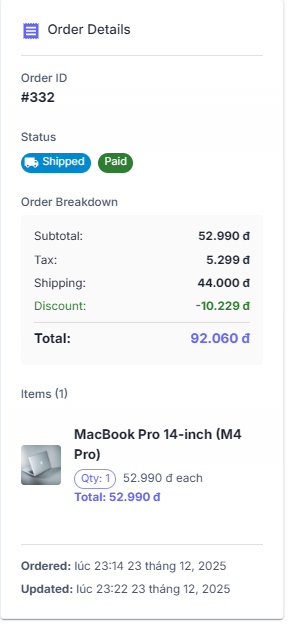
* Cho phép admin duyệt đơn , kiểm soát các đơn hàng.
* Có thể hủy nhận đơn.



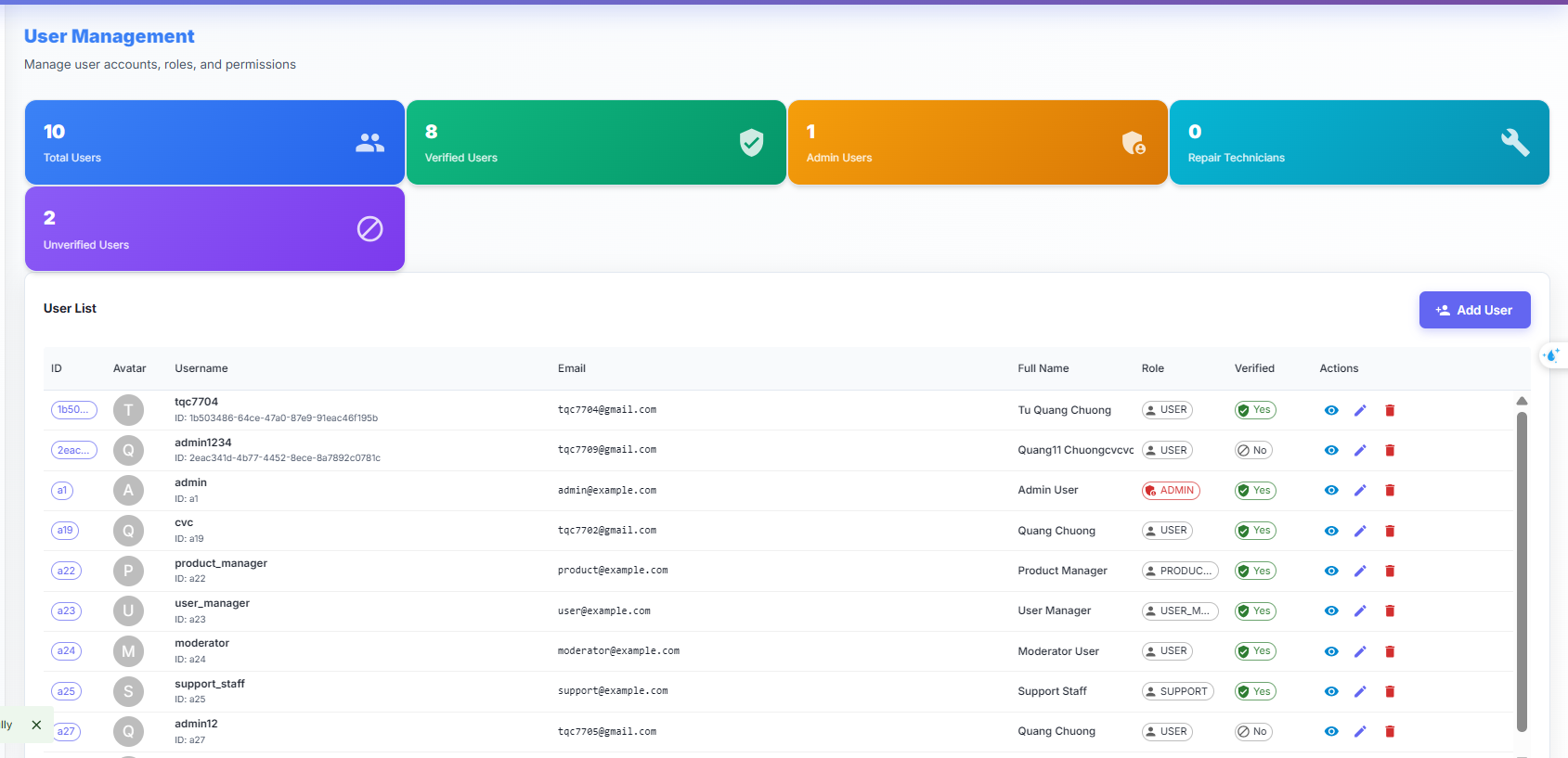
*Hình 30. Màn hình chỉnh sửa trạng thái đơn.*

* Trạng thái đơn hàng bên user được update ngay lập tức real-time luôn.
* User biết ngay trạng thái đơn hàng của mình như thế nào để tiện theo dõi đơn hàng.

Phía User:

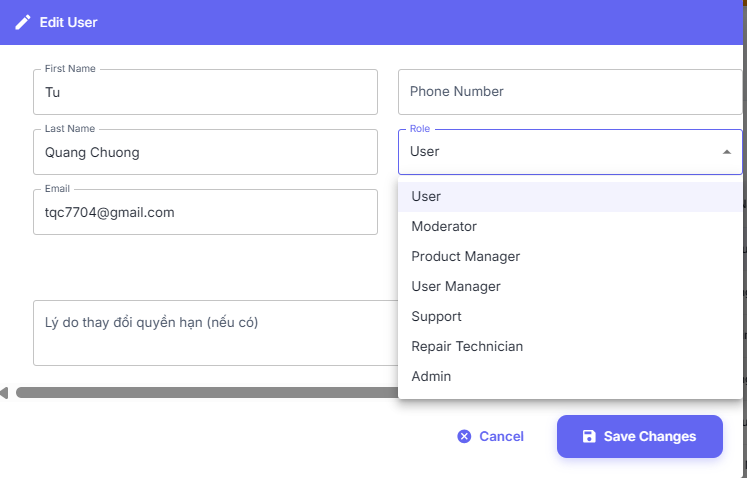


*Hình 31. Trạng thái đơn hàng đã được update*



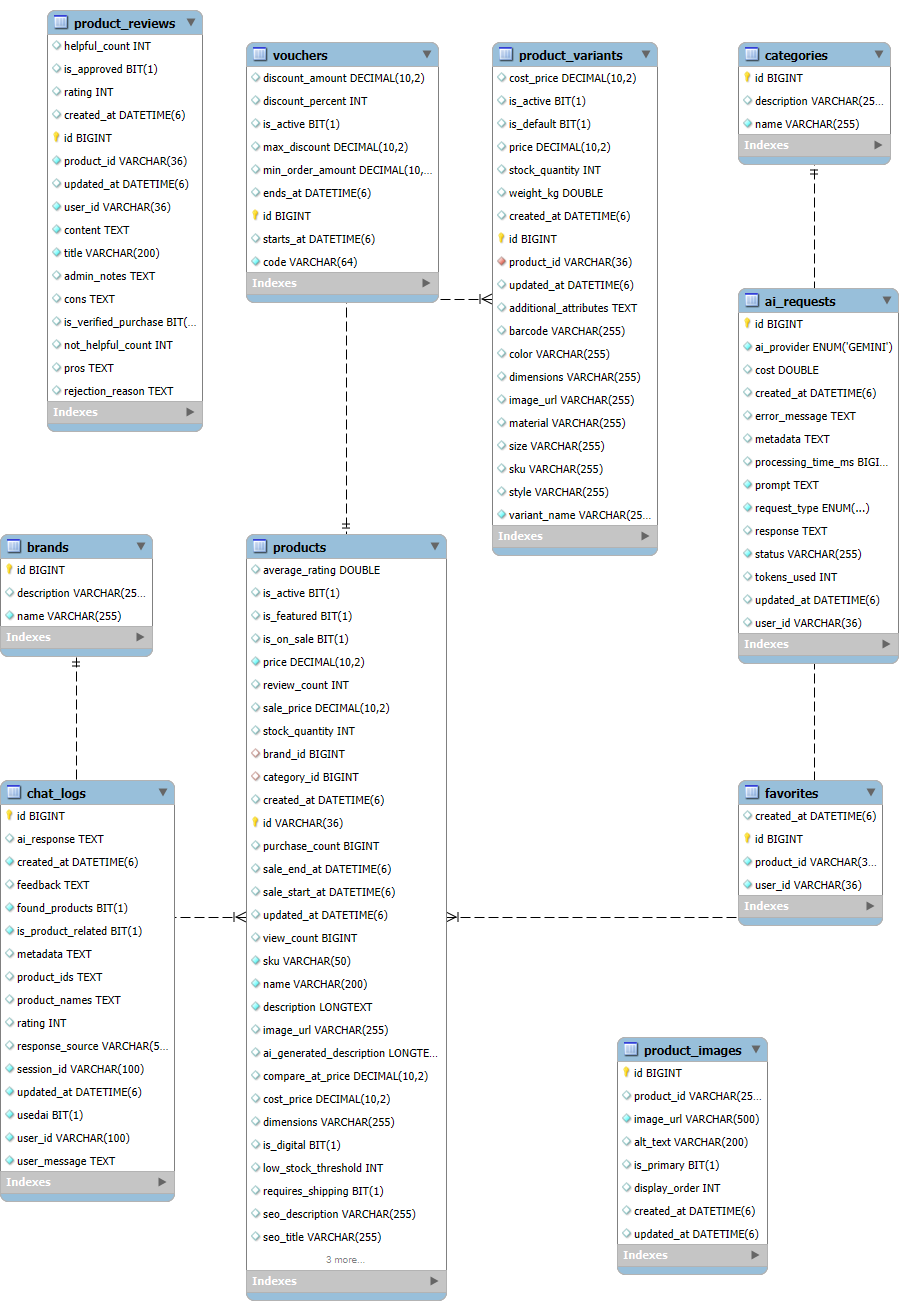
*Hình 33. Màn hình danh sách người dùng của hệ thống.*

* Tại đây admin có thể hạ quyền hoặc nâng quyền cho các account mà mình muốn.



*Hình 34. Màn hình phân quyền.*

#### 3.3.5. Lược đồ cơ sở dữ liệu



*Hình 35. Lược đồ cơ sở dữ liệu của hệ thống products.*

Phân tích các bảng trong cơ sở dữ liệu:

* *products*: Bảng lưu trữ thông tin sản phẩm chính của hệ thống, gồm tên, mô tả, giá, giá khuyến mãi, tồn kho tổng, cờ đang giảm giá/đang bán, điểm đánh giá trung bình, số lượt xem, SKU, các thông tin SEO và cấu hình khác.
* *votproduct\_variants*: Bảng lưu trữ các biến thể của sản phẩm (size, màu, kiểu dáng…), mỗi dòng gắn với một product\_id, có giá riêng, tồn kho riêng, SKU riêng, thuộc tính bổ sung (màu, kích thước, trọng lượng, chất liệu…).
* *product\_images*: Bảng lưu trữ danh sách hình ảnh của sản phẩm, mỗi ảnh gắn với một product\_id, gồm đường dẫn ảnh, text thay thế, cờ ảnh chính (is\_primary), thứ tự hiển thị, thời gian tạo/cập nhật.
* *product\_reviews:*Bảng lưu trữ các đánh giá của người dùng cho sản phẩm, gồm user\_id, product\_id, số sao (rating), tiêu đề, nội dung đánh giá, ưu/nhược điểm, cờ đã duyệt, đếm số lượt đánh giá hữu ích, lý do từ chối, v.v.
* *vouchers:* Bảng lưu trữ thông tin mã giảm giá của hệ thống, gồm code, phần trăm/giá trị giảm, giá trị giảm tối đa, điều kiện đơn hàng tối thiểu, thời gian bắt đầu/kết thúc, trạng thái đang hoạt động hay không.
* *categories:* Bảng lưu trữ danh mục sản phẩm, gồm tên và mô tả danh mục; dùng để phân loại sản phẩm và liên kết với bảng products.
* Brands: Bảng lưu trữ thông tin thương hiệu, gồm tên và mô tả thương hiệu; được tham chiếu bởi bảng products để gắn mỗi sản phẩm với một brand cụ thể
* Favorites: Bảng lưu trữ danh sách sản phẩm yêu thích (wishlist) của người dùng, mỗi dòng chứa user\_id và product\_id, cùng thời gian tạo; cho phép người dùng lưu lại sản phẩm quan tâm
* Chat\_logs: Bảng lưu trữ lịch sử hội thoại của chatbot với người dùng, gồm câu hỏi của user, phản hồi AI, feedback, metadata, cờ hội thoại liên quan sản phẩm, danh sách product\_ids/tên sản phẩm mà AI đã gợi ý, điểm rating, v.v.
* Ai\_requests: Bảng lưu trữ log các request gửi tới dịch vụ AI (ví dụ Gemini), gồm nhà cung cấp AI, prompt, loại request, chi phí, thời gian xử lý, số token, metadata, trạng thái (thành công/thất bại) và user\_id liên quan.

#### 3.3.6. Xây dựng các endpoint API

Các endpoint API của hệ thống:

+ Xác thực người dùng:

|  |  |  |
| --- | --- | --- |
| api/auth | /register | Đăng ký tài khoản mới |
| /login | Đăng nhập vào hệ thống |
| /logout | Đăng xuất khỏi hệ thống |
| /changePassword | Đổi mật khẩu người dùng |

+ Quản lý các sản phẩm:

|  |  |  |
| --- | --- | --- |
| api/products | /{id}/variants, product-variants/ | Biến thể của sản phẩm |
| /:id | Danh sách, chi tiết từng sản phẩm |
| /categories, /brands | Danh mục , thương hiệu |
| /{id}/reviews | Đánh giá sản phẩm |

+ Quản lý giỏ hàng:

|  |  |  |
| --- | --- | --- |
| api/cart | /{userId} | Lấy giỏ hàng |
| /{userId}/add | Thêm sản phẩm vào giỏ hàng |
| /guest/add?sessionId | Giỏ hàng guest (chưa login). |
| {userId}/count, /empty, /health | Đếm item, kiểm tra trống, health. |
| {userId}/remove/{productId} | Xóa item / xóa giỏ. |
| /{userId}/update | Cập nhập số lượng |

+ Quản lý đơn hàng:

|  |  |  |
| --- | --- | --- |
| api/orders | /orders | Danh sách đơn (phân trang, filter theo status). |
| {id}, /number/{orderNumber}, /my-orders?userId | Chi tiết / đơn của user |
| /user/{userId}/purchased-products | Sản phẩm user đã mua (check review). |
| /{orderId}/confirm | Admin xác nhận đơn hàng |
| /confirm-payment | Backend cập nhật đơn sau khi thanh toán thành công. |
| /by-number/{orderNumber}/cancel, /{orderId}/cancel | Hủy đơn (kể cả hủy khi thanh toán lỗi). |

+ Tương tác với blockchain:

|  |  |  |
| --- | --- | --- |
| api/recommendations | /behavior | Ghi lại hành vi người dùng (view, add\_to\_cart, purchase, favorite...). |
| /personalized/{userId}?limit=...: | Gợi ý cá nhân hóa cho user. |
| /similar/{productId}?limit=...: | Gợi ý sản phẩm tương tự. |

#### 3.3.8. Cơ chế bảo mật JWT

Xác thực người dùng:

* Khi người dùng đăng nhập vào hệ thống, hệ thống sẽ xác thực thông tin đăng nhập (email và mật khẩu).
* Nếu thông tin đăng nhập hợp lệ, máy chủ sẽ tạo ra một JWT (JSON Web Token) bao gồm thông tin nhận dạng của người dùng là id và mã hóa nó bằng một khóa bí mật. Token này sẽ được ghi vào cookies.

Phân quyền và quản lý phiên đăng nhập:

* JWT được gửi từ máy chủ đến người dùng qua HTTP cookie. Mỗi khi người dùng gửi yêu cầu API đến máy chủ, JWT sẽ được gửi kèm theo để xác thực.
* Máy chủ nhận JWT, giải mã nó bằng khóa bí mật, và kiểm tra tính hợp lệ của token.
* Nếu JWT hợp lệ, máy chủ sẽ cho phép người dùng thực hiện các yêu cầu đã được phân quyền dựa trên thông tin trong JWT (ví dụ: quyền truy cập vào các route hay dịch vụ cụ thể).

# CHƯƠNG 4. KẾT QUẢ, KẾT LUẬN VÀ KIẾN NGHỊ

### 4.1. KẾT QUẢ

Sau quá trình tìm hiểu và thực hiện đề tài “Xây dựng hệ thống bán hàng trực tuyến tích hợp AI và công nghệ microservices” em đã đạt được những kết quả sau:

- Kiến thức tìm hiểu được:

+ Kiến thức về kiến trúc microservices: hiểu cách phân tách hệ thống thành các service độc lập, giao tiếp qua API Gateway, service discovery (Eureka), và centralized configuration (Config Server).

+ Kiến thức về Spring Boot và Spring Cloud: nắm cách xây dựng REST API, tích hợp Spring Security với JWT, quản lý transaction, và xử lý async với Feign Client.

+ Kiến thức về AI/ML: tích hợp Gemini API cho chatbot và content generation; xây dựng recommendation service (collaborative filtering, content-based filtering) bằng Python/FastAPI.

+ Kỹ năng phát triển web: Nâng cao kỹ năng phát triển web bằng React.js và Spring Boot, tích hợp với MySQL, VNPAY và các công cụ khác.

+ Kỹ năng tích hợp dịch vụ bên thứ ba: GHN API (vận chuyển), OAuth2 (Google/Facebook), và các dịch vụ AI.

- Hệ thống đã đạt được các chức năng:

+ Quản lý người dùng: đăng ký, đăng nhập (username/password, OAuth2), quản lý hồ sơ, đổi mật khẩu, xác thực email/OTP.

+ Quản lý sản phẩm: CRUD, biến thể, hình ảnh, danh mục, thương hiệu, tìm kiếm và lọc.

+ Giỏ hàng: thêm/sửa/xóa, áp voucher, tính tổng tiền, lưu giỏ cho guest và user.

+ Đặt hàng và thanh toán: tạo đơn, chọn địa chỉ (GHN), tính phí ship, thanh toán VNPay/COD, xử lý callback, quản lý trạng thái đơn.

+ Gợi ý sản phẩm: gợi ý cá nhân hóa dựa trên lịch sử, sản phẩm tương tự, tracking hành vi.

+ Chatbot hỗ trợ: tích hợp Gemini, trả lời câu hỏi, gợi ý sản phẩm từ ngữ cảnh, phân tích sentiment.

+ Đánh giá và review: tạo/sửa/xóa review, rating, kiểm tra quyền review (đã mua).

+ Quản lý tồn kho: kiểm tra tồn kho, reserve/release khi đặt/hủy đơn.

+ Admin dashboard: quản lý sản phẩm, đơn hàng, user, thống kê, analytics.

Ưu điểm:

* Kiến trúc microservices: dễ mở rộng, bảo trì, triển khai độc lập từng service.
* Tích hợp AI: chatbot và gợi ý sản phẩm nâng cao trải nghiệm người dùng
* Bảo mật: JWT, OAuth2, Spring Security, validation đầu vào, xử lý lỗi.
* Hiệu năng: Redis cache, React Query, lazy loading, tối ưu truy vấn DB.
* Tích hợp thanh toán và vận chuyển: VNPay, Stripe, GHN API.

Nhược điểm:

* Độ phức tạp: nhiều service, cần quản lý cấu hình, monitoring, logging tập trung.
* Chi phí vận hành: nhiều service, database riêng, Redis, dịch vụ bên thứ ba (AI, payment, shipping).
* Phụ thuộc dịch vụ bên ngoài: VNPay, GHN, Gemini API; cần xử lý fallback khi lỗi.
* Độ trễ mạng: giao tiếp giữa các service có thể tăng độ trễ so với monolith.

## 4. 2. KẾT LUẬN

Tiểu luận đã nghiên cứu và xây dựng thành công hệ thống bán hàng trực tuyến với kiến trúc microservices, tích hợp AI/ML và các dịch vụ thanh toán/vận chuyển. Hệ thống sử dụng Spring Boot (Java 17), React + TypeScript, MySQL, Redis, và tích hợp Gemini API, VNPay, GHN API để tạo giải pháp toàn diện.

Kết quả cho thấy:

* Kiến trúc microservices phù hợp cho hệ thống quy mô lớn, dễ mở rộng và bảo trì.
* Tích hợp AI/ML cải thiện trải nghiệm người dùng qua chatbot và gợi ý sản phẩm.
* Tích hợp thanh toán và vận chuyển tạo luồng mua hàng hoàn chỉnh.

Hệ thống đáp ứng các yêu cầu cơ bản của một nền tảng thương mại điện tử hiện đại và có thể mở rộng thêm tính năng.

## 4. 3. KIẾN NGHỊ

Để tiếp tục phát triển hệ thống, đề xuất:

Về kiến trúc và hiệu năng:

* Triển khai monitoring và logging tập trung (Prometheus, Grafana, ELK Stack).
* Tối ưu database: indexing, query optimization, connection pooling.
* Cân nhắc message queue (Kafka/RabbitMQ) cho async processing và event-driven.
* Triển khai caching strategy tốt hơn: cache invalidation, distributed caching.

Về tính năng:

* Mở rộng AI: phân tích sentiment review, tự động tạo mô tả sản phẩm, tối ưu SEO.
* Cải thiện recommendation: deep learning, real-time personalization.
* Tích hợp thêm phương thức thanh toán: ví điện tử, trả góp.

Về bảo mật:

* Rate limiting và DDoS protection.
* Audit logging cho các thao tác quan trọng.
* Mã hóa dữ liệu nhạy cảm (PCI DSS compliance).
* Security scanning và penetration testing định kỳ.

Về triển khai:

* Containerization với Docker và orchestration với Kubernetes.
* CI/CD pipeline tự động (Jenkins/GitLab CI/GitHub Actions).
* Blue-green deployment để giảm downtime.
* Backup và disaster recovery.

Về trải nghiệm người dùng:

* Progressive Web App (PWA) cho mobile.
* Tối ưu SEO và performance (lazy loading, code splitting).
* A/B testing cho UI/UX.

Các đề xuất này giúp hệ thống hoàn thiện hơn, sẵn sàng cho production và có khả năng mở rộng quy mô lớn.

# *TÀI LIỆU THAM KHẢO*

**TÀI LIỆU THAM KHẢO**

1. Spring Boot Official Documentation. "Building Production-Ready Applications." Spring Framework, 2024. [Online]. Available: https://spring.io/projects/spring-boot.
2. Spring Cloud Documentation. "Microservices with Spring Cloud." Spring Framework, 2024. [Online]. Available: https://spring.io/projects/spring-cloud
3. Spring Security Documentation. "Authentication and Authorization." Spring Framework, 2024. [Online]. Available: https://spring.io/projects/spring-security
4. Google Gemini API Documentation. "Gemini AI Integration Guide." Google AI, 2024. [Online]. Available: https://ai.google.dev/docs.
5. React Official Documentation. "Building User Interfaces." Meta, 2024. [Online]. Available: https://react.dev.
6. TypeScript Documentation. "JavaScript with Syntax for Types." Microsoft, 2024. [Online]. Available: https://www.typescriptlang.org/docs.
7. Material-UI Documentation. "React Component Library." MUI Team, 2024. [Online]. Available: https://mui.com.
8. VNPay Integration Documentation. "Payment Gateway Integration Guide." VNPay, 2024. [Online]. Available: https://sandbox.vnpayment.vn/apis/.
9. GHN API Documentation. "Shipping Integration Guide." Giao Hàng Nhanh, 2024. [Online]. Available: https://api.ghn.vn/.
10. Ricci, F., Rokach, L., & Shapira, B. (2015). Recommender Systems Handbook. Springer Science & Business Media.
11. Aggarwal, C. C. (2016). Recommender Systems: The Textbook. Springer International Publishing.
12. Chen, H., Chiang, R. H., & Storey, V. C. (2012). Business intelligence and analytics: From big data to big impact. MIS quarterly, 1165-1188.
13. Zhang, S., Yao, L., Sun, A., & Tay, Y. (2019). Deep learning based recommender system: A survey and new perspectives. ACM Computing Surveys, 52(1), 1-38.
14. Lu, J., Wu, D., Mao, M., Wang, W., & Zhang, G. (2015). Recommender system application developments: a survey. Decision Support Systems, 74, 12-32.
15. Isinkaye, F. O., Folajimi, Y. O., & Ojokoh, B. A. (2015). Recommendation systems: Principles, methods and evaluation. Egyptian Informatics Journal, 16(3), 261-273.
16. FastAPI Documentation. "Modern Python Web Framework." FastAPI, 2024. [Online]. Available: https://fastapi.tiangolo.com/.
17. Scikit-learn Documentation. "Machine Learning in Python." Scikit-learn, 2024. [Online]. Available: https://scikit-learn.org/stable/.
18. MySQL Documentation. "Relational Database Management System." Oracle, 2024. [Online]. Available: https://dev.mysql.com/doc/.
19. Redis Documentation. "In-Memory Data Structure Store." Redis, 2024. [Online]. Available: https://redis.io/docs/.
20. React Query Documentation. "Data Fetching for React." TanStack, 2024. [Online]. Available: https://tanstack.com/query/latest.
21. React Hook Form Documentation. "Performant Forms with Easy Validation." React Hook Form, 2024. [Online]. Available: https://react-hook-form.com/.
22. Axios Documentation. "Promise-based HTTP Client." Axios, 2024. [Online]. Available: https://axios-http.com/docs/intro.
23. JWT.io Documentation. "JSON Web Token Introduction." Auth0, 2024. [Online]. Available: https://jwt.io/introduction.
24. OAuth 2.0 Documentation. "Authorization Framework." IETF, 2024. [Online]. Available: https://oauth.net/2/.
25. Chart.js Documentation. "Simple yet flexible JavaScript charting." Chart.js, 2024. [Online]. Available: https://www.chartjs.org/docs/.

*TÀI LIỆU THAM KHẢO*