

**TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN**



**VÕ TẤN PHÁT – 518H0239
NGUYỄN VIỆT MINH NHẬT – 518H0542**

XÂY DỰNG WEBSITE THƯƠNG MẠI ĐIỆN TỬ SỬ DỤNG JAVA SPRING THEO KIẾN TRÚC MICRO-SERVICES

DỰ ÁN CÔNG NGHỆ THÔNG TIN 2

NGÀNH KỸ THUẬT PHẦN MỀM

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

**TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN**



**VÕ TẤN PHÁT – 518H0239
NGUYỄN VIỆT MINH NHẬT – 518H0542**

**XÂY DỰNG WEBSITE THƯƠNG MẠI
ĐIỆN TỬ SỬ DỤNG JAVA SPRING
THEO KIẾN TRÚC MICRO-SERVICES**

DỰ ÁN CÔNG NGHỆ THÔNG TIN 2

NGÀNH KỸ THUẬT PHẦN MỀM

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

LỜI CẢM ƠN

Chúng em muốn gửi lời cảm ơn sâu sắc đến Thầy Võ Văn Thành đã hướng dẫn nhóm trong quá trình tìm hiểu về kiến trúc mircoservices bằng Java Spring Boot. Thầy đã giúp chúng em hiểu được nhiều kiến thức quan trọng về kiến mircoservices. Bên cạnh đó, thầy cũng đã giúp đỡ nhóm, theo sát tiến độ trong suốt quá trình hoàn thành dự án này. Chúng em rất cảm ơn Thầy đã dành thời gian để hướng dẫn và giúp đỡ nhóm trong quá trình học tập. Chúng em tin tưởng rằng kiến thức đã học được sẽ giúp bọn em trong sự nghiệp sau này. Xin cảm ơn Thầy một lần nữa và nhóm mong muốn sẽ có cơ hội học tập với Thầy trong tương lai.

TP. Hồ Chí Minh, ngày 08 tháng 02 năm 2023

Tác giả

(Ký tên và ghi rõ họ tên)

Võ Tấn Phát

Nguyễn Viết Minh Nhật

PHIẾU ĐÁNH GIÁ CỦA GIẢNG VIÊN HƯỚNG DẪN

Tên giảng viên hướng dẫn: Th.s. Võ Văn Thành.....

Ý kiến nhận xét:

.....

.....

.....

Điểm tổng theo phiếu đánh giá rubrik:

TP. Hồ Chí Minh, ngày 08 tháng 02 năm 2023

Giảng viên hướng dẫn

(Ký tên và ghi rõ họ tên)

Thành

Võ Văn Thành

ĐỒ ÁN ĐƯỢC HOÀN THÀNH TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Tôi xin cam đoan đây là sản phẩm đồ án của chúng tôi và được sự hướng dẫn của thầy Võ Văn Thành. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong đồ án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung đồ án của mình. Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

TP. Hồ Chí Minh, 08 ngày 02 tháng năm 2023

Tác giả

(ký tên và ghi rõ họ tên)

Võ Tấn Phát

Nguyễn Viết Minh Nhật

TÓM TẮT

Nhóm em tìm hiểu về kiến trúc Microservices và xây dựng website thương mại điện tử bằng Java Spring Boot. Nhóm em sẽ chia ra back end chia thành các service gồm UserService, ProductService, OrderService, PaymentService, ImageService và HistoryService. Phần front-end cũng sẽ chia ra làm 2 dự án riêng biệt là website bán hàng và website quản lý.

Trước tiên chúng em tìm hiểu về kiến trúc Microservices và Domain Driven Design. Sau khi tìm hiểu các chủ đề trên thì tụi em lại tiếp tục tìm hiểu về Java Spring Boot và implement thử 1 service bằng công nghệ này. Sau khi thành công thì tiếp tục với các services tiếp theo. Sau khi hoàn thành các services và hoàn thành back-end thì bọn em tiếp tục làm front-end cho dự án.

Sau khi hoàn thành dự án bọn em đã có được 1 trang website thương mại điện tử với đủ các chức năng cơ bản. Nhóm bọn em cũng hiểu được kiến trúc microservices, DDD và cũng cải thiện được khả năng code bằng công nghệ Java Spring Boot

MỤC LỤC

| | |
|--|-----------|
| CHƯƠNG I – GIỚI THIỆU..... | 3 |
| 1.1 Giới thiệu đề tài..... | 3 |
| 1.2 Mục tiêu và ý nghĩa đề bài | 3 |
| 1.3 Mô tả tổng quát..... | 3 |
| CHƯƠNG II – PHÂN TÍCH..... | 4 |
| 2.1 Use case | 4 |
| 2.2 Đặc tả hệ thống..... | 6 |
| CHƯƠNG III – SƠ ĐỒ CẤU TRÚC | 7 |
| 3.1 Sơ đồ hệ thống | 7 |
| 3.2 Cơ sở dữ liệu trong các Services | 8 |
| 3.2.1 Product Service..... | 8 |
| 3.2.2.User Service | 9 |
| 3.2.3 Payment Service | 9 |
| 3.2.4 Order Service..... | 10 |
| 3.2.5 HistoryService | 10 |
| 3.2.6 Image Service | 10 |
| 3.3 Flowchart của hệ thống..... | 11 |
| CHƯƠNG IV – CƠ SỞ LÝ THUYẾT | 12 |
| 4.1 Kiến trúc Mircoservice..... | 12 |
| 4.1.1 Khái niệm | 12 |
| 4.1.2 Những đặc điểm của mircoservice | 13 |
| 4.1.3 Ưu, nhược điểm của microservice..... | 14 |
| 4.2. Kiến trúc Domain Driven Design (DDD)..... | 15 |
| 4.2.1 Khái niệm DDD..... | 15 |
| 4.2.2 Các layer có trong DDD | 15 |

| | |
|--|-----------|
| CHƯƠNG V – HIỆN THỰC HỆ THỐNG..... | 17 |
| 5.1 Hiện thực back-end theo kiến trúc microservices bằng Java Spring Boot: | 17 |
| 5.1.1 Giới thiệu chung: | 17 |
| 5.1.2 User Service | 17 |
| 5.1.3 Product Service..... | 29 |
| 5.1.4 Order Service..... | 35 |
| 5.1.5 Payment Service | 42 |
| 5.1.6 History Service | 44 |
| 5.1.7 Image Service | 46 |
| 5.1.8 API Gateway | 48 |
| 5.1.9 Eureka Server | 48 |
| 5.2 Hiện thực website bán hàng..... | 48 |
| 5.3 Hiện thực website quản lý | 53 |
| CHƯƠNG VI – KẾT LUẬN | 61 |
| TÀI LIỆU THAM KHẢO | 62 |

CHƯƠNG I – GIỚI THIỆU

1.1 Giới thiệu đề tài

Với đề tài được giao, chúng em sẽ tìm hiểu về kiến trúc Microservice đang được nhiều dự án hiện nay áp dụng. Sau đó nhóm em sẽ hiện thực lại kiến trúc này bằng Java Spring Boot ở backend và dùng framework ReactJS, Angular để xây dựng lên một website thương mại điện tử

1.2 Mục tiêu và ý nghĩa đề bài

Kiến trúc microservice là kiến trúc đang thịnh hành gần đây và đang được rất nhiều doanh nghiệp và dự án sử dụng. Microservice giúp các hệ thống website có thể dễ dàng được mở rộng và maintance nên đây là một kiến trúc rất được ưu chuộng. Nhóm em sẽ dùng kiến trúc này để tạo ra website thương mại điện tử, một loại website cũng đang rất phát triển và nó có thể thể hiện được các ưu điểm của kiến trúc microservice.

Website thương mại điện tử sẽ bao gồm các chức năng chính sau:

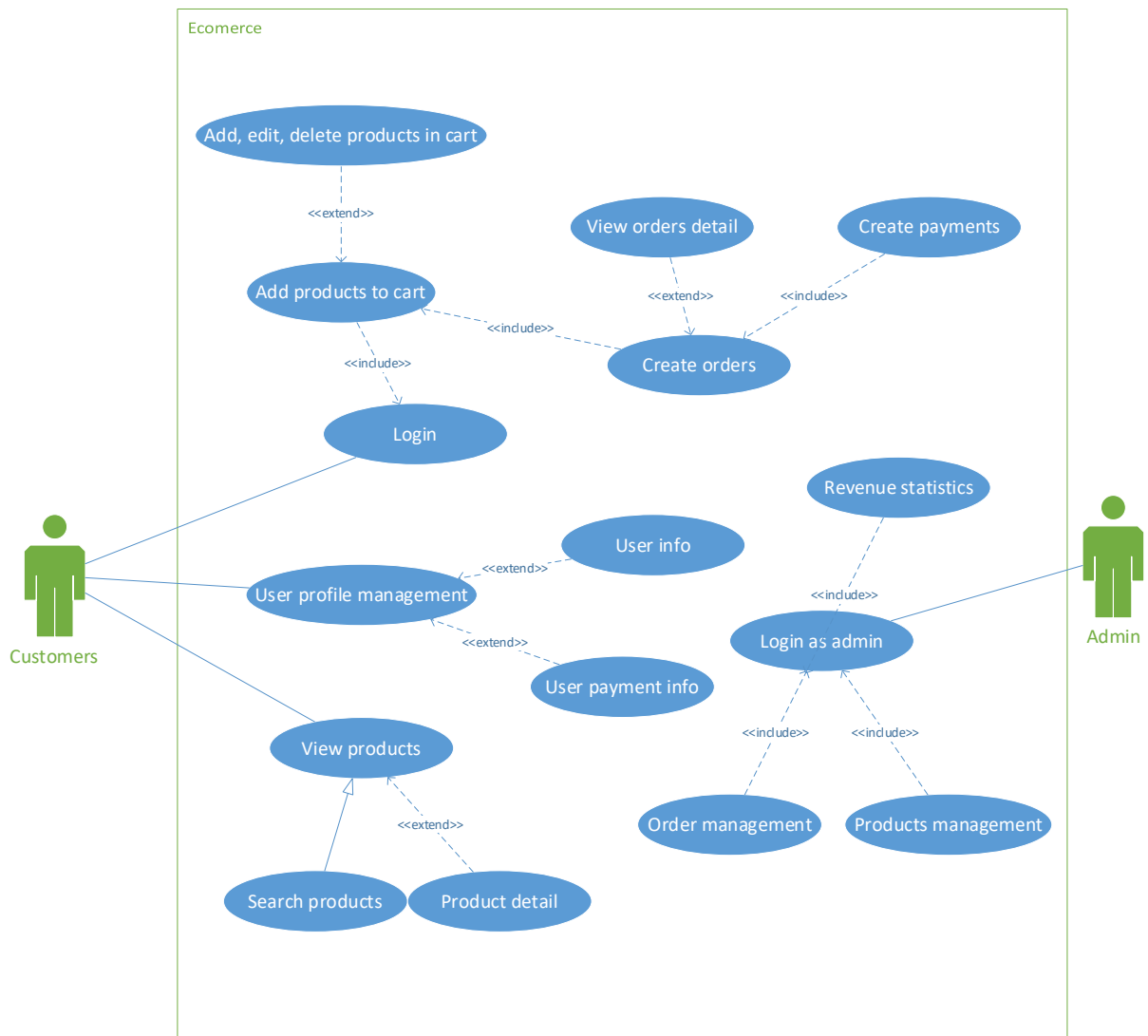
- Quản lý sản phẩm.
- Quản lý thông tin khách hàng
- Quản lý giỏ hàng
- Quản lý danh sách đơn hàng
- Quản lý thanh toán đơn hàng

1.3 Mô tả tổng quát

Website thương mại điện tử được xây dựng theo kiến trúc microservice có thể dễ dàng mở rộng sau này và khi có một chức năng bị lỗi vẫn sẽ không ảnh hưởng đến các chức năng khác.

CHƯƠNG II – PHÂN TÍCH

2.1 Use case



Hình 2.1.1. Use case diagram

Website thương mại điện tử sẽ có các chức năng chính sau có các use case sau:

Bảng 2.1.1. Chi tiết use cases

| STT | Use case | Mô tả |
|-----|-------------------------|---|
| 1 | Login | Cho phép users login để lấy thông tin người dùng |
| 2 | User profile management | Quản lý các thông tin cơ bản của user cũng như các tài khoản, phương thức thanh toán. |
| 3 | View products | Hiện thị danh sách sản phẩm của hệ thống đang có |
| 4 | Search products | Hiện thị danh sách sản phẩm theo các điều kiện của users. |
| 5 | Product detail | Chức năng hiện thị chi tiết về sản phẩm như giá tiền, màu sắc, số lượng, mô tả... |
| 6 | Add product to cart | Chức năng cho phép user lưu trữ các sản phẩm có dự định mua vào giỏ hàng |
| 7 | Edit cart | User có thể thay đổi cart trong quá trình sử dụng website. Các thao tác có thể là thêm, sửa, xóa sản phẩm trong giỏ hàng. |
| 8 | Create orders | User sẽ tạo ra các order dựa trên số sản phẩm có trong cart. |
| 9 | View order detail | Chức năng cho phép user theo dõi chi tiết đơn hàng như là trạng thái đơn hàng, số tiền cần thanh toán, thời gian dự kiến giao hàng, số lượng sản phẩm có trong đơn hàng ... |
| 10 | Create payment | Chức năng cho phép người dùng thanh toán đơn hàng đã tạo và đơn hàng sẽ được đưa lên hệ thống cửa hàng để bắt đầu giao sau khi thực hiện payment |
| 11 | Login as admin | Chức năng cho admin đăng nhập và website sẽ điều hướng admin đến website quản lý. |
| 12 | Products management | Chức năng cho admin thêm, sửa, xóa số lượng, thông tin của sản phẩm |
| 13 | Order management | Chức năng cho admin thêm, sửa, xóa thông tin trên đơn hàng cũng như theo dõi quá trình của đơn hàng |

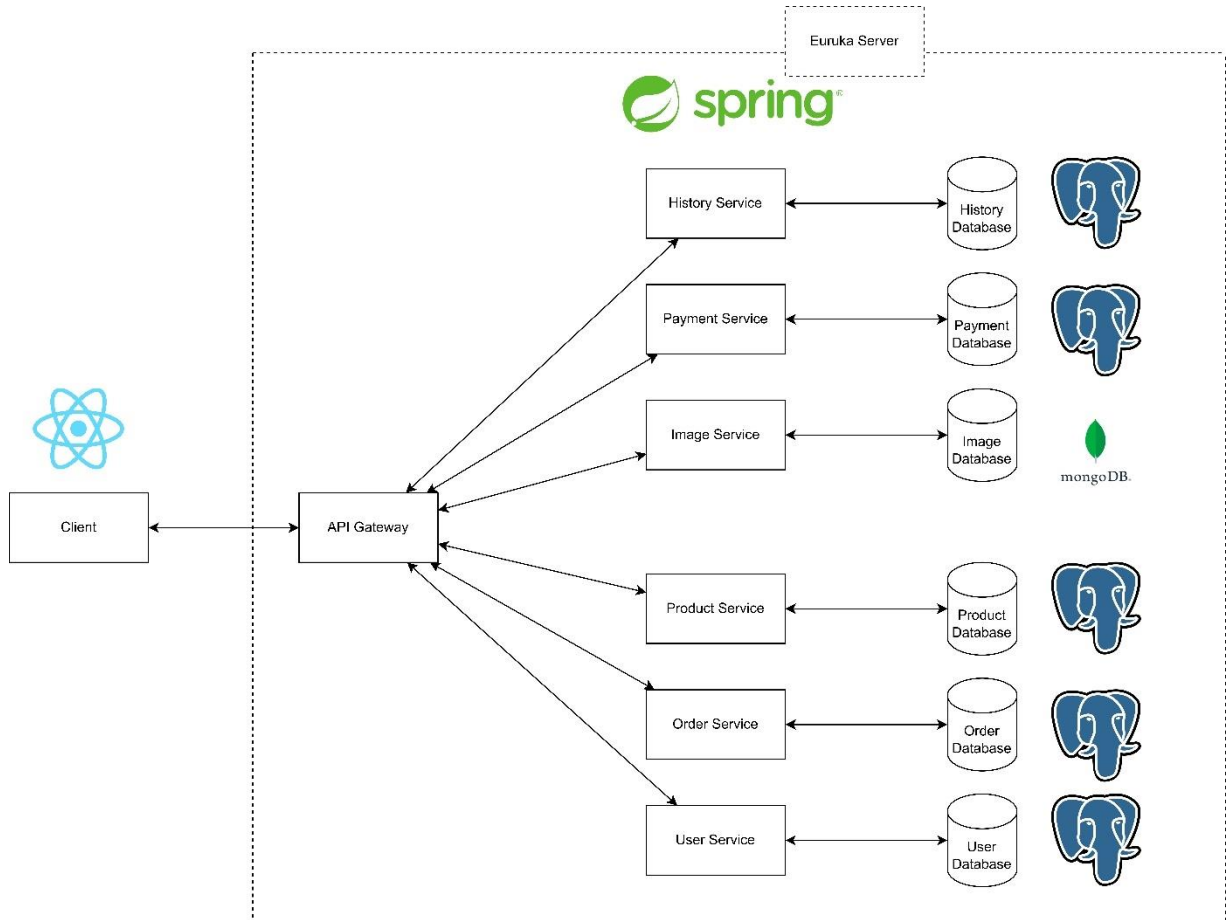
| | | |
|----|--------------------|---|
| 14 | User management | Chức năng cho phép admin quản lý thông tin người dùng |
|----|--------------------|---|

2.2 Đặc tả hệ thống

- Tiếp nhận các đơn hàng và lưu lại các đơn hàng đó
- Quản lý thông tin cơ bản và thông tin thẻ, tài khoản ngân hàng của người dùng.
- Thông kê doanh thu từ các đơn hàng.
- Theo dõi tình trạng, số lượng của các sản phẩm còn trong kho.

CHƯƠNG III – SƠ ĐỒ CẤU TRÚC

3.1 Sơ đồ hệ thống



Hình 3.1.1

Website thương mại điện tử được xây theo kiến trúc microservices sẽ được chia ra nhiều service độc lập nhau, mỗi service cũng sẽ có một database riêng. Website thương mại điện tử của nhóm em sẽ chia hệ thống thành sáu services là:

- Product Service: Quản lý sản phẩm, số lượng sản phẩm trong kho.
- Order Service: Quản lý đơn hàng, trạng thái đơn hàng.
- Image Service: Quản lý hình ảnh, được dung chủ yếu với sản phẩm.
- User Service: Quản lý thông tin khách hàng số lượng đơn hàng đã tạo.
- Payment Service: Quản lý thông tin thanh toán của đơn hàng.

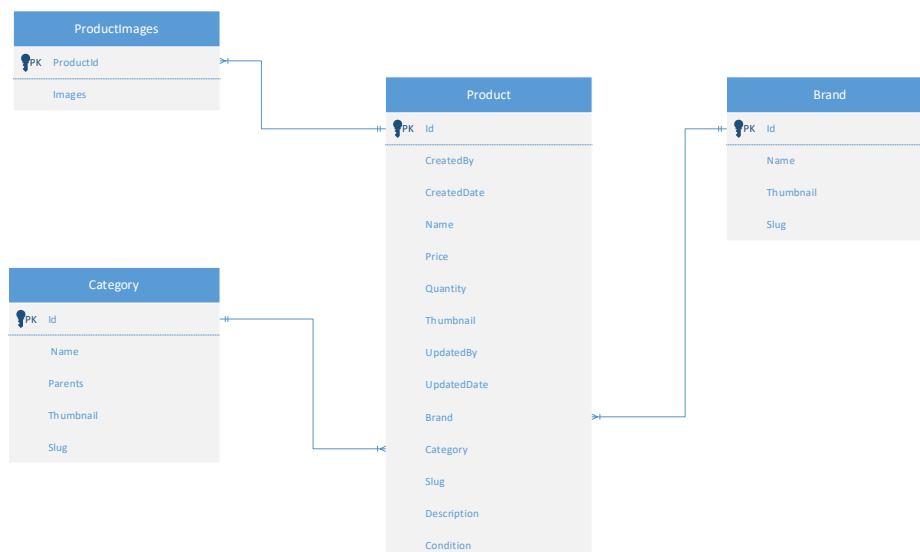
- History Service: Quản lý thông tin lịch sử sản phẩm, đánh giá của khách hàng.

Về phần hệ thống, nhóm em sẽ sử dụng Java Spring Boot để hiện thức các Service và database sẽ dùng PostgreSQL. Riêng Image Service sẽ được dùng MongoDB. Các Services sẽ liên lạc nhau qua thông qua RestAPIs để nhận thông tin từ Service khác và trả về kết quả. Trong mỗi Service sẽ được xây dựng theo kiến trúc Domain Driven Design(DDD)

Còn về phía Client, Client sẽ liên lạc với hệ thống mircoservices thông qua API gateway, đây là một cổng trung gian để đi vào hệ thống, API Gateway sẽ nhận requests từ Client để chỉnh sửa, xác thực và điều hướng chúng đến các APIs cụ thể vào các Services ở phía sau hệ thống. Về phía client sẽ được xây dựng bằng ReactJS, một library mạnh mẽ đang được đánh giá cao và được rất nhiều dự án đang dùng hiện tại, dành cho trang website bán hàng. Và Angular, một framework được xây dựng bởi Google, dành cho trang quản lý bán hàng. Việc chia hệ thống bán hàng ra thành 2 website khác nhau cũng như là một kiến trúc mircoservices cho front-end.

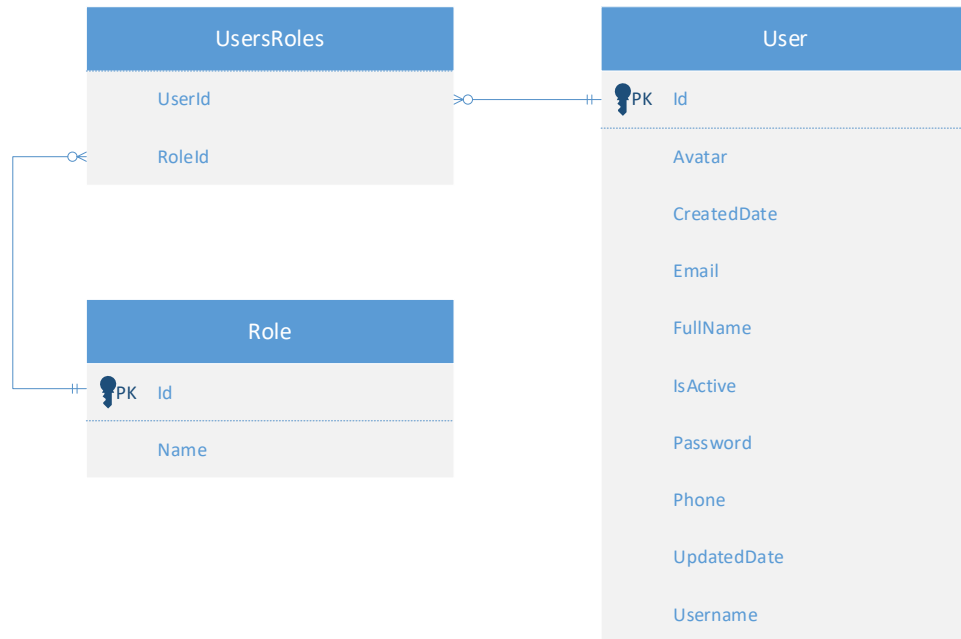
3.2 Cơ sở dữ liệu trong các Services

3.2.1 Product Service



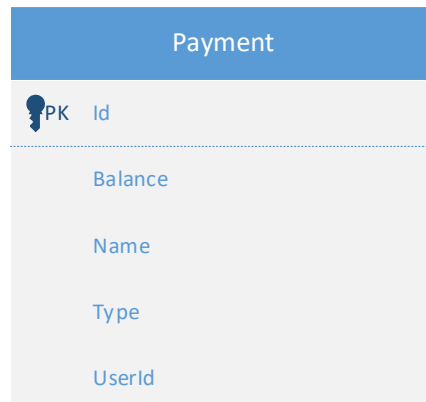
3.2.1 Sơ đồ cơ sở dữ liệu của Product Service

3.2.2. User Service



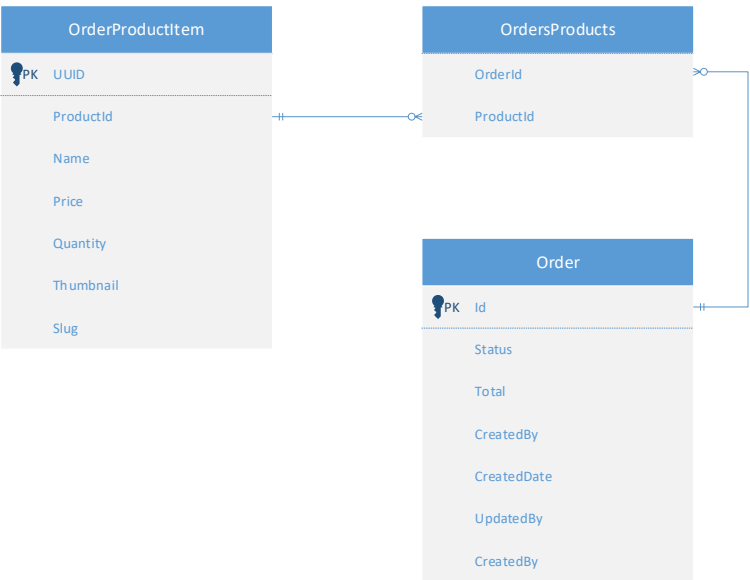
3.2.2 Sơ đồ cơ sở dữ liệu của User Service

3.2.3 Payment Service



3.2.3 Sơ đồ cơ sở dữ liệu của Payment Service

3.2.4 Order Service



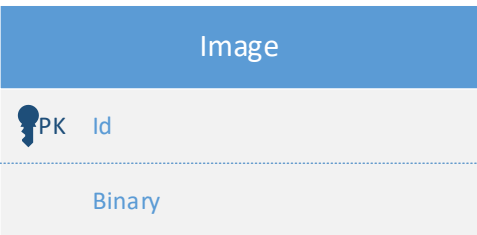
Hình 3.2.4 Sơ đồ cơ sở dữ liệu của Order Service

3.2.5 HistoryService



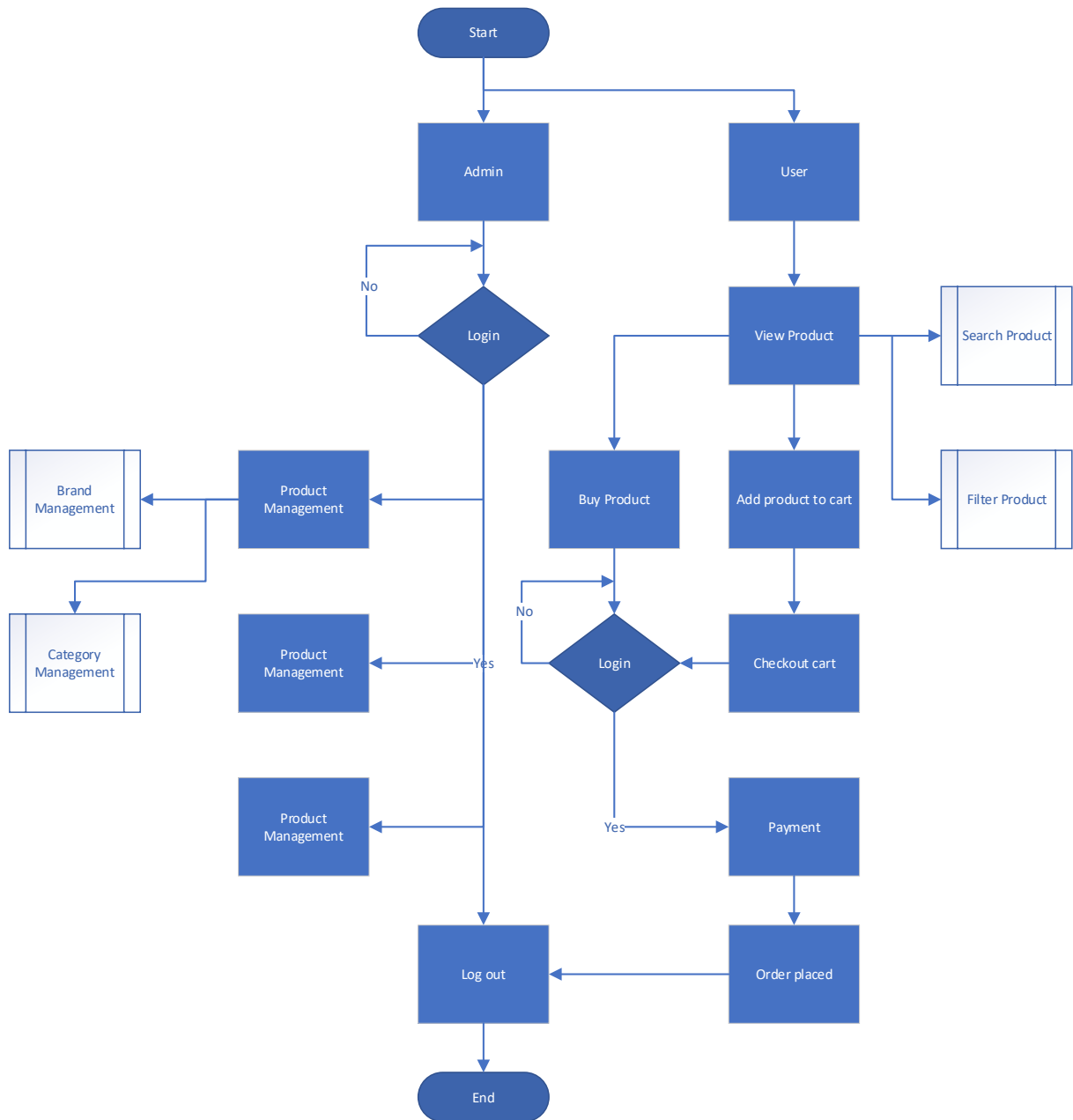
Hình 3.1.5 Sơ đồ dữ liệu của History Service

3.2.6 Image Service



Hình 3.2.6 Cơ sở dữ liệu của Image Service

3.3 Flowchart của hệ thống



Hình 3.3.1 Flowchart của hệ thống

CHƯƠNG IV – CƠ SỞ LÝ THUYẾT

4.1 Kiến trúc Mircoservice

4.1.1 Khái niệm

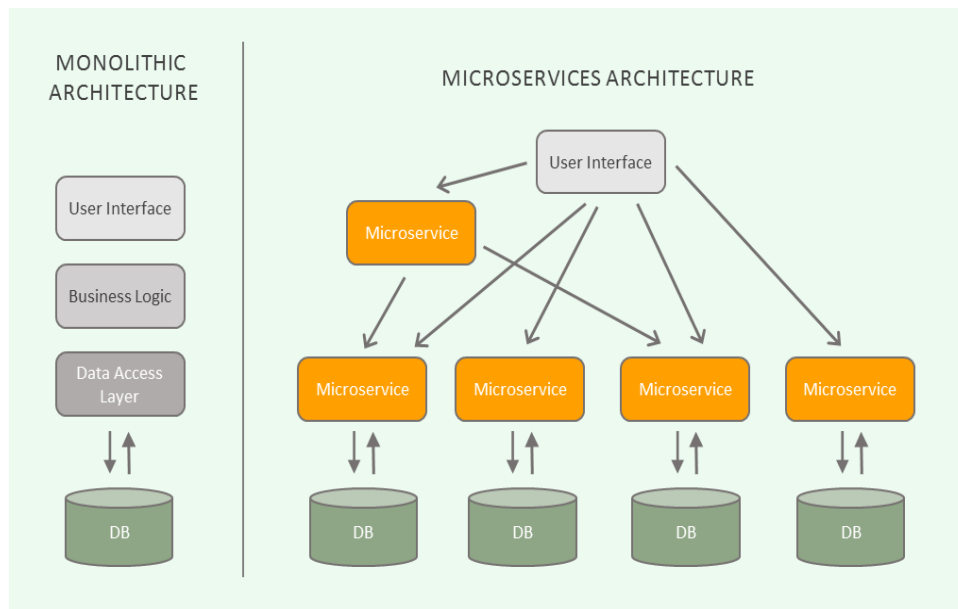
Microservices là một architectural pattern để xây dựng các hệ thống phần mềm bao gồm các services nhỏ, có thể triển khai độc lập. Mỗi service chịu trách nhiệm về một business cụ thể, chẳng hạn như quản lý người dùng, xử lý thanh toán hoặc lưu trữ hình ảnh.

Nhiều trang web và ứng dụng đã áp dụng kiến trúc mircoservices để cải thiện khả năng mở rộng và độ tin cậy. Một số ví dụ bao gồm:

- Netflix: Là một trong những công ty lớn nhất và nổi tiếng sử dụng mircoservices, Netflix sử dụng mircoservices để cung cấp service phát trực tuyến của mình cho hơn 200 triệu người dùng trên toàn cầu.
- Amazon: Amazon sử dụng các service siêu nhỏ để hỗ trợ các service bán lẻ và thương mại điện tử khác nhau của mình, bao gồm Amazon Prime, Amazon Fresh và Amazon Web Services.
- Uber: Uber sử dụng các mircoservices để cung cấp năng lượng cho ứng dụng gọi xe của mình, cho phép các phần khác nhau của ứng dụng, chẳng hạn như bản đồ, cơ sở dữ liệu tài xế và xử lý thanh toán, chạy dưới dạng các service riêng biệt.
- Airbnb: Airbnb sử dụng microservice để quản lý các phần khác nhau trên nền tảng của mình, bao gồm danh sách tài sản, hồ sơ người dùng và quản lý đặt phòng.
- SoundCloud: SoundCloud, nền tảng phân phối âm thanh trực tuyến, sử dụng các mircoservices để quản lý hệ thống tải lên, lưu trữ và phân phối âm thanh lớn và phức tạp của nó.

Đây chỉ là một vài ví dụ và nhiều trang web và ứng dụng khác cũng đang sử dụng microservice để cải thiện hệ thống của họ.

Mục tiêu của microservices là cải thiện khả năng mở rộng, độ tin cậy và tốc độ phát triển cho các hệ thống phần mềm phức tạp. Bằng cách chia nhỏ hệ thống thành các service nhỏ hơn, độc lập, các nhóm có thể cung cấp các tính năng mới nhanh hơn, mở rộng quy mô service khi cần, đồng thời cải thiện hiệu suất tổng thể và tính ổn định của hệ thống.



Hình 4.1.1.1. Hình minh họa kiến trúc mircoservice¹

4.1.2 Những đặc điểm của mircoservice

- Modularity: Mỗi service đều nhỏ và có một nhiệm vụ duy nhất, được xác định rõ ràng, giúp dễ hiểu, phát triển và bảo trì hơn.
- Independence: Các service có thể được triển khai, cập nhật và tạo phiên bản một cách độc lập, cho phép chu kỳ phát triển nhanh hơn và giảm thời gian ngừng hoạt động.

¹ [Microservices là gì và hiểu tường tận về Microservices \(levanphu.info\)](http://levanphu.info)

- Loose Coupling: Các service giao tiếp với nhau thông qua các cơ chế gọn nhẹ, thường sử dụng API, cho phép chúng phát triển và mở rộng quy mô một cách độc lập.
- Polyglot: Các service có thể được xây dựng bằng các ngôn ngữ lập trình, cơ sở dữ liệu và công cụ khác nhau, cho phép các nhóm chọn công nghệ tốt nhất cho nhu cầu của họ.
- Phi tập trung: Trách nhiệm ra quyết định và quản lý được phân cấp, cho phép các nhóm đưa ra quyết định và hành động nhanh chóng.

4.1.3 Ưu, nhược điểm của microservice

4.1.3.1 Ưu điểm

- Khả năng mở rộng: Các service có thể được mở rộng độc lập dựa trên các yêu cầu cụ thể.
- Khả năng phục hồi: Nếu một service bị lỗi, các service khác sẽ tiếp tục hoạt động.
- Agility: Các service có thể được cập nhật, triển khai và tạo phiên bản độc lập.
- Không cần đồng nhất về công nghệ: Các service khác nhau có thể được xây dựng bằng các ngôn ngữ lập trình, cơ sở dữ liệu và công cụ khác nhau.
- Phân tách: Mỗi service có một trách nhiệm cụ thể, được xác định rõ ràng, giúp ứng dụng dễ hiểu, phát triển và bảo trì hơn.

4.1.3.2 Nhược điểm

- Độ phức tạp: Microservices có thể gây ra sự phức tạp về mặt triển khai, thử nghiệm và quản lý.
- Giao tiếp giữa các service: Các service phải giao tiếp với nhau và có thể có những thách thức trong việc đảm bảo giao tiếp liên mạch giữa các services, đặc biệt là về hiệu suất, độ tin cậy và bảo mật.
- Tìm kiếm: Việc tìm kiếm service phù hợp để giao tiếp có thể là một thách thức, đặc biệt là trong một hệ thống lớn và phức tạp.

- Quản lý dữ liệu phân tán: Quản lý tính nhất quán của dữ liệu trên nhiều service có thể khó khăn, đặc biệt là trong các trường hợp cập nhật dữ liệu xảy ra trong nhiều service.
- Debug và giám sát: Gỡ lỗi và giám sát ứng dụng dựa trên microservice có thể phức tạp hơn so với trong kiến trúc nguyên khối, vì có nhiều bộ phận chuyển động hơn để giám sát và gỡ lỗi.
- Đầu tư ban đầu: Việc thiết lập kiến trúc microservices có thể yêu cầu đầu tư ban đầu đáng kể, bao gồm thời gian và tài nguyên để tái cấu trúc và triển khai ứng dụng.

4.2. Kiến trúc Domain Driven Design (DDD)

4.2.1 Khái niệm DDD

Domain Driven Design (DDD) là một phương pháp phát triển phần mềm tập trung vào việc lập mô hình và hiểu các vấn đề và khái niệm kinh doanh cốt lõi, sau đó xác định kiến trúc và thiết kế của hệ thống dựa trên các mô hình đó.

DDD nhằm mục đích cải thiện giao tiếp giữa các bên liên quan về kỹ thuật và kinh doanh, dẫn đến sự thể hiện chính xác và phù hợp hơn về lĩnh vực kinh doanh trong hệ thống phần mềm. Nó cũng thúc đẩy việc sử dụng các mẫu và thực tiễn hỗ trợ thiết kế các hệ thống có thể bảo trì, có thể mở rộng và có thể kiểm tra.

DDD thường được sử dụng cùng với các microservices để tạo ra các hệ thống linh hoạt, có thể mở rộng và có khả năng phát triển theo nhu cầu kinh doanh thay đổi. Các microservices có thể được liên kết với các bối cảnh có giới hạn trong miền, giúp dễ dàng quản lý các phần phụ thuộc giữa các services và cải thiện thiết kế hệ thống tổng thể.

4.2.2 Các layer có trong DDD

Domain-Driven Design (DDD) định nghĩa một kiến trúc phân lớp bao gồm các lớp sau:

- Domain Layer: Đây là cốt lõi của hệ thống và đại diện cho các khái niệm, entities và rules. Domain Layer chịu trách nhiệm xác định hành vi và logic nghiệp vụ.
- Application Layer: Layer này đóng vai trò là giao diện giữa domain layer và infrastructure layer. Nó thực hiện các trường hợp sử dụng dành riêng cho ứng dụng và giao tiếp với domain layer để thực hiện hành vi mong muốn.
- Infrastructure layer: Lớp này cung cấp cơ sở hạ tầng kỹ thuật cơ bản cho hệ thống, chẳng hạn như cơ sở dữ liệu, hệ thống nhắn tin và máy chủ web. Infrastructure layer giao tiếp với application layer để thực hiện các hoạt động kỹ thuật cần thiết.
- Presentation layer: Tầng này chịu trách nhiệm trình bày thông tin cho người dùng và nhận thông tin đầu vào từ người dùng. Nó giao tiếp với application layer để lấy thông tin cần thiết và trình bày cho người dùng.

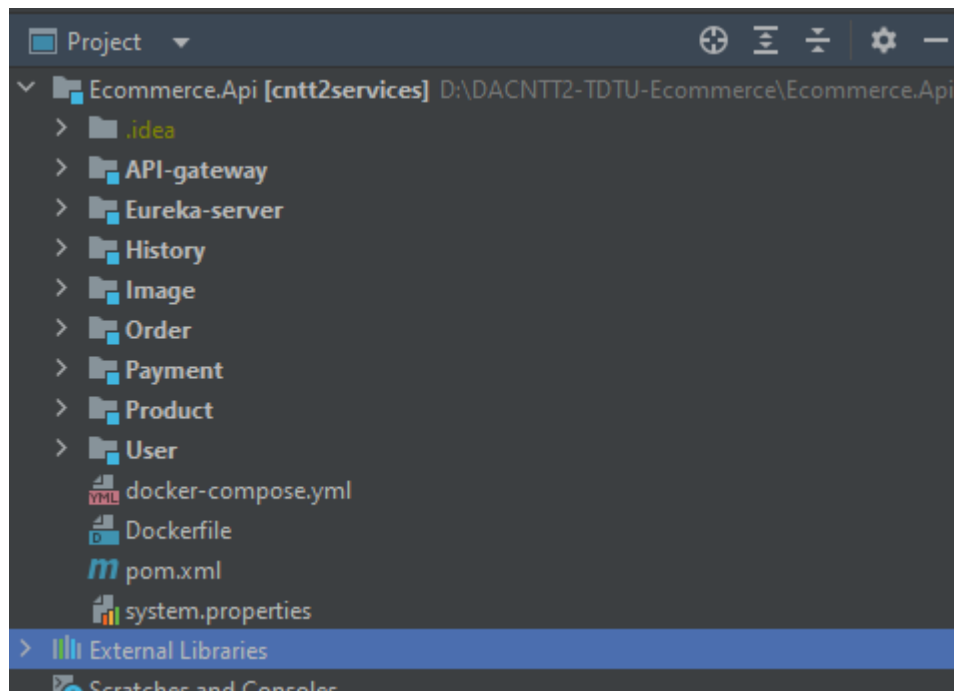
Các layer này được kết hợp lỏng lẻo và có thể được phát triển và triển khai độc lập, cho phép hệ thống có tính linh hoạt và khả năng mở rộng cao hơn. Domain layer là cốt lõi của hệ thống và các layer khác được thiết kế để hỗ trợ nó. Kiến trúc phân layer này giúp đảm bảo rằng hệ thống có thể bảo trì, có thể mở rộng và có thể kiểm tra cũng như logic nghiệp vụ được gói gọn tốt.

CHƯƠNG V – HIỆN THỰC HỆ THỐNG

5.1 Hiện thực back-end theo kiến trúc microservices bằng Java Spring Boot:

5.1.1 Giới thiệu chung:

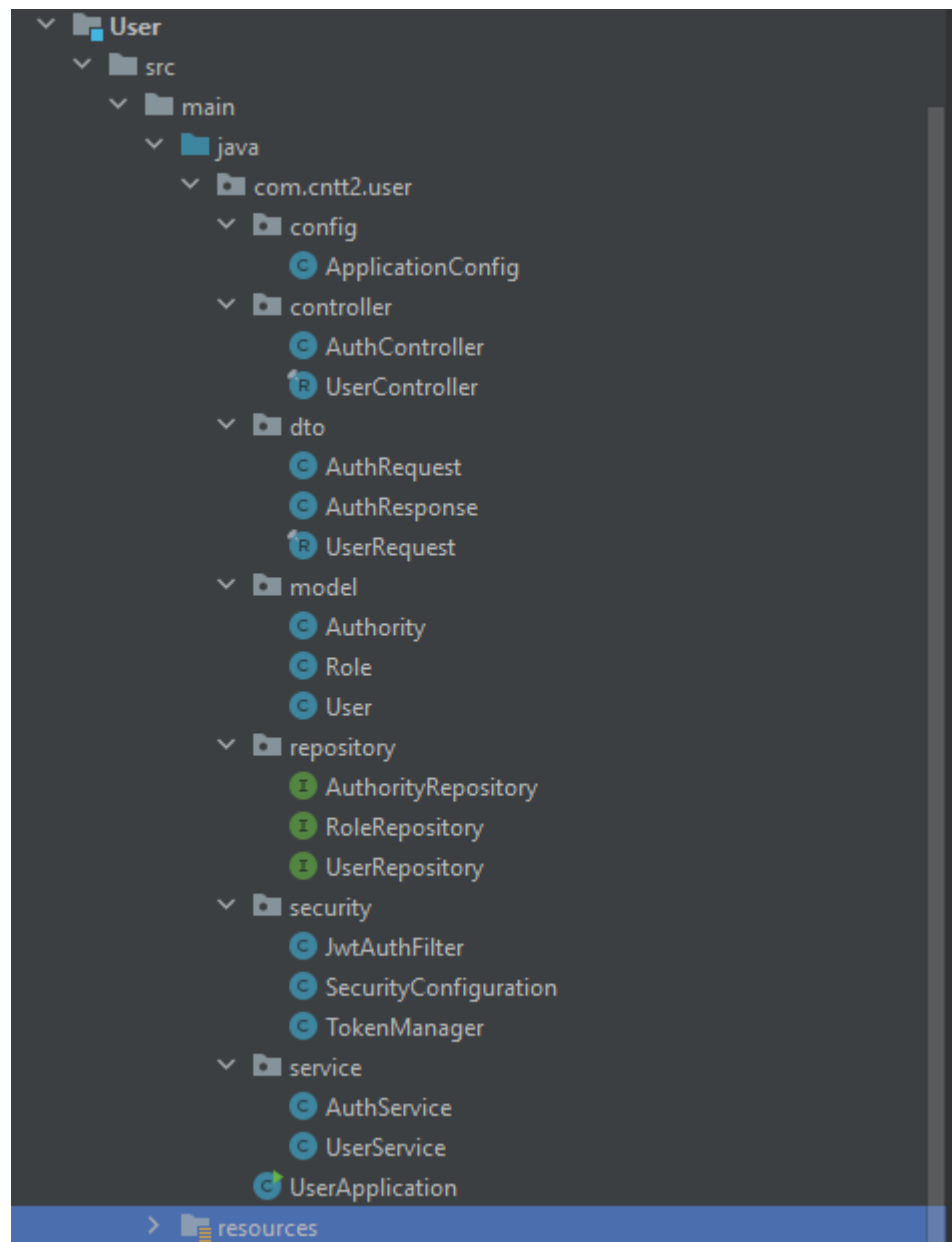
Như đã tóm tắt sơ qua ở phần Sơ đồ cấu trúc, kiến trúc của bọn em được chia thành 5 services gồm: Product Service, User Service, Order Service, Payment Service, History Service và Image Service. Nhóm em dùng API Gateway để điều hướng các request từ client với các services. Ngoài ra nhóm em dùng Eureka để thống nhất các services với nhau.



Hình 5.1.1.1 Kiến trúc của toàn bộ dự án

5.1.2 User Service

User Service là phần service quan trọng nhất trong hệ thống này, service được chia làm hai phần gồm User và Auth. Cấu trúc tổ chức trong User Service sẽ như sau:



Hình 5.1.2.1 Cấu trúc tổ chức User Service

Service này được tổ chức theo cấu trúc DDD với các layer như sau:

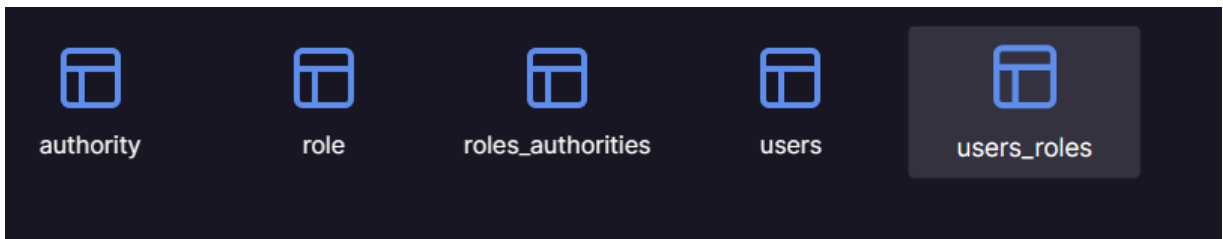
- Application layer: Controllers.
- Domain layer: Models, DTOs, Repositories, Services
- Infrastructure layer: Config

Sau khi chia cấu trúc của UserService như trên thì nhóm em bắt đầu implement các chức năng của service này.

5.1.2.1 Model

Trong UserService có năm models khác nhau, chi tiết mỗi model:

- Authority: model đệm để dùng cho việc xác thực.
- Role: chứa các role của user trong hệ thống
- Roles_authorities: model đệm để xác thực role.
- Users: chứa thông tin users
- User_roles: chứa roles của users



5.1.2.2 Security

Trong service này có layer là security, layer này là 1 middleware để nhận các request từ client và các services khác. Nó sẽ lấy token được gắn trong request header để rồi dùng để xác thực thông qua API authentication mà ta sẽ nói ở bên dưới. Trong layer này còn có file TokenManager dùng để setting cách generate token cũng như phương thức giải mã token (hình 5.1.2.2).

```

@Component
public class TokenManager implements Serializable {

    private static final long serialVersionUID = 7008375124389347049L;
    1 usage
    public static final long TOKEN_VALIDITY = 24 * 60 * 60 * 30;
    3 usages
    @Value("${secret}")
    private String jwtSecret;

    2 usages ButMinhNhat
    public String generateJwtToken(String userId) {
        Map<String, Object> claims = new HashMap<>();
        return Jwts.builder().setClaims(claims).setSubject(userId)
            .setIssuedAt(new Date(System.currentTimeMillis()))
            .setExpiration(new Date(System.currentTimeMillis() + TOKEN_VALIDITY * 1000 * 14))
            .signWith(SignatureAlgorithm.HS512, jwtSecret).compact();
    }

    1 usage ButMinhNhat
    public Boolean validateJwtToken(String token) {
        Claims claims = Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(token).getBody();
        Boolean isTokenExpired = claims.getExpiration().before(new Date());
        return !isTokenExpired;
    }

    1 usage ButMinhNhat
    public String getUserIDFromToken(String token) {
        final Claims claims = Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(token).getBody();
        return claims.getSubject();
    }
}

```

Hình 5.1.2.2 TokenManager

Trong file JWTAuthFilter sẽ hiện thực các để layer này có thể lấy được token từ request. Ở đây ta thấy được token sẽ được lấy từ header request có header là “Authorization” và token được bắt đầu từ từ khoá “Bearer”. Sau đó sẽ đi tới API authenticate để kiểm tra token mà ta sẽ được tìm hiểu bên dưới. Nếu API trên trả về được thông tin thì request đó sẽ được tiếp tục truyền xuống controller. Còn nếu API trả về kết quả thất bại hoặc không có dữ liệu thì request sẽ bị chặn và không tiếp tục đi tiếp nữa (hình 5.1.2.3, 5.1.2.4)

```

@Override
protected void doFilterInternal(
    HttpServletRequest request,
    HttpServletResponse response,
    FilterChain filterChain
) throws ServletException, IOException {
    final String authHeader = request.getHeader("Authorization");

    if(authHeader == null || !authHeader.startsWith("Bearer ")) {
        filterChain.doFilter(request, response);
        return;
    }

    if(SecurityContextHolder.getContext().getAuthentication() == null) {
        UserInfo userData = isAuthTokenValid(authHeader);
        UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(
            userData, null, userData.getAuthorities());
        authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
        SecurityContextHolder.getContext().setAuthentication(authToken);

        //set userId to header
        request.setAttribute("userId", userData.getId());
    }
    filterChain.doFilter(request, response);
}

```

Hình 5.1.2.3 Phương thức lấy token từ request

```

private UserInfo isAuthTokenValid(String token){
    UserInfo userData = null;
    try {
        userData = WebClientBuilder.build().get() RequestHeadersUriSpec<capture of ?>
            .uri(uri: "http://user/api/v1/auth/authenticate?token="+token) capture of ?
            .retrieve() ResponseSpec
            .bodyToMono(UserInfo.class) Mono<UserInfo>
            .block();
    }
    catch(HttpClientErrorException ex){
        if (ex.getStatusCode()== HttpStatus.UNAUTHORIZED) {
            return null;
        }
        throw ex;
    }
    return userData;
}

```

Hình 5.1.2.4 Phương thức để kiểm tra token bằng API authenticate

Trong file `SecurityConfiguration` sẽ config các setting để chạy security middleware đặt biệt là quy định các APIs nào trong service sẽ được bỏ qua bước kiểm tra token này. Trong đây các APIs thuộc auth được bỏ qua bước xác thực này bởi các APIs này chính là các phương thức tạo ra token cũng như xác thực token. Chi tiết sẽ được trình bày ở phía dưới (hình 5.1.2.5)

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http = http.csrf().disable();

    http = http.sessionManagement()
        .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
        .and();

    http = http.exceptionHandling().authenticationEntryPoint((req, rsp, e) -> rsp.sendError(HttpServletResponse.SC_UNAUTHORIZED)).and();

    http = http.addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class);

    http.authorizeRequests()
        .antMatchers( ...antPatterns: "/api/v1/auth/*").permitAll()
        .anyRequest().authenticated();

    http.cors();
}
```

Hình 5.1.2.5 `SecurityConfiguration`

5.1.2.3 Controller (Routes)

Trong `UserService` có 2 Controllers (Routes), đó là `AuthController` và `UserController`. Hai controller này được route như sau:

- `AuthController`: `http://localhost:8081/api/v1/auth`
- `UserController`: `http://localhost:8081/api/v1/user`

a) `AuthController`

APIs:

- POST: `SignIn`
- POST: `SignUp`
- GET: `Authenticate`

Giải thích các thức hoạt động của các APIs:

- `SignIn`: Ta bắt đầu từ request từ client, client sẽ request đến API này và API sẽ nhận request theo model `SignInRequest` (hình 5.1.2.6). Sau đó, request sẽ được

AuthController tiếp nhận và đẩy qua tiếp AuthService (hình 5.1.2.7). Ở trong AuthService, thông tin gồm username và password được dùng để AuthRepository đi xuống dưới hệ thống dữ liệu để kiểm tra xem có tồn tại dữ liệu hay không, nếu AuthService sẽ generate ra token và trả về thông tin cho request gồm thông tin của User đó kèm theo token đã được generate. Trường hợp không có dữ liệu dưới cơ sở dữ liệu. AuthService sẽ trả về status code 401: Unauthorized (hình 5.1.2.8)

```
6 usages ButMinhNhat
public class AuthRequest {
    2 usages ButMinhNhat
    public record SignInRequest(
        2 usages
        String username,
        2 usages
        String password
    ) {
    }
}
```

Hình 5.1.2.6 Model của SignInRequest

```
@PostMapping(path = "signin")
public ResponseEntity<AuthResponse> signIn(@RequestBody AuthRequest.SignInRequest signInRequestRequest) {
    return authService.signIn(signInRequestRequest);
}
```

Hình 5.1.2.7 SignIn trong AuthController

```

1 usage  📌 Võ Tấn Phát +1
public ResponseEntity<AuthResponse> signIn(AuthRequest.SignInRequest request) {
    if(request.username().isEmpty() || request.password().isEmpty()) {
        return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
    }
    //get user
    Optional<User> userData = userRepository.findByUsername(request.username());

    if(userData.isEmpty()) {
        return new ResponseEntity<>(HttpStatus.UNAUTHORIZED);
    }
    if(!passwordEncoder.matches(request.password(), userData.get().getPassword())) {
        return new ResponseEntity<>(HttpStatus.UNAUTHORIZED);
    }

    //generate token
    final String jwtToken = tokenManager.generateJwtToken(userData.get().getId());

    //generate data response
    AuthResponse response = new AuthResponse(userData.get());
    response.setToken(jwtToken);

    return new ResponseEntity<AuthResponse>(response, HttpStatus.OK);
}

```

Hình 5.1.2.8 Xử lý logic trong AuthService

- SignUp: Tương tự như flow trên thì ở signUp API, API cũng có cùng luồng đi với signin API nhưng khác là API này sẽ thêm tạo ra thêm user mới dựa trên thông tin được truyền vào từ request, controller nhận được dữ liệu và truyền tiếp tới authService. Việc tạo user mới được diễn ra ở AuthService và thêm vào thông qua UserRepository. Sau đó API này sẽ trả về thông tin user được tạo và token. (hình 5.1.2.10, 5.1.2.11)

```

@PostMapping(path = "signup")
public ResponseEntity<AuthResponse> signUp(@RequestBody AuthRequest.SignUpRequest signUpRequest) {
    return authService.signUp(signUpRequest);
}

```

Hình 5.1.2.10 Controller truyền params xuống AuthService và nhận lại result từ đó, sau đó trả về client

```

1 usage  ButMinhNhat +1
public ResponseEntity<AuthResponse> signUp(AuthRequest.SignUpRequest request) {
    List<Role> userRoles = setRoles(Arrays.asList("USER"));

    User user = User.builder()
        .username(request.username())
        .password(passwordEncoder.encode(request.password()))
        .fullname(request.fullname())
        .email(request.email())
        .phone(request.phone())
        .roles(userRoles)
        .build();

    //save data in db
    User saveUser = userRepository.save(user);
    System.out.println(saveUser);

    //generate token
    final String jwtToken = tokenManager.generateJwtToken(user.getId());

    //generate data response
    AuthResponse response = new AuthResponse(user);
    response.setToken(jwtToken);

    return new ResponseEntity<AuthResponse>(response, HttpStatus.OK);
}

```

Hình 5.1.2.11 AuthService kiểm tra và chạy UserRepository để tạo user mới và generate token

- Authenticate: Khác với 2 APIs phía trên dùng để generate token thì API này dùng để xác thực token đó với các request. API sẽ lấy token trong các request từ phía client cũng như các services. API này cũng có luồng như các API khác khi sẽ nhận được request từ Controller. Controller sẽ truyền các params xuống AuthService để thực hiện logic. Ở đây AuthService sẽ giải token ra lại userId, sau đó dùng param này để truyền vào UserRepository để tìm dưới cơ sở dữ liệu có dữ liệu đó hay không. Nếu có thì sẽ trả về thông tin của User đó, nếu không thì AuthService sẽ trả về null cho Controller. Controller nếu thấy giá trị là null thì sẽ return về status code 400: Bad Request về phía client. Nếu có giá trị, Controller

sẽ trả về thông tin user nhận được từ AuthService cho client. (hình 5.1.2.12, hình 5.1.2.13).

```
ButMinhNhat
@GetMapping(value = {"authenticate"})
public ResponseEntity<UserDetails> authenticate(@RequestParam(name = "token", required = true) String tokenHeader) throws Exception {
    UserDetails userData = authService.checkAuth(tokenHeader);

    if(userData != null) {
        return new ResponseEntity<UserDetails>(userData, HttpStatus.OK);
    }

    return new ResponseEntity<UserDetails>(HttpStatus.BAD_REQUEST);
}
```

Hình 5.1.2.12 Controller nhận request và truyền params xuống AuthService, sau đó trả về kết quả cho client nếu AuthService trả về dữ liệu, nếu không thì trả về Bad Request cho client

```
public UserDetails checkAuth(String tokenHeader) {
    String userId = null;
    String token = null;

    if (tokenHeader != null && tokenHeader.startsWith("Bearer ")) {
        token = tokenHeader.substring(7);
        try {
            userId = tokenManager.getUserIDFromToken(token);
        } catch (IllegalArgumentException e) {
            throw new IllegalStateException("Unable to get JWT Token");
        } catch (ExpiredJwtException e) {
            throw new IllegalStateException("JWT Token has expired");
        }
    } else {
        throw new IllegalStateException("Bearer String not found in token");
    }

    if (null != userId) {
        User userData = userRepository.findById(userId).orElseThrow(
            () -> new IllegalStateException("UserID not found!")
        );
        if (tokenManager.validateJwtToken(token) && userData != null) {
            return userData;
        }
    }

    return null;
}
```

Hình 5.1.2.13 AuthService giải mã token thành userId và kiểm tra user có tồn tại hay không thông qua userRepository



Hình 5.1.2.14 Flow trong Auth APIs

b) UserController

APIs:

- GET: GetUsers
- GET: GetUser
- PUT: UpdateUser
- DELETE: DeleteUser

Flow từ lúc API được thực hiện cho đến khi nhận được kết quả là User cũng tương tự như Auth, nhưng điểm khác biệt ở đây thì các APIs thuộc User phải đi qua middleware security. Như đã giải thích ở trên, middleware này sẽ lấy tìm trong request đến các APIs này để lấy ra header “Authorization” để kiểm tra token thông qua phương thức checkAuth ở AuthService. Nếu token không có hoặc không xác thực được. Request sẽ không được tiếp tục và trả về Bad Request. Ngược lại, request sẽ được đi đến Controller. Từ đây, yêu cầu của request sẽ được UserController tiếp tục điều hướng đến layer UserService, UserService sẽ tương tác với data ở database thông qua UserRepository, để có thể thực hiện yêu cầu và trả về kết quả mong muốn (hình 5.1.2.15, 5.1.2.16, 5.1.2.17)

```

@RequestMapping("/api/v1/user")
public record UserController(UserService userService) {

    //get all users
    ▲ ButMinhNhat +1
    @GetMapping
    public ResponseEntity<List<User>> getUsers() { return userService.getUsers(); }

    //get single user
    ▲ ButMinhNhat +1
    @GetMapping(path = "{userId}")
    public ResponseEntity<User> getSingleUser(@PathVariable("userId") String id) {
        return userService.getSingleUser(id);
    }

    //update user
    ▲ ButMinhNhat +1
    @PutMapping(path = "{userId}")
    public ResponseEntity<User> updateUser(@PathVariable("userId") String id, @RequestBody UserRequest userRequest) {
        return userService.updateUser(id, userRequest);
    }

    //delete user
    ▲ ButMinhNhat +1
    @DeleteMapping(path = "{userId}")
    public ResponseEntity deleteUser(@PathVariable("userId") String id) { return userService.deleteUser(id); }
}

```

Hình 5.1.2.15 Các APIs trong User có trong controller

```

1 usage ▲ Vô Tấn Phát +1
public ResponseEntity<List<User>> getUsers() { return ResponseEntity.ok(userRepository.findAll()); }

1 usage ▲ Vô Tấn Phát
public ResponseEntity<User> getSingleUser(String userId) {
    Optional<User> userData = userRepository.findById(userId);
    if (userData.isEmpty()) {
        return new ResponseEntity<> (HttpStatus.NOT_FOUND);
    }
    return new ResponseEntity<User> (userData.get(), HttpStatus.OK);
}

1 usage ▲ Vô Tấn Phát +1
public ResponseEntity<User> updateUser(String userId, UserRequest request) {
    Optional<User> userData = userRepository.findById(userId);
    if (userData.isEmpty()) {
        return new ResponseEntity<> (HttpStatus.NOT_FOUND);
    }

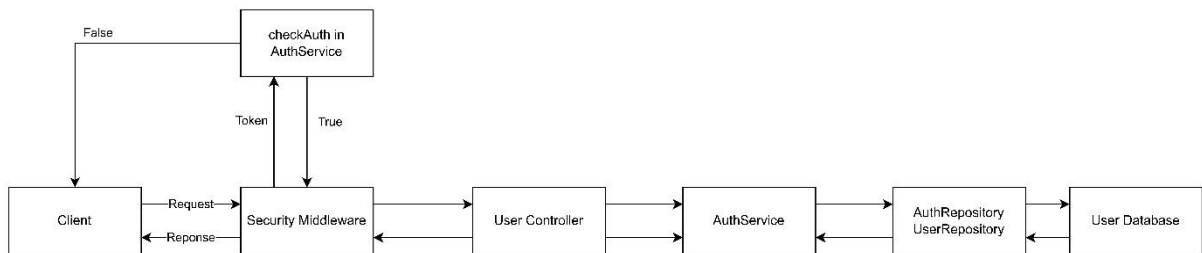
    if (request.username() != null)
        userData.get().setUsername(request.username());
    if (request.password() != null)
        userData.get().setPassword(request.password());
    if (request.fullname() != null)
        userData.get().setFullname(request.fullname());
    if (request.avatar() != null)
        userData.get().setAvatar(request.avatar());

    return new ResponseEntity<User> (
        userRepository.save(userData.get()),
        HttpStatus.OK );
}

1 usage ▲ Vô Tấn Phát +1
public ResponseEntity deleteUser(String userId) {
    Optional<User> userData = userRepository.findById(userId);
    if (userData.isEmpty()) {
        return new ResponseEntity<> (HttpStatus.NOT_FOUND);
    }
}

```

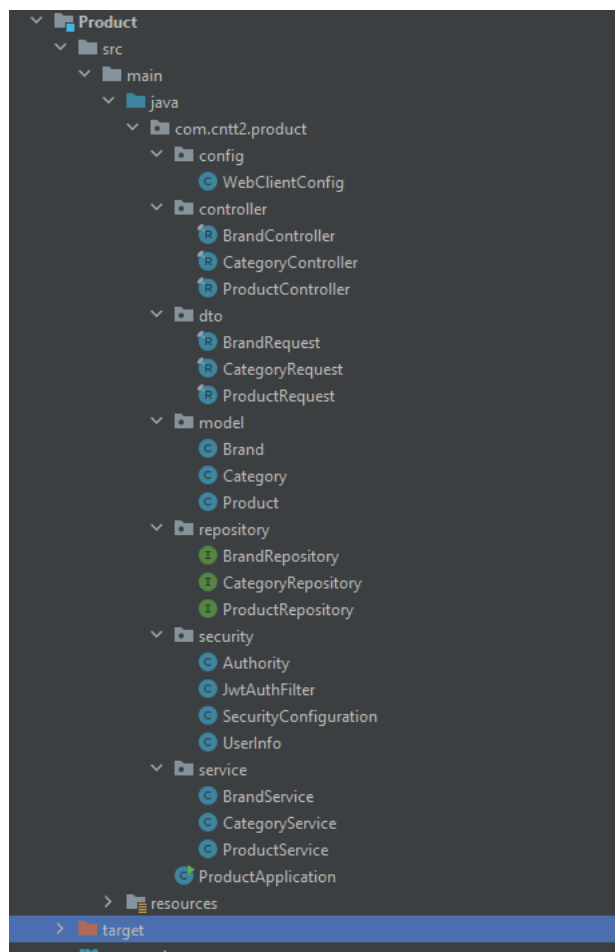
Hình 5.1.2.16 Các phương thức có trong UserService để thực hiện các logic để trả về kết quả thông qua UserRepository



Hình 5.1.2.17 Flow của các request đến User

5.1.3 Product Service

Tương tự như UserService, Product Service được tổ chức theo DDD và được chia thành 3 phần đó là: Product, Category và Brand (hình 5.1.3.1).

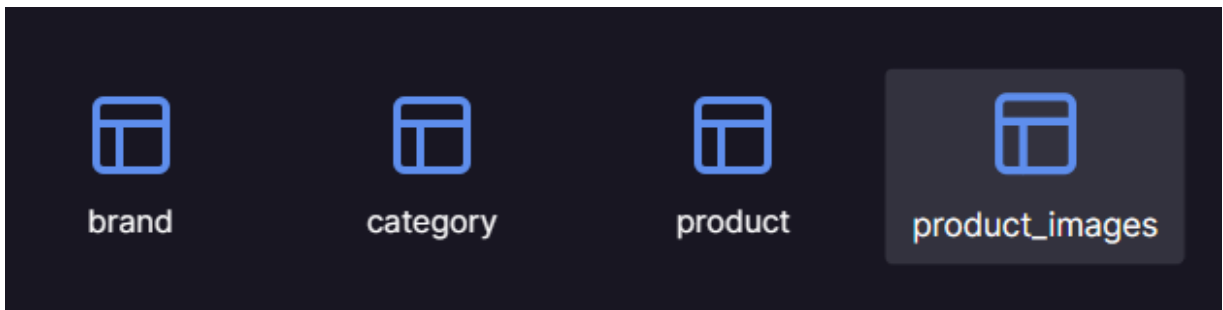


Hình 5.1.3.1 Cấu trúc tổ chức của Product Service

5.1.3.1 Model

Trong ProductService có bốn model, chi tiết các models:

- Product: chứa thông tin của products
- Category: chứa thông tin của categories
- Brand: chứa thông tin của brands
- Product_images: chứa thông tin sản phẩm có các hình nào



Hình 5.1.3.2 Các models trong ProductService

5.1.3.2 Security

Tương tự như UserService, ProductService cũng có security middleware. ProductService cũng dùng JWTAuthFilter để có thể lấy ra token từ request header trong token thông qua class AuthService có trong chính nó, thì ở ProductService, security middleware này phải chuyển request thông qua API Gateway để đi đến API authenticate ở UserService (hình 5.1.3.3). Request sẽ được xác thực ở tại API này và trả về lại kết quả. Security middleware ở ProductService sẽ tiếp nhận được response từ API authenticate của UserService. Từ đây cũng như logic cũ ở UserService, nếu token được xác thực thành công thì request sẽ được đi đến Controllers, nếu không thì request sẽ bị chặn lại (hình 5.1.3.4).

Ngoài ra các APIs get trong ProductService không cần xác thực token, rule này được config trong SecurityConfiguration (hình 5.1.3.5).

```

public class SecurityConfiguration extends WebSecurityConfigurerAdapter {
    1 usage
    @Autowired
    private JwtAuthFilter jwtAuthFilter;

    2 ButMinhNhat +2
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http = http.csrf().disable();

        http = http.sessionManagement()
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            .and();

        http = http.exceptionHandling().authenticationEntryPoint((req, rsp, e) -> rsp.sendError(HttpServletResponse.SC_UNAUTHORIZED)).and();

        http = http.addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class);

        http.authorizeRequests()
            .antMatchers(HttpMethod.GET, ...antPatterns: "/api/v1/product/**", "/api/v1/category/**", "/api/v1/brand/**").permitAll()
            .anyRequest().authenticated();

        http.cors();
    }
}

```

```

1 usage 2 ButMinhNhat
private UserInfo isAuthTokenValid(String token){
    UserInfo userData = null;
    try {
        userData = webClientBuilder.build().get() RequestHeadersUriSpec<capture of ?>
            .uri( uri: "http://user/api/v1/auth/authenticate?token="+token) capture of ?
            .retrieve() ResponseSpec
            .bodyToMono(UserInfo.class) Mono<UserInfo>
            .block();
    }
    catch(HttpClientErrorException ex){
        if (ex.getStatusCode()== HttpStatus.UNAUTHORIZED) {
            return null;
        }
        throw ex;
    }
    return userData;
}
}

```

Hình 5.1.3.3 Middleware điều hướng đến UserService để validate token

```

protected void doFilterInternal(
    HttpServletRequest request,
    HttpServletResponse response,
    FilterChain filterChain
) throws ServletException, IOException {
    final String authHeader = request.getHeader("Authorization");

    if(authHeader == null || !authHeader.startsWith("Bearer")) {
        filterChain.doFilter(request, response);
        return;
    }

    if(SecurityContextHolder.getContext().getAuthentication() == null) {
        UserInfo userData = isAuthTokenValid(authHeader);
        UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(
            userData, credentials: null, userData.getAuthorities());
        authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
        SecurityContextHolder.getContext().setAuthentication(authToken);

        //set userId to header
        request.setAttribute("userId", userData.getId());
    }
    filterChain.doFilter(request, response);
}

1 usage  ButMinhNhat
private UserInfo isAuthTokenValid(String token){
    UserInfo userData = null;
    try {
        userData = webClientBuilder.build().get() RequestHeadersUriSpec<capture of ?>
            .uri( uri: "http://user/api/v1/auth/authenticate?token="+token) capture of ?
            .retrieve() ResponseSpec
            .bodyToMono(UserInfo.class) Mono<UserInfo>
            .block();
    }
    catch(HttpClientErrorException ex){
        if (ex.getStatusCode() == HttpStatus.UNAUTHORIZED) {
            return null;
        }
        throw ex;
    }
}

```

Hình 5.1.3.4 Security middleware

```

public class SecurityConfiguration extends WebSecurityConfigurerAdapter {
    1 usage
    @Autowired
    private JwtAuthFilter jwtAuthFilter;

    1 ButMinhNhat +2
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http = http.csrf().disable();

        http = http.sessionManagement()
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            .and();

        http = http.exceptionHandling().authenticationEntryPoint((req, rsp, e) -> rsp.sendError(HttpServletResponse.SC_UNAUTHORIZED)).and();

        http = http.addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class);

        http.authorizeRequests()
            .antMatchers(HttpMethod.GET, ...antPatterns: "/api/v1/product/**", "/api/v1/category/**", "/api/v1/brand/**").permitAll()
            .anyRequest().authenticated();

        http.cors();
    }
}

```

Hình 5.1.3.5 Security Configuration

5.1.3.3 Controller (Routes)

Trong Product có ba Controllers (Routes), đó là ProductController, CategoryController và BrandController. Ba controller này được route như sau:

- ProductController: http://localhost:8082/api/v1/product
- CategoryController: http://localhost:8082/api/v1/category
- BrandController: http://localhost:8082/api/v1/brand

a) ProductController

APIs:

- GET: GetProducts
- GET: GetProduct
- POST: CreateProduct
- PUT: UpdateProduct
- DELETE: DeleteProduct

Flow thực hiện 1 request của Product cũng không có gì khác biệt với User. Request sẽ phải đi qua Security middleware để xác thực token rồi mới đi tiếp hoặc bị chặn lại. Riêng các request của phương thức get thì sẽ không cần xác thực

token. Sau đó flow của request cũng đi từ ProductController để điều hướng sau đó đến ProductService để xử lý logic và tương tác với database thông qua ProductRepository. Sau đó kết quả sẽ được trả lại theo chiều ngược lại. Chi tiết tham khảo source code ².

b) CategoryController

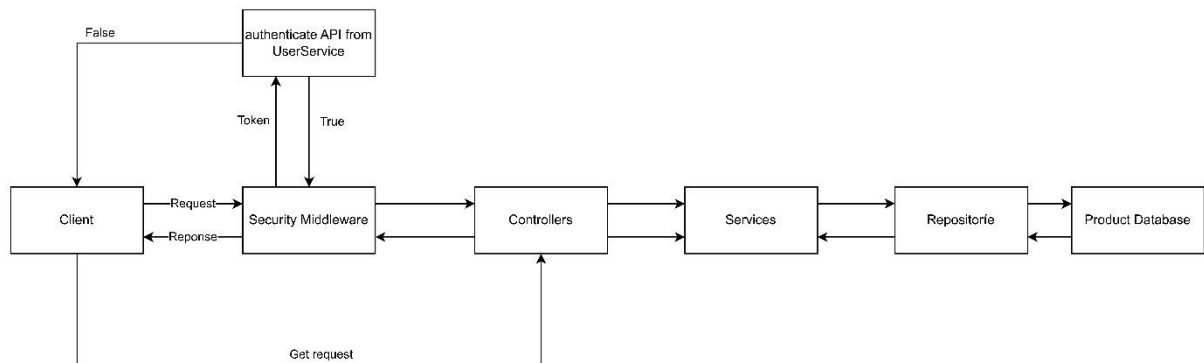
Tương tự như ProductController, CategoryController cũng có 5 APIs để thực hiện CRUD cho model Category và flow cũng tương tự như vậy.

c) BrandController

Tương tự như ProductController và CategoryController, BrandController cũng có 5 APIs để thực hiện CRUD cho model Category và flow cũng tương tự như vậy.

5.1.3.4 Kết luận

ProductService có 3 Controller đảm nhiệm quản lý thông tin cho Product, Category và Brand. Các request trong ProductService đều đi theo flow dưới đây:

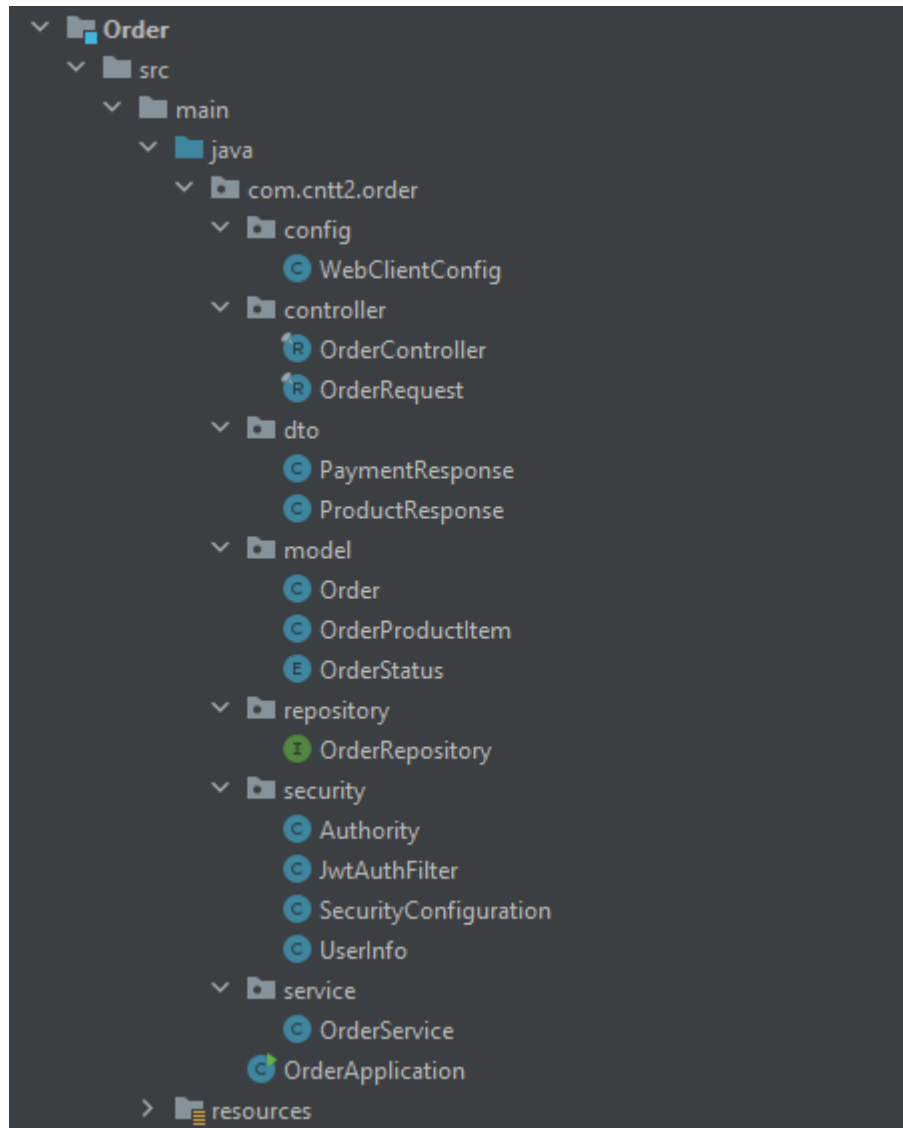


Hình 5.1.3.6 Flow của các request trong ProductService

² <https://github.com/recca5p/DACNTT2-TDTU-Ecommerce/tree/master/Ecommerce.Api/Product/src/main/java/com/cntt2/product>

5.1.4 Order Service

Order Service cũng được tổ chức theo kiến thức DDD và có 1 Security Middleware tương tự như ProductService và UserService:



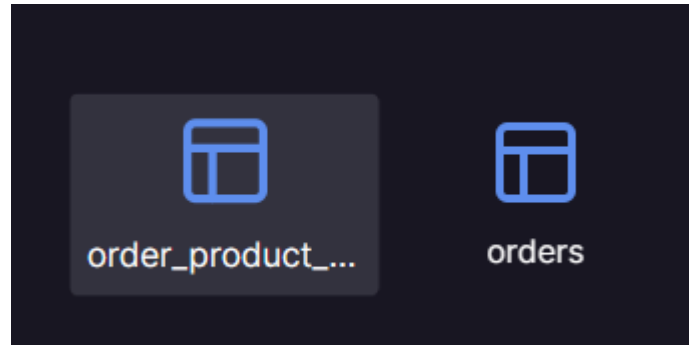
Hình 5.1.4.1 Tổ chức của OrderService

5.1.4.1 Model

OrderService có ba models, chi tiết các model:

- Order: chứa thông tin về các Orders đang có trong hệ thống

- Order_product_item: chứa danh sách sản phẩm có trong Order
- Order_status: các trạng thái sản phẩm (được hardcode)



Hình 5.1.4.2 Các models có trong OrderService

5.1.4.2 Security

Tương tự như ProductService, OrderService cũng có 1 middleware để lấy token trong header và truyền đến API authenticate của UserService. Điều khác biệt ở đây ở OrderService so với ProductService đó là tất cả APIs của OrderService đều cần có token (hình 5.1.4.3)

```
package com.cntt2.order.security;

import ...

@ButMinhNhat
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {
    1 usage
    @Autowired
    private JwtAuthFilter jwtAuthFilter;

    @ButMinhNhat
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http = http.csrf().disable();

        http = http.sessionManagement()
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            .and();

        http = http.exceptionHandling().authenticationEntryPoint((req, rsp, e) -> rsp.sendError(HttpServletResponse.SC_UNAUTHORIZED)).and();

        http = http.addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class);

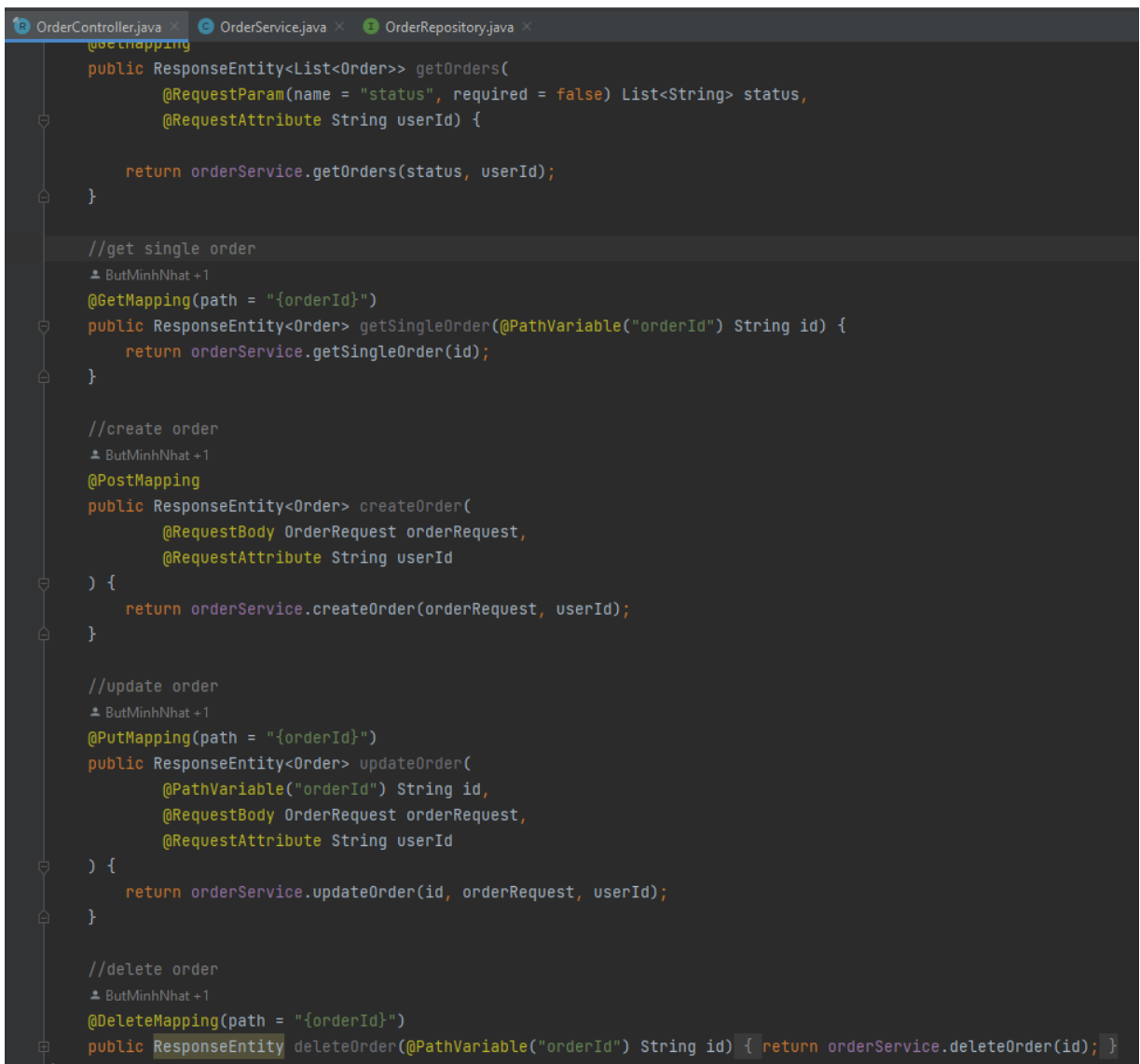
        http.authorizeRequests().anyRequest().authenticated();
    }
}
```

Hình 5.1.4.3 SecurityConfiguration

5.1.4.3 Controller (Routes)

APIs:

- GET: GetOrders
- GET: GetOrder
- POST: CreateOrder
- PUT: UpdateOrder
- DELETE: DeleteOrder



```
OrderController.java x OrderService.java x OrderRepository.java x
@GetMapping
public ResponseEntity<List<Order>> getOrders(
    @RequestParam(name = "status", required = false) List<String> status,
    @RequestAttribute String userId) {

    return orderService.getOrders(status, userId);
}

//get single order
ButMinhNhat +1
@GetMapping(path = "{orderId}")
public ResponseEntity<Order> getSingleOrder(@PathVariable("orderId") String id) {
    return orderService.getSingleOrder(id);
}

//create order
ButMinhNhat +1
@PostMapping
public ResponseEntity<Order> createOrder(
    @RequestBody OrderRequest orderRequest,
    @RequestAttribute String userId
) {
    return orderService.createOrder(orderRequest, userId);
}

//update order
ButMinhNhat +1
@PutMapping(path = "{orderId}")
public ResponseEntity<Order> updateOrder(
    @PathVariable("orderId") String id,
    @RequestBody OrderRequest orderRequest,
    @RequestAttribute String userId
) {
    return orderService.updateOrder(id, orderRequest, userId);
}

//delete order
ButMinhNhat +1
@DeleteMapping(path = "{orderId}")
public ResponseEntity deleteOrder(@PathVariable("orderId") String id) { return orderService.deleteOrder(id); }
```

Hình 5.1.4.4 OrderController

Các phương thức ở trong OrderController này chỉ gọi tiếp phương thức kế tiếp của nó ở trong OrderService (hình 5.1.4.4). Tuy nhiên trong OrderService thì lại có ba phương thức khác là containsStatus, getTotalPrice (hình 5.1.4.5) và checkExistProducts (hình 5.1.4.6).

```
public class OrderService {  
    8 usages  
    private final OrderRepository orderRepository;  
    1 usage  
    private final WebClient.Builder webClientBuilder;  
  
    1 usage ButMinhNhat  
    public static boolean containsStatus(String input) {  
        for (OrderStatus o : OrderStatus.values()) {  
            if (o.name().equals(input)) {  
                return true;  
            }  
        }  
        return false;  
    }  
  
    2 usages ButMinhNhat  
    @Transient  
    public BigDecimal getTotalOrderPrice(List<OrderProductItem> orderProducts) {  
        BigDecimal sum = BigDecimal.valueOf(0);  
        for (OrderProductItem op : orderProducts) {  
            sum = sum.add(op.getPrice().multiply(BigDecimal.valueOf(op.getQuantity())));  
        }  
        return sum;  
    }  
}
```

Hình 5.1.4.5 Phương thức containsStatus và getTotalPrice

```

2 usages ButMinhNhat
public void checkExistProducts(OrderRequest request) {
    //request product => list product id
    List<String> idList = request.products().stream() Stream<OrderProductItem>
        .map(OrderProductItem::getId) Stream<String>
        .toList();

    //get products by id
    ProductResponse[] productResponseArray = webClientBuilder.build().get() RequestHeadersUriSpec<capture of ?>
        .uri(uri: "http://product/api/v1/product",
            uriBuilder -> uriBuilder.queryParam( name: "id", idList).build()) capture of ?
        .retrieve() ResponseSpec
        .bodyToMono(ProductResponse[].class) Mono<ProductResponse[]>
        .block();

    //check quantity
    Arrays.stream(productResponseArray).forEach(product -> {
        OrderProductItem productInStock = (OrderProductItem) request.products().stream() Stream<OrderProductItem>
            .filter(p -> p.getId().equals(product.getId()) && product.getQuantity() >= p.getQuantity())
            .findFirst() Optional<OrderProductItem>
            .orElse( other: null);

        if(productInStock == null) {
            throw new IllegalArgumentException("Product is not in stock, please try again later");
        }
    });
}

```

Hình 5.1.4.6 Phương thức checkExistProducts

Hai phương thức containsStatus và getTotalPrice chỉ đơn giản là hàm nhỏ được implement để dùng cho các phương thức chính. Nhưng phương thức checkExistProducts lại là hàm đặc biệt. Nó dùng để kiểm tra liệu Product đó có tồn tại hay không và liệu số lượng (quantity) của product đó có còn hay không. Phương thức này gọi đến API getsingleproduct của ProductService để lấy thông tin của sản phẩm cần tìm dựa trên product-slug mà phương thức này được truyền vào. Nếu tìm thấy sản phẩm thì thông tin nhận được từ ProductService, nó sẽ kiểm tra tiếp là product có cần số lượng hay không? Nếu có product và số lượng vẫn còn thì sẽ tiếp tục đi tiếp, nếu không thì sẽ ngay lập tức ngắt request và trả về lỗi kèm message “Product is not in stock, please try again later.” Phương thức này được dùng trong 2 request đó là createOrder và updateOrder. Khi tạo mới và sửa Order, thì OrderService sẽ cần phải tương tác với ProductService để lấy thông

tin sản phẩm và dựa vào đó, OrderService sẽ tiếp tục xử lý logic và tương tác với database thông qua OrderRepository. (hình 5.1.4.7, 5.1.4.8, 5.1.4.9)

```
public ResponseEntity<List<Order>> getOrders(List<String> status, String userId) {
    if(status != null) {
        return ResponseEntity.ok(orderRepository.findByStatusInAndCreatedBy(status, userId));
    }
    return ResponseEntity.ok(orderRepository.findByCreatedBy(userId));
}

1 usage  ▲ Võ Tấn Phát +1
public ResponseEntity<Order> getSingleOrder(String orderId) {
    Optional<Order> orderData = orderRepository.findById(orderId);
    if (orderData.isEmpty()) {
        return new ResponseEntity<>(HttpStatus.NOT_FOUND);
    }
    return new ResponseEntity<Order>(orderData.get(), HttpStatus.OK);
}

1 usage  ▲ ButMinhNhat +1
public ResponseEntity<Order> createOrder(OrderRequest request, String userId) {
    //check order request
    checkExistProducts(request);

    Order order = Order.builder()
        .status(OrderStatus.PENDING.name())
        .total(getTotalOrderPrice(request.products()))
        .products(request.products())
        .createdBy(userId)
        .updatedBy(userId)
        .build();

    return new ResponseEntity<Order>(orderRepository.save(order), HttpStatus.OK);
}
```

Hình 5.1.4.7 OrderService [1]

```

public ResponseEntity<Order> updateOrder(String orderId, OrderRequest request, String userId) {
    Optional<Order> orderData = orderRepository.findById(orderId);
    if(orderData.isEmpty()) {
        return new ResponseEntity<>(HttpStatus.NOT_FOUND);
    }

    if(!containStatus(request.status())) {
        return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
    }

    if(request.status().equalsIgnoreCase(OrderStatus.PAID.name())) {
        //check exist products
        checkExistProducts(request);
    }

    //update
    if(request.status() != null) { orderData.get().setStatus(request.status()); }
    if(request.products() != null) {
        orderData.get().setTotal(getTotalOrderPrice(request.products()));
        orderData.get().setProducts(request.products());
    }
    orderData.get().setUpdatedBy(userId);

    return new ResponseEntity<Order>(orderRepository.save(orderData.get()), HttpStatus.OK);
}

1 usage  🧑 Võ Tấn Phát +1
public ResponseEntity deleteOrder(String orderId) {
    Optional<Order> orderData = orderRepository.findById(orderId);
    if (orderData.isEmpty()) {
        return new ResponseEntity<>(HttpStatus.NOT_FOUND);
    }
    orderRepository.deleteById(orderId);
    return new ResponseEntity<>(HttpStatus.OK);
}

```

Hình 5.1.4.8 OrderService [2]

```

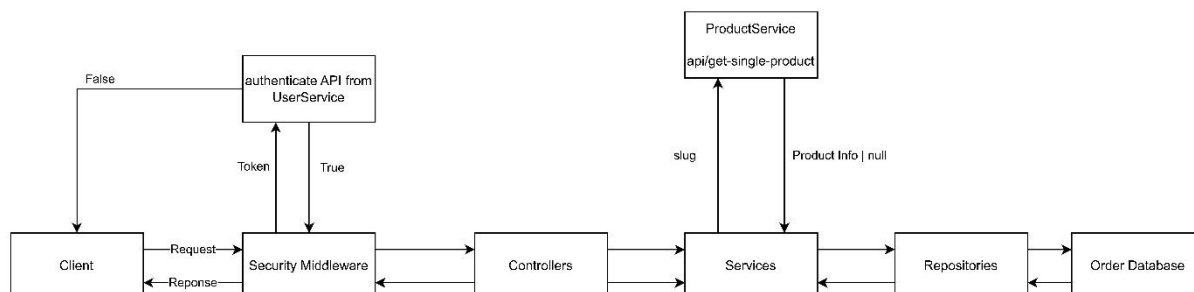
2 📄 pages  🧑 ButMinhNhat
public interface OrderRepository extends JpaRepository<Order, String> {
    1 usage  🧑 ButMinhNhat
    List<Order> findByCreatedBy(String createdBy);
    1 usage  🧑 ButMinhNhat
    List<Order> findByStatusInAndCreatedBy(List<String> status, String createdBy);
}

```

Hình 5.1.4.9 OrderRepository

5.1.4.4 Kết luận

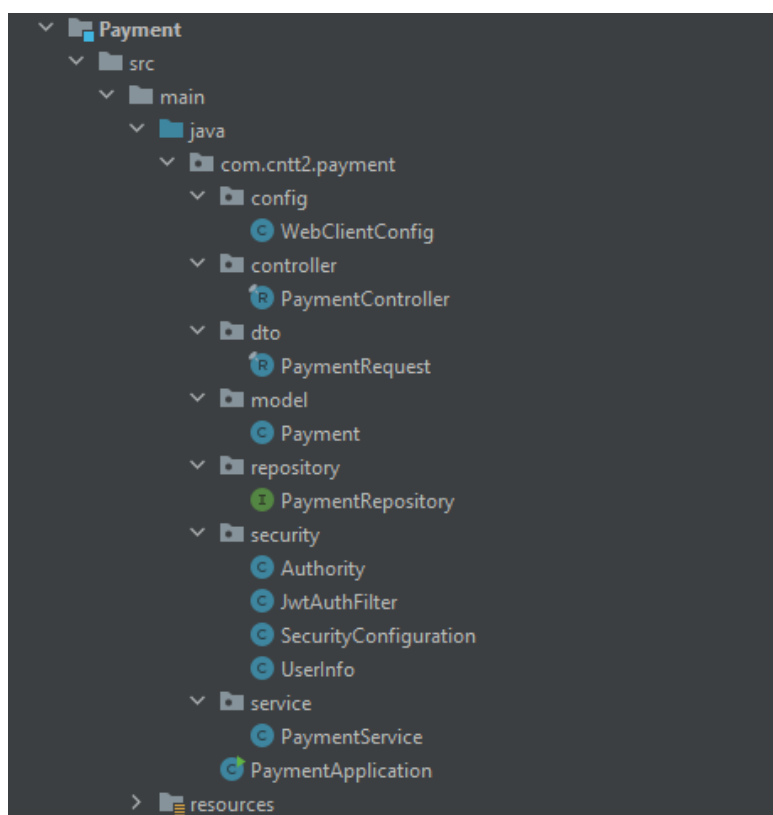
Các request của OrderService được biểu diễn như hình dưới đây:



Hình 5.1.4.10 Flow request trong OrderService

5.1.5 Payment Service

Payment Service cũng được tổ chức theo kiến thức DDD và có Security Middleware tương tự như ProductService, UserService và OrderService:



Hình 5.1.5.1 Tổ chức của Payment Service

5.1.5.1 Model

PaymentService chỉ có 1 model, chi tiết model đó như sau:

```
public class Payment {  
    @Id  
    @GeneratedValue(generator = "uuid2")  
    @GenericGenerator(name = "uuid2", strategy = "org.hibernate.id.UUIDGenerator")  
    @Column(name = "id", columnDefinition = "VARCHAR(255)")  
    private String id;  
  
    @Column(name="name")  
    private String name;  
  
    @Column(name="type")  
    private String type;  
  
    @Column(name="balance")  
    private BigDecimal balance;  
  
    @Column(name="userId")  
    private String userId;  
}
```

Hình 5.1.5.2 PaymentModel

5.1.5.2 Security

Cũng như OrderService, Payment Service cũng tương tác với UserService để xác thực token và request nào trong service này cũng cần xác thực.

5.1.5.3 Controller (Routes)

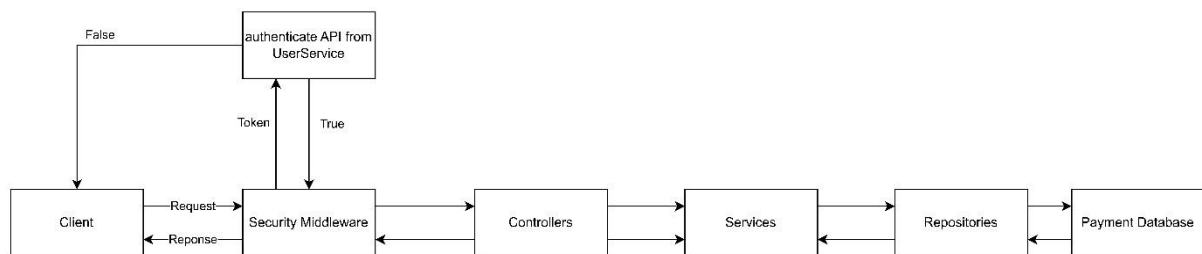
APIs:

- GET: GetPayments
- GET: GetPayment
- POST: CreatePayment
- PUT: UpdatePayment
- DELETE: DeletePayment

APIs sẽ nhận request và đưa tới Controller sau khi qua bước xác thực token. Controller sẽ tiếp tục điều hướng request đến Service layer để xử lý logic và layer service sẽ tương tác với database bằng PaymentRepository tương tự như các Service khác. Payment Service ở đây không có tương tác với các service khác để xử lý logic. Tham khảo source code ³ để biết thêm chi tiết

5.1.4.3. Kết luận

Các request tới PaymentService sẽ được biểu diễn như dưới đây:

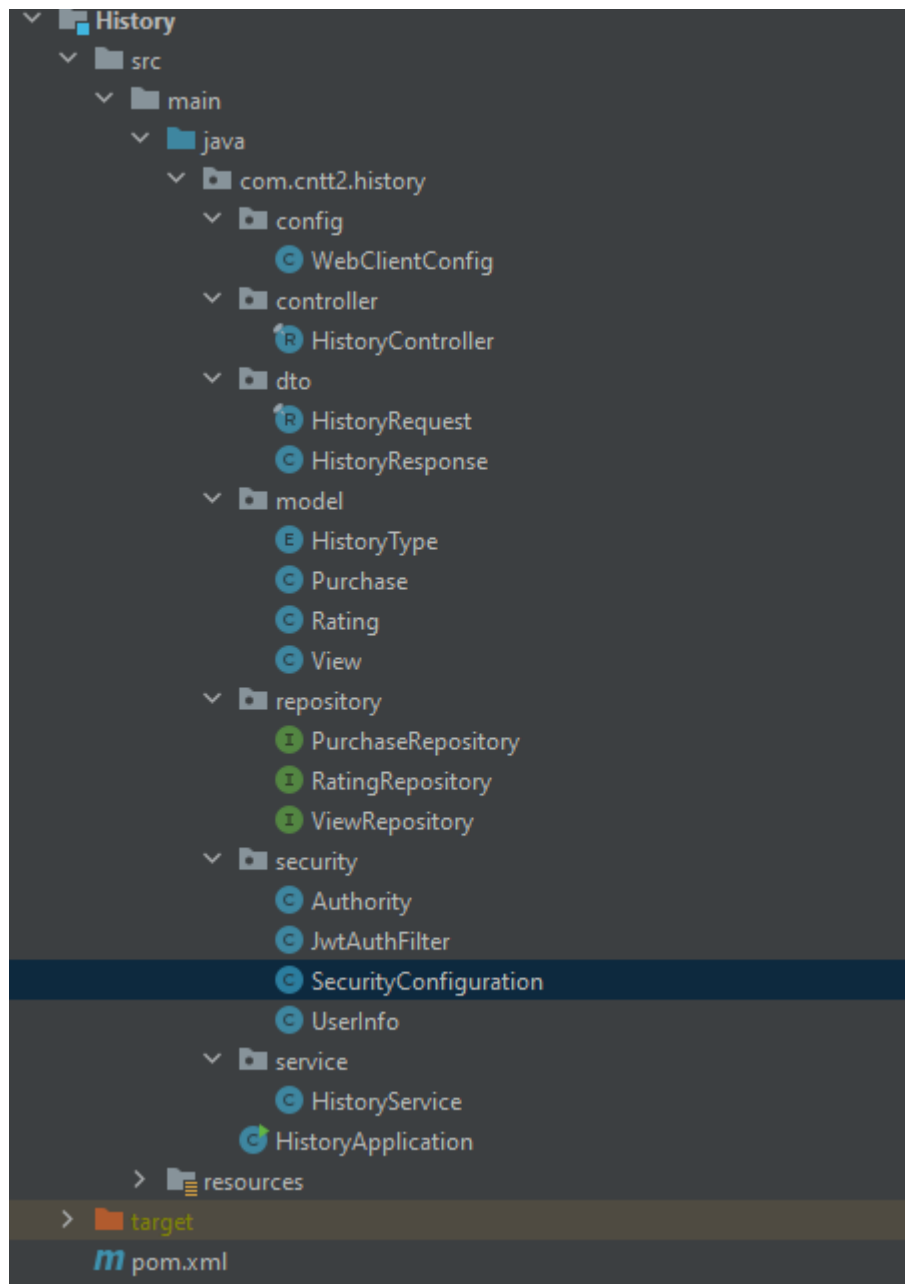


Hình 5.1.5.3 Flow request của Payment Service

5.1.6 History Service

History Service cũng được tổ chức theo cấu trúc DDD và cũng có security kiểm tra middleware tương tự như các services khác.

³ <https://github.com/recca5p/DACNTT2-TDTU-Ecommerce/tree/master/Ecommerce.Api/Payment/src/main>



Hình 5.1.6.1 Tổ chức của History Service

5.1.6.1 Model

History Service có 3 models:

- Purchase_product: các sản phẩm đã được mua
- Rating_product: đánh giá của các sản phẩm

- View_product: số lượt xem của sản phẩm

5.1.6.2 Security

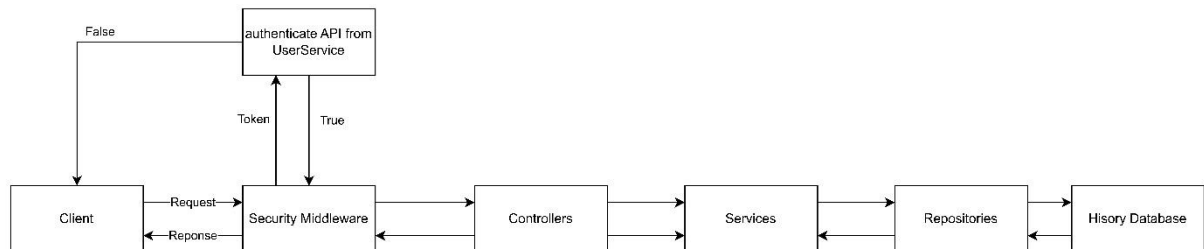
History Service cũng cần xác thực ở tất cả các APIs và dùng UserService để xác thực.

5.1.6.3 Controller (Routes)

APIs:

- GET: GetHistories
- POST: PostHistory

History Service cũng đi theo flow của các service và History Service cũng chỉ tương tác với database của nó thông qua Repository chứ không tương tác với Service khác. Chi tiết có thể tham khảo source code ⁴.



Hình 5.1.6.2 Flow request của History Service

5.1.7 Image Service

Image Service cũng được tổ chức theo các service khác nhưng Image Service không có Security Middleware.

5.1.7.1 Model

Model của Service này chỉ có mỗi Image Model:

⁴ <https://github.com/recca5p/DACNTT2-TDTU-Ecommerce/tree/master/Ecommerce.Api/History/src/main/java/com/cntt2/history>

```

@Entity
@Document(collection = "images")
@NoArgsConstructor
public class Image {
    @Id
    private String id;

    private Binary image;
}

```

Hình 5.1.7.1 Image Model

5.1.7.2 Security

Do Image Service không có Security Middleware nên không cần phải xác thực gì cả.

5.1.7.3 Controller (Routes)

APIs:

- GET: GetPhoto
- POST: AddPhoto

Image Service cũng không có gì khác biệt so với các service khác trong hệ thống. Nó cũng nhận request ở controller và đẩy tiếp request xuống layer service để xử lý logic và tương tác với database bằng repository. Chi tiết tham khảo source code ⁵.

5.1.7.4 Kết luận

Image Service có được biểu diễn như sau:



Hình 5.1.7.2 Flow của request trong Image Service

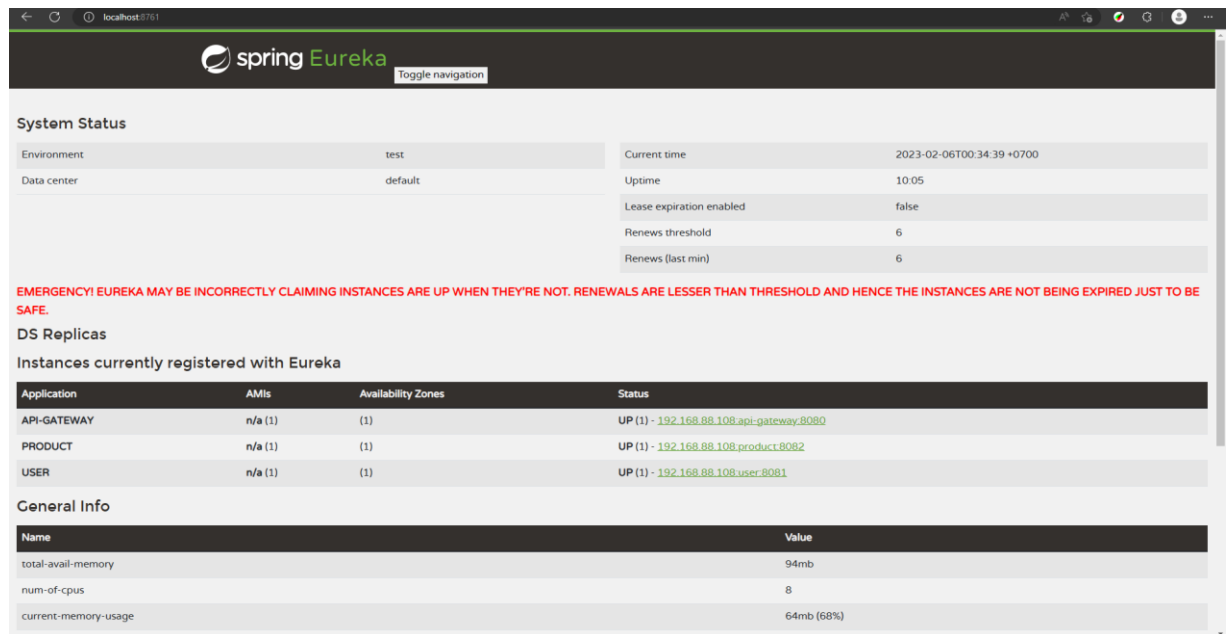
⁵ <https://github.com/recca5p/DACNTT2-TDTU-Ecommerce/tree/master/Ecommerce.Api/Image/src/main/java/com/cntt2/image>

5.1.8 API Gateway

API được dùng để khai báo port cho các service, chi tiết tham khảo source code ⁶.

5.1.9 Eureka Server

Eureka Server dùng để quản lý các service, chi tiết tham khảo source code ⁷.



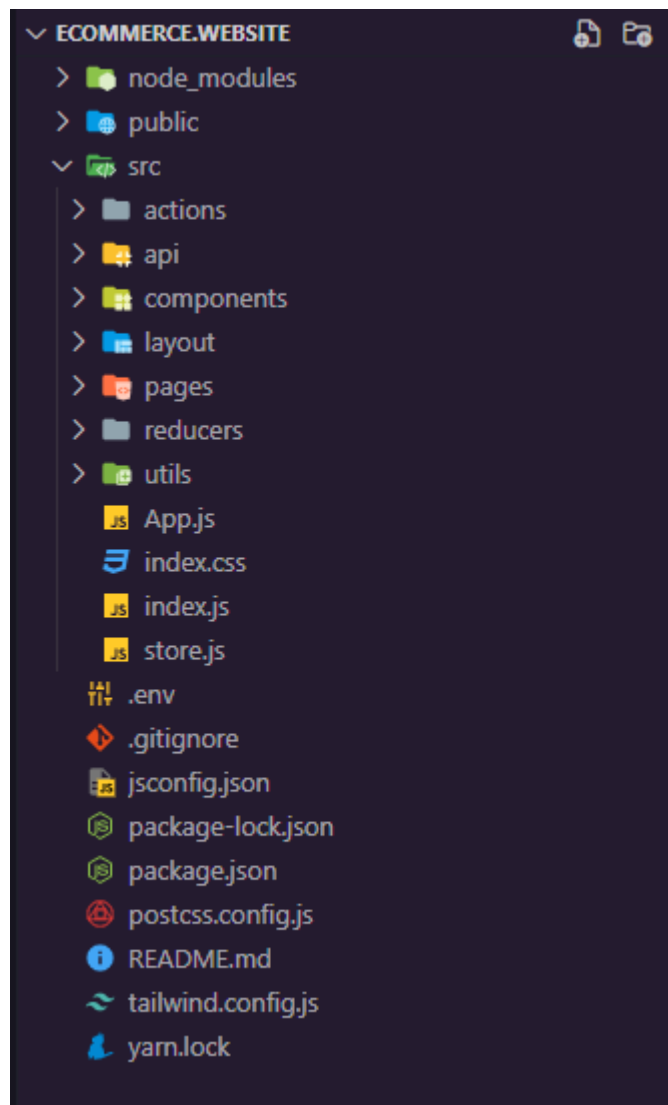
Hình 5.1.9.1 Eureka Server

5.2 Hiện thực website bán hàng

Website bán hàng được implements bằng library của Javascript là ReactJS. Cấu trúc của một project React được tổ chức như sau:

⁶ <https://github.com/recca5p/DACNTT2-TDTU-Ecommerce/blob/master/Ecommerce.Api/API-gateway/src/main/resources/application.yml>

⁷ <https://github.com/recca5p/DACNTT2-TDTU-Ecommerce/tree/master/Ecommerce.Api/Eureka-server>



Hình 5.2.1 Tổ chức project trong ReactJS

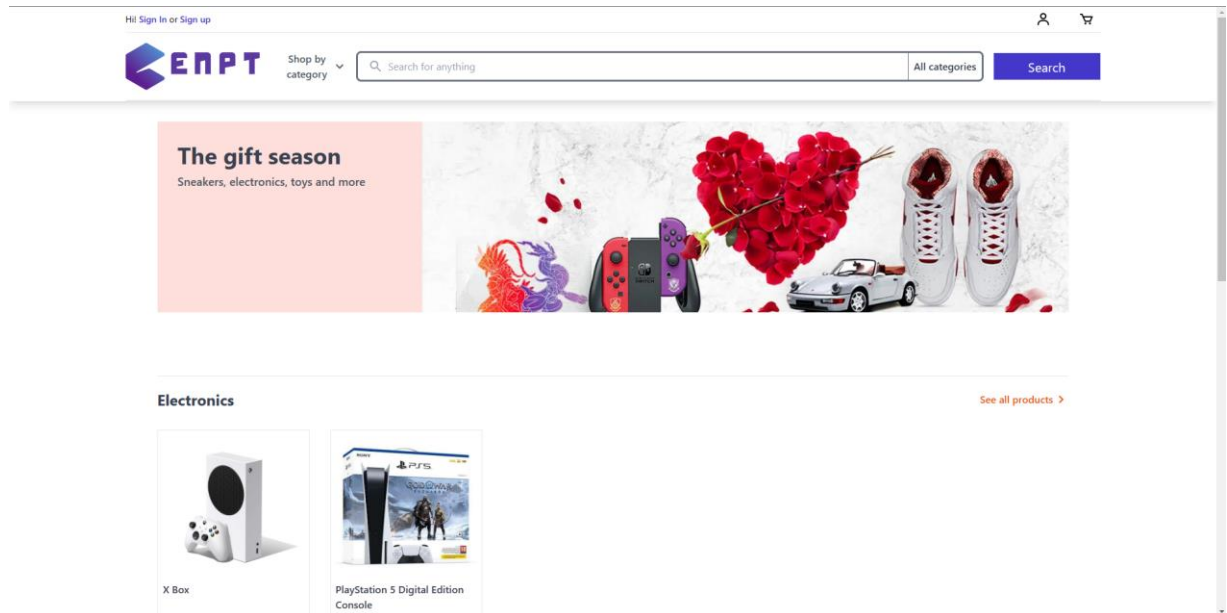
Các page có trong website gồm:

- Home: “/”
- Product: “/product”
- ProductDetail: “/product/slug”
- Category: “/category/slug”
- Cart: “/cart”
- Checkout: “/checkout/orderId”

- SignIn: “/auth/sign-in”
- SignUp: “auth/sign-up”
- NotFound “/*”

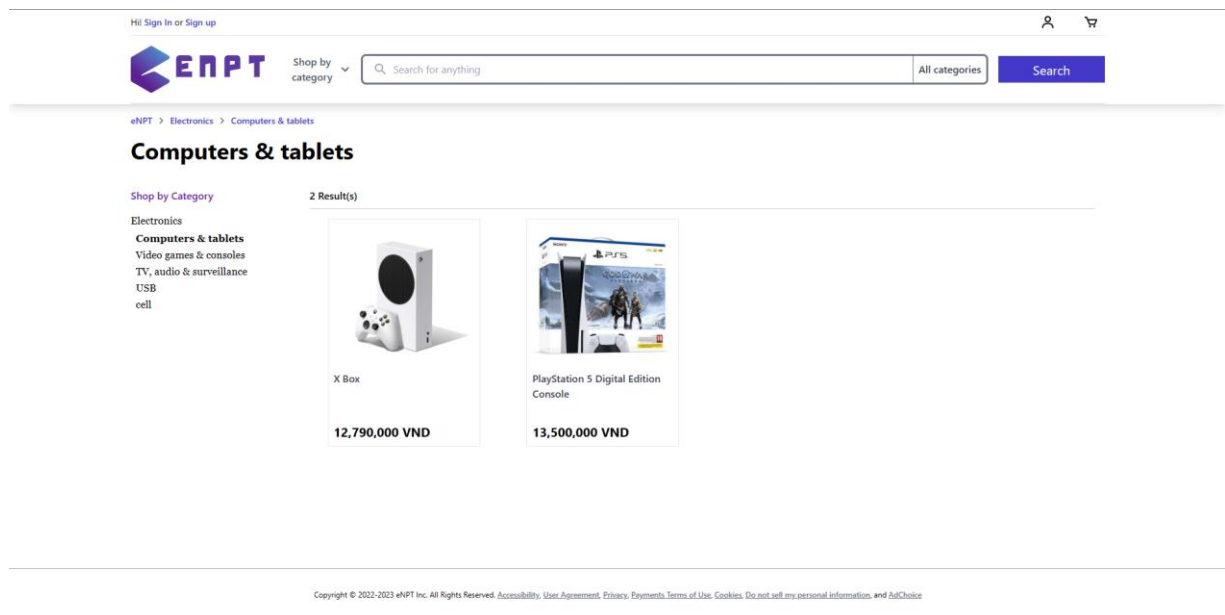
Chi tiết các implement giao diện có thể tham khảo source code ⁸.

Một vài giao diện của website bán hàng:

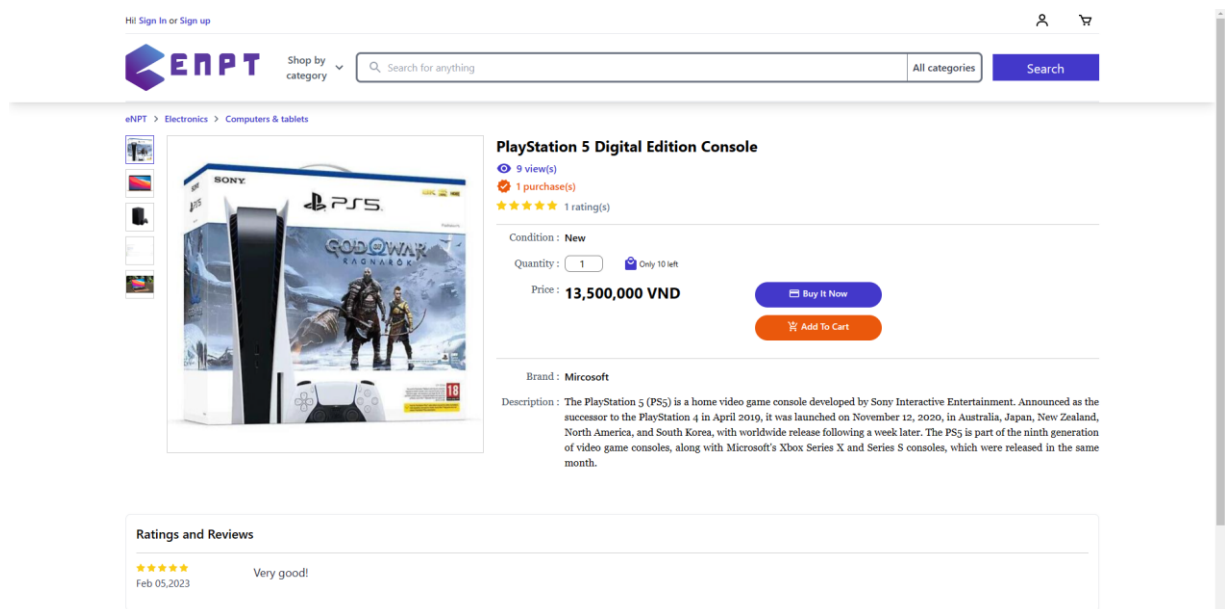


Hình 5.2.2 Home Page

⁸ <https://github.com/recca5p/DACNTT2-TDTU-Ecommerce/tree/master/Ecommerce.Website>



Hình 5.2.3 Search filter



Hình 5.2.4 Product detail



Hello

Sign in to eNTP or [create an account](#)

Username

Password

☒ Stay signed in

Sign in

Copyright © 2022-2023 eNPT Inc. All Rights Reserved. [Accessibility](#), [User Agreement](#), [Privacy](#), [Payments Terms of Use](#), [Cookies](#), [Do not sell my personal information](#), and [AdChoice](#)

Hình 5.2.5 Sign In Page



Create an account

Already a member ? [Sign in](#)

Your name

Username

Password

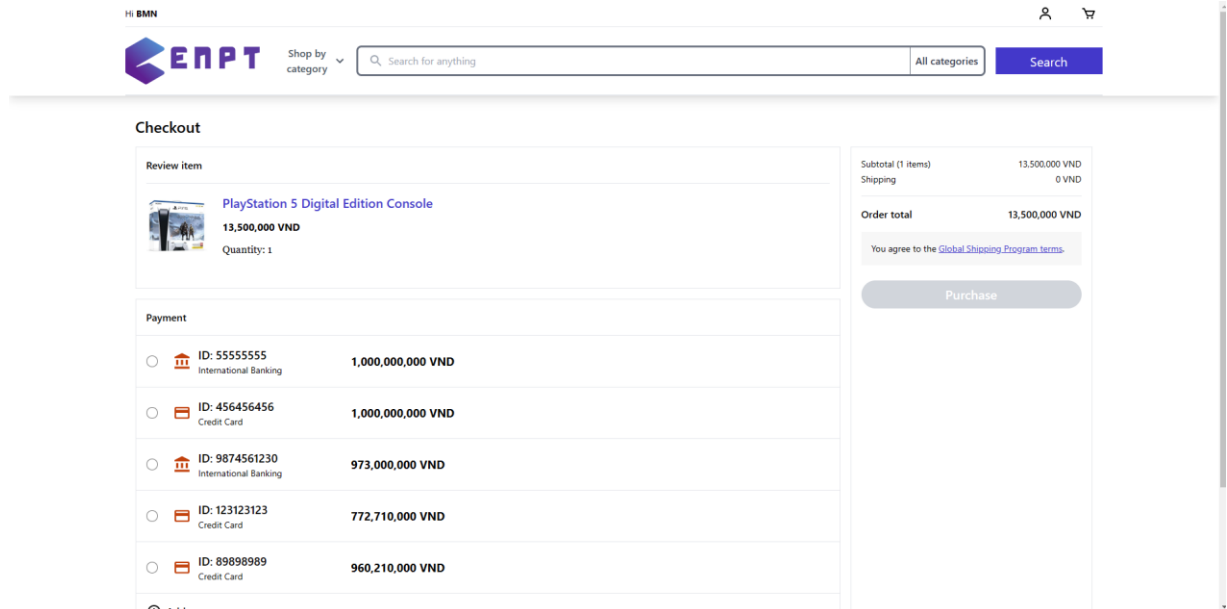
Email

Phone number

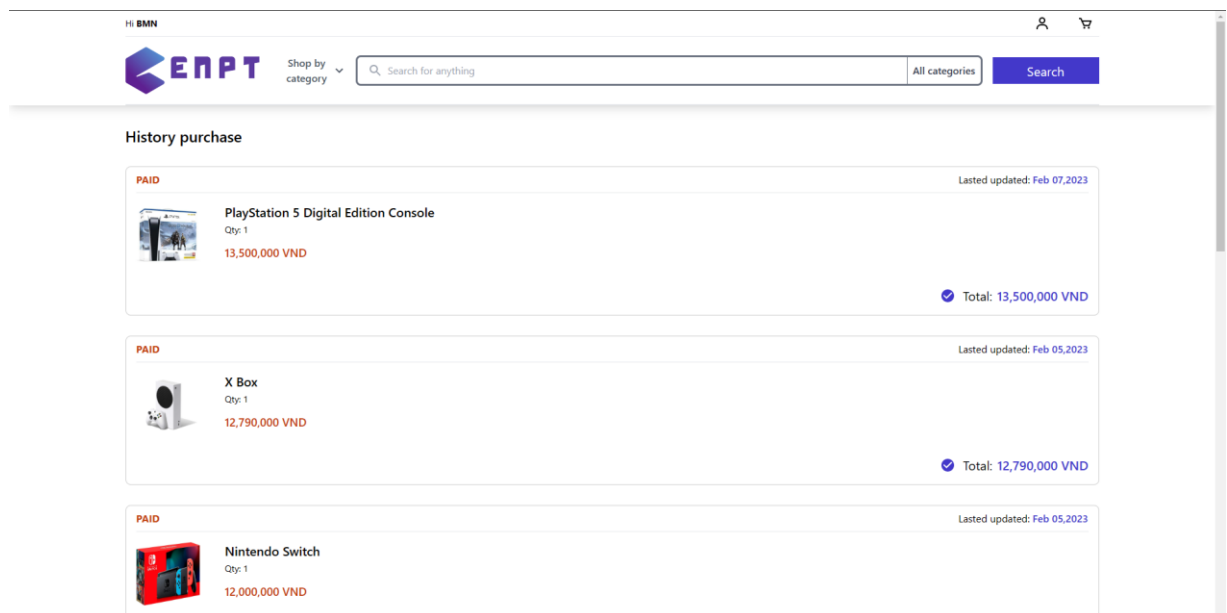
Sign Up

Copyright © 2022-2023 eNPT Inc. All Rights Reserved. [Accessibility](#), [User Agreement](#), [Privacy](#), [Payments Terms of Use](#), [Cookies](#), [Do not sell my personal information](#), and [AdChoice](#)

Hình 5.2.6 SignUp Page



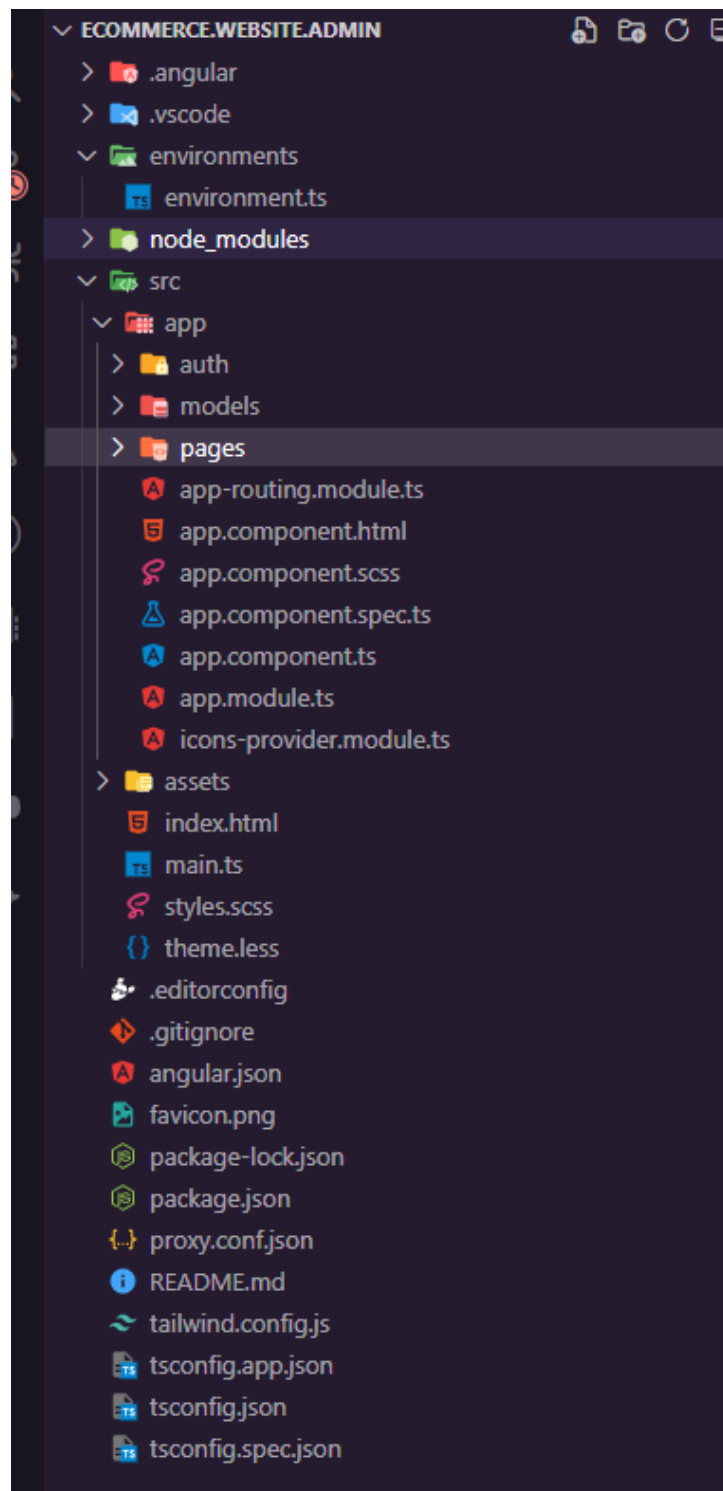
Hình 5.2.7 Checkout Page



Hình 5.2.8 History Page

5.3 Hiện thực website quản lý

Website quản lý được implements bằng framework là Angular, ngôn ngữ của framework này là typescript. Cấu trúc của một project Angular được tổ chức như sau:



Hình 5.3.1 Tổ chức trong project Angular

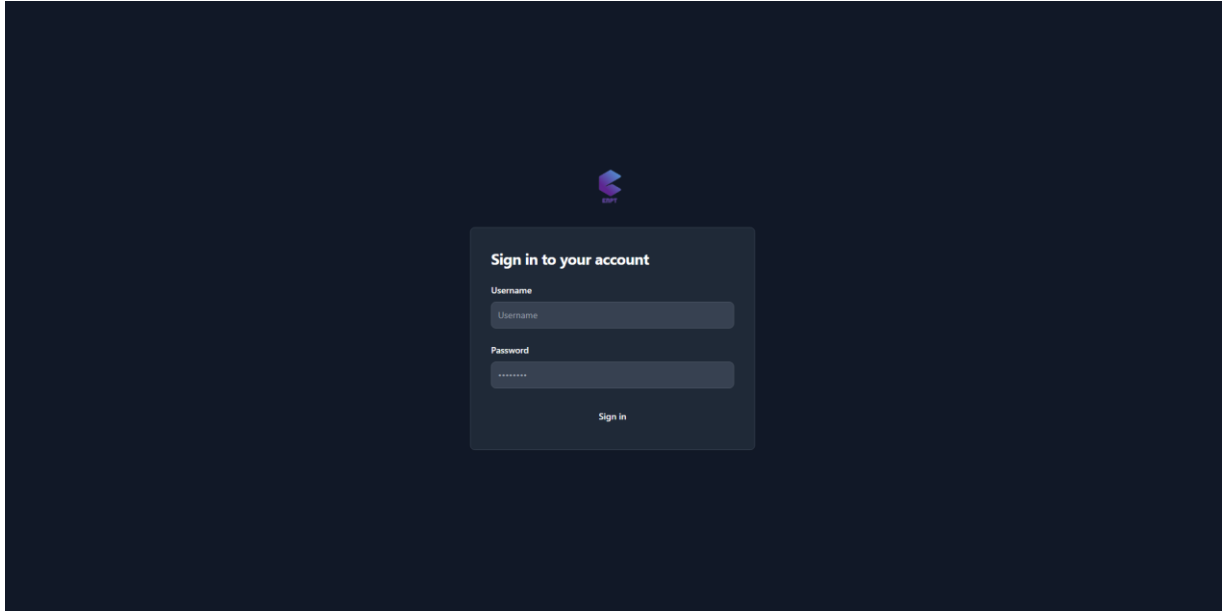
Các page có trong website gồm:

- Home: “dashboard/welcome”
- Product: “/product-manage/product”
- ProductDetail: “/product-manage/product-detail/slug”
- CreateProduct: “product-manage/create-product”
- Category: “/product-manage/category”
- CategoryDetail: /product-manage/category-detail/categoryId”
- CreateCategory: “product-manage/create-category”
- Brand: “/product-manage/brand”
- BrandDetail: /product-manage/brand-detail/brandId”
- CreateBrand: “product-manage/create-brand”
- Order: “/order-manage/order”
- OrderDetail: /order-manage/ order -detail/orderId”
- User: “/user-manage/user”
- CreateUser: “user-manage/create-user”
- SignIn: “/auth”
- NotFound “/**”

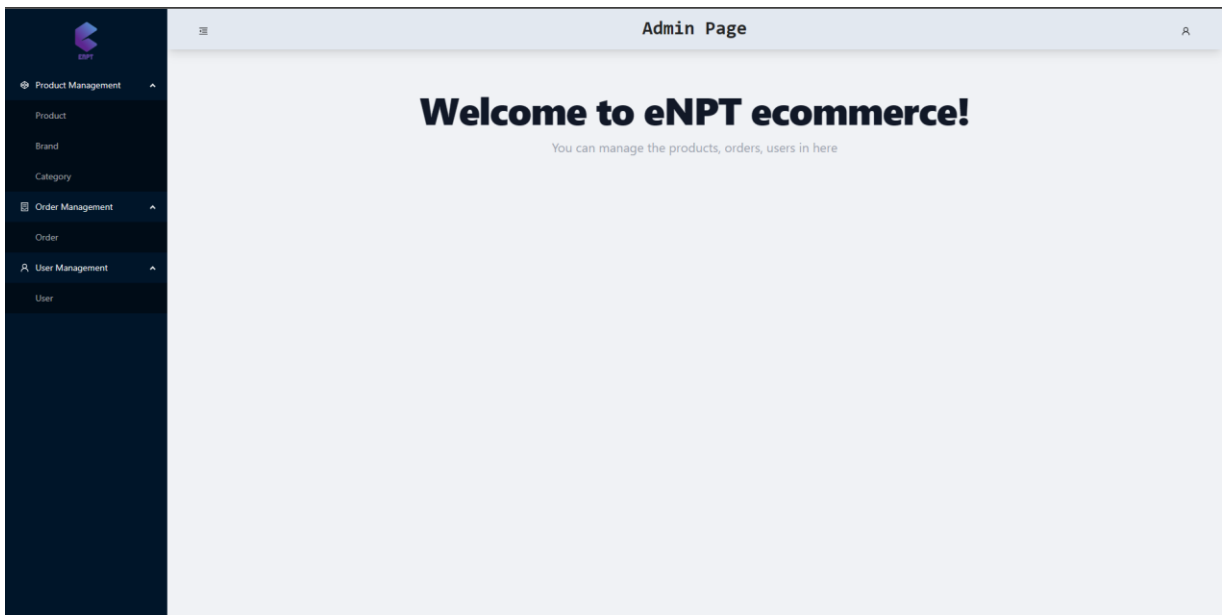
Chi tiết các implement giao diện có thể tham khảo source code ⁹.

Một vài giao diện của website quản lý:

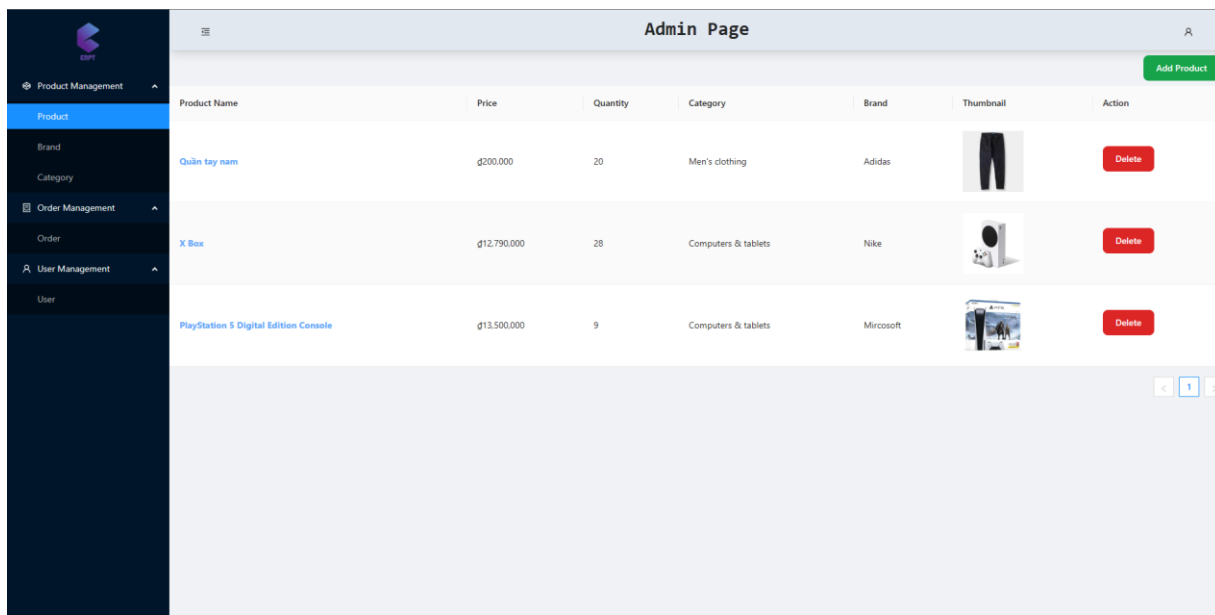
⁹ <https://github.com/recca5p/DACNTT2-TDTU-Ecommerce/tree/master/Ecommerce.Website.Admin>



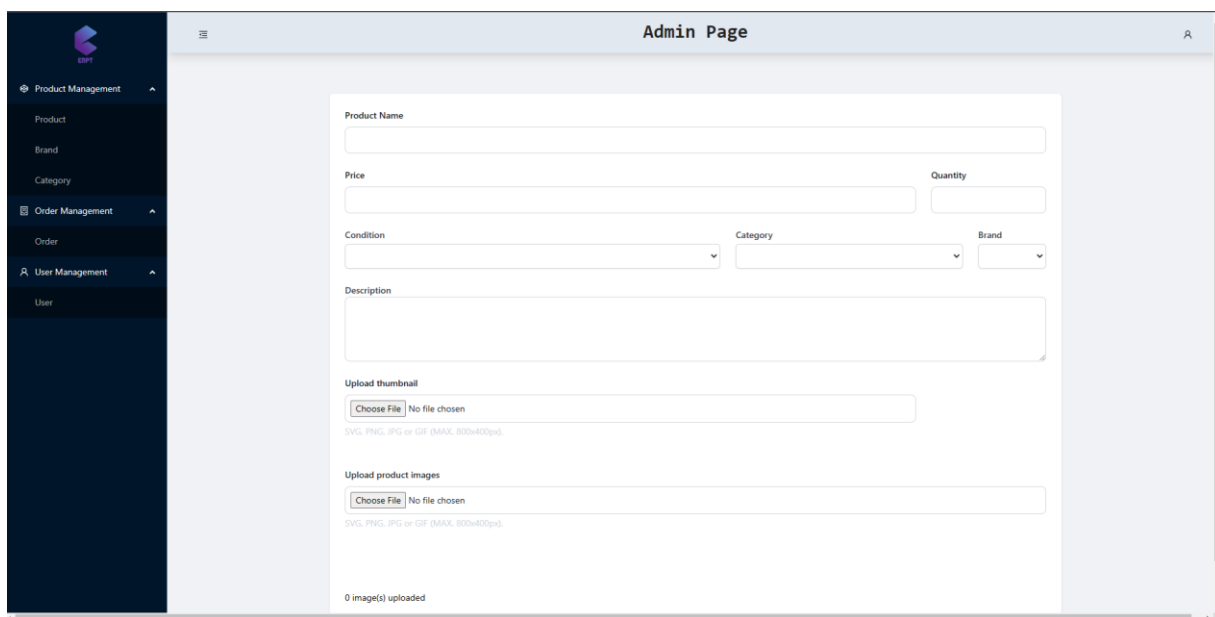
Hình 5.3.2 SignIn Page




Hình 5.3.3 Homepage



Hình 5.3.4 Product Page



Hình 5.3.5 Create Product Page



- Product Management
 - Product
 - Brand
 - Category
- Order Management
 - Order
- User Management
 - User

Admin Page

Product Name

PlayStation 5 Digital Edition Console

Price

13500000

Quantity

9

Condition

New

Category

Computers & tablets


Brand

Microsoft

Description

The PlayStation 5 (PS5) is a home video game console developed by Sony Interactive Entertainment. Announced as the successor to the PlayStation 4 in April 2019, it was launched on November 12, 2020, in Australia, Japan, New Zealand, North America, and South Korea, with worldwide release following a week later. The PS5 is part of the ninth generation of video game consoles, along with Microsoft's Xbox Series X and Series S consoles, which were released in the same month.

Thumbnail



Images

Choose File


No file chosen

image(s) uploaded

Upload

Delete

Hình 5.3.6 Product Detail



- Product Management
 - Product
 - Brand
 - Category
- Order Management
 - Order
- User Management
 - User

Admin Page

Add Brand

| Brand | Action |
|-----------|--------|
| Nike | Delete |
| Adidas | Delete |
| Sony | Delete |
| Microsoft | Delete |

<

1

>

Hình 5.3.7 Brand Page

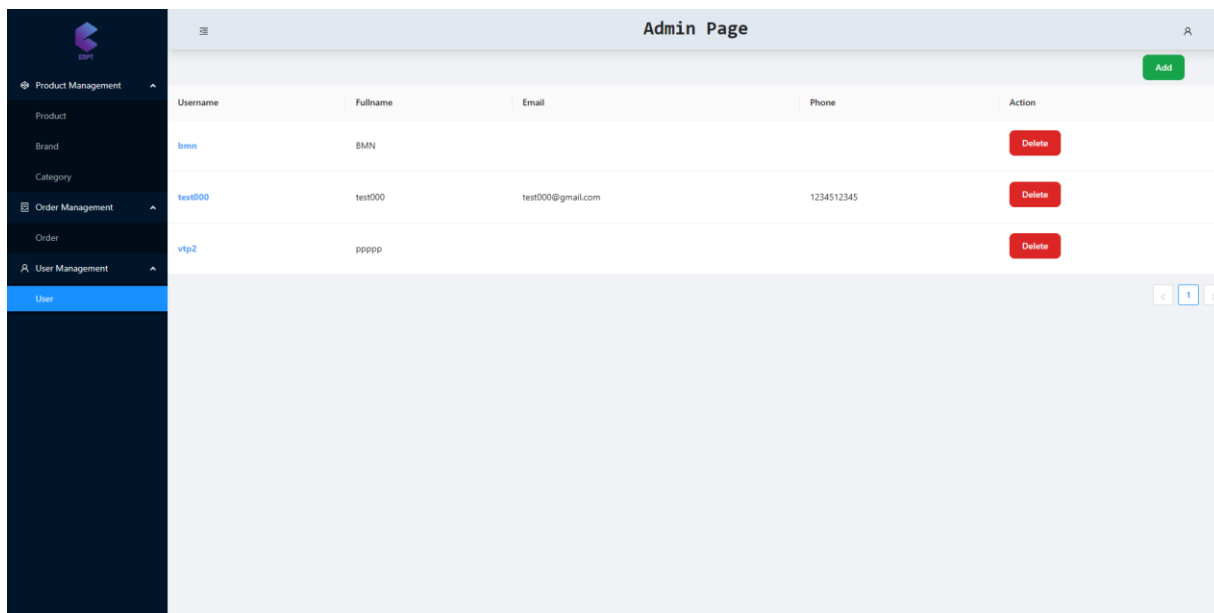
58

| Admin Page | | | 🔍 |
|--------------------------|-------------|------------------------|---------------------|
| | | | Add |
| Category Name | Parent | Action | |
| Computers & tablets | Electronics | Delete | |
| Video games & consoles | Electronics | Delete | |
| TV, audio & surveillance | Electronics | Delete | |
| Men's clothing | Fashion | Delete | |
| Women's clothing | Fashion | Delete | |
| Kid's clothing | Fashion | Delete | |
| Football | Sports | Delete | |
| Basketball | Sports | Delete | |
| Running | Sports | Delete | |
| Swimming | Sports | Delete | |

Hình 5.3.8 Category Page

| Admin Page | | | | 🔍 |
|--------------------------------------|---------|-----------|----------|---|
| OrderId | Status | Total | Products | |
| 88b86551-d9d9-4dbe-bcc7-49b9b9f04d8c | PAID | 27000000 | 1 | |
| 66f99546-5447-4587-a3d2-3f5590b4c8a3 | PENDING | 0 | 0 | |
| fa7581b7-2a4e-40c7-830b-27885aadf5e6 | PENDING | 0 | 0 | |
| 0c277d8a-fdc5-4559-972b-56419fa81024 | PAID | 135000000 | 1 | |
| 7dfcd10a-f19f-42df-860e-9ad48de6cb13 | PAID | 54000000 | 1 | |
| 09fc098a-286a-46f1-bba0-198d76bd0266 | PAID | 27000000 | 1 | |
| d81cf084-1ecc-4641-8e30-1aa48fc5a857 | PENDING | 0 | 0 | |
| ceccf078-5186-4057-8b1c-361f6ed251ce | PENDING | 0 | 0 | |
| 339bdfc4-050b-4ad5-a51a-39590d14863f | PAID | 38290000 | 3 | |
| bb2b8b0f-7796-4a23-9f16-32dde308f045 | PAID | 12790000 | 1 | |
| | | | | < 1 2 > |

Hình 5.3.9 Order Page



Hình 5.3.10 User Page



Hình 5.3.11 Statistic Page

CHƯƠNG VI – KẾT LUẬN

Sau khi hoàn thành dự án, nhóm em đã implements được 1 hệ thống thương mại điện tử theo kiến trúc microservices. Hệ thống đã đạt được các chức năng cơ bản để dùng cho việc bán hàng. Hệ thống gồm 2 website là website bán hàng và website quản lý được hoạt động riêng biệt. Đây là lợi thế của kiến trúc microservices khi nó tách các services để dễ dàng bảo trì, nâng cấp và khi có một services gặp trục trặc thì các services khác vẫn có thể hoạt động.

Bên cạnh các mặt mà nhóm em đạt được, hệ thống của nhóm còn thiếu vài chức năng chưa thể hoàn thành được. Nhóm em đã học hỏi và rút kinh nghiệm cho các dự án sau này.

TÀI LIỆU THAM KHẢO

1. Dat Nguyen (2020). “Tìm hiểu về microservice”, Viblo Blog, <https://viblo.asia/p/tim-hieu-ve-microservice-Do754PD45M6>
2. Kong. “What are microservices?”, Mircoservices IO, <https://microservices.io/>
3. “Domain-Driven-Design(DDD)”,GeeksForGeeks, <https://www.geeksforgeeks.org/domain-driven-design-ddd/>
4. Tu Bean (2018), “[Microservice] Dụng Microservice web bằng Spring Boot và Eureka”, Github, <https://tubean.github.io/2018/12/microservice-springboot-eureka>
5. Omar Elgabry, “Microservices with Spring Boot”, OmarElgabry's Blog, <https://medium.com/omarelgabrys-blog/microservices-with-spring-boot-intro-to-microservices-part-1-c0d24cd422c3>
6. Binildas Christudas (2019), Practical Mircoservices Architectural Patterns