|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ĐẠI HỌC BÁCH KHOA HÀ NỘI  **TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**  ──────── \* ───────  Trường Công nghệ Thông tin và Truyền thông - Đại học Bách khoa Hà Nội  **BÁO CÁO**  **Môn học:**  **PROJECT III**  **IT3943**   |  |  | | --- | --- | | **Giảng viên hướng dẫn:** | Đinh Thị Hà Ly | | **Mã học phần:** | IT3943 | | **Tên học phần:** | Project III | | **Học kỳ:** | 2024.1 | | **Sinh viên thực hiện:** | **MSSV** | | Nguyễn Đình Tuấn Đạt | 20215562 |   ***Hà Nội, tháng 10 năm 2024*** |

Contents

[I. Tổng quan về đề tài 5](#_Toc187413257)

[1.1. Mô tả đề tài 5](#_Toc187413258)

[1.2. Những khó khăn khi giải quyết bài toán. 6](#_Toc187413259)

[1.2.1. Quản lý service. 6](#_Toc187413260)

[1.2.2. Quản lý cấu hình. 6](#_Toc187413261)

[1.2.3. Vấn đề khi gọi tới một service. 7](#_Toc187413262)

[1.2.4. Giao tiếp giữa các service với nhau. 7](#_Toc187413263)

[II. Kiến trúc hệ thống. 8](#_Toc187413264)

[2.1. Tác nhân hệ thống. 8](#_Toc187413265)

[2.2. Các tính năng của hệ thống. 8](#_Toc187413266)

[2.3. Sơ đồ Usecase tổng quan 11](#_Toc187413267)

[2.3.1. Usecase đăng ký/đăng nhập 11](#_Toc187413268)

[2.3.2. Usecase Tìm kiếm 11](#_Toc187413269)

[2.3.3. Usecase quản lý Order 12](#_Toc187413270)

[2.3.4. Usecase become a seller 12](#_Toc187413271)

[2.3.5. Usecase quản lý gig 13](#_Toc187413272)

[2.3.6. Usecase process order 14](#_Toc187413273)

[2.3.7. Usecase process offer 14](#_Toc187413274)

[2.3.8. Usecase process extension 15](#_Toc187413275)

[2.4. Sơ đồ activities cho các luồng nghiệp vụ. 15](#_Toc187413276)

[2.4.1. Nghiệp vụ đăng ký và xác thực tài khoản. 15](#_Toc187413277)

[2.4.2. Nghiệp vụ tìm kiếm và mua dịch vụ. 16](#_Toc187413278)

[2.4.3. Nghiệp vụ hoàn thiện sản phẩm và đánh giá 16](#_Toc187413279)

[2.5. Thiết kế cơ sở dữ liệu 17](#_Toc187413280)

[2.5.1. Gig domain 17](#_Toc187413281)

[2.5.2. User domain 17](#_Toc187413282)

[2.5.3. Chat domain 18](#_Toc187413283)

[2.5.4. Order domain 18](#_Toc187413284)

[2.5.5. Payment domain 19](#_Toc187413285)

[III. Dữ liệu truyền vào trong các usecase 19](#_Toc187413286)

[3.1. Đăng ký 19](#_Toc187413287)

[3.2. Đăng nhập 19](#_Toc187413288)

[3.3. Verify email ( bỏ qua ) 19](#_Toc187413289)

[3.4. Reset password 19](#_Toc187413290)

[3.5. Change password 20](#_Toc187413291)

[III. Công nghệ sử dụng. 20](#_Toc187413292)

[3.1. Kiến trúc monolithic. 20](#_Toc187413293)

[3.2. Kiến trúc microservice. 22](#_Toc187413294)

[3.3. Service Discovery. 23](#_Toc187413295)

[3.4. Centralized configuration. 24](#_Toc187413296)

[3.5. API Gateway. 26](#_Toc187413297)

[3.6. Message Queue. 27](#_Toc187413298)

[IV. Kết quả sản phẩm. 28](#_Toc187413299)

[V. Kết luận. 31](#_Toc187413300)

[5.1. Độ hoàn thiện sản phẩm. 31](#_Toc187413301)

[5.2. Những phần có thể cải tiến thêm trong tương lai. 31](#_Toc187413302)

[5.3. Mã nguồn. 31](#_Toc187413303)

## Tổng quan về đề tài

## Mô tả đề tài

* Jobber là một nền tảng marketplace đột phá, được thiết kế để tạo ra cầu nối hiệu quả giữa cộng đồng freelancer và khách hàng trong khu vực Đông Nam Á, với trọng tâm ban đầu tại thị trường Việt Nam. Được xây dựng dựa trên kiến trúc microservice hiện đại, Jobber không chỉ đơn thuần là một sàn giao dịch việc làm tự do thông thường, mà còn là một hệ sinh thái toàn diện được thiết kế để đáp ứng những nhu cầu đặc thù của thị trường nội địa.
* Trong bối cảnh thị trường freelance đang bùng nổ tại Việt Nam và khu vực Đông Nam Á, với tốc độ tăng trưởng ước tính đạt 15-20% mỗi năm, các nền tảng hiện có như Fiverr, Toptal hay Freelancer.com vẫn còn nhiều hạn chế. Những platform quốc tế này thường được xây dựng trên kiến trúc monolithic truyền thống, khiến việc tùy biến và mở rộng tính năng cho thị trường địa phương trở nên khó khăn và tốn kém. Bên cạnh đó, các rào cản về ngôn ngữ, phương thức thanh toán, và mức phí cao cũng là những thách thức đáng kể đối với người dùng trong khu vực.
* Jobber được phát triển với tầm nhìn khắc phục những hạn chế này thông qua việc áp dụng kiến trúc microservice tiên tiến. Kiến trúc này cho phép platform có khả năng scale linh hoạt theo nhu cầu thị trường, dễ dàng thích ứng với các yêu cầu đặc thù của từng vùng miền, và nhanh chóng triển khai các tính năng mới mà không ảnh hưởng đến các thành phần khác của hệ thống.
* Hơn nữa, với việc tập trung vào thị trường nội địa, Jobber không chỉ đơn thuần là một nền tảng kết nối, mà còn là một giải pháp toàn diện được tối ưu hóa cho người dùng Việt Nam. Platform này cung cấp giao diện người dùng thân thiện. Điều này tạo ra một lợi thế cạnh tranh đáng kể so với các đối thủ quốc tế, đồng thời đáp ứng được nhu cầu ngày càng tăng của thị trường freelance nội địa đang phát triển mạnh mẽ.

## Những khó khăn khi giải quyết bài toán.

## Quản lý service.

* Trong quá trình phát triển hệ thống microservice, việc quản lý và điều phối giao tiếp giữa các service là một thách thức cực kỳ phức tạp. Hãy tưởng tượng một hệ thống với 10 service khác nhau, mỗi service có 3-4 instance chạy song song để đảm bảo tính sẵn sàng cao. Mỗi service cần lưu trữ và quản lý một danh sách dài các địa chỉ IP và port của tất cả các service khác mà nó cần giao tiếp. Ví dụ, Payment Service cần biết chính xác địa chỉ của User Service, Order Service và Notification Service. Điều này dẫn đến việc phải quản lý và cập nhật hàng chục, thậm chí hàng trăm endpoint khác nhau, đặc biệt phức tạp trong môi trường cloud nơi các instance có thể được tạo hoặc hủy động để đáp ứng tải.
* Khả năng mở rộng của hệ thống cũng bị ảnh hưởng nghiêm trọng. Khi lượng người dùng tăng đột biến, việc thêm mới các instance trở thành một quy trình phức tạp và dễ xảy ra lỗi. Team vận hành phải cập nhật cấu hình của tất cả các service liên quan mỗi khi một instance mới được tạo ra. Trong môi trường production, quá trình này có thể dẫn đến downtime không mong muốn hoặc routing sai, ảnh hưởng trực tiếp đến trải nghiệm người dùng và độ tin cậy của hệ thống.
* Vấn đề về khả năng chịu lỗi cũng trở nên nghiêm trọng hơn trong môi trường phân tán. Khi một instance của một service gặp sự cố và không thể phục vụ request, các service khác vẫn tiếp tục gửi request đến instance này cho đến khi cấu hình được cập nhật thủ công. Điều này có thể dẫn đến tình trạng người dùng gặp lỗi liên tục khi thực hiện các thao tác như đăng nhập, thanh toán hay gửi tin nhắn. Đặc biệt nghiêm trọng với các giao dịch tài chính, nơi sự cố có thể dẫn đến mất mát dữ liệu hoặc trạng thái không nhất quán giữa các service.

## Quản lý cấu hình.

* Trong quá trình phát triển hệ thống microservice, việc quản lý cấu hình tập trung cho toàn bộ hệ thống là một thách thức đáng kể. Khi mỗi service lưu trữ cấu hình riêng của mình trong các file properties hoặc YAML, việc thay đổi một thông số đơn giản như connection string database hay API key của bên thứ ba trở thành một quy trình phức tạp và dễ xảy ra sai sót. Ví dụ, khi cần thay đổi mật khẩu database cho 15 service khác nhau, team vận hành phải cập nhật thủ công từng file cấu hình, sau đó triển khai lại tất cả các service liên quan. Trong môi trường production với nhiều instance của mỗi service, việc đảm bảo tất cả các instance đều nhận được cấu hình mới một cách đồng bộ là cực kỳ khó khăn. Tình huống còn trở nên phức tạp hơn khi các service được triển khai trên nhiều môi trường khác nhau như development, staging và production, mỗi môi trường lại có một bộ cấu hình riêng biệt. Việc theo dõi lịch sử thay đổi cấu hình cũng trở nên khó khăn, đặc biệt khi cần phải xác định nguyên nhân của các sự cố liên quan đến cấu hình sai. Hơn nữa, vấn đề bảo mật thông tin nhạy cảm như credentials hay API keys trở nên nghiêm trọng khi chúng được lưu trữ trực tiếp trong các file cấu hình, có thể bị lộ thông qua các repository code hoặc log files. Khi hệ thống cần scale up nhanh chóng để đáp ứng tải cao, việc đảm bảo các instance mới đều nhận được cấu hình chính xác và cập nhật là một thách thức lớn, có thể dẫn đến tình trạng một số instance hoạt động không đúng do sử dụng cấu hình cũ hoặc không đồng bộ.

## Vấn đề khi gọi tới một service.

* Trong quá trình phát triển hệ thống microservice, việc quản lý và điều phối các request API mà không có một điểm vào tập trung tạo ra nhiều thách thức phức tạp. Client (như ứng dụng web hay mobile) phải biết và quản lý địa chỉ endpoint của từng service riêng biệt, dẫn đến việc phải duy trì nhiều kết nối và xử lý authentication/authorization riêng cho từng service. Ví dụ, một ứng dụng mobile đơn giản cần tương tác với User Service để đăng nhập, Order Service để đặt hàng, và Payment Service để thanh toán sẽ phải quản lý ba endpoint khác nhau, mỗi endpoint có protocol và cơ chế bảo mật riêng. Điều này không chỉ làm tăng độ phức tạp của code client mà còn tạo ra các vấn đề về bảo mật khi mỗi service phải tự xử lý việc xác thực người dùng. Việc monitoring và debugging cũng trở nên khó khăn khi logs và metrics bị phân tán across nhiều service. Khi cần thực hiện các thay đổi về protocol hay format của API, team phải cập nhật trên từng service riêng lẻ và đồng bộ những thay đổi này với tất cả các client, tạo ra rủi ro về tính không nhất quán của API. Đặc biệt, việc implement các tính năng cross-cutting như rate limiting, caching, hay request transformation trở nên cực kỳ phức tạp vì phải được triển khai riêng lẻ trên từng service, dẫn đến việc duplicate code và khó maintain. Trong trường hợp có sự cố, việc xác định nguồn gốc vấn đề (root cause analysis) trở nên phức tạp hơn nhiều vì không có một điểm quan sát tập trung để theo dõi luồng request từ đầu đến cuối.

## Giao tiếp giữa các service với nhau.

* Trong quá trình phát triển hệ thống microservice, việc giao tiếp trực tiếp giữa các service thông qua REST API hoặc gRPC tạo ra nhiều thách thức về độ tin cậy và khả năng mở rộng. Khi Order Service cần thông báo cho Payment Service và Inventory Service về một đơn hàng mới, việc gọi API đồng bộ có thể dẫn đến tình trạng blocking và timeout nếu một trong các service gặp sự cố hoặc quá tải. Ví dụ, trong giờ cao điểm Black Friday, khi hàng nghìn đơn hàng được tạo cùng lúc, việc xử lý đồng bộ có thể khiến toàn bộ hệ thống bị chậm hoặc ngừng hoạt động. Đặc biệt nghiêm trọng khi một service trong chuỗi xử lý gặp sự cố, dẫn đến mất dữ liệu hoặc trạng thái không nhất quán giữa các service. Ví dụ, một đơn hàng có thể đã được thanh toán nhưng không cập nhật được trạng thái trong hệ thống kho do Inventory Service tạm thời không phản hồi. Việc retry các request thất bại cũng trở nên phức tạp, đòi hỏi mỗi service phải tự implement cơ chế retry logic, dễ dẫn đến việc duplicate code và khó maintain. Khi cần scale hệ thống, việc đảm bảo tất cả các service có thể xử lý tải tăng đột biến một cách đồng bộ là một thách thức lớn, có thể dẫn đến tình trạng một số service trở thành bottleneck, ảnh hưởng đến hiệu suất toàn bộ hệ thống. Hơn nữa, việc theo dõi và debug các vấn đề liên quan đến giao tiếp giữa các service trở nên khó khăn khi không có cơ chế theo dõi message flow một cách rõ ràng.

## Kiến trúc hệ thống.

## Tác nhân hệ thống.

* Khách (Guest/Visitor): Khách truy cập là những người dùng chưa đăng ký tài khoản trên hệ thống, họ có thể tự do duyệt và tìm kiếm các dịch vụ được cung cấp trên nền tảng. Với vai trò này, họ được phép xem thông tin công khai về các gig, đọc đánh giá và nhận xét từ người dùng khác, cũng như tìm hiểu về người bán thông qua hồ sơ công khai. Tuy nhiên, để thực hiện các giao dịch hay tương tác sâu hơn với nền tảng, họ cần phải đăng ký tài khoản để trở thành người dùng chính thức.
* Người mua (Buyer): Người mua là những người dùng đã đăng ký tài khoản với mục đích tìm kiếm và sử dụng dịch vụ trên nền tảng. Họ có thể tương tác trực tiếp với người bán thông qua hệ thống chat, đặt câu hỏi về dịch vụ, thực hiện đặt hàng và thanh toán. Sau khi nhận được dịch vụ, người mua có quyền đánh giá và để lại nhận xét về trải nghiệm của họ, góp phần xây dựng hệ thống đánh giá uy tín trên nền tảng. Họ cũng có thể quản lý danh sách order và theo dõi lịch sử đơn hàng của mình.
* Người bán (Seller): Người bán đóng vai trò then chốt trong việc cung cấp dịch vụ trên nền tảng, họ là những chuyên gia trong lĩnh vực của mình và có khả năng tạo ra các gig chất lượng. Thông qua việc xây dựng portfolio chuyên nghiệp và duy trì mức độ dịch vụ cao, họ tương tác với người mua, thực hiện các đơn đặt hàng, và gửi sản phẩm hoàn thiện. Người bán có thể theo dõi thu nhập, quản lý đơn hàng, và quản lý gig thông qua các tính năng được cung cấp.

## Các tính năng của hệ thống.

* Đăng ký và đăng nhập: Hệ thống cung cấp quy trình đăng ký và đăng nhập linh hoạt cho người dùng. Người dùng có thể tạo tài khoản mới thông qua form đăng ký với các thông tin cơ bản như email, mật khẩu và thông tin cá nhân. Sau khi đăng ký, người dùng cần xác thực email để kích hoạt tài khoản. Đối với đăng nhập, hệ thống hỗ trợ các tính năng bảo mật như sử dụng access token, refresh token, khôi phục mật khẩu qua email, duy trì phiên đăng nhập an toàn.
* Quản lý profile: Người bán có thể xây dựng và tùy chỉnh hồ sơ chuyên nghiệp của mình trên nền tảng. Profile bao gồm các thông tin chi tiết như giới thiệu bản thân, kỹ năng chuyên môn, kinh nghiệm làm việc, chứng chỉ và portfolio. Hệ thống cũng hiển thị các chỉ số quan trọng như lượng rating trung bình, thời gian phản hồi trung bình, và đánh giá từ khách hàng, giúp xây dựng uy tín và thu hút người mua tiềm năng.
* Tạo và quản lý gig: Người bán có thể tạo và quản lý các gig - đơn vị dịch vụ cơ bản trên nền tảng. Khi tạo gig, họ cần cung cấp thông tin chi tiết về dịch vụ như tiêu đề, mô tả, hình ảnh minh họa, giá cả và category của gig. Họ có thể thiết lập các tùy chọn bổ sung (subcategory), thời gian hoàn thành, và các yêu cầu đặc biệt từ người mua. Hệ thống cho phép người bán theo dõi hiệu suất của từng gig, cập nhật thông tin, và tạm ngưng hoặc xóa gig khi cần thiết.
* Tìm kiếm gig: Hệ thống tìm kiếm được thiết kế để giúp người dùng dễ dàng tìm thấy dịch vụ phù hợp với nhu cầu. Người dùng có thể tìm kiếm bằng từ khóa, lọc theo danh mục, giá cả, thời gian giao hàng. Kết quả tìm kiếm được sắp xếp theo độ liên quan và có thể được sắp xếp lại theo các tiêu chí khác như giá cả, thời gian giao hàng. Hệ thống cũng đề xuất các gig tương tự và gợi ý thông minh dựa trên lịch sử tìm kiếm của người dùng.
* Tạo và quản lý order: Khi người mua quyết định sử dụng một dịch vụ, họ có thể tạo order thông qua gig đã chọn. Sau khi tạo order, cả người mua và người bán có thể theo dõi tiến độ thông qua các trạng thái như: đang thực hiện, đã hủy, và hoàn thành. Hệ thống cung cấp công cụ để quản lý thời gian, trao đổi tệp tin, và yêu cầu chỉnh sửa nếu cần.
* Nhắn tin giữa những người dùng: Hệ thống chat tích hợp cho phép người dùng trao đổi trực tiếp với nhau một cách thuận tiện. Người mua có thể đặt câu hỏi về dịch vụ trước khi đặt hàng, trong khi người bán có thể làm rõ các yêu cầu và cung cấp thông tin bổ sung. Tính năng chat hỗ trợ gửi hình ảnh và offer cho người mua. Các cuộc trò chuyện được lưu trữ và có thể truy cập lại bất cứ lúc nào, đồng thời hệ thống gửi thông báo khi có tin nhắn mới.
* Rating và review: Sau khi hoàn thành đơn hàng, người mua có cơ hội đánh giá và viết nhận xét về trải nghiệm của họ. Hệ thống đánh giá bao gồm xếp hạng sao (thường từ 1-5 sao) Người mua cũng có thể viết nhận xét chi tiết về trải nghiệm của họ. Những đánh giá và nhận xét này được hiển thị công khai trên trang gig và profile của người bán, giúp xây dựng độ tin cậy và hỗ trợ quyết định cho những người mua tiềm năng trong tương lai.
* Thanh toán: Hệ thống thanh toán được thiết kế để đảm bảo an toàn và thuận tiện cho cả người mua và người bán. Người mua có thể thanh toán thông qua nhiều phương thức như thẻ tín dụng, ví điện tử, hay chuyển khoản ngân hàng. Hệ thống tự động tính toán và khấu trừ phí dịch vụ, đồng thời cung cấp các báo cáo tài chính chi tiết cho cả hai bên.

## Sơ đồ Usecase tổng quan

A diagram of a system

Description automatically generated

## Usecase đăng ký/đăng nhập

A diagram of a system

Description automatically generated

## Usecase Tìm kiếm

A diagram of a system

Description automatically generated

## Usecase quản lý Order

A diagram of a system

Description automatically generated

## Usecase become a seller

A diagram of a system

Description automatically generated

## Usecase quản lý gig

A diagram of a diagram

Description automatically generated

## Usecase process order

A diagram of a person

Description automatically generated

## Usecase process offer

A diagram of a person with a person's figure and a person's figure

Description automatically generated

## Usecase process extension

A diagram of a diagram

Description automatically generated

## Sơ đồ activities cho các luồng nghiệp vụ.

## Nghiệp vụ đăng ký và xác thực tài khoản.

A diagram of a system

Description automatically generated

## Nghiệp vụ tìm kiếm và mua dịch vụ.

A diagram with text and images

Description automatically generated with medium confidence

## Nghiệp vụ hoàn thiện sản phẩm và đánh giá

A diagram of a diagram

Description automatically generated with medium confidence

## Thiết kế cơ sở dữ liệu

## Gig domain



## User domain

A screenshot of a computer

Description automatically generated

## Chat domain

A screenshot of a computer

Description automatically generated

## Order domain

A screenshot of a computer

Description automatically generated

## Payment domain

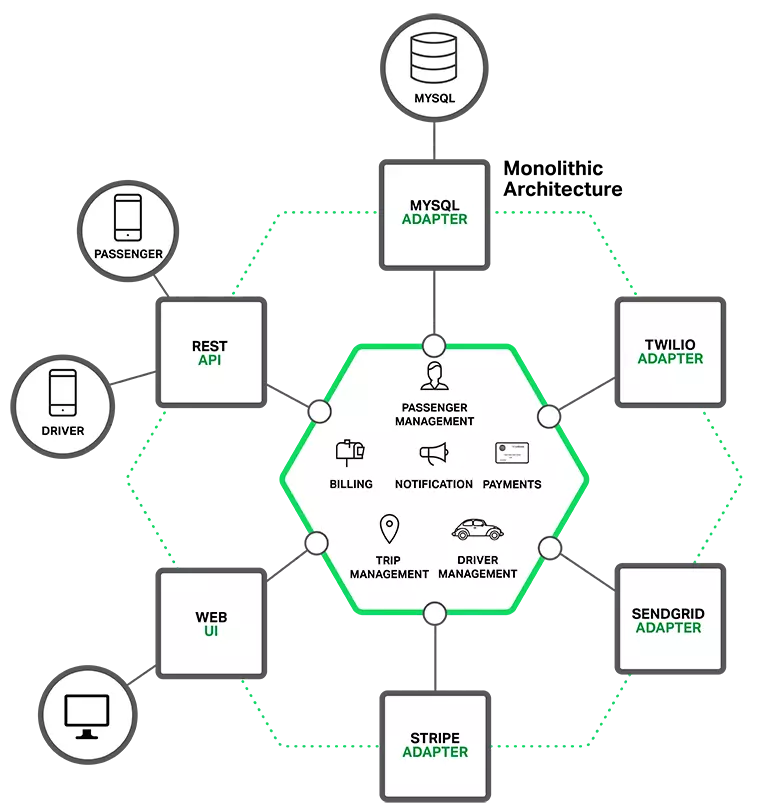
A screenshot of a computer

Description automatically generated

## Công nghệ sử dụng.

## Kiến trúc monolithic.

* Kiến trúc Monolithic là một phương pháp thiết kế phần mềm truyền thống, trong đó toàn bộ ứng dụng được xây dựng như một thể thống nhất, không thể tách rời. Trong mô hình này, tất cả các thành phần chức năng của ứng dụng được đóng gói chặt chẽ và triển khai như một đơn vị duy nhất. Điều này có nghĩa là mọi thay đổi, dù nhỏ hay lớn, đều yêu cầu phải build và triển khai lại toàn bộ ứng dụng.



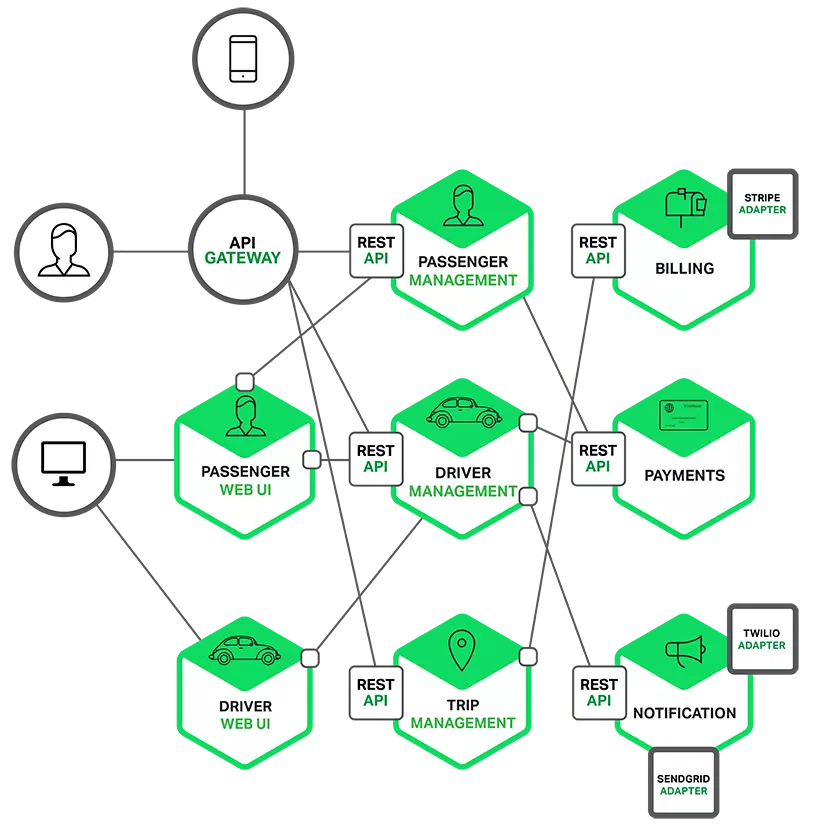
* Đặc điểm nổi bật của kiến trúc Monolithic là tính đơn giản trong việc phát triển và triển khai ban đầu. Các nhà phát triển có thể dễ dàng debug và test ứng dụng vì mọi thứ đều nằm trong cùng một codebase. Tuy nhiên, khi ứng dụng phát triển lớn dần, kiến trúc này bộc lộ nhiều hạn chế đáng kể. Việc maintain và mở rộng trở nên phức tạp hơn, thời gian build và deploy kéo dài hơn, và một lỗi nhỏ trong bất kỳ module nào cũng có thể ảnh hưởng đến toàn bộ hệ thống.
* Mặc dù có những hạn chế, kiến trúc Monolithic vẫn phù hợp với các dự án nhỏ và vừa, đặc biệt là trong giai đoạn đầu khi yêu cầu về quy mô và độ phức tạp của ứng dụng chưa cao. Đối với các doanh nghiệp khởi nghiệp hoặc các ứng dụng có quy mô nhỏ, việc sử dụng kiến trúc Monolithic có thể giúp tiết kiệm thời gian và nguồn lực, đồng thời đơn giản hóa quy trình phát triển và vận hành.

A yellow and black rectangular box with black text

Description automatically generated

## Kiến trúc microservice.

* Kiến trúc Microservice là một phương pháp thiết kế phần mềm hiện đại, trong đó ứng dụng được chia thành nhiều service nhỏ, độc lập. Như minh họa trong hình ảnh, một hệ thống có thể bao gồm nhiều thành phần riêng biệt như Passenger Management, Driver Management, Payment, và Notification, mỗi thành phần đều được kết nối thông qua API Gateway và giao tiếp qua REST API.



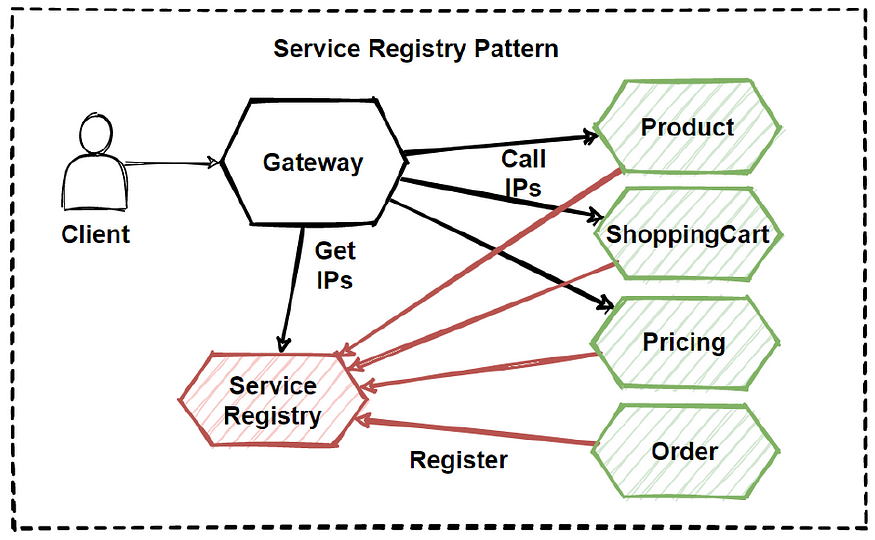
* Kiến trúc này mang đến nhiều ưu điểm nổi bật. Thứ nhất, khả năng mở rộng linh hoạt khi mỗi service có thể được phát triển và triển khai độc lập. Thứ hai, hệ thống có khả năng chống chịu lỗi tốt hơn vì sự cố ở một service không ảnh hưởng đến toàn bộ hệ thống. Thứ ba, việc hiểu và bảo trì trở nên dễ dàng hơn do mỗi service có phạm vi trách nhiệm rõ ràng. Cuối cùng, các team có thể phát triển và triển khai các service một cách độc lập, tăng tính linh hoạt trong quá trình phát triển.
* Tuy nhiên, kiến trúc này cũng đối mặt với một số thách thức đáng kể. Việc triển khai và vận hành trở nên phức tạp hơn do số lượng service lớn cần quản lý. Chi phí vận hành cao hơn do mỗi service cần có tài nguyên riêng. Đặc biệt, việc giao tiếp giữa các service trở nên phức tạp và có thể gây ra độ trễ trong hệ thống, đòi hỏi cần có chiến lược thiết kế và tối ưu hóa phù hợp.

A white and yellow rectangular sign with black text

Description automatically generated

## Service Discovery.

* Service Discovery là một thành phần quan trọng trong kiến trúc microservice, đóng vai trò như một "danh bạ thông minh" cho toàn bộ hệ thống. Đây là nơi lưu trữ và quản lý thông tin chi tiết của tất cả các service trong hệ thống, bao gồm số lượng instance được triển khai, phiên bản, và trạng thái deploy. Điểm đặc biệt của Service Discovery là khả năng cung cấp cơ chế định danh cho từng service thông qua tên gọi cụ thể, giúp các service có thể giao tiếp với nhau một cách linh hoạt mà không cần quan tâm đến địa chỉ IP - vốn thường xuyên thay đổi trong môi trường triển khai động. Hiện nay, có nhiều công cụ phổ biến hỗ trợ Service Discovery như Netflix Eureka, Apache ZooKeeper, và HashiCorp Consul, mỗi công cụ đều có những đặc điểm và ưu điểm riêng phù hợp với các nhu cầu khác nhau của hệ thống microservice.



* Netflix Eureka là một giải pháp Service Discovery mã nguồn mở được phát triển bởi Netflix, và đã trở thành một trong những công cụ phổ biến nhất trong hệ sinh thái Spring Cloud. Hoạt động theo mô hình client-server, Eureka Server đóng vai trò như một registry trung tâm, nơi các microservice (Eureka Client) có thể đăng ký thông tin của mình và khám phá các service khác trong hệ thống. Một ưu điểm nổi bật của Eureka là khả năng tự phục hồi cao (self-healing) thông qua cơ chế heartbeat, cho phép nó tự động phát hiện và loại bỏ các instance không hoạt động, đồng thời cập nhật danh sách service một cách liên tục. Eureka còn hỗ trợ nhiều zone và region khác nhau, giúp tăng tính sẵn sàng của hệ thống, đặc biệt trong môi trường cloud. Với việc tích hợp chặt chẽ vào hệ sinh thái Spring Cloud, Eureka trở thành lựa chọn tự nhiên cho các dự án Java Spring Boot, cho phép các nhà phát triển dễ dàng triển khai service discovery chỉ với vài cấu hình đơn giản.



## Centralized configuration.

* Centralized Configuration (Cấu hình tập trung) là một giải pháp quản lý cấu hình hiệu quả trong môi trường phân tán của microservices. Thay vì lưu trữ các cấu hình riêng lẻ trong mã nguồn hoặc file cấu hình local của từng service, hệ thống sử dụng một server cấu hình trung tâm (Config Server) để quản lý tất cả các cấu hình. Config Server kết nối với một kho lưu trữ cấu hình (Configuration Repository) có thể là SVN, Git hoặc FileSystem, từ đó phân phối các cấu hình tới các service khác nhau (Service A, B, Y,...) trong hệ thống. Spring Cloud Config là một trong những công cụ phổ biến được sử dụng để triển khai giải pháp này. Cách tiếp cận tập trung này mang lại nhiều lợi ích như dễ dàng quản lý, cập nhật và đồng bộ cấu hình across các service, đồng thời giảm thiểu rủi ro từ việc quản lý cấu hình phân tán.

A diagram of a service

Description automatically generated

* Spring Cloud Config là một giải pháp quản lý cấu hình tập trung mạnh mẽ trong hệ sinh thái Spring Cloud, được thiết kế đặc biệt cho các ứng dụng phân tán và microservices. Hoạt động theo mô hình client-server, Spring Cloud Config Server có khả năng tích hợp liền mạch với các hệ thống quản lý phiên bản như Git, cho phép lưu trữ và quản lý cấu hình dưới dạng mã nguồn (configuration as code). Một trong những tính năng nổi bật của Spring Cloud Config là khả năng cập nhật cấu hình động (dynamic configuration updates) thông qua Spring Cloud Bus, cho phép các service nhận được các thay đổi cấu hình mà không cần khởi động lại. Ngoài ra, nó còn hỗ trợ việc quản lý cấu hình theo môi trường (development, staging, production), theo service, và theo profile khác nhau, cùng với khả năng mã hóa các thông tin nhạy cảm thông qua tích hợp với Spring Cloud Vault. Với những tính năng này, Spring Cloud Config trở thành công cụ không thể thiếu trong việc xây dựng và vận hành các hệ thống microservice dựa trên Spring Boot.

A diagram of a company

Description automatically generated

## API Gateway.

* API Gateway đóng vai trò như một "cửa ngõ thông minh" trong kiến trúc microservices, hoạt động như điểm vào duy nhất cho tất cả các requests từ phía client. Thay vì client phải giao tiếp trực tiếp với nhiều service khác nhau như Shopping Cart, Shipping, Inventory, Product Catalog, Review, hay Order Service, mọi request đều đi qua API Gateway để được xử lý, xác thực và điều hướng đến service phù hợp. Hiện nay, có nhiều giải pháp API Gateway phổ biến được sử dụng trong thực tế như Spring Cloud Gateway (phát triển bởi Spring), Kong (một API Gateway mã nguồn mở), và APISIX (một API Gateway hiệu năng cao). Việc sử dụng API Gateway không chỉ giúp tăng cường bảo mật và kiểm soát truy cập, mà còn đơn giản hóa việc quản lý và mở rộng hệ thống microservices, đồng thời cung cấp các tính năng bổ sung như load balancing, logging, và monitoring.

A diagram of a product

Description automatically generated

* Spring Cloud Gateway là một API Gateway hiện đại, được xây dựng trên nền tảng Spring Framework, Spring Boot và Project Reactor, mang đến giải pháp mạnh mẽ cho việc định tuyến và xử lý request trong hệ thống microservices. Với kiến trúc non-blocking được xây dựng trên Netty, Spring Cloud Gateway cung cấp hiệu năng cao và khả năng xử lý đồng thời tốt hơn so với các gateway truyền thống như Zuul 1.0. Gateway này nổi bật với khả năng cấu hình linh hoạt thông qua các Predicate và Filter: Predicate xác định các điều kiện để một request được chuyển tiếp đến service cụ thể (như path, header, query parameter), trong khi Filter cho phép thực hiện các tác vụ xử lý trước và sau khi request được chuyển tiếp (như authentication, rate limiting, logging). Đặc biệt, Spring Cloud Gateway tích hợp mượt mà với các thành phần khác trong hệ sinh thái Spring Cloud như Service Discovery (Eureka) và Configuration Server, tạo nên một hệ thống gateway toàn diện và dễ dàng mở rộng cho các ứng dụng enterprise.

A green cloud with arrows and black text

Description automatically generated

## Message Queue.

* Message Queue đóng một vai trò thiết yếu trong hệ thống phân tán hiện đại, hoạt động như một "hộp thư thông minh" để lưu trữ và quản lý luồng thông điệp. Về bản chất, nó là nơi trung chuyển tin nhắn, nhận các message từ publisher (người gửi) và phân phối chúng đến các consumer (người nhận) đã đăng ký. Hiện nay, có nhiều giải pháp Message Queue phổ biến như Apache Kafka - nổi tiếng với khả năng xử lý dữ liệu lớn, RabbitMQ - được ưa chuộng bởi tính linh hoạt và dễ sử dụng, và Apache Pulsar - một nền tảng messaging hiện đại với nhiều tính năng tiên tiến. Việc sử dụng Message Queue giúp các hệ thống đạt được tính loose coupling (liên kết lỏng lẻo), đảm bảo tin nhắn được gửi đi một cách đáng tin cậy, và cho phép các thành phần trong hệ thống hoạt động độc lập mà không cần biết về sự tồn tại của nhau, từ đó tăng tính ổn định và khả năng mở rộng của toàn hệ thống.

A graphic of a mail delivery

Description automatically generated with medium confidence

## Kết quả sản phẩm.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

## Kết luận.

## Độ hoàn thiện sản phẩm.

* Dự án đã đạt được một cột mốc quan trọng khi hoàn thành toàn bộ các chức năng cốt lõi theo đúng đề xuất ban đầu đã được thống nhất với giáo viên hướng dẫn. Không chỉ dừng lại ở việc đáp ứng các yêu cầu cơ bản, nhóm đã tích cực tiếp thu và triển khai thành công các tính năng bổ sung được đề xuất thông qua các buổi họp hàng tuần. Những buổi gặp mặt định kỳ này đã tạo ra một kênh tương tác hiệu quả, nơi giáo viên có thể theo dõi tiến độ, đưa ra những góp ý có giá trị, và gợi ý các hướng phát triển mới để nâng cao chất lượng sản phẩm.
* Quá trình làm việc chặt chẽ với giáo viên hướng dẫn không chỉ giúp dự án đi đúng hướng mà còn mở rộng phạm vi của sản phẩm so với kế hoạch ban đầu. Những đề xuất được đưa ra trong các buổi họp hàng tuần đã được em nghiêm túc phân tích, đánh giá tính khả thi và triển khai một cách có hệ thống. Kết quả là một sản phẩm hoàn thiện, không chỉ đáp ứng đầy đủ các mục tiêu đề ra ban đầu mà còn được nâng cao về cả chiều rộng lẫn chiều sâu của các tính năng.
* Tuy nhiên phần code vẫn còn chưa tốt, tổ chức code chưa được tối ưu, nhiều phần bị lặp lại hoặc chưa viết được các unit test hoàn chỉnh. Chưa viết được các api doc để người khác có thể hiểu hơn về hệ thống.
* Mặc dù dự án đã đạt được các mục tiêu về chức năng, song phần triển khai kỹ thuật vẫn còn một số điểm cần được cải thiện và tối ưu hóa. Về mặt tổ chức mã nguồn, cấu trúc code hiện tại chưa thực sự theo các best practices của clean architecture, dẫn đến việc một số module còn chồng chéo và thiếu tính module hóa rõ ràng. Đáng chú ý là hiện tượng code duplication xuất hiện ở nhiều nơi, đặc biệt là trong các thành phần xử lý logic nghiệp vụ và truy cập dữ liệu, điều này không chỉ làm tăng độ phức tạp của mã nguồn mà còn gây khó khăn trong việc bảo trì và mở rộng hệ thống trong tương lai.
* Một thiếu sót đáng kể khác là phần unit test chưa được triển khai một cách toàn diện. Mặc dù đã có một số test case cơ bản, nhưng độ bao phủ (test coverage) vẫn còn thấp, chưa đảm bảo việc kiểm tra đầy đủ các trường hợp edge case và các luồng xử lý đặc biệt. Điều này tiềm ẩn rủi ro về mặt chất lượng và độ tin cậy của hệ thống, đặc biệt khi cần thực hiện các thay đổi hoặc nâng cấp trong tương lai.
* Ngoài ra, một hạn chế quan trọng là việc thiếu vắng tài liệu API một cách có hệ thống. Hiện tại, các API endpoint chưa được mô tả chi tiết thông qua các công cụ như Swagger hay OpenAPI, khiến cho việc tích hợp và sử dụng hệ thống trở nên khó khăn đối với các developer khác. Thiếu sót này không chỉ ảnh hưởng đến khả năng mở rộng và maintain của dự án, mà còn tạo ra rào cản đối với việc onboarding thành viên mới vào team hoặc việc tích hợp với các hệ thống khác.

## Những phần có thể cải tiến thêm trong tương lai.

* Để nâng cao tính cạnh tranh và đáp ứng nhu cầu ngày càng đa dạng của người dùng, dự án cần được mở rộng với nhiều tính năng hiện đại và thiết yếu. Một trong những tính năng quan trọng cần được tích hợp là hệ thống video call thời gian thực, cho phép freelancer và khách hàng có thể tương tác trực tiếp thông qua các cuộc họp online, tạo điều kiện thuận lợi cho việc trao đổi yêu cầu, demo sản phẩm và đánh giá tiến độ công việc. Tính năng này có thể được xây dựng dựa trên các công nghệ WebRTC hiện đại, đảm bảo chất lượng stream ổn định và bảo mật cao.
* Về mặt thanh toán, việc tích hợp đa dạng các phương thức thanh toán nội địa là một ưu tiên hàng đầu. Điều này bao gồm việc kết nối với các cổng thanh toán phổ biến tại Việt Nam như VNPay, Momo, ZaloPay, và các ngân hàng nội địa thông qua API banking. Hệ thống thanh toán cần được thiết kế với tính năng escrow (ký quỹ) để đảm bảo an toàn cho cả người thuê và freelancer, đồng thời tích hợp các tính năng theo dõi giao dịch và báo cáo tài chính chi tiết.
* Về mặt DevOps, dự án cần được nâng cấp toàn diện với việc thiết lập quy trình CI/CD (Continuous Integration/Continuous Deployment) chuyên nghiệp. Điều này bao gồm việc tự động hóa các quy trình build, test và deploy code lên các môi trường khác nhau (development, staging, production) thông qua các công cụ như Jenkins hoặc GitLab CI. Việc tích hợp SonarQube vào pipeline sẽ giúp đảm bảo chất lượng code thông qua việc phân tích tĩnh mã nguồn, phát hiện các lỗi tiềm ẩn, đo lường độ phức tạp của code và đảm bảo tuân thủ các coding standards.
* Quy trình deployment cần được tự động hóa và tối ưu hóa thông qua việc sử dụng container hóa với Docker và orchestration với Kubernetes, đảm bảo khả năng scale linh hoạt và high availability cho hệ thống. Bên cạnh đó, việc thiết lập hệ thống monitoring và logging comprehensive với các công cụ như Prometheus, Grafana và ELK stack sẽ giúp team có thể theo dõi hiệu suất hệ thống, phát hiện và xử lý các vấn đề một cách chủ động. Các metrics quan trọng như response time, error rate, resource utilization cần được theo dõi và cảnh báo kịp thời để đảm bảo trải nghiệm người dùng tốt nhất.

## Mã nguồn.