MỤC LỤC

1. Why do we need to study Software Architecture and Design?

1.1. Giới thiệu

Trong kỷ nguyên kỹ thuật số hiện đại, phần mềm đã trở thành động lực quan trọng thúc đẩy đổi mới và hiệu quả trong mọi lĩnh vực. Từ các tác vụ cá nhân hàng ngày đến các quy trình công nghiệp phức tạp, phần mềm đóng vai trò trung tâm trong hầu hết các khía cạnh của xã hội hiện đại. Việc nghiên cứu kiến trúc và thiết kế phần mềm là cần thiết để hiểu rõ cách xây dựng các hệ thống phần mềm có thể đáp ứng được sự phức tạp ngày càng tăng và thích ứng với các thách thức công nghệ trong tương lai.

Các doanh nghiệp hiện đại, cơ quan chính phủ và các tổ chức giáo dục đều phụ thuộc vào phần mềm để vận hành một cách suôn sẻ và hiệu quả. Việc có kiến thức vững về kiến trúc và thiết kế phần mềm giúp đảm bảo rằng các hệ thống này có thể phát triển, duy trì và mở rộng theo nhu cầu kinh doanh và công nghệ.

Việc nghiên cứu kiến trúc và thiết kế phần mềm là không thể thiếu để phát triển các hệ thống có thể mở rộng, dễ bảo trì và đáng tin cậy. Hiểu biết toàn diện về các lĩnh vực này trang bị cho các chuyên gia khả năng xây dựng các hệ thống có thể quản lý hiệu quả sự phức tạp ngày càng tăng và thích ứng với các thách thức công nghệ trong tương lai. Điều này không chỉ định hướng cho phân tích tiếp theo mà còn nhấn mạnh lợi ích lâu dài của việc thành thạo các lĩnh vực thiết yếu này, đảm bảo rằng phần mềm có thể phát triển cùng với các đổi mới và nhu cầu thị trường mới nổi.

1.2. Hiểu biết về Kiến trúc và thiết kế phần mềm

Kiến trúc phần mềm là bản thiết kế tổng thể của một hệ thống—nó xác định cấu trúc cấp cao, phác thảo các thành phần chính, mối quan hệ của chúng và khung tổng thể trong đó hệ thống hoạt động. Định nghĩa này bao gồm cái nhìn khái niệm về hệ thống, quy định cách các phần khác nhau kết nối, giao tiếp và hợp tác để đáp ứng các yêu cầu chức năng và phi chức năng tổng thể. Ngược lại, thiết kế phần mềm đi sâu vào quá trình chi tiết hóa các giải pháp cụ thể để đáp ứng các yêu cầu của hệ thống. Nó liên quan đến việc đưa ra các quyết định cụ thể về thuật toán, cấu trúc dữ liệu, hệ thống phân cấp lớp và giao diện, cùng nhau biến tầm nhìn kiến trúc trừu tượng thành một hình thức cụ thể và có thể thực thi. Trong khi kiến trúc thiết lập nền tảng với góc nhìn vĩ mô, thiết kế tập trung vào các chi tiết vi mô đảm bảo mỗi thành phần hoạt động như mong muốn.

Mặc dù khác biệt, kiến trúc phần mềm và thiết kế phần mềm có mối quan hệ sâu sắc và đóng vai trò bổ sung trong quá trình phát triển. Kiến trúc đặt nền móng bằng cách thiết lập một cấu trúc mạch lạc hướng dẫn cách hệ thống nên được xây dựng, giống như bản thiết kế cho một tòa nhà. Nó xác định cách sắp xếp tổng thể và các con đường quan trọng mà dữ liệu và điều khiển lưu thông trong hệ thống. Mặt khác, thiết kế tiếp nhận các quyết định cấp cao này và điền vào chi tiết—xử lý các cụ thể về cách mỗi thành phần được triển khai, tinh chỉnh và tích hợp. Sự phân chia trách nhiệm này đảm bảo rằng hệ thống vừa vững chắc vừa linh hoạt. Về bản chất, trong khi kiến trúc cung cấp tầm nhìn chiến lược và tính toàn vẹn cấu trúc

của hệ thống, thiết kế chuyển hóa tầm nhìn đó thành mã thực tế, hoạt động. Cùng nhau, chúng tạo thành một mối quan hệ cộng sinh quan trọng để phát triển các hệ thống phần mềm có thể mở rộng, dễ bảo trì và hiệu quả.

1.3. Lợi ích của việc nghiên cứu SAD

Nghiên cứu toàn diện về kiến trúc và thiết kế phần mềm trực tiếp góp phần nâng cao chất lượng tổng thể của các hệ thống phần mềm. Bằng cách lập kế hoạch và cấu trúc hệ thống một cách cẩn thận, các kiến trúc sư và nhà thiết kế có thể cải thiện đáng kể hiệu suất thông qua việc tối ưu hóa quản lý tài nguyên và luồng công việc. Tính mở rộng được giải quyết bằng cách thiết kế các hệ thống có thể phát triển và xử lý tải tăng mà không làm giảm hiệu quả. Hơn nữa, bảo mật được củng cố khi các kiến trúc mạnh mẽ tích hợp các biện pháp bảo mật và thực hành tốt ngay từ đầu quá trình phát triển. Cuối cùng, khả năng bảo trì được đảm bảo bằng cách tạo ra các cấu trúc rõ ràng, được tài liệu hóa tốt, giúp đơn giản hóa việc cập nhật và khắc phục sự cố, qua đó giảm chi phí dài hạn và cải thiện độ bền của hệ thống.

Tài liệu rõ ràng và các ký hiệu thiết kế chuẩn hóa đóng vai trò như một ngôn ngữ chung cho tất cả các bên liên quan trong dự án, bao gồm các nhà phát triển, quản lý dự án và khách hàng. Sự hiểu biết chung này rất quan trọng trong việc điều chỉnh kỳ vọng và đảm bảo rằng tất cả những người tham gia dự án đều hiểu rõ về mục tiêu và tiến trình công việc. Khi các quyết định về kiến trúc và chi tiết thiết kế được tài liệu hóa tốt, chúng không chỉ giúp đơn giản hóa quá trình phát triển mà còn giúp giải quyết các xung đột và hiểu lầm. Điều này dẫn đến sự hợp tác hiệu quả hơn, giải quyết vấn đề nhanh chóng, và khả năng sản phẩm cuối cùng đáp ứng đúng yêu cầu và tiêu chuẩn chất lượng.

Nghiên cứu về kiến trúc và thiết kế phần mềm trang bị cho các chuyên gia kỹ năng để dự đoán các thách thức tiềm ẩn trước khi chúng trở thành các vấn đề lớn. Bằng cách phân tích yêu cầu hệ thống và sự phụ thuộc trong giai đoạn phát triển ban đầu, các kiến trúc sư có thể xác định các rủi ro như tắc nghẽn hiệu suất, vấn đề tích hợp hoặc lỗ hổng bảo mật. Tầm nhìn này giúp quản lý rủi ro một cách chủ động và đưa ra các quyết định có cơ sở, từ đó giúp giảm nợ kỹ thuật — chi phí tái thiết kế các phần của hệ thống do thiết kế ban đầu không đủ tốt. Cuối cùng, những thực hành này dẫn đến các hệ thống bền vững hơn, tốt hơn trong việc thích ứng với các thay đổi và giảm thiểu các vấn đề không lường trước.

Thiết kế mô-đun là một nguyên lý cơ bản hỗ trợ phát triển phần mềm hiệu quả và bền vững. Bằng cách phân tách hệ thống thành các thành phần nhỏ, tự chứa, các nhà phát triển có thể tái sử dụng các mô-đun này qua các dự án khác nhau, giúp giảm đáng kể thời gian và chi phí phát triển. Phương pháp mô-đun này không chỉ đơn giản hóa việc bảo trì — vì các mô-đun riêng lẻ có thể được cập nhật hoặc thay thế độc lập — mà còn tăng cường khả năng thích ứng của phần mềm để đáp ứng các tính năng mới hoặc thay đổi công nghệ. Về cơ bản, việc tập trung vào tính tái sử dụng và mô-đun dẫn đến các hệ thống linh hoạt hơn, có thể phát triển theo thời gian mà không cần thay đổi toàn bộ.

1.4. Nghiên cứu và ứng dụng thực tế

Trong lĩnh vực phần mềm hiện đại, nhiều hệ thống quy mô lớn minh họa sức mạnh của các nguyên lý kiến trúc vững chắc. Ví dụ, các công ty như Netflix và Amazon đã triển khai kiến trúc microservices để xử lý hàng triệu giao dịch và luồng dữ liệu đồng thời. Những

hệ thống này được thiết kế với ranh giới mô-đun rõ ràng, cho phép mỗi dịch vụ hoạt động độc lập trong khi tương tác liền mạch với các dịch vụ khác. Thành công của các kiến trúc này nằm ở khả năng hỗ trợ tính mở rộng, khả năng chịu lỗi và triển khai nhanh chóng. Bằng cách tách rời các dịch vụ, những hệ thống này không chỉ đạt hiệu suất cao dưới tải nặng mà còn tạo điều kiện cho việc bảo trì và phát triển theo thời gian.

Ngược lại, có những trường hợp đáng chú ý khi các quyết định kiến trúc kém đã dẫn đến những thất bại nghiêm trọng và tốn kém. Trong nhiều trường hợp, các hệ thống được thiết kế mà không xem xét đúng đắn về mô-đun hóa hoặc khả năng mở rộng đã gặp phải các nút thắt hiệu suất, nợ kỹ thuật tăng cao và thậm chí là sự cố hệ thống hoàn toàn. Ví dụ, các ứng dụng monolithic kế thừa không được thiết kế để xử lý tải người dùng tăng đã thường xuyên yêu cầu cải tiến tốn kém và mất thời gian. Những thất bại này nhấn mạnh tầm quan trọng của một kiến trúc được suy nghĩ kỹ lưỡng, dự đoán trước các yêu cầu và thách thức trong tương lai. Chúng đóng vai trò như những câu chuyện cảnh báo, nhắc nhở chúng ta rằng việc bỏ qua các thực hành kiến trúc tốt có thể dẫn đến sự kém hiệu quả, chi phí bảo trì cao hơn và giảm khả năng cạnh tranh trong một thị trường phát triển nhanh chóng.

Kiến trúc phần mềm vững chắc đóng vai trò quan trọng trong việc thúc đẩy đổi mới. Một nền tảng kiến trúc vững chắc không chỉ hỗ trợ các nhu cầu hiện tại của hệ thống mà còn cung cấp tính linh hoạt để tích hợp các công nghệ mới nổi và mô hình kinh doanh mới. Ví dụ, các công ty đầu tư vào kiến trúc linh hoạt có thể nhanh chóng tích hợp các tiến bộ trong trí tuệ nhân tạo, điện toán đám mây và Internet vạn vật (IoT) vào sản phẩm của họ. Khả năng thích ứng này cho phép các tổ chức phản ứng hiệu quả hơn với xu hướng thị trường và nhu cầu của khách hàng, thúc đẩy văn hóa cải tiến liên tục và đổi mới. Cuối cùng, một kiến trúc hướng tới tương lai cho phép các nhà phát triển thử nghiệm và đổi mới mà không làm nguy hiểm đến sự ổn định và hiệu suất của hệ thống tổng thể.

1.5. Ý nghĩa tương lai trong phát triển phần mềm

Sự phát triển nhanh chóng của công nghệ đã mang đến những xu hướng chuyển đổi như Internet vạn vật (IoT), trí tuệ nhân tạo (AI) và điện toán đám mây. Những công nghệ này đòi hỏi các phương pháp mới trong kiến trúc và thiết kế phần mềm để giải quyết hiệu quả các thách thức đặc thù của chúng. Ví dụ, các hệ thống IoT yêu cầu kiến trúc có khả năng xử lý hàng triệu thiết bị kết nối, đảm bảo dữ liệu được xử lý một cách đáng tin cậy và theo thời gian thực. Tương tự, các ứng dụng AI cần kiến trúc hiệu suất cao để quản lý khối lượng dữ liệu lớn và các tác vụ học máy phức tạp. Điện toán đám mây đã định nghĩa lại việc cung cấp dịch vụ bằng cách nhấn mạnh tính mở rộng và đàn hồi, điều này đòi hỏi các hệ thống phải vừa bền bỉ vừa linh hoạt. Trong bối cảnh này, thiết kế kiến trúc vững chắc đóng vai trò quan trọng, cung cấp nền tảng vững chắc để tích hợp và hỗ trợ các công nghệ mới nổi, đảm bảo hệ thống duy trì tính linh hoạt và hiệu quả trong môi trường thay đổi.

Khi công nghệ tiếp tục phát triển với tốc độ chưa từng có, việc học tập liên tục và phát triển kỹ năng trở nên thiết yếu đối với các chuyên gia trong lĩnh vực này. Sự xuất hiện của các khung công tác mới, ngôn ngữ lập trình và phương pháp thiết kế có nghĩa là kiến thức có thể nhanh chóng trở nên lỗi thời. Bằng cách cập nhật với các xu hướng và thực hành tốt nhất trong kiến trúc và thiết kế phần mềm, các nhà phát triển và kiến trúc sư không chỉ nâng cao khả năng xây dựng các hệ thống đổi mới mà còn đảm bảo họ có thể thích ứng với các thách

thức chưa lường trước. Việc học tập liên tục này thúc đẩy tư duy linh hoạt và khả năng phục hồi, cho phép các chuyên gia tích hợp các giải pháp mới vào hệ thống hiện có và duy trì tính cạnh tranh trong một môi trường công nghệ thay đổi nhanh chóng. Cuối cùng, cam kết học tập liên tục là rất quan trọng để duy trì hiểu biết sâu sắc và thực tế về các nguyên lý cơ bản của thiết kế hệ thống hiệu quả.

Việc thành thạo các chi tiết của kiến trúc và thiết kế phần mềm mang lại lợi thế đáng kể cho sự phát triển nghề nghiệp lâu dài. Hiểu biết sâu sắc về các lĩnh vực này mở ra cơ hội nghề nghiệp nâng cao, như vai trò kiến trúc sư hệ thống, vị trí lãnh đạo dự án và vai trò cố vấn kỹ thuật chiến lược trong các tổ chức. Các chuyên gia sở hữu kỹ năng kiến trúc và thiết kế vững chắc thường có vị trí tốt hơn để thúc đẩy đổi mới, hướng dẫn các thành viên trong nhóm và ảnh hưởng đến hướng đi tổng thể của các sáng kiến phát triển phần mềm. Chuyên môn của họ không chỉ dẫn đến các hệ thống bền bỉ và có thể mở rộng hơn mà còn trở thành tài sản quan trọng trong việc đưa ra các quyết định chiến lược định hình tương lai công nghệ trong tổ chức của họ. Theo cách này, nền tảng vững chắc trong kiến trúc và thiết kế phần mềm không chỉ mang lại lợi ích cho con đường sự nghiệp cá nhân mà còn đóng góp vào sự tiến bộ và phát triển của toàn bộ lĩnh vực.

1.6. Kết luận

Việc nghiên cứu kiến trúc và thiết kế phần mềm đóng vai trò quan trọng trong việc phát triển các hệ thống phần mềm hiệu quả và bền vững. Kiến trúc phần mềm cung cấp cấu trúc tổng thể, xác định các thành phần chính và mối quan hệ giữa chúng, trong khi thiết kế phần mềm tập trung vào việc triển khai chi tiết các thành phần đó. Sự hiểu biết sâu sắc về cả hai lĩnh vực này giúp cải thiện hiệu suất, khả năng mở rộng, bảo mật và khả năng bảo trì của hệ thống. Ngoài ra, việc nghiên cứu còn hỗ trợ giao tiếp hiệu quả giữa các bên liên quan và quản lý rủi ro một cách chủ động.

Trong thực tế, nhiều hệ thống lớn như Netflix và Amazon đã áp dụng kiến trúc microservices để xử lý hàng triệu giao dịch và luồng dữ liệu đồng thời, cho thấy tầm quan trọng của việc thiết kế kiến trúc phù hợp. Ngược lại, những quyết định kiến trúc kém có thể dẫn đến các vấn đề nghiêm trọng như tắc nghẽn hiệu suất và tăng nợ kỹ thuật. Do đó, việc nghiên cứu và áp dụng kiến trúc và thiết kế phần mềm là cần thiết để phát triển các hệ thống phần mềm mạnh mẽ, có khả năng thích ứng với các công nghệ mới và đáp ứng nhu cầu thay đổi của thị trường.

Tóm lại, việc nghiên cứu kiến trúc và thiết kế phần mềm không chỉ là một hoạt động học thuật mà còn là thực hành thiết yếu, giúp các chuyên gia điều hướng sự phức tạp của phát triển phần mềm hiện đại, đảm bảo rằng các hệ thống phần mềm có thể phát triển và duy trì hiệu quả trong môi trường công nghệ thay đổi nhanh chóng.

2. Monolithic and Microservice. Microservice and AGILE

2.1. So sánh giữa Kiến trúc Monolithic và Microservices

2.1.1. Định nghĩa và Tổng quan

2.1.1.1. Kiến trúc Monolithic

Kiến trúc Monolithic là một phương pháp phát triển phần mềm truyền thống, trong đó toàn bộ ứng dụng được xây dựng như một khối duy nhất. Tất cả các thành phần và chức năng của ứng dụng, bao gồm giao diện người dùng, logic nghiệp vụ và truy cập dữ liệu, đều được tích hợp chặt chẽ trong một mã nguồn duy nhất. Điều này có nghĩa là mọi phần của ứng dụng đều liên kết mật thiết với nhau, tạo thành một thể thống nhất.

Với cấu trúc đơn khối, việc thiết kế, phát triển và triển khai ban đầu trở nên dễ dàng hơn. Tất cả các thành phần được quản lý trong một dự án duy nhất, giúp giảm thiểu sự phức tạp trong giai đoạn đầu. Các nhà phát triển có thể tập trung vào một mã nguồn chung, đơn giản hóa quy trình phát triển và kiểm thử. Điều này đặc biệt hữu ích cho các ứng dụng nhỏ, nơi mà yêu cầu về tính linh hoạt và mở rộng không quá cao.

Tuy nhiên, khi ứng dụng ngày càng phức tạp, kích thước mã nguồn tăng lên, dẫn đến khó khăn trong việc quản lý, bảo trì và mở rộng. Một thay đổi nhỏ trong một phần của ứng dụng có thể ảnh hưởng đến toàn bộ hệ thống, đòi hỏi phải kiểm tra và triển khai lại toàn bộ ứng dụng. Điều này không chỉ tốn kém thời gian mà còn tăng nguy cơ phát sinh lỗi trong quá trình triển khai. Hơn nữa, việc mở rộng ứng dụng để đáp ứng nhu cầu ngày càng tăng trở nên khó khăn, do tất cả các thành phần đều phụ thuộc lẫn nhau và không thể mở rộng một cách độc lập.

2.1.1.2. Kiến trúc Microservices

Kiến trúc Microservices là một phương pháp phát triển phần mềm hiện đại, trong đó ứng dụng được chia thành nhiều dịch vụ nhỏ, độc lập. Mỗi dịch vụ đảm nhận một chức năng cụ thể và có thể được phát triển, triển khai một cách độc lập. Các dịch vụ này giao tiếp với nhau thông qua các giao diện lập trình ứng dụng (API) hoặc các giao thức truyền thông nhẹ như HTTP.

Mỗi dịch vụ có thể được phát triển, triển khai và mở rộng độc lập: Do tính độc lập của từng dịch vụ, các nhóm phát triển có thể làm việc song song trên các dịch vụ khác nhau, sử dụng các công nghệ và ngôn ngữ lập trình phù hợp nhất cho từng dịch vụ. Điều này giúp tăng tốc độ phát triển và linh hoạt trong việc triển khai.

Yêu cầu quản lý phức tạp hơn do số lượng dịch vụ tăng lên: Việc chia nhỏ ứng dụng thành nhiều dịch vụ dẫn đến sự phức tạp trong quản lý, giám sát và giao tiếp giữa các dịch vụ. Cần có các công cụ và chiến lược phù hợp để quản lý hiệu quả hệ thống phân tán này.

2.1.2. Ưu và Nhược điểm

2.1.2.1. Kiến trúc Monolithic

***Ưu điểm**:*

Phát triển ban đầu nhanh chóng và đơn giản: Với cấu trúc đơn khối, việc phát triển ban đầu trở nên dễ dàng hơn do tất cả các thành phần được tích hợp trong một dự án duy nhất. Điều này giúp giảm thiểu sự phức tạp và tăng tốc quá trình phát triển ban đầu.

Dễ dàng kiểm tra và triển khai khi ứng dụng còn nhỏ: Vì toàn bộ ứng dụng nằm trong một khối duy nhất, việc kiểm tra và triển khai trở nên đơn giản hơn. Các công cụ và quy trình kiểm thử có thể áp dụng trực tiếp lên toàn bộ ứng dụng mà không cần phải xử lý sự phức tạp của nhiều dịch vụ riêng lẻ.

***Nhược điểm**:*

Khó mở rộng và duy trì khi ứng dụng trở nên phức tạp: Khi ứng dụng phát triển và trở nên phức tạp, kích thước mã nguồn tăng lên, dẫn đến khó khăn trong việc quản lý, bảo trì và mở rộng. Việc thêm tính năng mới hoặc sửa lỗi có thể ảnh hưởng đến toàn bộ hệ thống, làm tăng nguy cơ phát sinh lỗi và giảm hiệu suất.

Một thay đổi nhỏ có thể ảnh hưởng đến toàn bộ hệ thống: Do tất cả các thành phần được tích hợp chặt chẽ, một thay đổi nhỏ trong một phần của ứng dụng có thể gây ra hiệu ứng domino, ảnh hưởng đến các phần khác và thậm chí toàn bộ hệ thống. Điều này đòi hỏi phải kiểm tra và triển khai lại toàn bộ ứng dụng sau mỗi thay đổi, tốn kém thời gian và nguồn lực.

2.1.2.1. Kiến trúc Microservices

***Ưu điểm**:*

Tăng khả năng mở rộng và linh hoạt trong phát triển: Mỗi dịch vụ trong kiến trúc Microservices có thể được phát triển, triển khai và mở rộng độc lập. Điều này cho phép các nhóm phát triển làm việc song song trên các dịch vụ khác nhau, tăng tốc độ phát triển và dễ dàng mở rộng hệ thống khi cần thiết.

Mỗi dịch vụ có thể được phát triển bằng ngôn ngữ và công nghệ phù hợp nhất: Kiến trúc Microservices cho phép mỗi dịch vụ được xây dựng bằng ngôn ngữ lập trình và công nghệ phù hợp nhất với yêu cầu cụ thể của dịch vụ đó. Điều này mang lại sự linh hoạt trong việc lựa chọn công nghệ và tối ưu hóa hiệu suất cho từng phần của ứng dụng.

***Nhược điểm**:*

Phức tạp trong quản lý và triển khai do số lượng dịch vụ lớn: Việc chia ứng dụng thành nhiều dịch vụ nhỏ dẫn đến sự phức tạp trong quản lý, giám sát và triển khai. Cần có các công cụ và quy trình phù hợp để quản lý hiệu quả các dịch vụ độc lập này, đảm bảo chúng hoạt động hài hòa và hiệu quả trong hệ thống tổng thể.

Yêu cầu cơ sở hạ tầng và công cụ hỗ trợ phù hợp: Để triển khai và vận hành kiến trúc Microservices hiệu quả, cần có cơ sở hạ tầng mạnh mẽ và các công cụ hỗ trợ như hệ thống quản lý container (ví dụ: Docker, Kubernetes), dịch vụ khám phá (service discovery), và các giải pháp giám sát, logging. Việc thiết lập và duy trì cơ sở hạ tầng này đòi hỏi chi phí và nguồn lực đáng kể.

2.1.3. Khi nào nên sử dụng từng kiến trúc

2.1.3.1. Sử dụng Kiến trúc Monolithic

Phù hợp với ứng dụng nhỏ hoặc khi bắt đầu dự án với nguồn lực hạn chế: Đối với các dự án có phạm vi nhỏ, yêu cầu đơn giản và đội ngũ phát triển hạn chế, kiến trúc Monolithic là lựa chọn hợp lý. Cấu trúc đơn khối giúp giảm thiểu sự phức tạp trong phát triển và triển khai ban đầu, tiết kiệm thời gian và chi phí.

Khi yêu cầu về mở rộng và thay đổi không cao: Nếu ứng dụng dự kiến không cần mở rộng quy mô lớn hoặc thay đổi thường xuyên, kiến trúc Monolithic cung cấp sự ổn định và đơn giản trong quản lý. Việc duy trì và kiểm thử cũng trở nên dễ dàng hơn khi toàn bộ ứng dụng nằm trong một khối duy nhất.

2.1.3.2. Sử dụng Kiến trúc Microservices

Khi ứng dụng dự kiến phát triển lớn và phức tạp: Đối với các hệ thống lớn, phức tạp và có khả năng mở rộng cao, kiến trúc Microservices cho phép chia nhỏ ứng dụng thành các dịch vụ độc lập, giúp dễ dàng quản lý, phát triển và mở rộng từng phần mà không ảnh hưởng đến toàn bộ hệ thống.

Khi cần linh hoạt trong phát triển, triển khai và mở rộng: Kiến trúc Microservices cho phép các nhóm phát triển làm việc song song trên các dịch vụ khác nhau, sử dụng công nghệ phù hợp cho từng dịch vụ. Điều này tăng tính linh hoạt trong phát triển và triển khai, đồng thời cho phép mở rộng từng dịch vụ độc lập theo nhu cầu.

2.2. Mối quan hệ giữa Microservices và Phương pháp Agile

2.2.1. Tổng quan về Phương pháp Agile

Phương pháp Agile là một triết lý và phương pháp phát triển phần mềm linh hoạt, được thiết kế để đáp ứng nhanh chóng các thay đổi và yêu cầu từ khách hàng thông qua các chu kỳ phát triển ngắn gọn và liên tục cải tiến. Ra đời vào năm 2001 với bản Tuyên ngôn Agile, trở thành một tiêu chuẩn trong lĩnh vực phát triển phần mềm hiện đại.

Agile tập trung vào việc phát hành nhanh chóng các phiên bản phần mềm thông qua các chu kỳ phát triển ngắn, thường được gọi là "sprint". Mỗi chu kỳ kéo dài từ 1-4 tuần, trong đó một phần chức năng hoàn chỉnh của sản phẩm được phát triển, cung cấp cho khách hàng. Điều này cho phép thu thập phản hồi kịp thời, điều chỉnh sản phẩm theo nhu cầu thực tế.

Nguyên tắc chính: Phương pháp Agile dựa trên bốn giá trị cốt lõi sau:

1. **Cá nhân và sự tương tác hơn là quy trình và công cụ:** Agile đề cao vai trò của con người và sự giao tiếp trong nhóm. Mặc dù quy trình và công cụ hỗ trợ quan trọng,

nhưng sự hợp tác và tương tác giữa các thành viên mới là yếu tố quyết định thành công của dự án.

2. **Phần mềm hoạt động hơn là tài liệu đầy đủ:** Thay vì tập trung vào việc tạo ra tài liệu chi tiết, Agile ưu tiên việc phát triển phần mềm có chức năng và đáp ứng nhu cầu người dùng. Tài liệu chỉ nên đủ để hỗ trợ phát triển và sử dụng sản phẩm.
3. **Hợp tác với khách hàng hơn là đàm phán hợp đồng:** Agile khuyến khích sự tham gia liên tục của khách hàng trong quá trình phát triển. Thay vì chỉ dựa trên các điều khoản hợp đồng, việc hợp tác chặt chẽ với khách hàng giúp đảm bảo sản phẩm cuối cùng phù hợp với mong đợi và yêu cầu thực tế.
4. **Phản hồi với thay đổi hơn là tuân theo kế hoạch:** Trong môi trường kinh doanh luôn biến đổi, Agile chấp nhận và linh hoạt trước các thay đổi. Thay vì cố gắng tuân thủ một kế hoạch cố định, nhóm phát triển sẵn sàng điều chỉnh hướng đi dựa trên phản hồi và tình hình mới.

2.2.2. Cách Microservices hỗ trợ Phương pháp Agile

2.2.2.1. Phát triển và Triển khai Nhanh chóng

Kiến trúc Microservices đóng vai trò quan trọng trong việc hỗ trợ phương pháp Agile, đặc biệt trong việc phát triển và triển khai nhanh chóng. Dưới đây là phân tích chi tiết về cách Microservices thúc đẩy quá trình này:

1. Phát triển và triển khai độc lập

- **Phát triển độc lập:** Trong kiến trúc Microservices, ứng dụng được chia thành nhiều dịch vụ nhỏ, mỗi dịch vụ đảm nhận một chức năng cụ thể và có thể được phát triển độc lập. Điều này cho phép các nhóm phát triển làm việc song song trên các dịch vụ khác nhau mà không gây ảnh hưởng lẫn nhau, tăng tốc độ phát triển tổng thể của dự án.
- **Triển khai độc lập:** Mỗi dịch vụ có thể được triển khai riêng biệt, giúp giảm thiểu rủi ro và thời gian khi triển khai các tính năng mới hoặc cập nhật. Nếu một dịch vụ gặp sự cố, chỉ cần triển khai lại dịch vụ đó mà không ảnh hưởng đến toàn bộ hệ thống. Điều này hỗ trợ mạnh mẽ cho việc triển khai liên tục (Continuous Deployment) trong phương pháp Agile.

2. Thử nghiệm và phát hành nhanh chóng

- **Thử nghiệm nhanh chóng:** Với Microservices, các dịch vụ nhỏ gọn và độc lập dễ dàng được kiểm thử riêng lẻ. Việc này giúp phát hiện và khắc phục lỗi nhanh chóng, đảm bảo chất lượng phần mềm trước khi tích hợp vào hệ thống chính.
- **Phát hành tính năng mới:** Do tính chất độc lập của từng dịch vụ, các tính năng mới có thể được phát triển và triển khai mà không cần chờ đợi các phần khác của ứng dụng. Điều này cho phép doanh nghiệp đáp ứng nhanh chóng nhu cầu thị trường và phản hồi từ khách hàng, phù hợp với nguyên tắc linh hoạt và phản ứng nhanh của Agile.

Tóm lại, sự kết hợp giữa kiến trúc Microservices và phương pháp Agile tạo ra một môi trường phát triển linh hoạt, hiệu quả và nhanh chóng, giúp các tổ chức đáp ứng kịp thời các thay đổi và yêu cầu từ thị trường.

2.2.2.2. Tăng cường Tính Linh Hoạt và Thay đổi

Kiến trúc Microservices đóng vai trò quan trọng trong việc tăng cường tính linh hoạt và khả năng thích ứng của hệ thống phần mềm. Dưới đây là phân tích chi tiết về cách Microservices hỗ trợ trong việc thay đổi và phản ứng nhanh với yêu cầu từ khách hàng hoặc thị trường:

1. Dễ dàng thay đổi hoặc thay thế các dịch vụ mà không ảnh hưởng đến toàn bộ hệ thống:

- **Tính độc lập của dịch vụ:** Trong kiến trúc Microservices, mỗi dịch vụ được thiết kế để hoạt động độc lập, đảm nhận một chức năng cụ thể trong hệ thống. Điều này cho phép các nhà phát triển thay đổi, cập nhật hoặc thậm chí thay thế một dịch vụ mà không cần phải can thiệp vào các phần khác của ứng dụng. Sự tách biệt này giúp giảm thiểu rủi ro và đảm bảo rằng các thay đổi cục bộ không gây ảnh hưởng đến toàn bộ hệ thống.
- **Triển khai độc lập:** Mỗi dịch vụ có thể được triển khai riêng lẻ, cho phép các nhóm phát triển cập nhật hoặc mở rộng một phần cụ thể của ứng dụng mà không cần phải triển khai lại toàn bộ hệ thống. Điều này không chỉ tăng tốc độ triển khai mà còn giảm thiểu thời gian ngừng hoạt động và rủi ro liên quan đến việc triển khai.

2. Hỗ trợ phản ứng nhanh với yêu cầu thay đổi từ khách hàng hoặc thị trường

- **Phát triển song song:** Kiến trúc Microservices cho phép các nhóm phát triển làm việc đồng thời trên các dịch vụ khác nhau. Khi có yêu cầu thay đổi hoặc bổ sung tính năng từ khách hàng, nhóm phát triển có thể nhanh chóng thực hiện trên dịch vụ liên quan mà không phải chờ đợi các phần khác của dự án. Sự linh hoạt này giúp rút ngắn thời gian phản hồi và đáp ứng nhanh chóng nhu cầu thị trường.
- **Sử dụng công nghệ phù hợp:** Với Microservices, mỗi dịch vụ có thể được phát triển bằng ngôn ngữ lập trình và công nghệ phù hợp nhất với yêu cầu cụ thể. Điều này cho phép các nhóm phát triển tận dụng các công nghệ mới và hiệu quả, đáp ứng nhanh chóng các thay đổi và cải tiến cần thiết.

2.2.2.3. Tự chủ cho Các Nhóm Phát triển

Kiến trúc Microservices đóng vai trò quan trọng trong việc tăng cường tính tự chủ cho các nhóm phát triển phần mềm. Dưới đây là phân tích chi tiết về cách Microservices hỗ trợ sự tự chủ và giảm thiểu sự phụ thuộc giữa các nhóm:

1. Tăng cường sự tự chủ và trách nhiệm của các nhóm phát triển

- **Phân chia dịch vụ theo chức năng:** Trong kiến trúc Microservices, ứng dụng được chia thành nhiều dịch vụ nhỏ, mỗi dịch vụ đảm nhận một chức năng cụ thể. Mỗi nhóm phát triển có thể chịu trách nhiệm hoàn toàn về một hoặc một vài dịch vụ này, từ khâu thiết kế, phát triển đến triển khai và bảo trì. Sự phân chia này giúp các nhóm có quyền

tự quyết định về công nghệ, quy trình và tiến độ cho phần việc của mình, tăng cường tính tự chủ và trách nhiệm.

- **Quyền sở hữu dịch vụ:** Việc mỗi nhóm sở hữu một dịch vụ cụ thể giúp họ hiểu sâu sắc về chức năng, hiệu suất và các yêu cầu kỹ thuật của dịch vụ đó. Điều này dẫn đến việc cải thiện chất lượng code, giảm thiểu lỗi và tăng hiệu quả trong việc phát hiện và khắc phục sự cố.

2. Giảm thiểu sự phụ thuộc giữa các nhóm, cải thiện hiệu suất làm việc

- **Phát triển và triển khai độc lập:** Mỗi dịch vụ trong kiến trúc Microservices có thể được phát triển, kiểm thử và triển khai một cách độc lập. Điều này có nghĩa là các nhóm có thể làm việc song song trên các dịch vụ khác nhau mà không cần chờ đợi lẫn nhau, giảm thiểu sự phụ thuộc và tắc nghẽn trong quy trình phát triển.
- **Giao tiếp thông qua API:** Các dịch vụ giao tiếp với nhau thông qua các giao diện lập trình ứng dụng (API) được xác định rõ ràng. Việc này giúp giảm thiểu sự phụ thuộc chặt chẽ giữa các nhóm, vì mỗi nhóm chỉ cần tuân thủ các hợp đồng API mà không cần quan tâm đến chi tiết triển khai của các dịch vụ khác.
- **Tăng tốc độ phản hồi:** Với khả năng phát triển và triển khai độc lập, các nhóm có thể nhanh chóng phản hồi trước các yêu cầu thay đổi hoặc cải tiến từ khách hàng hoặc thị trường, cải thiện hiệu suất làm việc và đáp ứng nhanh chóng nhu cầu kinh doanh.

2.2.3. Thách thức khi Kết hợp Microservices và Agile

Khi kết hợp kiến trúc Microservices với phương pháp Agile, các tổ chức có thể tận dụng lợi ích của cả hai để phát triển phần mềm linh hoạt và hiệu quả hơn. Tuy nhiên, việc này cũng đặt ra một số thách thức, đặc biệt trong việc quản lý và giám sát hệ thống. Dưới đây là phân tích chi tiết về các thách thức liên quan đến quản lý phức tạp khi kết hợp Microservices và Agile

2.2.3.1. Quản lý Phức tạp

1. Quản lý Phức tạp

- **Sự gia tăng số lượng dịch vụ:** Khi áp dụng kiến trúc Microservices, ứng dụng được chia thành nhiều dịch vụ nhỏ, độc lập. Mỗi dịch vụ có thể được phát triển, triển khai và mở rộng độc lập, điều này dẫn đến việc tăng số lượng dịch vụ trong hệ thống. Việc quản lý và giám sát một số lượng lớn dịch vụ đòi hỏi hệ thống quản lý và giám sát hiệu quả để đảm bảo hoạt động trơn tru của toàn bộ hệ thống.
- **Phức tạp trong giao tiếp giữa các dịch vụ:** Các dịch vụ trong Microservices giao tiếp với nhau thông qua các giao thức như HTTP hoặc message queues. Việc quản lý và giám sát các giao tiếp này trở nên phức tạp, đặc biệt khi hệ thống mở rộng, do cần đảm bảo tính nhất quán và độ tin cậy trong giao tiếp giữa các dịch vụ.
- **Quản lý dữ liệu phân tán:** Trong Microservices, mỗi dịch vụ thường có cơ sở dữ liệu riêng biệt. Việc quản lý và đồng bộ hóa dữ liệu giữa các dịch vụ trở thành một thách thức lớn, đặc biệt khi cần đảm bảo tính nhất quán và toàn vẹn dữ liệu trong toàn bộ hệ thống.

2. Yêu cầu về tự động hóa trong kiểm thử và triển khai

- **Tự động hóa kiểm thử:** Với số lượng dịch vụ lớn, việc kiểm thử thủ công trở nên không khả thi. Do đó, cần thiết lập các quy trình tự động hóa kiểm thử để đảm bảo chất lượng phần mềm. Tuy nhiên, việc thiết lập và duy trì các quy trình tự động hóa kiểm thử cho từng dịch vụ đòi hỏi nguồn lực và công sức đáng kể.
- **Tự động hóa triển khai:** Việc triển khai từng dịch vụ độc lập đòi hỏi hệ thống tự động hóa triển khai mạnh mẽ để đảm bảo tính nhất quán và giảm thiểu rủi ro trong quá trình triển khai. Điều này đòi hỏi đầu tư vào công cụ và quy trình tự động hóa phù hợp.

3. Đảm bảo chất lượng và tốc độ phát triển

- **Duy trì chất lượng trong môi trường phân tán:** Việc duy trì chất lượng phần mềm trong môi trường Microservices đòi hỏi các biện pháp kiểm soát chất lượng chặt chẽ, bao gồm kiểm thử liên tục, tích hợp liên tục và triển khai liên tục. Điều này có thể tăng độ phức tạp và yêu cầu nguồn lực đáng kể.
- **Quản lý sự thay đổi:** Trong môi trường Agile, yêu cầu thay đổi thường xuyên. Việc quản lý và triển khai các thay đổi trong một hệ thống Microservices phức tạp đòi hỏi quy trình và công cụ hỗ trợ hiệu quả để đảm bảo các thay đổi được triển khai một cách an toàn và hiệu quả.

4. Đảm bảo tính nhất quán và đồng bộ

- **Quản lý trạng thái và phiên bản:** Việc quản lý trạng thái và phiên bản của từng dịch vụ trong Microservices đòi hỏi hệ thống quản lý cấu hình và phiên bản mạnh mẽ để đảm bảo tính nhất quán và đồng bộ giữa các dịch vụ.
- **Giám sát và phản hồi:** Cần thiết lập hệ thống giám sát và phản hồi hiệu quả để phát hiện và xử lý sự cố kịp thời, đảm bảo hệ thống hoạt động ổn định và đáp ứng yêu cầu kinh doanh.

2.2.3.2. Yêu cầu Kỹ năng và Văn hóa Tổ chức

Khi kết hợp kiến trúc Microservices với phương pháp Agile, tổ chức cần chú trọng đến việc phát triển kỹ năng và xây dựng văn hóa tổ chức phù hợp để đảm bảo thành công. Dưới đây là phân tích chi tiết về yêu cầu kỹ năng và văn hóa tổ chức trong môi trường này:

1. Yêu cầu Kỹ năng

- **Kỹ năng đa dạng và chuyên sâu:** Đội ngũ phát triển cần có kiến thức vững về cả Microservices và Agile. Điều này bao gồm hiểu biết sâu sắc về thiết kế và triển khai Microservices, cũng như khả năng áp dụng các nguyên tắc Agile trong quá trình phát triển phần mềm. Việc này giúp đảm bảo rằng các dịch vụ được phát triển hiệu quả và phù hợp với yêu cầu kinh doanh.
- **Kỹ năng giao tiếp và hợp tác:** Trong môi trường Microservices và Agile, việc giao tiếp hiệu quả giữa các nhóm phát triển là rất quan trọng. Mỗi nhóm cần hiểu rõ về các dịch vụ mà họ phát triển và cách chúng tương tác với các dịch vụ khác. Kỹ năng hợp tác giúp giảm thiểu sự phụ thuộc giữa các nhóm và tăng cường hiệu suất làm việc.

- **Kỹ năng quản lý dự án và tự động hóa:** Việc quản lý nhiều dịch vụ độc lập đòi hỏi kỹ năng quản lý dự án tốt và khả năng thiết lập các quy trình tự động hóa trong kiểm thử và triển khai. Điều này giúp đảm bảo chất lượng và tốc độ phát triển phần mềm.

2. Văn hóa Tổ chức

- **Hỗ trợ sự tự chủ và trách nhiệm:** Mỗi nhóm phát triển cần có quyền tự chủ trong việc quyết định công nghệ và quy trình làm việc cho dịch vụ của mình. Điều này khuyến khích sự sáng tạo và trách nhiệm, giúp tăng cường hiệu suất và chất lượng sản phẩm.
- **Khuyến khích học hỏi liên tục:** Văn hóa tổ chức nên khuyến khích việc học hỏi và cải tiến liên tục. Điều này giúp đội ngũ phát triển cập nhật kiến thức mới, áp dụng các công nghệ và phương pháp tiên tiến, từ đó nâng cao hiệu quả công việc.
- **Chấp nhận và thích nghi với thay đổi:** Trong môi trường Agile và Microservices, thay đổi là điều không thể tránh khỏi. Văn hóa tổ chức cần chấp nhận và hỗ trợ việc thay đổi, giúp đội ngũ phát triển linh hoạt và phản ứng nhanh chóng với yêu cầu mới từ khách hàng hoặc thị trường.

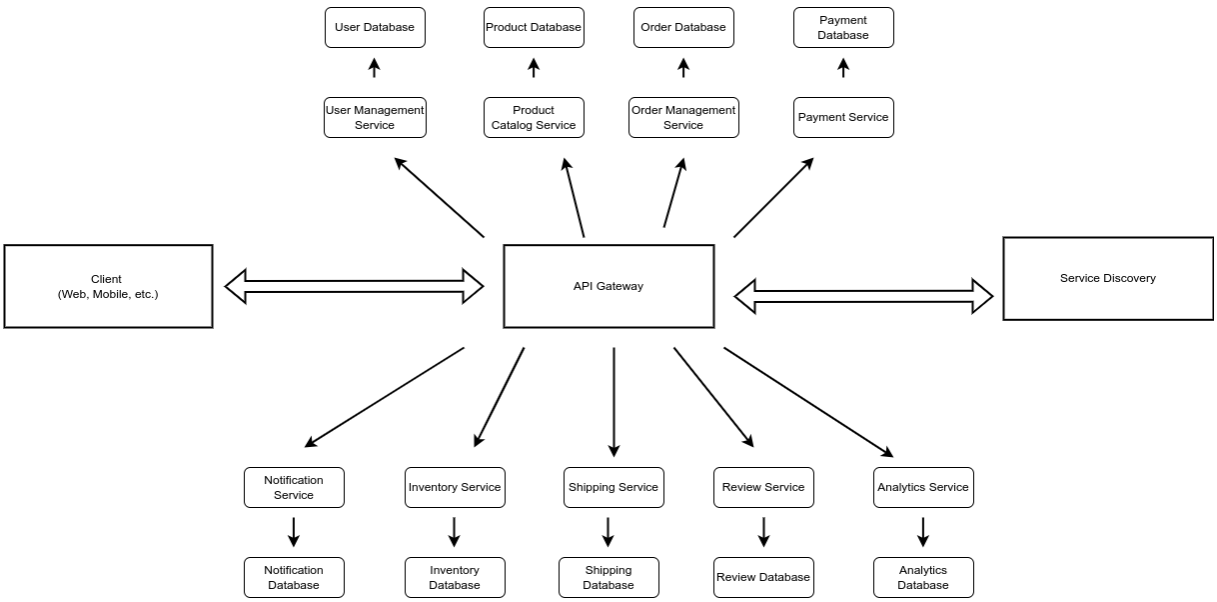
3. Decompose a software system in microservice

3.1. Phân tách Hệ thống Thương mại Điện tử (E-commerce System)

Hệ thống thương mại điện tử bao gồm các chức năng như quản lý người dùng, danh mục sản phẩm, xử lý đơn hàng, thanh toán, quản lý kho, v.v.

Phân tách Microservices:

1. **Dịch vụ Quản lý Người dùng (User Management Service)**: Quản lý đăng ký, đăng nhập, thông tin cá nhân và vai trò của người dùng.
2. **Dịch vụ Danh mục Sản phẩm (Product Catalog Service)**: Quản lý thông tin sản phẩm, bao gồm mô tả, giá cả, danh mục và hình ảnh.
3. **Dịch vụ Quản lý Đơn hàng (Order Management Service)**: Quản lý việc tạo đơn hàng, xử lý đơn hàng, theo dõi trạng thái đơn hàng và lịch sử đơn hàng.
4. **Dịch vụ Quản lý Kho (Inventory Service)**: Theo dõi tồn kho, cập nhật số lượng sản phẩm và quản lý quy trình nhập hàng.
5. **Dịch vụ Thanh toán (Payment Service)**: Tích hợp với các cổng thanh toán để xử lý giao dịch, hoàn tiền và thanh toán cho đơn hàng.
6. **Dịch vụ Vận chuyển & Giao hàng (Shipping & Delivery Service)**: Quản lý thông tin vận chuyển, phương thức giao hàng và trạng thái giao hàng.
7. **Dịch vụ Đánh giá & Nhận xét (Review & Rating Service)**: Cho phép người dùng đánh giá sản phẩm, viết nhận xét và xem các đánh giá của người dùng khác.
8. **Dịch vụ Thông báo (Notification Service)**: Gửi thông báo cho người dùng về cập nhật đơn hàng, chương trình khuyến mãi và thông tin quan trọng khác.
9. **Dịch vụ Phân tích & Báo cáo (Analytics & Reporting Service)**: Xử lý và báo cáo về hành vi người dùng, doanh thu, hiệu suất sản phẩm và các chỉ số quan trọng khác.



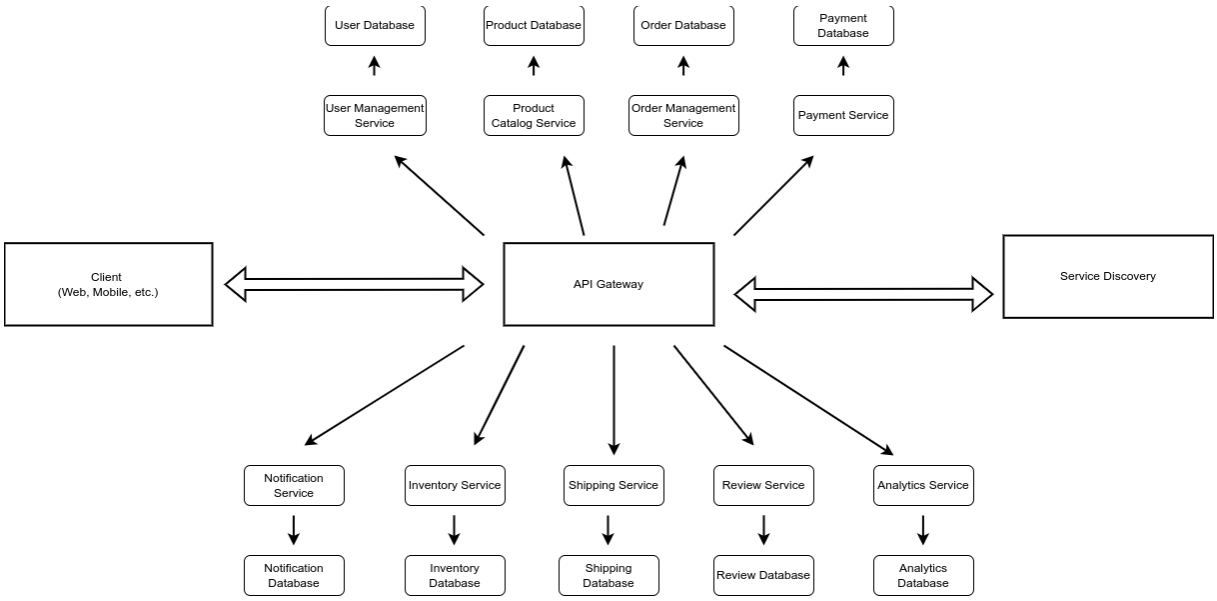
Hệ thống thương mại điện tử

3.2. Phân tách Hệ thống Y tế (Medicine System)

Hệ thống y tế quản lý hồ sơ bệnh án, đơn thuốc, thông tin bệnh nhân, dịch vụ dược phẩm và các quy trình liên quan.

Phân tách Microservices:

1. **Dịch vụ Quản lý Bệnh nhân (Patient Management Service)**: Quản lý hồ sơ bệnh nhân, thông tin y tế và tiền sử bệnh.
2. **Dịch vụ Đơn thuốc (Prescription Service)**: Xử lý đơn thuốc, các lần tái kê đơn và kế hoạch điều trị cho bệnh nhân.
3. **Dịch vụ Quản lý Dược phẩm (Pharmacy Management Service)**: Quản lý kho thuốc, đơn thuốc và quá trình phát thuốc.
4. **Dịch vụ Quản lý Lịch hẹn (Appointment Management Service)**: Quản lý việc đặt lịch khám, hủy lịch và thông báo cho bệnh nhân.
5. **Dịch vụ Hồ sơ Y tế (Medical History Service)**: Lưu trữ hồ sơ y tế, bao gồm các chẩn đoán, phương pháp điều trị và các thủ tục đã thực hiện.
6. **Dịch vụ Thanh toán (Billing Service)**: Xử lý hóa đơn, thanh toán của bệnh nhân và các giao dịch bảo hiểm.
7. **Dịch vụ Y tế từ xa (Telemedicine Service)**: Cung cấp các dịch vụ khám bệnh trực tuyến qua video call hoặc chat.
8. **Dịch vụ Quản lý Bác sĩ (Doctor Management Service)**: Quản lý hồ sơ bác sĩ, chuyên môn, lịch làm việc và đánh giá của bệnh nhân.



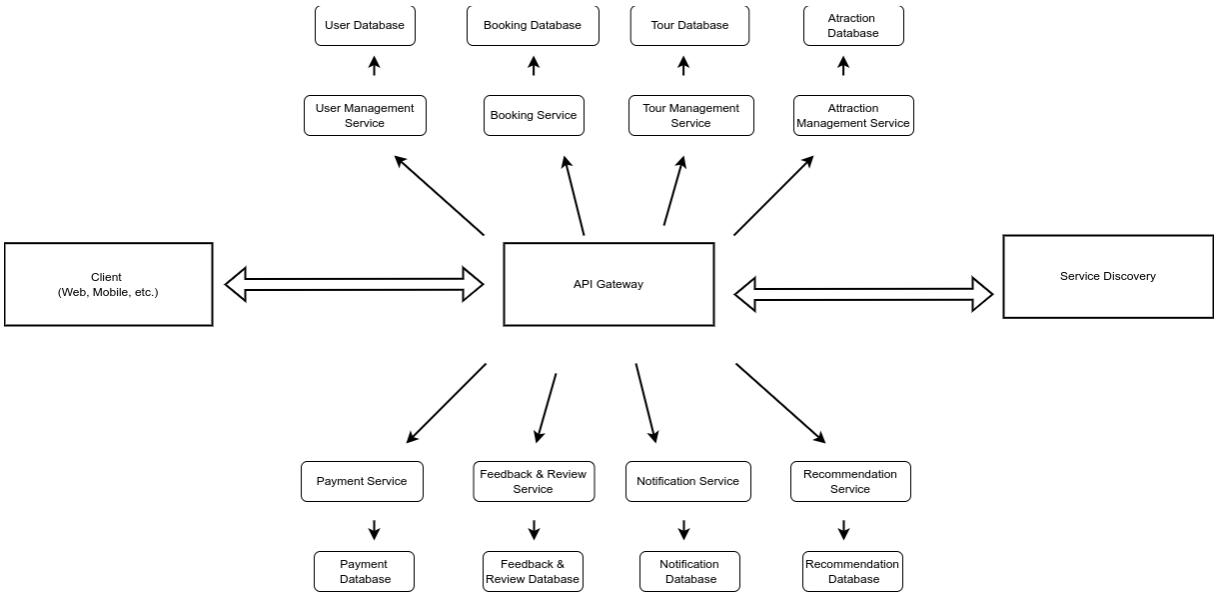
Hệ thống y tế

3.3. Phân tách Hệ thống Trợ lý Du lịch (Tourist Assistant System)

Hệ thống Trợ lý Du lịch giúp người dùng tìm kiếm các địa điểm tham quan, đặt tour và các dịch vụ du lịch khác.

Phân tách Microservices:

1. **Dịch vụ Quản lý Người dùng (User Management Service)**: Quản lý đăng ký, thông tin cá nhân và sở thích của người dùng.
2. **Dịch vụ Quản lý Tour (Tour Management Service)**: Quản lý các tour du lịch, lịch trình, và thông tin đặt tour.
3. **Dịch vụ Quản lý Địa điểm (Attraction Management Service)**: Cung cấp thông tin về các địa điểm du lịch, giờ mở cửa, giá vé, v.v.
4. **Dịch vụ Đặt chỗ (Booking Service)**: Quản lý việc đặt các tour, địa điểm tham quan và các dịch vụ du lịch.
5. **Dịch vụ Thanh toán (Payment Service)**: Xử lý thanh toán cho các dịch vụ đặt chỗ, tích hợp với các cổng thanh toán.
6. **Dịch vụ Phản hồi & Đánh giá (Feedback & Review Service)**: Thu thập phản hồi và đánh giá của người dùng về các tour, địa điểm và trải nghiệm.
7. **Dịch vụ Thông báo (Notification Service)**: Gửi thông báo về việc xác nhận đặt chỗ, thông báo về các chương trình khuyến mãi, và các thông tin quan trọng khác.
8. **Dịch vụ Đề xuất (Recommendation Service)**: Đưa ra các đề xuất về tour, địa điểm tham quan dựa trên sở thích và lịch sử của người dùng.



Hệ thống trợ lý du lịch

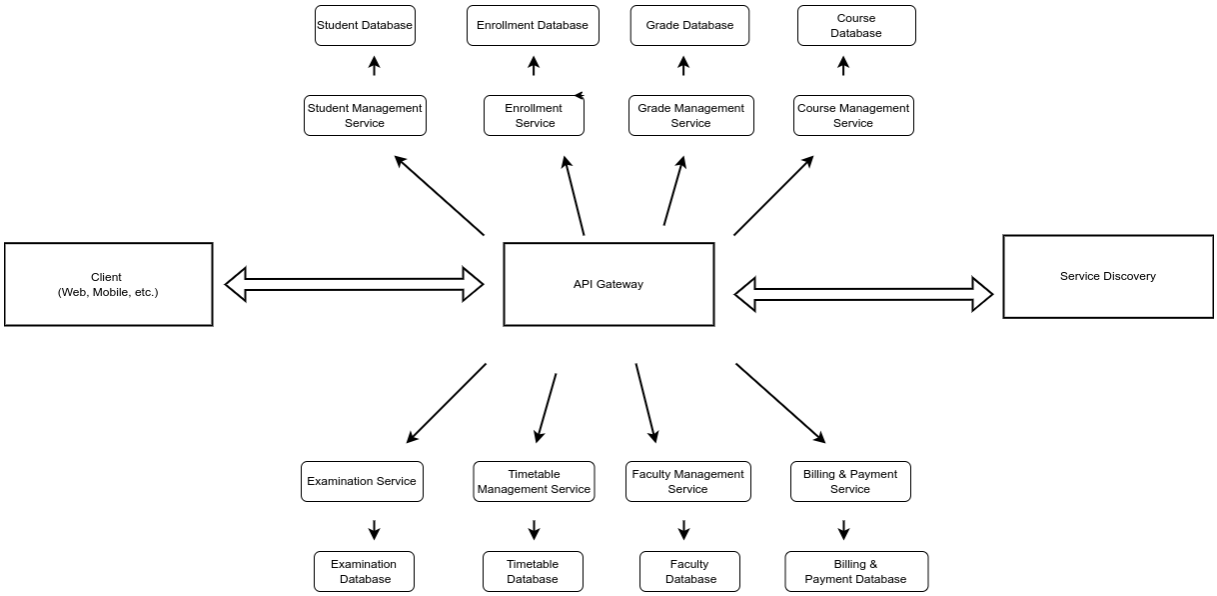
3.4. Phân tách Hệ thống Quản lý Đại học (University Management System)

Hệ thống này quản lý thông tin sinh viên, khóa học, việc đăng ký môn học, bảng điểm, lịch thi, v.v.

Phân tách Microservices:

1. **Dịch vụ Quản lý Sinh viên (Student Management Service)**: Quản lý thông tin sinh viên, đăng ký học và hồ sơ học tập.
2. **Dịch vụ Quản lý Khóa học (Course Management Service)**: Quản lý thông tin về các khóa học, lịch học và việc đăng ký môn học.

3. **Dịch vụ Đăng ký Môn học (Enrollment Service)**: Quản lý việc đăng ký và hủy đăng ký môn học, danh sách chờ.
4. **Dịch vụ Quản lý Điểm (Grade Management Service)**: Quản lý điểm số, bảng điểm và kết quả học tập.
5. **Dịch vụ Quản lý Thi cử (Examination Service)**: Xử lý lịch thi, kết quả thi và việc đánh giá sinh viên.
6. **Dịch vụ Quản lý Lịch học (Timetable Management Service)**: Quản lý thời gian biểu cho giảng viên và sinh viên.
7. **Dịch vụ Quản lý Giảng viên (Faculty Management Service)**: Quản lý thông tin giảng viên, phân công giảng dạy và lịch làm việc.
8. **Dịch vụ Thanh toán (Billing & Payment Service)**: Xử lý học phí, thanh toán và các khoản trợ cấp.



Hệ thống quản lý đại học

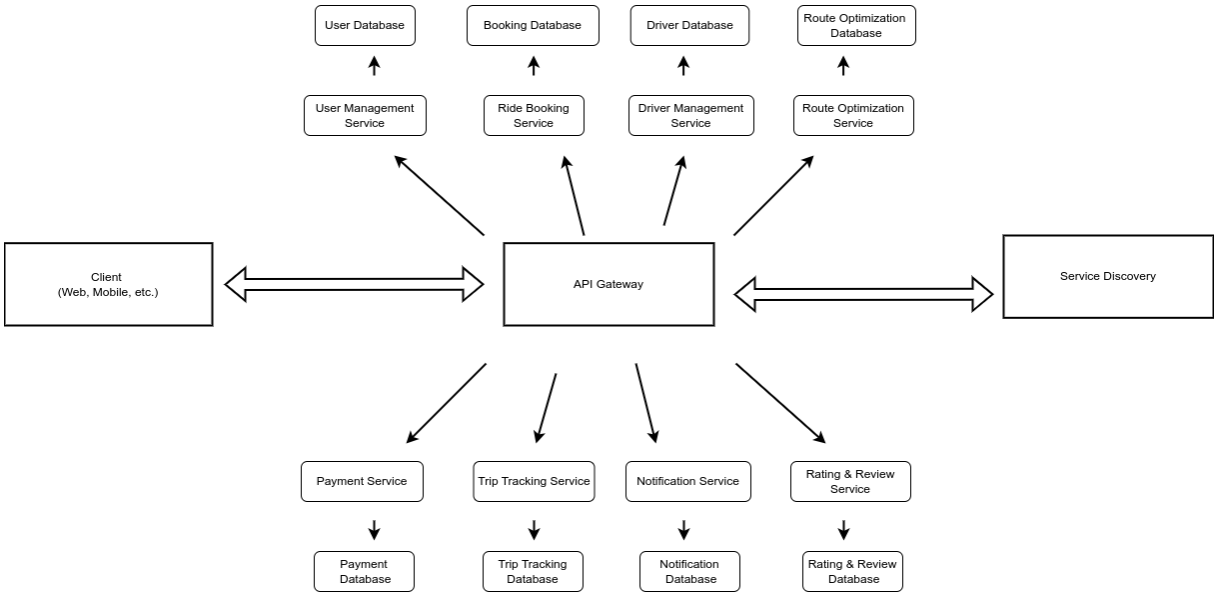
3.5. Phân tách Hệ thống Quản lý Grab Car (Grab Car Management System)

Hệ thống quản lý Grab Car liên quan đến việc đặt xe, quản lý tài xế, tối ưu hóa lộ trình, thanh toán, và đánh giá.

Phân tách Microservices:

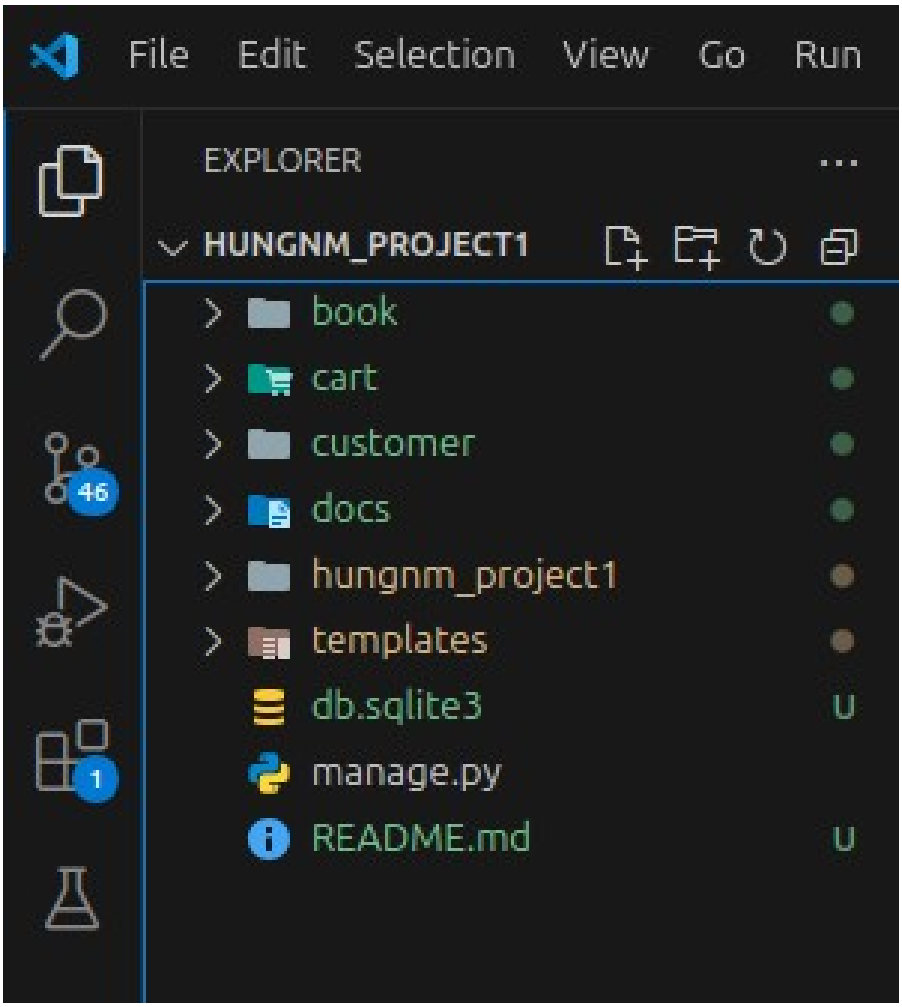
1. **Dịch vụ Quản lý Người dùng (User Management Service)**: Quản lý tài khoản của khách hàng và tài xế, sở thích và các cài đặt cá nhân.
2. **Dịch vụ Đặt xe (Ride Booking Service)**: Quản lý việc đặt xe, tính toán cước phí, và ghép xe với khách hàng.
3. **Dịch vụ Quản lý Tài xế (Driver Management Service)**: Quản lý thông tin tài xế, xe của họ, tình trạng sẵn sàng và các đánh giá.
4. **Dịch vụ Thanh toán (Payment Service)**: Xử lý thanh toán cho mỗi chuyến đi, bao gồm cổng thanh toán.

5. **Dịch vụ Tối ưu hóa Lộ trình (Route Optimization Service)**: Cung cấp lộ trình tối ưu cho tài xế dựa trên tình trạng giao thông và khoảng cách.
6. **Dịch vụ Theo dõi Chuyến đi (Trip Tracking Service)**: Theo dõi vị trí chuyến đi theo thời gian thực.
7. **Dịch vụ Đánh giá & Nhận xét (Rating & Review Service)**: Thu thập và hiển thị các đánh giá cho tài xế và khách hàng.
8. **Dịch vụ Thông báo (Notifications Service)**: Gửi thông báo về chuyến đi, thông tin cập nhật và khuyến mãi.



Hệ thống quản lý Grab Car

4. Cài đặt Python và Django. Tạo dự án hungnm_project1, app customer, book, cart.



5. Phát triển module customer, book, cart với Django

Link github: https://github.com/HungNM1486/hungnm_project1.git

5.1. Bảng thuộc tính và mối quan hệ giữa các thực thể

5.1.1. Module customer

Entity	Thuộc tính	Kiểu Dữ liệu / Quan hệ	Mô tả
CustomerProfile	user	OneToOneField (liên kết với User)	Liên kết tới tài khoản người dùng Django.
	first_name	CharField	Tên của khách hàng.
	last_name	CharField	Họ của khách hàng.
	email	EmailField (hoặc từ User)	Email liên hệ của khách hàng.
	phone_number	CharField	Số điện thoại liên hệ.
	address	CharField / TextField	Địa chỉ (hoặc có thể tách ra làm model Address).
	date_joined	DateTimeField (auto_now_add hoặc từ User)	Ngày tham gia.
	updated_at	DateTimeField (auto_now)	Thời gian cập nhật thông tin lần cuối.

5.1.2. Module cart

Entity	Thuộc tính	Kiểu Dữ liệu / Quan hệ	Mô tả
Cart	customer	ForeignKey hoặc OneToOneField (tới CustomerProfile)	Liên kết giỏ hàng với khách hàng.
	created_at	DateTimeField (auto_now_add)	Thời gian tạo giỏ hàng.
	updated_at	DateTimeField (auto_now)	Thời gian cập nhật giỏ hàng.

Entity	Thuộc tính	Kiểu Dữ liệu / Quan hệ	Mô tả
CartItem	cart	ForeignKey (tới Cart)	Nhiều mặt hàng thuộc về 1 giỏ hàng.
	product	ForeignKey (tới Book)	Liên kết sản phẩm (sách) được thêm vào giỏ hàng.
	quantity	IntegerField	Số lượng sản phẩm được chọn.
	added_at	DateTimeField (auto_now_add)	Thời gian sản phẩm được

			thêm vào giỏ hàng.

5.1.3.Module book

Entity	Thuộc tính	Kiểu Dữ liệu / Quan hệ	Mô tả
Book	title	CharField	Tiêu đề của sách.
	author	CharField (hoặc ForeignKey tới model Author)	Tác giả của sách.
	description	TextField	Mô tả chi tiết về sách.
	price	DecimalField	Giá bán của sách.
	ISBN	CharField	Mã ISBN định danh sách.
	publisher	CharField	Nhà xuất bản.
	publication_date	DateField	Ngày xuất bản.
	stock	IntegerField	Số lượng tồn kho.
	category	ForeignKey hoặc ManyToManyField (tới Category)	Danh mục hoặc thể loại sách.
	image	ImageField	Hình ảnh bìa của sách.
	created_at	DateTimeField (auto_now_add)	Thời gian tạo bản ghi sách.
	updated_at	DateTimeField (auto_now)	Thời gian cập nhật thông tin sách.

5.2. Các phương thức theo từng module

5.2.1. Module customer

Phương thức	Mô tả
update_profile(data)	Cập nhật thông tin cá nhân của khách hàng (số điện thoại, địa chỉ, …).
get_profile()	Lấy thông tin chi tiết của khách hàng.
add_address(address_data)	Thêm địa chỉ mới cho khách hàng (nếu quản lý nhiều địa chỉ).
remove_address(address_id)	Xóa một địa chỉ khỏi danh sách địa chỉ của khách hàng.

5.2.2. Module cart

Phương thức	Mô tả
add_item(product, quantity=1)	Thêm sản phẩm vào giỏ hàng. Nếu sản phẩm đã tồn tại, tăng số lượng.
remove_item(product)	Xóa sản phẩm khỏi giỏ hàng.
update_quantity(product, quantity)	Cập nhật số lượng của một sản phẩm trong giỏ hàng.
get_total()	Tính tổng số tiền của các sản phẩm trong giỏ hàng.
clear()	Xóa toàn bộ sản phẩm trong giỏ hàng.

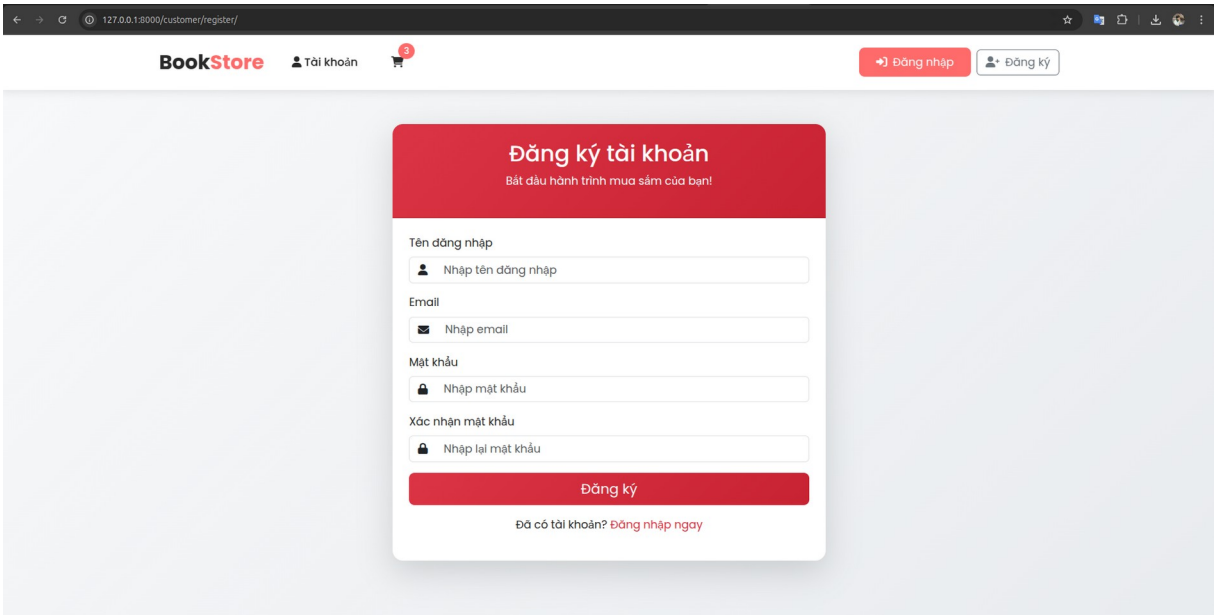
5.2.3. Module book

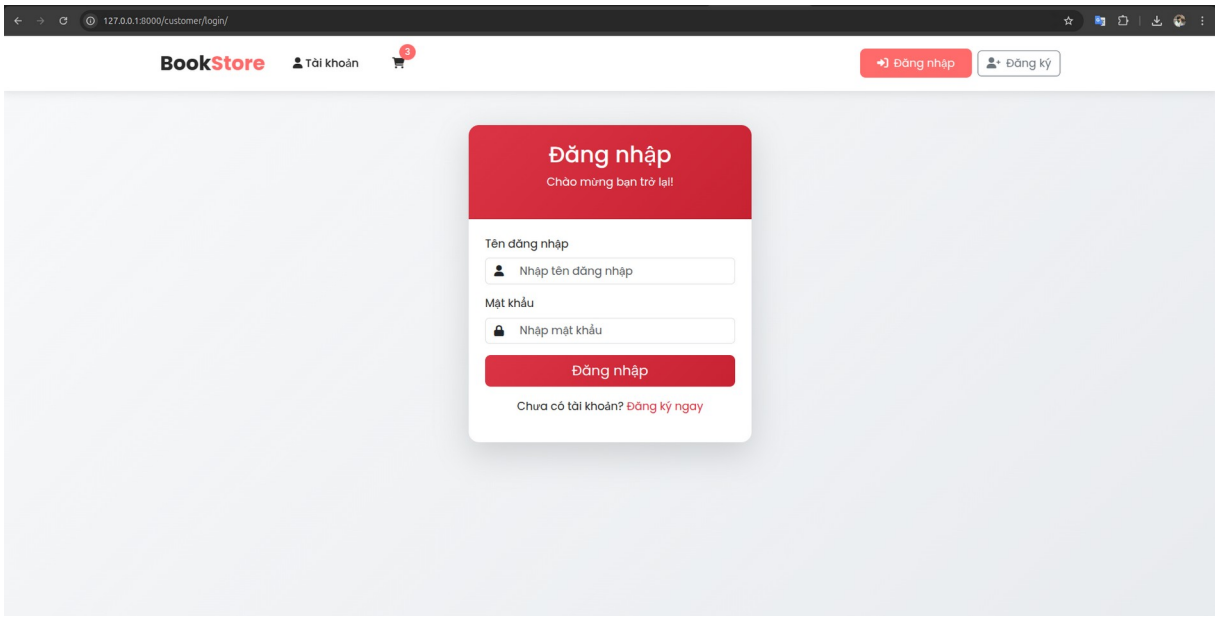
Phương thức	Mô tả
update_stock(quantity)	Cập nhật tồn kho khi có đơn hàng hoặc nhập hàng mới (cộng/trừ số lượng).
get_discounted_price(discount)	Tính giá sau khi áp dụng giảm giá cho sách.
get_detail()	Lấy thông tin chi tiết về sách.

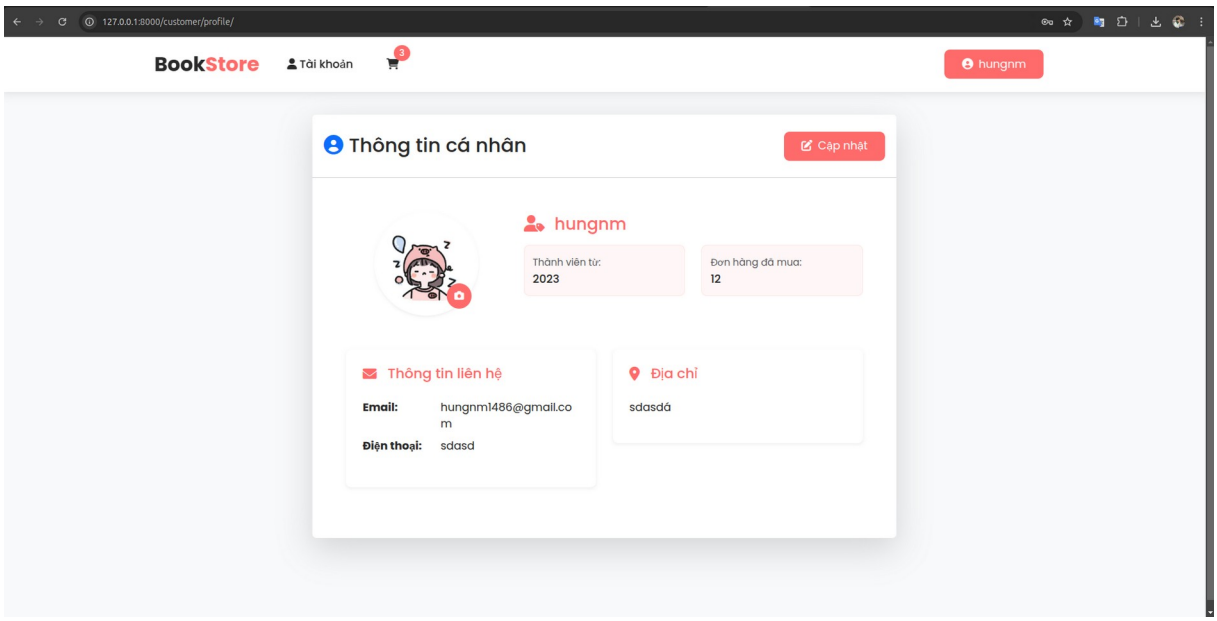
5.3. Mã nguồn & Giao diện các chức năng

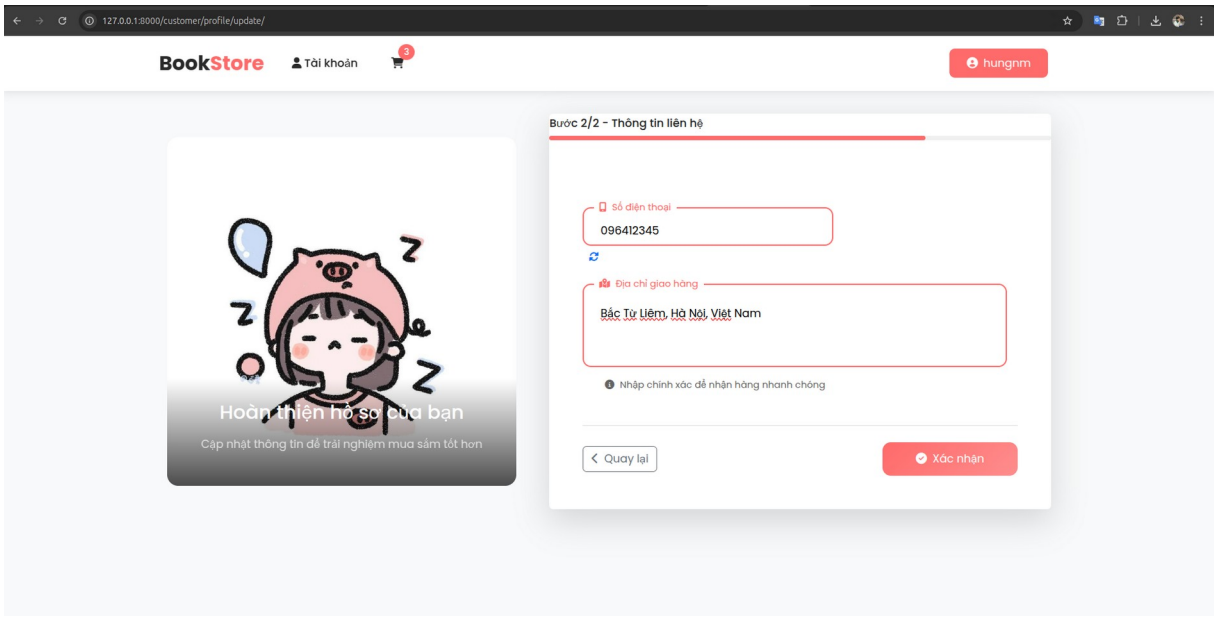
Link github: https://github.com/HungNM1486/hungnm_project1.git

5.3.1. Customer

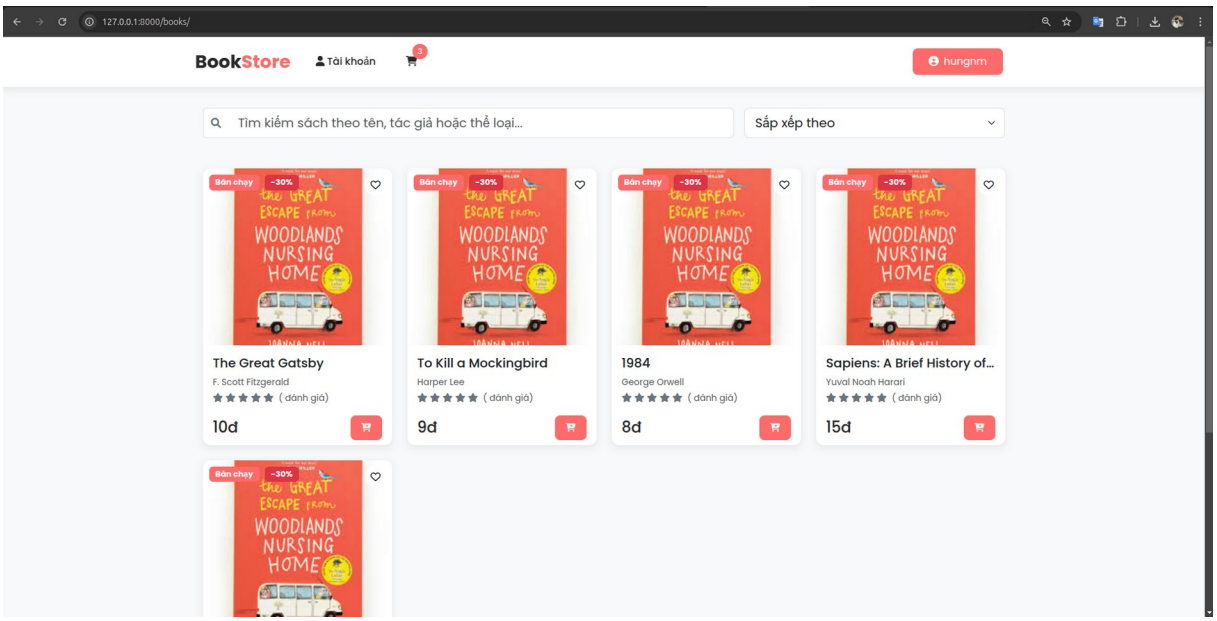


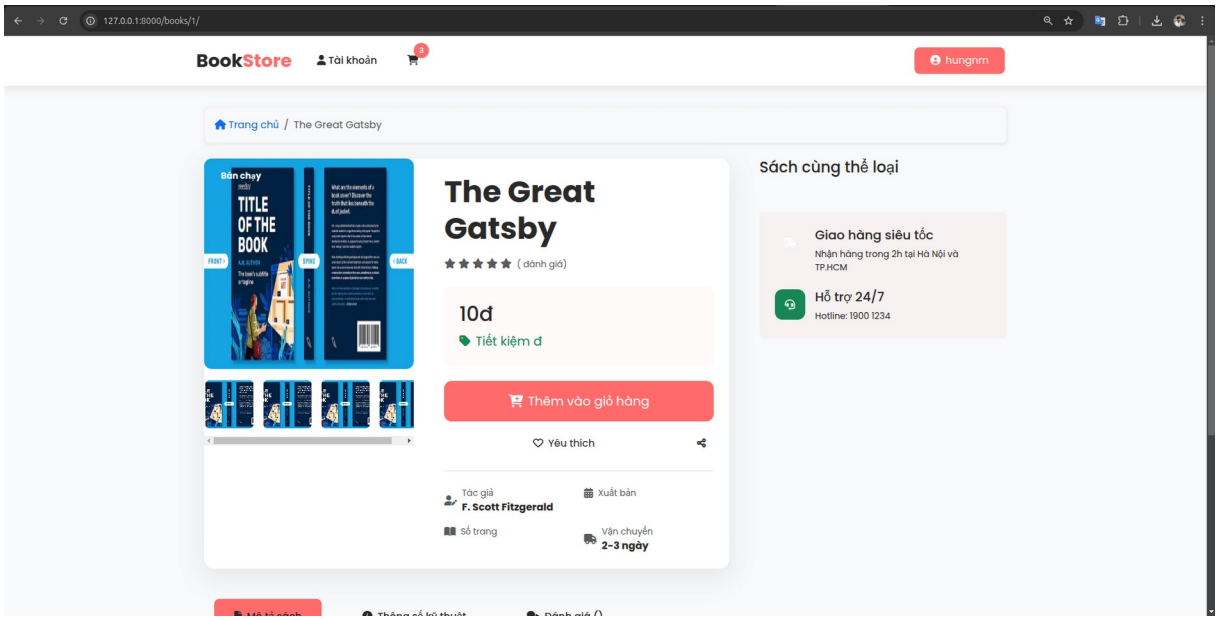

127.0.0.1:8000/customer/login/
BookStore
Tài khoản
3
Đăng nhập
Đăng ký
Đăng nhập
Chào mừng bạn trở lại!
Tên đăng nhập
Nhập tên đăng nhập
Mật khẩu
Nhập mật khẩu
Đăng nhập
Chưa có tài khoản? Đăng ký ngay


127.0.0.1:8000/customer/profile/
BookStore
Tài khoản
hungnm
Thông tin cá nhân
Cập nhật
hungnm
Thành viên từ:
2023
Đơn hàng đã mua:
12
Thông tin liên hệ
Email:
hungnm1486@gmail.com
Điện thoại:
sdasd
Địa chỉ
sdasdá

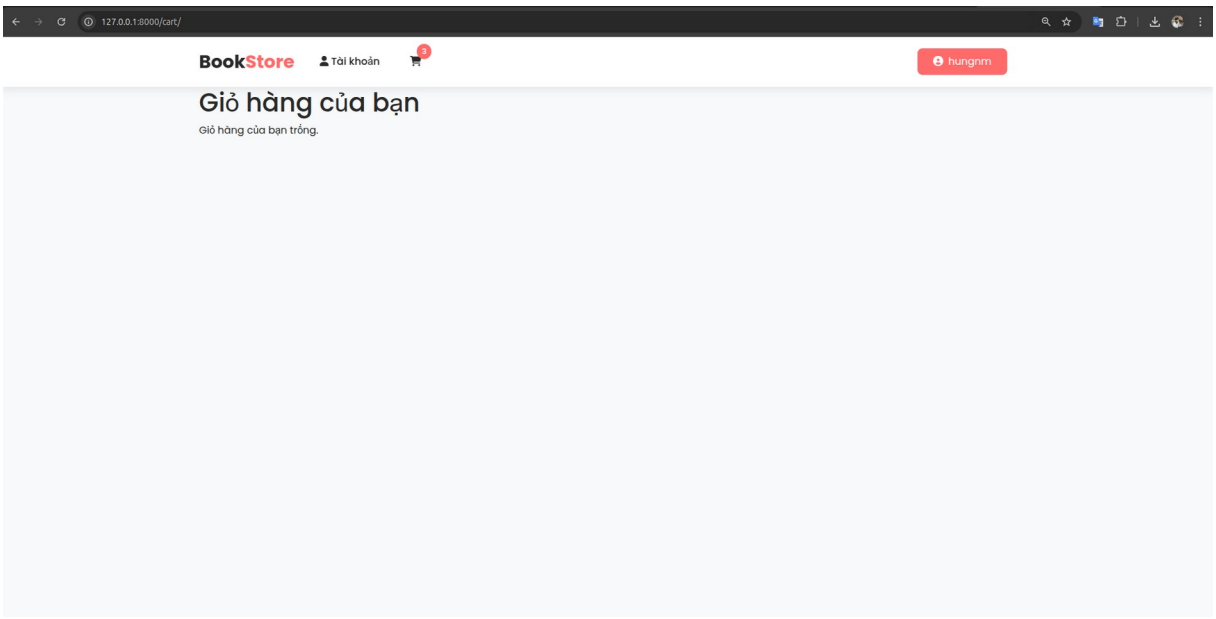

127.0.0.1:8000/customer/profile/update/
BookStore
Tài khoản
hungnm
Hoàn thiện hồ sơ của bạn
Cập nhật thông tin để trải nghiệm mua sắm tốt hơn
Bước 2/2 - Thông tin liên hệ
Số điện thoại
096412345
Địa chỉ giao hàng
Bắc Từ Liêm, Hà Nội, Việt Nam
Nhập chính xác để nhận hàng nhanh chóng
Quay lại
Xác nhận

5.3.2. Book





5.3.3. CartCart


127.0.0.1:8000/cart/
BookStore
Tài khoản
3
hungnm
Giỏ hàng của bạn
Giỏ hàng của bạn trống.