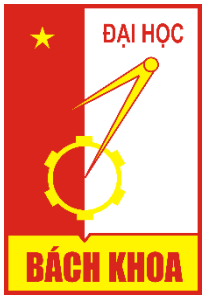
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

———— * ————



ĐỒ ÁN

TỐT NGHIỆP ĐẠI HỌC

NGÀNH CÔNG NGHỆ THÔNG TIN

TÌM HIỂU VÀ XÂY DỰNG ỨNG DỤNG TRÊN KIẾN TRÚC MICROSERVICE

Sinh viên thực hiện: **Đinh Hoàng Hải Hảo**
Lớp ATTT – K59
Giáo viên hướng dẫn: [TS] **Hoàng Văn Hiệp**

HÀ NỘI 5-2019

PHIẾU GIAO NHẬN ĐỒ ÁN TỐT NGHIỆP

1. Thông tin về sinh viên

Họ và tên sinh viên: Đinh Hoàng Hải Hảo
Điện thoại liên lạc: 0973408415 Email: haodinh0123@gmail.com
Lớp: ATTT-K59 Hệ đào tạo: Kỹ sư
Thời gian làm đồ án tốt nghiệp: Từ ngày 11/02/2019 đến 24/05/2019

2. Mục đích nội dung của đồ án tốt nghiệp

- Tìm hiểu kiến trúc Microservice

- Áp dụng kiến trúc Microservice xây dựng một ứng dụng minh họa

3. Các nhiệm vụ cụ thể của đồ án tốt nghiệp

- Tìm hiểu kiến trúc Microservice

- Thiết kế và xây dựng website quản lý bán hàng theo mô hình Microservice

4. Lời cam đoan của sinh viên:

Tôi - *Đinh Hoàng Hải Hảo* - cam kết ĐATN là công trình nghiên cứu của bản thân tôi dưới sự hướng dẫn của *TS.Hoàng Văn Hiệp*.
Các kết quả nêu trong ĐATN là trung thực, không phải là sao chép toàn văn của bất kỳ công trình nào khác.

Hà Nội, ngày ... tháng ... năm .....

Tác giả ĐATN

Đinh Hoàng Hải Hảo

5. Xác nhận của giáo viên hướng dẫn về mức độ hoàn thành của ĐATN và cho phép bảo vệ:

Hà Nội, ngày ... tháng ... năm .....

Giáo viên hướng dẫn

TS. Hoàng Văn Hiệp

LỜI CẢM ƠN

Năm tháng trôi qua tựa như một cơn gió lướt qua tuổi thanh xuân. Năm năm gắn với Bách Khoa, không dài cũng không ngắn, nhưng đó sẽ mãi là ký ức của một thời tuổi trẻ khao khát và dại khờ.

Lời đầu tiên tôi xin gửi lời cảm ơn tới TS. Hoàng Văn Hiệp, thày giáo hướng dẫn đồ án này. Cảm ơn thày đã gợi ý đề tài cũng như đưa gia các giải pháp, hướng đi và đông hành giúp tôi hoàn thành đồ án trong thời gian cho phép và đạt kết quả tốt nhất.

Bên cạnh đó tôi xin gửi lời cảm ơn đến gia đình và bạn bè đã luôn là chỗ dựa tinh thần và giúp đỡ trong thời gian làm đồ án cũng như suốt những năm tháng học tập và gắn bó với Bách Khoa. Cảm ơn tất cả thày cô giáo trong trường và đặc biệt là các thày cô trong viện đã đồng hành, dìu dắt em qua những năm tháng học tập tại ngôi trường Bách Khao thân yêu.

Do thời gian làm đồ án có hạn và kiến thức cũng như trình độ còn hạn chế nên không thể tránh khỏi những thiếu sót. Vì vậy, sự đóng góp ý kiến, nhận xét của thày cô cũng như các bạn sinh viên là vô cùng quý giá giúp đồ án có thể hoàn thiện hơn!

Xin chân thành cảm ơn!

TÓM TẮT NỘI DUNG ĐỒ ÁN TỐT NGHIỆP

Khi một ứng dụng web phát triển, số lượng người dùng tăng lên, ứng dụng cần nhiều tính năng mới, dữ liệu tăng, logic của ứng dụng phức tạp hơn, ứng dụng cần giao tiếp với các hệ thống khác. Và sau một thời gian không lâu, các ứng dụng đơn giản được thiết kế theo kiến trúc nguyên khối sẽ trở thành một con quái vật, đi kèm theo đó sẽ là kinh phí bỏ ra để xây dựng, duy trì, nâng cấp là không hề nhỏ.

Qua việc tìm kiếm giải pháp, tôi nhận thấy nhiều tập đoàn lớn như Amazon, eBay, Netflix,… đã giải quyết vấn đề kiến trúc nguyên khối bằng kiến trúc microservices (tập hợp các dịch vụ nhỏ). Ý tưởng là chia nhỏ ứng dụng lớn thành các dịch vụ nhỏ kết nối với nhau, điều này làm cho ứng dụng trở nên dễ hiểu, dễ phát triển, thử nghiệm, triển khai và trở nên linh hoạt hơn. Và nội dung đồ án này quyết định triển khai giải pháp này bằng cách xây dựng một ứng dụng theo kiến trúc microservice để thực hiện thử nghiệm, đánh giá.

Với quy mô đồ án, việc triển khai sẽ được thực hiện với ứng dụng mang tính chất là mô hình tham khảo, học tập và thực hành việc thiết kế, xây dựng và triển khai một ứng dụng microservice. Ứng dụng hoàn thành được đặt tên "Quản lý bán hàng" (Quản lý kinh doanh - MS.BusinessManagement) sẽ có các khối chức năng chính là dịch vụ quản lý tài khoản cung cấp chức năng đăng ký, đăng nhập và cung cấp token cho người dùng sử dụng các dịch vụ khác của ứng dụng; dịch vụ quản lý kho hàng cung cấp chức năng quản lý sản phẩm, cập nhật thông tin đơn hàng với số lượng sản phẩm tương ứng; dịch vụ quản lý đơn nhập hàng cung cấp chức năng quản lý nhà cung cấp, quản lý đơn nhập hàng và chi tiết đơn nhập hàng; dịch vụ quản lý đơn bán hàng cung cấp chức năng quản lý khách hàng, quản lý đơn bán hàng và chi tiết đơn bán hàng. Ứng dụng được phát triển trên nền tảng Asp.Net Core, các dịch vụ của ứng dụng lớn sẽ giao tiếp với nhau thông qua một message broker (phần mềm trung gian giúp trao đổi thông điệp) là RabbitMQ, sử dụng cơ sở dữ liệu là SQL Server Express 2017 và Entity Framework Core để thao tác với nó.

ABSTRACT

As a web application grows, the number of users increases, the application needs more new features, increased data, more complex application logic, and applications that need to communicate with other systems. And after a long time, simple applications designed according to monolithic architecture will become a monster, accompanied by the cost of building, maintaining and upgrading is not small.

Through finding solutions, I found that many large corporations such as Amazon, eBay, Netflix, etc. have solved the problem of monolithic architecture with microservices architecture (a collection of small services). The idea is to break up large applications into interconnected small services, which makes the application easier to understand, develop, test, deploy, and become more flexible. And the contents of this project decided to implement this solution by building a microservice architecture application to perform testing and evaluation.

With the project scale, the implementation will be implemented with the application of nature as a reference model, learning and practicing the design, construction and deployment of a microservice application. The completed application is named "Sales Manager" (Business Management - MS.BusinessManagement) will have the main function blocks are account management services that provide functions to register, log in and provide token for users using other services of the application; warehouse management service provides product management function, updating order information with the corresponding number of products; Goods import management service provides supplier management functions, purchase order management and purchase order details; Sales application management service provides customer management functions, sales order management and sales order details. The application is developed on the Asp.Net Core platform, the services of large applications will communicate with each other through a message broker is RabbitMQ, using the database is SQL Server Express 2017 and Entity Framework Core to manipulate.

MỤC LỤC

DANH MỤC HÌNH VẼ

DANH MỤC BẢNG

CHƯƠNG 1: GIỚI THIỆU ĐỀ TÀI

1.1. Đặt vấn đề

Ban đầu khi xây dựng một ứng dụng web, cũng như các dự án trước đó mà tôi đã thực hiện, tôi lựa chọn xây dựng trên kiến trúc nguyên khối. Tuy nhiên tại thời điểm đó tôi vẫn chưa có khái niệm và kiến trúc nguyên khối. Thế kiến trúc nguyên khối là gì? Kiến trúc nguyên khối chính là kiến trúc mà các thành phần của ứng dụng sẽ được gói gọn trong một dự án, nó có thể chia thành nhiều tầng giúp ứng dụng trở bên chuyên biệt mô hình dữ liệu và đầu vào, đầu ra, và có thể có cấu trúc module hóa hợp lý, nhưng ứng dụng kiểu này sẽ đóng gói và cài đặt thành một khối. Tôi lựa chọn nó là vì nó rất dễ xây dựng, dễ thử nghiệm, dễ sao chép bởi các công cụ lập trình IDE, nó có rất nhiều dự án mẫu được tối ưu cho các yêu cầu khác nhau mà bạn cần, nó tạo ra một ứng dụng duy nhất nên có thể dễ dàng kiểm thử tự động và dễ dàng triển khai.

Tuy nhiên, ứng dụng được xây dựng trên kiến trúc nguyên khối bắt đầu bộc lộ nhiều khuyết điểm khi số lượng người dùng tăng lên, ứng dụng cần nhiều tính năng mới, dữ liệu tăng, logic của ứng dụng phức tạp hơn, ứng dụng cần giao tiếp với các hệ thống khác. Và sau một thời gian không lâu, ứng dụng đơn giản sẽ trở thành một con quái vật, đi kèm theo đó sẽ là kinh phí bỏ ra để xây dựng, duy trì, nâng cấp là không hề nhỏ.

Bên cạnh đó ứng dụng nguyên khối chỉ dùng một loại ngôn ngữ lập trình duy nhất, làm cho tôi và lập trình viên nói chung ít phải học thêm nhiều cái mới, dễ làm quen với sự thuận tiện, dễ dàng khi chỉ cần nắm vững một ngôn ngữ và có thể giải quyết được hầu hết các vấn đề. Khi đó mọi người sẽ dần lệ thuộc vào ngôn ngữ lập trình đó và trở nên thiên vị, ngại học thêm hay tích hợp các công nghệ khác của ngôn ngữ khác.

Ứng dụng nguyên khối có một ưu điểm mà đa số lập trình viên rất thích là khả năng biên dịch và cho kết quả ngay (biên dịch nóng) mà không phụ vào bất kỳ ứng dụng nào khác. Tuy nhiên khi ứng dụng lớn lên, số lượng các chức năng tăng lên, thì việc biên dịch sẽ kéo dài hơn, khởi động lại chậm đi, và đặc biệt là khi mình chỉ thay đổi một chức năng nhỏ thì cũng phải khởi động lại toàn bộ hệ thống. Quá khó để triển khai lại cả một ứng dụng lớn chỉ vì một nâng cấp nhỏ, hoạt động của hệ thống sẽ bị ngưng trệ, việc kiểm thử sẽ mất nhiều công sức hơn rất nhiều.

Bài toán đặt ra là phải tìm giải pháp để giúp một hệ thống lớn website có thể hoạt động một cách hiệu quả, việc tạo dựng, duy trì và nâng cấp dễ dàng, các thành phần của hệ thống có thể hoạt động độc lập với nhau. Quan trọng nhất là thời gian và chi phí bỏ ra sẽ tiết kiệm được không ít. Nhiều lĩnh vực khác cũng có những hệ thống lớn và cũng cần một giải pháp tương tự như bài toán này tìm kiếm vì nó sẽ đem lại lợi ích không hề nhỏ.

Qua việc tìm kiếm giải pháp, tôi nhận thấy nhiều tập đoàn lớn như Amazon, eBay, Netflix,… đã giải quyết vấn đề kiến trúc nguyên khối bằng kiến trúc microservices (tập hợp các dịch vụ nhỏ). Ý tưởng là chia nhỏ ứng dụng lớn thành các dịch vụ nhỏ kết nối với nhau, điều này làm cho ứng dụng trở nên dễ hiểu, dễ phát triển, thử nghiệm, triển khai và trở nên linh hoạt hơn. Và tôi quyết định triển khai giải pháp này để thực hiện thử nghiệm, đánh giá.

1.2. Mục tiêu và phạm vi đề tài

Mục tiêu đầu tiên của đề tài chính là tìm ra được giải pháp giúp cho hệ thống website lớn có thể hoạt động một cách hiệu quả, giải quyết được những khuyết điểm của kiến trúc nguyên khối đã được nêu ra ở phần trên, sau đó xây dựng và thử nghiệm nó. Dựa vào kết quả đạt được mà tìm ra ưu điểm tốt mà giải pháp có cùng với những nhược điểm còn mắc phải, rồi tiếp tục đánh giá để có thể tìm được giải pháp tối ưu hơn.

Tuy nhiên do phạm vi đồ án có giới hạn thời gian, nên không đủ để xây dựng và triển khai được cả một hệ thống lớn trên thực tế, do đó đồ án này sẽ xây dựng một ứng dụng mô phỏng rằng nó là một hệ thống lớn và triển khai giải pháp tìm được trên nó. Ứng dụng được xem như một cách tiếp cận giải pháp, học hỏi cách xây dựng, triển khai và qua đó đánh giá tổng quan giải pháp để có thể tích lũy được kinh nghiệm và áp dụng vào các dự án cụ thể trên thực tế.

1.3. Định hướng giải pháp

Để giải quyết bài toán đặt ra, đồ án này sẽ thực hiện tìm hiểu, thiết kế và xây dựng một ứng dụng theo kiến trúc microservice. Kiến trúc microservices là tập hợp các dịch vụ nhỏ, trong đó mỗi dịch vụ nhỏ là một ứng dụng nhỏ thực hiện các chức năng chuyên biệt, có kiến trúc riêng, có thể lựa chọn ngôn ngữ lập trình, framework riêng và có thể cài đặt trên các nền tảng khác nhau. Do đó nó có thể được phát triển, thử nghiệm và triển khai độc lập mà không ảnh hưởng đến các phần khác của ứng dụng lớn. Khi vùng chức năng được đảm nhận bởi một dịch vụ nhỏ, đối tượng sử dụng có thể sẽ chỉ sử dụng đến một số ít dịch vụ nhỉ trong một thời điểm, giúp chức năng tối giản, dễ tương thích hơn, tốc độ nhanh hơn, tối ưu sử dụng hơn.

Để các dịch vụ nhỏ có thể giao tiếp với nhau, chúng có thể sẽ cung cấp REST API, và các dịch vụ khác sẽ gọi và sử dụng nó. Hiện nay có nhiều sự lựa chọn hỗ trợ khác như Google Protobuf, Apache thrift, Apache Avro tốn ít băng thông hơn. Đặc điểm của kết nối giữa các dịch vụ có thể là Synchronous (đồng bộ - gọi xong chờ) hoặc Asynchronous (bất đồng bộ - gọi xong chạy tiếp, khi có kết quả thì xử lý), cách gọi có thể là REST (tập lệnh gửi qua HTTP để truy vấn, thao tác dữ liệu, kiểu dữ liệu XML, JSON, JSONB), RPC (remote procedure call – lệnh gọi từ xa, kiểu dữ liệu binary, Thrift, Protobuf, Avro), SOAP (simple object access protocol).

Kiến trúc microservice ảnh hưởng lớn đến quan hệ bên trong ứng dụng và cơ sở dữ liệu. Thay vì dùng chung một sơ sở dữ liệu giữa các dịch vụ, mỗi dịch vụ sẽ có cơ sở dữ liệu riêng. Cách làm này đi ngược lại việc tập trung hóa cơ sở dữ liệu, hệ quả là sẽ có dư thừa, các ràng buộc quan hệ không thể áp dụng với các bảng ở hai cơ sở dữ liệu riêng biệt. Tuy nhiên, lưu dữ liệu ở từng dịch vụ rất quan trọng nếu muốn kiến trúc này đạt được hiệu quả cao và đảm bảo ít ràng buộc. Đây cũng chỉ là khuyến nghị của tác giả, trong thực tế thì các dịch vụ vẫn có thể dùng chung một cơ sở dữ liệu khi tính toàn vẹn dữ liệu cần ưu tiên cao nhất.

Áp dụng kiến trúc microservice vào ứng dụng trong đồ án này, để có được một thực nghiệm tốt để đánh giá mà vẫn đảm bảo hoàn thành với quy mô đồ án, tôi lựa chọn xây

dựng một ứng dụng web quản lý bán hàng triển khai kiến trúc microservice trên nền tảng .NetCore với ngôn ngữ lập trình C#. Việc xây dựng kết hợp sử dụng RabbitMQ là một loại message broker để các dịch vụ nhỏ giao tiếp với nhau, sử dụng SignalR để giao tiếp giữa các dịch vụ thuộc cùng khối chức năng, sử dụng Entity Framework Core để thao tác với cơ sở dữ liệu, sử dụng Angular JS để xây dựng ứng dụng single page application phía frontend cung cấp giao diện sử dụng cho người dùng.

1.4. Bố cục đồ án

Với định hướng giải pháp ở chương 1, chương 2 thực hiện khảo sát hiện trạng, đưa ra các ưu nhược điểm của microservice, trình bày tổng quan các chức năng của ứng dụng cần có thông qua các biểu đồ use case và đặc tả use case. Qua khảo sát và phân tích yêu cầu ở chương 2, chương 3 sẽ đưa ra sơ sở lý thuyết về các công nghệ mà ứng dụng sử dụng, trình bày định nghĩa, đánh giá ưu nhược điểm của công cụ và đưa ra lý do lựa chọn công nghệ. Sau khi đã phân tích tổng quan chức năng và lựa chọn công nghệ, chương 4 thực hiện phân tích và thiết kế chi tiết các chức năng của ứng dụng, sau đó xây dựng, kiểm thử và thực hiện triển khai ứng dụng. Cuối cùng là đưa ra kết luận và hướng phát triển sau đồ án tại chương 5.

CHƯƠNG 2: KHẢO SÁT VÀ PHÂN TÍCH YÊU CẦU

2.1. Khảo sát hiện trạng

Kiến trúc microservice cũng đã được nhiều công ty lớn áp dụng và đem lại lợi ích không hề nhỏ, đa số đó là các website thương mại điện tử như Amazon, eBay, Netflix,… Tuy nhiên chúng ta không nên chỉ nhìn vào điều này mà lao vào áp dụng cho dự án mủa mình, cần cân nhắc, khảo sát kỹ trước khi triển khai ứng dụng của mình theo kiến trúc này vì không có kiến trúc nào là hoàn hảo cả, tùy vào từng bài toán thực tế mà sẽ phù hợp loại kiến trúc nào đó hơn. Sau đây là một số ưu nhược điểm của kiến trúc microservice đã khảo sát được:

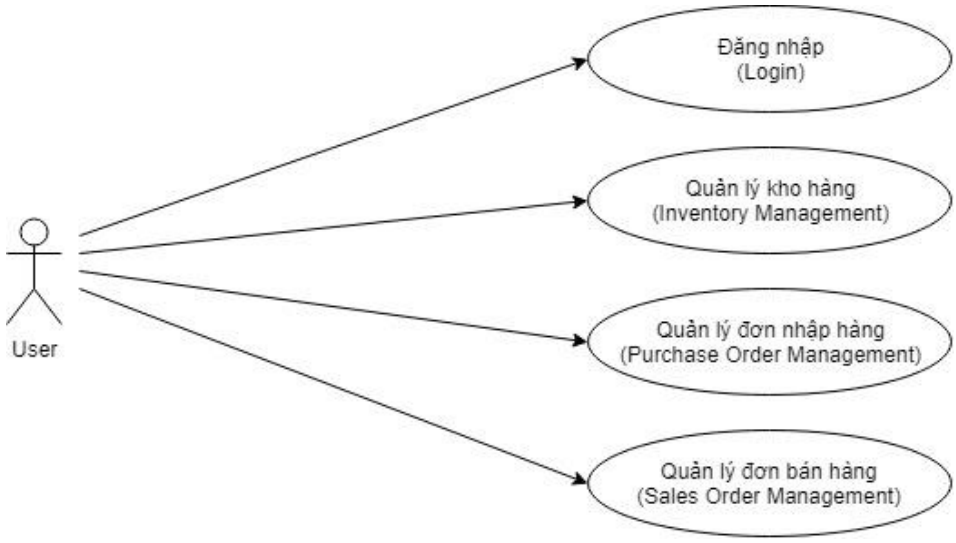
- *Ưu điểm:*
  - Giảm thiểu gia tăng sự phức tạp của một hệ thống lớn
  - Chia nhỏ ứng dụng nguyên khối cồng kềnh thành các dịch vụ nhỏ làm cho việc quản lý, bảo trì, nâng cấp trở nên dễ dàng hơn, tự do chọn và nâng cấp công nghệ mới.
  - Mỗi microservice sẽ định ra ranh giới rõ ràng dưới dạng RPC hay API hướng thông điệp.
  - Microservice thúc đẩy tách rời các khối chức năng, điều mà rất khó thực hiện với ứng dụng nguyên khối.
  - Một số microservice có thể thuê ngoài mà vẫn đảm bảo bảo mật hệ thống và mã nguồn phần dịch vụ còn lại. Đội phát triển có nhiều lựa chọn công nghệ mới, framework mới, cơ sở dữ liệu mới, môi trường tối ưu để chạy, đa dạng để nâng cấp từng dịch vụ nhỏ.
- *Nhược điểm:*
  - Cần phải tính toán tốt để phân chia ứng dụng lớn thành các microservice có kích thước hợp lý để có thể dễ kiểm soát.
  - Phải xử lý sự cố khi kết nối chậm, lỗi khi thông điệp không gửi được hoặc thông điệp đến nhiều đích vào các thời điểm khác nhau.
  - Đảm bảo giao dịch phân tán cập nhật dữ liệu đúng đắn vào nhiều dịch vụ nhỏ khác nhau khó hơn nhiều, đôi khi là không thể so với đảm bảo giao dịch cập nhật vào nhiều bảng trong cơ sở dữ liệu trung tâm.
  - Theo nguyên tắc CAP (CAP theorem) thì giao dịch phân tán sẽ không thể thỏa mãn cả 3 điều kiện: consistency (dữ liệu ở các điểm khác nhau trong mạng phải giống nhau), availablity (yêu cầu gửi đi phải có phúc đáp), partition tolerance (hệ thống vẫn hoạt động được ngay cả khi mạng bị lỗi). Những công nghệ cơ sở dữ liệu phi quan hệ (NoSQL) hay môi giới thông điệp (message broker) tốt nhất hiện nay cũng chưa vượt qua nguyên tắc CAP.
  - Kiểm thử tự động một dịch vụ trong kiến trúc microservice đôi khi yêu cầu phải chạy cả các dịch vụ nhỏ khác mà nó phụ thuộc. Do đó khi phân tách ứng dụng nguyên khối thành microservice cần luôn kiểm tra mức độ ràng buộc giữa các dịch vụ mềm dẻo hơn hay cứng nhắc hơn hay lệ thuộc hơn. Nếu ràng buộc ít đi, lỏng lẻo hơn thì bạn đang đi đúng hướng và ngược lại.

- Nếu các dịch vụ nhỏ thiết kế phụ thuộc vào nhau theo kiểu đường ống A gọi B, B gọi C, C gọi D, và một mắt xích có giao tiếp API thay đổi, liệu các mắt xích khác có phải thay đổi theo không? Nếu có thì việc bảo trì, kiểm thử sẽ phức tạp tương tự một ứng dụng nguyên khối. Thiết kế dịch vụ tốt sẽ giảm tối đa ảnh hưởng lan truyền đến các dịch vụ khác.
- Khi có quá nhiều microservice được phân tách thì số lượng kết nối giữa các microservice sẽ tăng lên đáng kể khiến khó kiểm soát hệ thống. Nếu không có quy tắc phân luồng – quản lý – đo đếm – theo dõi hợp lý thì số lượng kết nối giữa các dịch vụ sẽ gia tăng một cách tùy tiện, chất lượng kết nối không đảm bảo, khi đó hệ thống sẽ bị chậm mà không thể biết đoạn tắc nghẽn nằm ở vị trí nào.
- Triển khai ứng dụng microservices nếu làm thủ công theo cách tương tự với triển khai ứng dụng nguyên khối thì sẽ phức tạp hơn rất nhiều. Ứng dụng nguyên khối bỏ xung các server mới giống hệt nhau đằng sau bộ cân bằng tải. Trong khi ở kiến trúc microservice thì các dịch vụ nhỏ nằm trên nhiều máy ảo hay Docker container khác nhau, hoặc một dịch vụ có nhiều thực thể nằm phân tán

Tóm lại, kiến trúc nguyên khối sẽ hữu hiệu với ứng dụng đơn giản, ít chức năng. Nó bộc lộ nhiều nhược điểm khi ứng dụng phát triển lớn nhiều chức năng. Kiến trúc microservice chia nhỏ kiến trúc nguyên khối ra nhiều dịch vụ nhỏ. Microservice sẽ hiệu quả, phù hợp cho những ứng dụng phức tạp, liên tục phát triển nếu được thiết kế đúng và tận dụng các công nghệ quản lý và vận hành tự động. Hướng đi được coi là thích hợp và đơn giản là trước hết cần tập trung phát triển ứng dụng dạng nguyên khối, với nhiều module riêng biệt trước, khi vai trò của các module trở nên rõ ràng, ta có thể tách dần các module này thành các service riêng.

2.2. Tổng quan chức năng

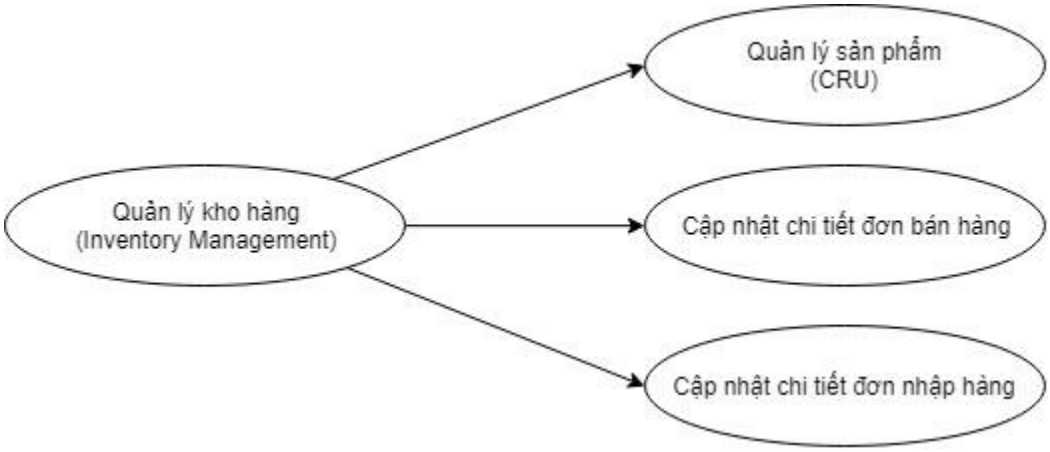
2.2.1. Biểu đồ use case tổng quan



Hình 2.1: Biểu đồ use case tổng quan

Biểu đồ mô tả các chức năng tổng quan mà người dùng có thể sử dụng trên hệ thống. Hệ thống có một tác nhân chính là người sử dụng, để sử dụng các chức năng của hệ thống, việc đầu tiên cần làm là đăng nhập, sau đó người dùng có thể sử dụng các khối chức năng là quản lý kho hàng, quản lý đơn nhập hàng và quản lý đơn bán hàng

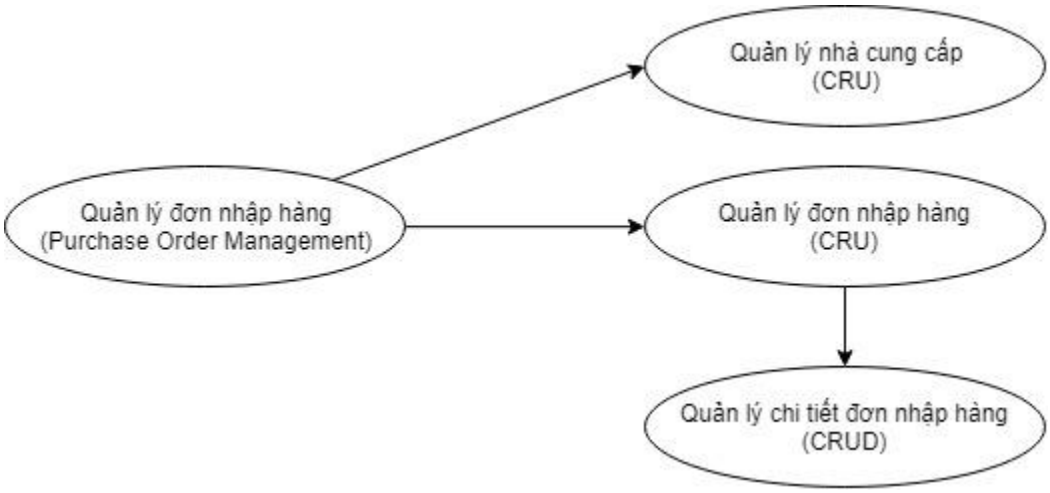
2.2.2. Biểu đồ use case phân rã chức năng quản lý kho hàng



Hình 2.2: Biểu đồ use case phân rã chức năng quản lý kho hàng

Mô tả các chức năng trong khối chức quản lý kho hàng mà người dùng có thể sử dụng. Sau khi đăng nhập và truy cập vào khối chức năng quản lý kho hàng, người dùng có thể quản lý danh sách sản phẩm (xem, thêm, sửa), cập nhật chi tiết đơn nhập hàng, đơn bán hàng, cụ thể là cập nhật số lượng sản phẩm có thể giao trong các đơn hàng.

2.2.3. Biểu đồ use case phân rã chức năng quản lý nhập hàng

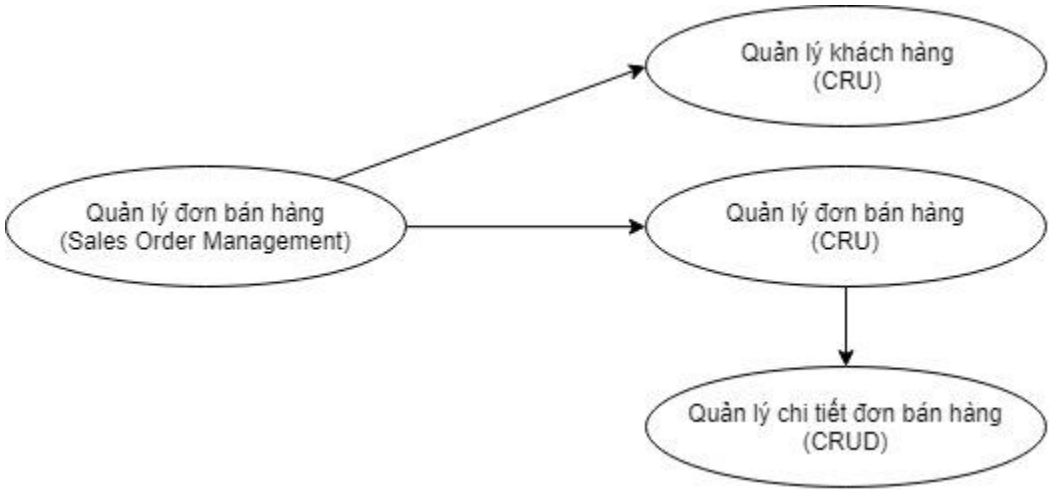


Hình 2.3: Biểu đồ use case phân rã chức năng quản lý đơn nhập hàng

Mô tả các chức năng trong khối chức năng quản lý đơn nhập hàng mà người dùng có thể sử dụng. Sau khi đăng nhập và truy cập vào khối chức năng quản lý đơn nhập

hàng, người dùng có thể quản lý danh sách nhà cung cấp (xem, thêm, sửa), quản lý danh sách đơn bán hàng (xem, thêm, sửa), quản lý chi tiết đơn bán hàng (xem, thêm, sửa, xóa).

2.2.4. Biểu đồ use case phân rã chức năng quản lý bán hàng



Hình 2.4: Biểu đồ use case phân rã chức năng quản lý đơn bán hàng

Mô tả các chức năng trong khối chức năng quản lý đơn bán hàng mà người dùng có thể sử dụng. Sau khi đăng nhập và truy cập vào khối chức năng quản lý đơn bán hàng, người dùng có thể quản lý danh sách khách hàng (xem, thêm, sửa), quản lý danh sách đơn bán hàng (xem, thêm, sửa), quản lý chi tiết của đơn bán hàng (xem, thêm, sửa, xóa).

2.3. Đặc tả chức năng

2.3.1. Đặc tả use case quản lý sản phẩm

Tác nhân hệ thống	Người dùng
Luồng sự kiện chính	1. Người dùng tạo mới sản phẩm • Input: thông tin sản phẩm • Output: tạo mới sản phẩm thành công 2. Người dùng xem danh sách sản phẩm 3. Người dùng click vào sản phẩm trong danh sách để xem chi tiết. 3. Người dùng thay đổi thông tin sản phẩm và cập nhật • Input: thông tin sản phẩm • Output: cập nhật sản phẩm thành công
Luồng sự kiện thay thế	1. Người dùng tạo mới sản phẩm không thành công và xuất thông báo lỗi 2. Người dùng cập nhật thông tin sản phẩm không thành công và xuất thông báo lỗi

Bảng 2.1: Đặc tả use case quản lý sản phẩm

2.3.2. Đặc tả use case quản lý đơn nhập hàng và chi tiết đơn nhập hàng

Tác nhân hệ thống	Người dùng
Luồng sự kiện chính	1. Người dùng tạo mới đơn nhập hàng • Input: thông tin đơn nhập hàng • Output: tạo mới đơn hàng thành công 2. Người dùng xem danh sách đơn nhập hàng 3. Người dùng click vào đơn để xem chi tiết 4. Người dùng thay đổi thông tin đơn nhập hàng và cập nhật • Input: Thông tin đơn nhập hàng • Output: Cập nhật đơn hàng thành công 5. Người dùng thêm, sửa, xóa chi tiết trong đơn nhập hàng và cập nhật • Input: Thông tin chi tiết đơn nhập hàng • Output: Cập nhật đơn hàng thành công
Luồng sự kiện thay thế	1. Người dùng tạo mới đơn nhập hàng không thành công và xuất thông báo lỗi 2. Người dùng cập nhật đơn hàng không thành công và xuất thông báo lỗi

Bảng 2.2: Đặc tả use case quản lý đơn nhập hàng

2.3.3. Đặc tả use case quản lý đơn bán hàng và chi tiết đơn bán hàng

Tác nhân hệ thống	Người dùng
Luồng sự kiện chính	1. Người dùng tạo mới đơn bán hàng • Input: thông tin đơn bán hàng • Output: tạo mới đơn hàng thành công 2. Người dùng xem danh sách đơn bán hàng 3. Người dùng click vào đơn để xem chi tiết 4. Người dùng thay đổi thông tin đơn bán hàng và cập nhật • Input: Thông tin chi tiết đơn bán hàng • Output: Cập nhật đơn hàng thành công 5. Người dùng thêm, sửa, xóa chi tiết trong đơn bán hàng và cập nhật • Input: thông tin chi tiết đơn bán hàng • Output: cập nhật đơn hàng thành công
Luồng sự kiện thay thế	1. Người dùng tạo mới đơn hàng bán không thành công và xuất thông báo lỗi 2. Người dùng cập nhật đơn hàng không thành công và xuất thông báo lỗi

Bảng 2.3: Đặc tả use case quản lý đơn bán hàng

CHƯƠNG 3: CÔNG NGHỆ SỬ DỤNG

3.1. Asp.Net Core 2.2

3.1.1. ASP.NET là gì ?

ASP.NET là nền tảng phát triển web (web application frameword), cung cấp một mô hình lập trình, cơ sở hạ tầng phần mềm toàn diện và các dịch vụ cần thiết để xây dựng các ứng dụng web động mạnh mẽ cho máy tính cũng như trên các thiết bị di động.

ASP.NET là một phần của nền tảng Microsoft.NET. Ứng dụng ASP.NET được biên dịch mã, được viết bằng cách sử dụng mở rộng và tái sử dụng các thành phần hoặc đối tượng trong nền tảng NET. Các mã này được sử dụng cho toàn bộ hệ thống phân cấp của các class trong .NET

Các ứng dụng ASP.NET có thể được viết bằng bất kỳ ngôn ngữ nào sao đây: C#, Visual Basic.Net, Jscript, J#

ASP.NET được sử dụng để tạo ra các tương tác, dữ liệu điều khiển các ứng dụng web trên internet. ASP.NET bao gồm một số lượng lớn các controls như là các text box, button và labels cho assembling, và các thao tác mã để tạo ra các trang HTML.

3.1.2. ASP.NET Core là gì?

ASP.NET core là một mã nguồn mở và là nền tảng mới cho xây dựng cloud trên internet kết nối các ứng dụng web, loT và mobile backends. ASP.NET Core có thể chạy trên .NET Core hoặc chạy đầy đủ trên .NET Framework. ASP.NET Core được kiến trúc để cung cấp một nền tảng phát triển tối ưu cho các ứng dụng được triển khai tới cloud hoặc chạy on-premises. ASP.NET Core bao gồm các thành phần mô-đun cần thiết tối thiểu, do đó bạn giữ lại được tính linh hoạt trong khi xây dựng các solution của bạn. Bạn có thể phát triển và chạy ASP.NET Core trên Windows, MAC và LINUX. ASP.NET Core là mã nguồn mở tại GitHub

3.1.3. Tại sao cần xây dựng ASP.NET Core ?

Các phiên bản đầu tiên của ASP.NET đã được ra đời gần 15 năm trước đây như là một phần của .NET Framework. Kể từ đó có hàng triệu nhà phát triển đã sử ASP.NET để xây dựng và chạy các ứng dụng web tuyệt vời, và trong những năm qua, đã được Microsoft liên tục phát triển.

ASP.NET Core có một số thay đổi kiến trúc làm cho nó gọn nhẹ hơn. ASP.NET Core không còn dựa trên System.Web.dll mà dựa trên tập hợp các granular và các NuGet. Nhờ thế cho phép bạn tối ưu hóa các ứng dụng của bạn nhờ chỉ cần sử dụng các gói NuGet mà bạn cần. Ngoài ra nó bảo mật hơn, cải thiện hiệu suất và giảm chi phí nhờ bạn chỉ cần chỉ trả cho những gì bạn sử dụng mà thôi

3.1.4. ASP.NET core có những cải tiến cơ bản nào ?

- Web UI và Web API được hợp nhất
- Tích hợp các frameworks hiện đại cho khách hàng và nhà phát triển
- A cloud-ready environment-based configuration system
- Built-in dependency injection
- New light-weight and modular HTTP request pipeline
- Khả năng lưu trữ trên IIS hoặc tự chủ trong tiến trình riêng của bạn
- Built on .NET Core, which supports true side-by-side app versioning
- Ships entirely as NuGet packages
- Công cụ vừa mới và đơn giản để phát triển các web hiện đại
- Xây dựng và chạy nền tảng ứng dụng ASP.NET không chỉ trên Windows mà còn trên MAC và Linux nữa
- Mã nguồn mở và có một cộng đồng lớn

3.1.5. Đặc điểm riêng biệt của ASP.NET Core:

- Hỗ trợ đa nền tảng: Ngày nay bạn có thể phát triển và chạy ASP.NET trên cả Windows, Mac, và Linux. Và nếu trên Windows bạn có thể thể sử dụng công cụ tốt nhất Visual Studio 2015 để tạo, quản lý và gỡ lỗi các ứng dụng ASP.NET Core của bạn, thì nay trên bất kỳ nền tảng nào bạn có thể sử dụng Visual Studio Code. Visual Studio Code là một trình soạn thoải với các plugin có hỗ trợ để chỉnh sửa các ứng dụng ASP.NET Core của bạn.
- Nguồn mở: Ngày nay mã nguồn và tài liệu ASP.NET đã được Microsoft mở tất cả. Các mã nguồn giờ đã có sẵn trong Github bạn giờ có thể tải hay thay đổi bất kỳ mã nào mà bạn thích. Nếu bạn có bất kỳ một góp ý cải tiến gì đó, bạn có thể gửi một yêu cầu đến cho microsoft để xem xét và kết hợp. Tương tự như vậy, tất cả các tài liệu cũng là mã nguồn mở và có sẵn trong bài viết docs.asp.net. Mỗi trang trên đó điều có chức năng “chỉnh sửa trang này” ở phía trên và bạn có thể chỉnh sửa các tài liệu từ Microsoft
- Hỗ trợ đầy đủ cho framework: Một thông tin hữu ích là asp.net core dẫu trở thành mã nguồn mở nhưng Microsoft vẫn sẽ cung cấp hỗ trợ trong 3 năm cho mỗi bản phát hành lớn nhỏ của họ.
- Hiệu suất: Microsoft giới thiệu máy chủ mới web Kestrel chạy trong host IIS của bạn hoặc chạy sau một host process khác. Kestrel hiện tại là máy chủ .NET chạy nhanh nhất hiện nay.
- Hỗ trợ xây dựng bằng Dependency Injection: Nếu mà giải thích thì chỉ vài dòng thì sẽ hơi khó hiểu. ASP.NET Core đã được xây dựng trong Dependency Injection.Dependency Injection là một mẫu thiết kế cho phép các phụ thuộc của một class được injected như các đối tượng được yêu cầu trong ứng dụng của bạn. Với ASP.NET Core, Microsoft đã cung cấp một Dependency Injection mà bạn có thể sử dụng để xác định sự phụ thuộc được đưa vào Controller, View của bạn, hoặc bất kỳ lớp học khác mà framework sẽ tạo ra cho bạn. Bạn có thể bắt đầu việc cấu hình thông qua phương thức ConfigureServices trong tập tin Startup.cs.

- Một Framework duy nhất: Trong ASP.NET Core, Microsoft đã đưa tất cả các framework vào một framework duy nhất vừa nhẹ hơn và vừa có những tính năng của MVC và WebAPI. Với việc sáp nhập này của MVC và Web API, mọi thứ đơn giản hơn nhiều khi bạn không cần phải cân nhắc định tuyến khác nhau, an toàn, hoặc các bộ lọc cho một ApiController so với MVC Controller. Tất cả các lớp Controller hiện giờ có thể xử lý các yêu cầu sử dụng API Web hoặc cách tiếp cận MVC.
- MVC Helpers Tag: Với ASP.NET Core, Microsoft đã giới thiệu tag helpers để tạo ra mã phía client từ .NET và làm cho nó dễ dàng hơn để tái sử dụng trong Razor markup. Nó được tham chiếu trong đánh dấu phía máy chủ của bạn như thể họ là một tag HTML mà bạn được sử dụng. Công cụ Razor sẽ nhận ra thẻ và thực thi các mã .NET có liên quan tương ứng với nó.

3.1.6. Phiên bản Asp.Net Core 2.2

Chủ đề chính cho bản release ASP.NET Core này là cải tiến hiệu năng cho developer và các tính năng nền tảng phục vụ việc xây dựng Web/ HTTP APIs. Như thường lệ, họ cũng có cải tiến hiệu năng. Các cải tiến nổi bật là:

- Tích hợp tốt hơn với một số Open API (Swagger) bao gồm kiểm tra chất lượng code thời điểm phát triển.
- Giới thiệu về Endpoint Routing với cải tiến 20% về hiệu năng cho routing trong MVC
- Cải tiến về cách generate URL với LinkGenerator class và hỗ trợ cho điều hướng Parameter Transformers.
- API kiểm tra tình trạng sức khoẻ của ứng dụng mới
- Cải tiến 400% việc xử lý hỗ trợ IIS hosting in-process
- Cải tiến 15% hiệu năng cho việc model validation
- Problem Details (RFC 7807) hỗ trợ trong MVC cho việc chi tiết hoá kết quả lỗi.
- Preview cho HTTP/2 hỗ trợ ASP.NET Core
- Template cập nhật Bootstrap 4 và Angular 6
- Java client cho ASP.NET Core SignalR
- Cải tiến hiệu năng đến 60% cho HTTP Client trên Linux và 20% trên Windows.

3.2. RabbitMQ

3.2.1. Giới thiệu

RabbitMQ là một message broker (message-oriented middleware) sử dụng giao thức AMQP - Advanced Message Queue Protocol (Đây là giao thức phổ biến, thực tế rabbitmq hỗ trợ nhiều giao thức). RabbitMQ được lập trình bằng ngôn ngữ Erlang. RabbitMQ cung cấp cho lập trình viên một phương tiện trung gian để giao tiếp giữa nhiều thành phần trong một hệ thống lớn (Ví dụ openstack). RabbitMQ sẽ nhận message đến từ các thành phần khác nhau trong hệ thống, lưu trữ chúng an toàn trước khi đẩy đến đích.

Thực sự, với lập trình viên thì rabbitmq rất đáng giá. Nếu không có các hệ thống message broker như rabbitmq thì bất cứ lúc nào cần đẩy data giữa các thành phần trong hệ thống, lập trình viên cần một kết nối trực tiếp. Một hệ thống càng lớn. Số thành phần càng nhiều, mức độ trao đổi message giữa càng thành phần cũng vì thế tăng lên khiến việc lập trình trở nên phức tạp. Tôi từng đọc vài bài báo về lập trình thì thấy họ khuyến cáo các lập trình viên chỉ nên tập trung vào business logic của ứng dụng còn các công tác hậu trường thì nên được tái sử dụng các giải pháp đã có. Rabbitmq cũng là một giải pháp rất tốt trong các kiến trúc hệ thống lớn.

3.2.2. Tại sao lại sử dụng RabbitMQ

Chúng ta thử xem các message broker như rabbitmq đem lại lợi ích gì trong việc thiết kế ứng dụng. Trong một hệ thống phân tán (distributed system), có rất nhiều thành phần. Nếu muốn các thành phần này giao tiếp được với nhau thì chúng phải biết nhau. Nhưng điều này gây rắc rối cho việc viết code. Một thành phần phải biết quá nhiều đâm ra rất khó maintain, debug. Giải pháp ở đây là thay vì các liên kết trực tiếp, khiến các thành phần phải biết nhau thì sử dụng một liên kết trung gian qua một message broker. Với sự tham gia của message broker thì producer sẽ không hề biết consumer. Nó chỉ việc gửi message đến các queue trong message broker. Consumer chỉ việc đăng ký nhận message từ các queue này.

Tất nhiên, có thể có một giải pháp là sử dụng database để lưu các message trong các temporary table. Tuy nhiên xét về hiệu năng thì không thể bằng message broker vì một số lý do: Tần xuất trao đổi message cao sẽ làm tăng load của database, giảm performance đáng kể. Trong môi trường multithread, database cần có cơ chế lock. Lock cũng làm giảm performance. Sử dụng message broker sẽ không có vấn đề này.

Vì producer nói chuyện với consumer trung gian qua message broker nên dù producer và consumer có khác biệt nhau về ngôn ngữ thì giao tiếp vẫn thành công. Dù viết bằng java, python, php hay ruby... thì chỉ cần thỏa mãn giao thức với message broker thì thông suốt hết. HIện nay, rabbitmq cũng đã cung cấp client library cho khá nhiều các ngôn ngữ rồi. Tính năng này cho phép tích hợp hệ thống linh hoạt.

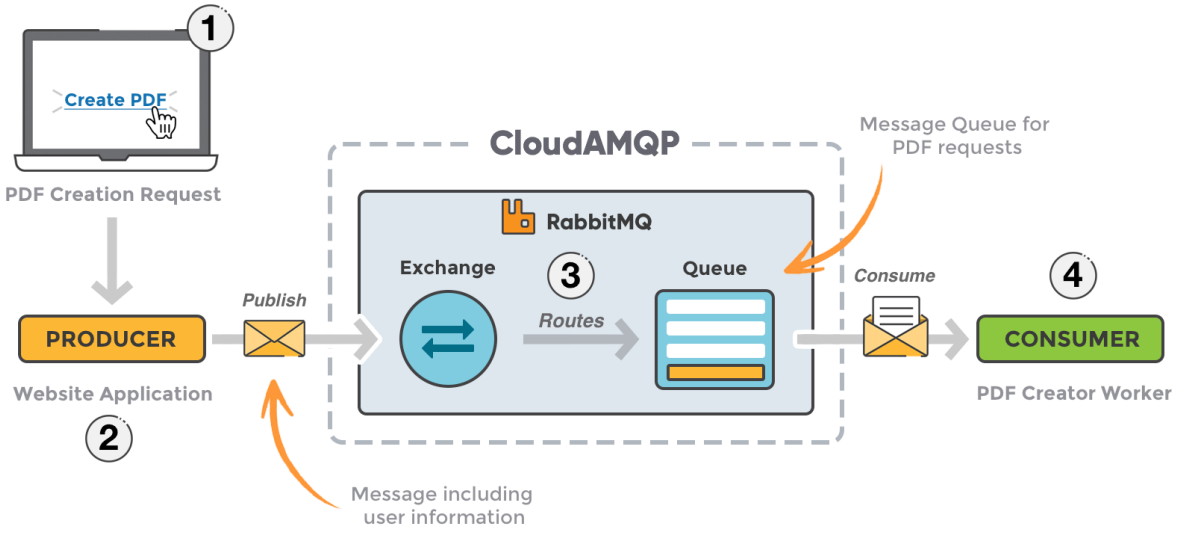
Một đặc tính của rabbitmq là asynchronous. Producer không thể biết khi nào message đến được consumer hay khi nào message được consumer xử lý xong. Đối với producer, đẩy message đến message broker là xong việc. Consumer sẽ lấy message về khi nó muốn. Đặc tính này có thể được tận dụng để xây dựng các hệ thống lưu trữ và xử lý log. ELK stack - Elasticsearch Logstash Kibana là một ví dụ.

Bên cạnh các lợi ích kể trên, rabbitmq còn có nhiều tính năng thú vị khác như:

- *cluster*: các bạn có thể gom nhiều rabbitmq instance vào một cluster. Một queue được đinh nghĩa trên một instance khi đó đều có thể truy xuất từ các instance còn lại. Có thể tận dụng để làm load balancing (tuy rằng có hạn chế, tôi sẽ nói sau)
- *high availibilty*: cho phép failover khi sử dụng mirror queue.
- *reliability*: có cơ chế ack để đảm bảo message được nhận bởi consumer đã được xử lý.

3.2.3. Mô hình

Hàng đợi tin nhắn cho phép các máy chủ web trả lời các yêu cầu một cách nhanh chóng thay vì buộc phải thực hiện các thủ tục nặng về tài nguyên ngay tại chỗ. Hàng đợi tin nhắn cũng tốt khi bạn muốn phân phối một tin nhắn cho nhiều người nhận để tiêu thụ hoặc để cân bằng tải giữa các công nhân. Người tiêu dùng có thể nhận một tin nhắn của hàng đợi và bắt đầu xử lý PDF cùng lúc với nhà sản xuất đang xếp hàng các tin nhắn mới trên hàng đợi. Người tiêu dùng có thể ở trên một máy chủ hoàn toàn khác với nhà xuất bản hoặc họ có thể được đặt trên cùng một máy chủ. Yêu cầu có thể được tạo bằng một ngôn ngữ lập trình và được xử lý bằng ngôn ngữ lập trình khác - hai ứng dụng sẽ chỉ liên lạc qua các tin nhắn mà chúng đang gửi cho nhau. Do đó, hai ứng dụng sẽ có sự kết hợp thấp giữa người gửi và người nhận.



Hình 3.1: Mô hình hoạt động của RabbitMQ

3.3. SignalR

ASP.NET SignalR là một thư viện cho các lập trình viên Asp.Net đơn giản hóa quá trình thêm chức năng web real-time trong phát triển ứng dụng. Real-time web functionality là gì ? Đó là khả năng server đẩy những nội dung tới client đã được kết nối một cách tức thì. Nó khác với giao thức HTTP thông thường: server đợi những yêu cầu từ client và trả về nội dung tương ứng.

SignalR có thể sử dụng trong bất kì chức năng web real-time nào. Trong đó ứng dụng chat trên web là một ví dụ điển hình. Ngoài ra, các ứng dụng cho dashboards, monitoring, collaborative là những gợi ý cho việc sử dụng SignalR.

SignalR cung cấp một API đơn giản cho việc tạo server-to-client remote procedure call (RPC) để gọi những hàm javascript trong trình duyệt (và những nền tảng khác) từ code .Net của server-side. SignalR cũng bao gồm API cho việc quản lý kết nối (connect và disconnect events) và những kết nối nhóm.

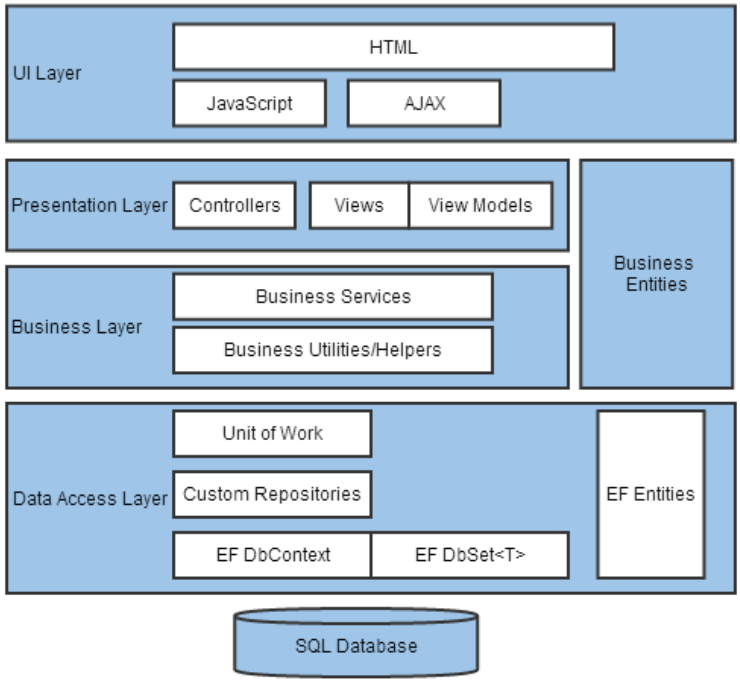
SignalR xử lý quản lý kết nối một cách tự động, và cho bạn truyền đi thông điệp tới tất cả các client đã được kết nối một cách đồng loạt, giống như một chat room. Bạn cũng có thể gửi những thông điệp tới những client được xác định. Kết nối giữa client và server là liên tục, không giống như kết nối HTTP cổ điển, cái mà sẽ thành lập lại kết nối cho mỗi lần giao tiếp.

SignalR hỗ trợ chức năng "server push", trong server code có thể gọi tới client code trong trình duyệt bởi "Remote Procedure Calls" (RPC), hơn là sử dụng Service Bus, SQL Server hay Redis. SignalR là mã nguồn mở, có thể truy cập thông qua GitHub.

3.4. Entity Framework Core 2.2

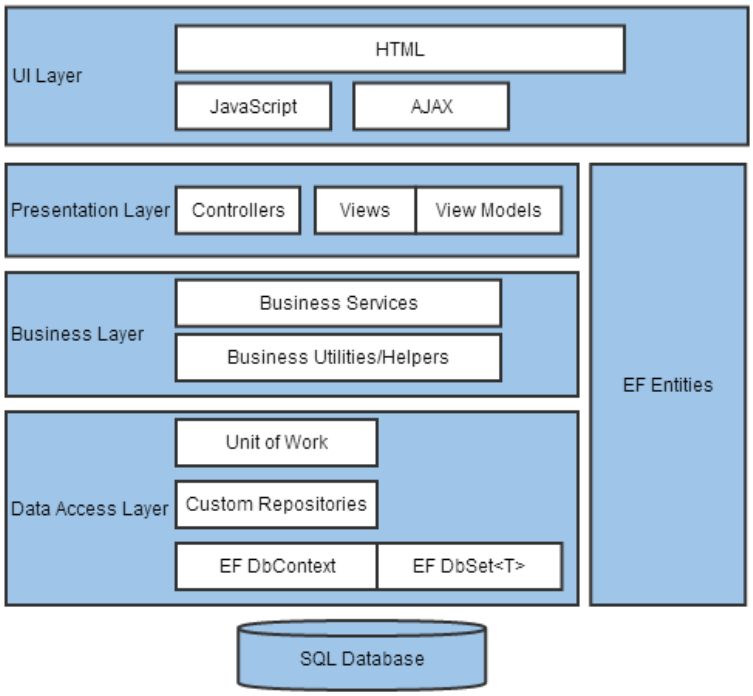
Entity Framework (EF) là một framework ánh xạ quan hệ đối tượng (ORM) dành cho ADO.NET, là 1 phần của .NET Framework. EF cho phép các nhà phát triển Web tương tác với dữ liệu quan hệ theo phương pháp hướng đối tượng đặc trưng. Lợi ích lớn nhất của EF là giúp lập trình viên giảm thiểu việc lập trình mã nguồn cần thiết để truy cập và tương tác với cơ sở dữ liệu. EF được Microsoft hỗ trợ phát triển lâu dài và bền vững, vì vậy EF là 1 framework mạnh nhất hiện nay để phát triển ứng dụng Web với sự hỗ trợ đông đảo của các nhà phát triển Web.

Chắc hẳn, bạn đã biết về mô hình Web 3 tầng (n tầng) khi đọc sách, giáo trình, bài giảng về nội dung lập trình Web. Tuy nhiên, vấn đề bạn cần phải hiểu rõ là: việc định rõ số lượng các tầng và các mối liên hệ giữa các tầng trong mô hình phát triển Web cũng rất đa dạng, tùy theo cách hiểu các lập trình viên và dự án Web. Do đó, việc hiểu Entity Framework nằm ở đâu trong mô hình Web 3 tầng cũng không hoàn toàn dễ dàng. Trong phần này, tôi có tham khảo từ Stephen M. Redd nhằm giúp các bạn hiểu rõ bản chất vấn đề hơn. Đầu tiên, chúng ta có vị trí của EF trong mô hình dự án Web ASP.NET MVC.

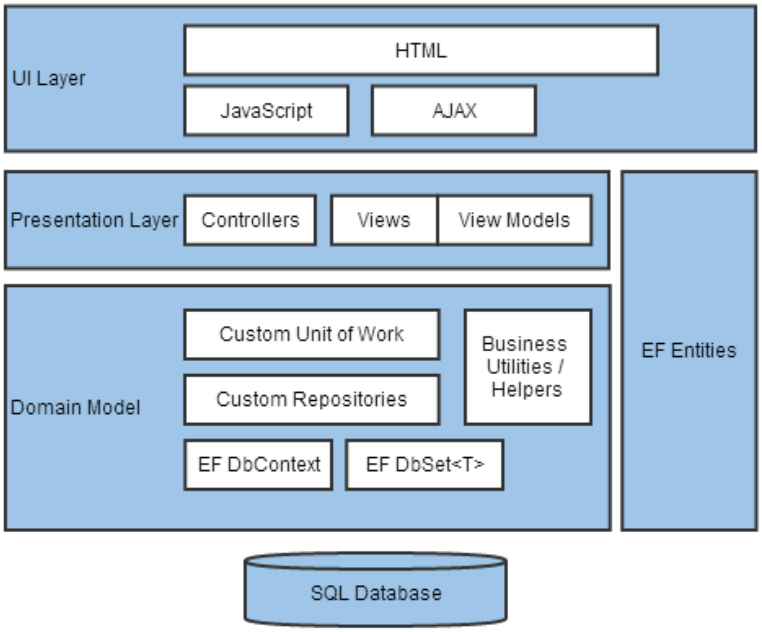


Hình 3.2: Vị trí của EF trong mô hình dự án Web Asp.Net MVC

Trong mô hình trên, chúng ta thấy vị trí của EF nằm trọn trong tầng Data Access Layer, tức là EF đóng vai trò là nơi trung gian để tương tác dữ liệu từ database với các tầng cao hơn, chẳng hạn như Business Layer. Hay nói cách khác, EF đóng vai trò như là phiên bản mới của ADO.NET. Tuy nhiên vị trí của EF trong mô hình này chỉ có trên lý thuyết. Thật sự, khi bắt tay xây dựng nhiều dự án Web, vị trí thường thấy EF ở các dự án website trên thực tế như mô hình sau.



Hình 3.3: Vị trí của EF trong các mô hình dự án thực tế



Hình 3.4: Vị trí của EF trong một vài mô hình khác

Trong hình 3.3, EF dường như có vị trí rất tự do, xuất hiện ở cả 3 tầng Data Access Layer, Business Layer, và cả Presentation Layer. Có thể hình dung EF như là 1 thư viện, hễ tầng nào cần thì chỉ cần gọi đến. Mô hình này xuất phát có thể là sự cẩu thả của lập trình viên khi không thích tuân theo tiêu chuẩn quan hệ giữa các tầng hoặc cách giải thích khác là sự lỏng lẻo giúp cho việc lập trình thuận tiện hơn

Trong vài mô hình người ta gom tầng Data Access Layer và Business Layer chỉ làm tầng Data Model, và vị trí EF cũng bao hàm tầng Data Model và Presentation Layer như hình trên. Đến đây, ta có thể hiểu 1 cách chung nhất EF có vị trí trung gian, đóng vai trò kết nối giữa cơ sở dữ liệu và các thành phần khác của 1 dự án Web khi cần đến. Ngoài ra, có nhiều cách hiểu về vị trí của EF ở đâu trong mô hình Web, ta sẽ dần khám phá để đưa ra cách hiểu và định nghĩa riêng trong quá trình thiết kế và xây dựng dự án Web ASP.NET

3.5. Angular JS

3.5.1. AngularJS là gì?

Đây là một mã nguồn mở, 1 framwork cho các ứng dụng web. Được phát triển từ năm 2009, hiện tại được duy trì bởi google và đã ra mắt phiên bản 2.0

Định nghĩa chính thức được đưa ra như sau : AngularJS là một framework có cấu trúc cho các ứng dụng web động. Nó cho phép bạn sử dụng HTML như là ngôn ngữ mẫu và cho phép bạn mở rộng cú pháp của HTML để diễn đạt các thành phần ứng dụng của bạn một cách rõ ràng và súc tích. Hai tính năng cốt lõi: Data binding và Dependency injection của AngularJS loại bỏ phần lớn code mà bạn thường phải viết. Nó xảy ra trong tất cả các trình duyệt, làm cho nó trở thành đối tác lý tưởng của bất kỳ công nghệ Server nào.

Để học được AngularJS bạn cần phải có những kiến thức cơ bản javascript, object, string …. Việc bạn có hiểu biết chuyên sâu về javascript sẽ giúp cho bạn dễ dàng học AngularJS. Bản chất của AngularJS là hoạt động dạng Single Page, sử dụng API để lấy data, cho nên bạn cần biết các kĩ thuật DHTML, AJAX.

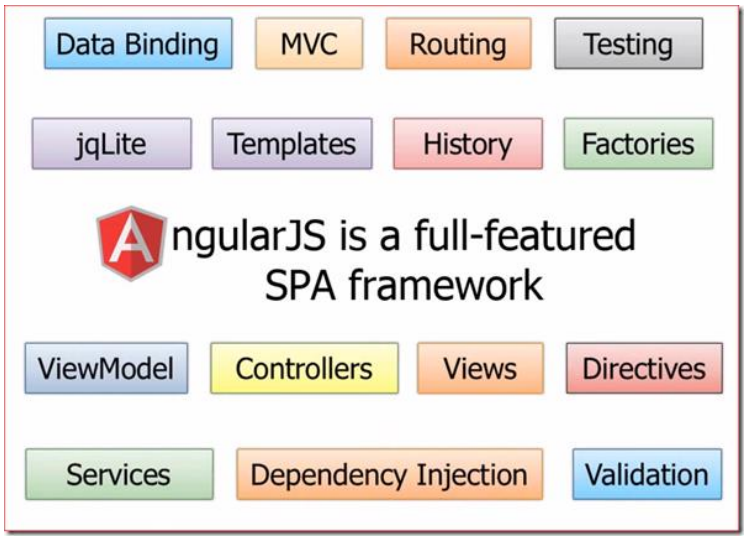
3.5.2. Đặc trưng của AngularJS

- Phát triển dự trên Javascript
- Tạo các ứng dụng client-side theo mô hình MVC.
- Khả năng tương thích cao, tự động xử lý mã javascript để phù hợp vúri mỗi trình duyệt.
- Mã nguồn mở, miễn phí hoàn toàn và được sủ dụng rộng rãi.

3.5.3. Các tính năng cơ bản

- *Scope* : là đối tượng có nhiệm vụ giao tiếp giữa controller và view của ứng dụng.
- *Controller* : xử lí dữ liệu cho đối tượng $scope, từ đây bên views sẽ sử dụng các dữ liệu trong scope để hiển thị ra tương ứng.

- *Data-binding* : tự động đồng bộ dữ liệu giữa model và view
- *Service* : là singleton object được khởi tạo 1 lần duy nhất cho mỗi ứng dụng, cung cấp các phương thức lưu trữ dữ liệu có sãn. ($http, $httpBackend, $sce, $controller, $document, $compile, $parse, $rootElement, $rootScope …..)
- *Filter* : Lọc các tập con từ tập item trong các mảng và trả về các mảng mới.
- *Directive* : dùng để tạo các thẻ HTML riêng phục vụ những mục đích riêng. AngularJS có những directive có sẵn như ngBind, ngModel…
- *Temple* : một thành phần của view, hiển thị thông tin từ controller
- *Routing* : chuyển đổi giữa các action trong controller, qua lại giữa các view.
- *MVC & MVVM* : mô hình thiết kế để phân chia các ứng dụng thành nhiều phần khác nhau (gọi là Model, View và Controller) mỗi phần có một nhiệm vụ nhất định. AngularJS không triển khai MVC theo cách truyền thống, mà gắn liền hơn với Model-View-ViewModel.
- *Deep link* : Liên kết sâu, cho phép bạn mã hóa trạng thái của ứng dụng trong các URL để nó có thể bookmark với công cụ tìm kiếm. Các ứng dụng có thể được phục hồi lại từ các địa chỉ URL với cùng một trạng thái.
- *Dependency Injection*: AngularJS có sẵn một hệ thống con dependency injection để giúp các lập trình viên tạo ra các ứng dụng dễ phát triển, dễ hiểu và kiểm tra.



Hình 3.5: Các tính năng cơ bản của Angular JS

3.5.4. Các components chính

- *ng-app* : định nghĩa và liên kết một ứng dụng AngularJS tới HTML.
- *ng-model* : gắn kết giá trị của dữ liệu ứng dụng AngularJS đến các điều khiển đầu vào HTML.
- *ng-bind* : gắn kết dữ liệu ứng dụng AngularJS đến các thẻ HTML.

3.5.5. Ưu điểm của angularJS

- Cung cấp khả năng tạo ra các Single Page Aplication dễ dàng.

- Cung cấp khả năng data binding tới HTML, khiến cho người dùng cảm giác linh hoạt, thân thiện.
- Dễ dàng Unit test.
- Dễ dàng tái sử dụng component.
- Giúp lập trình viên viết code ít hơn với nhiều chức năng hơn.
- Chạy được trên các loại trình duyệt, trên cả PC lẫn mobile.

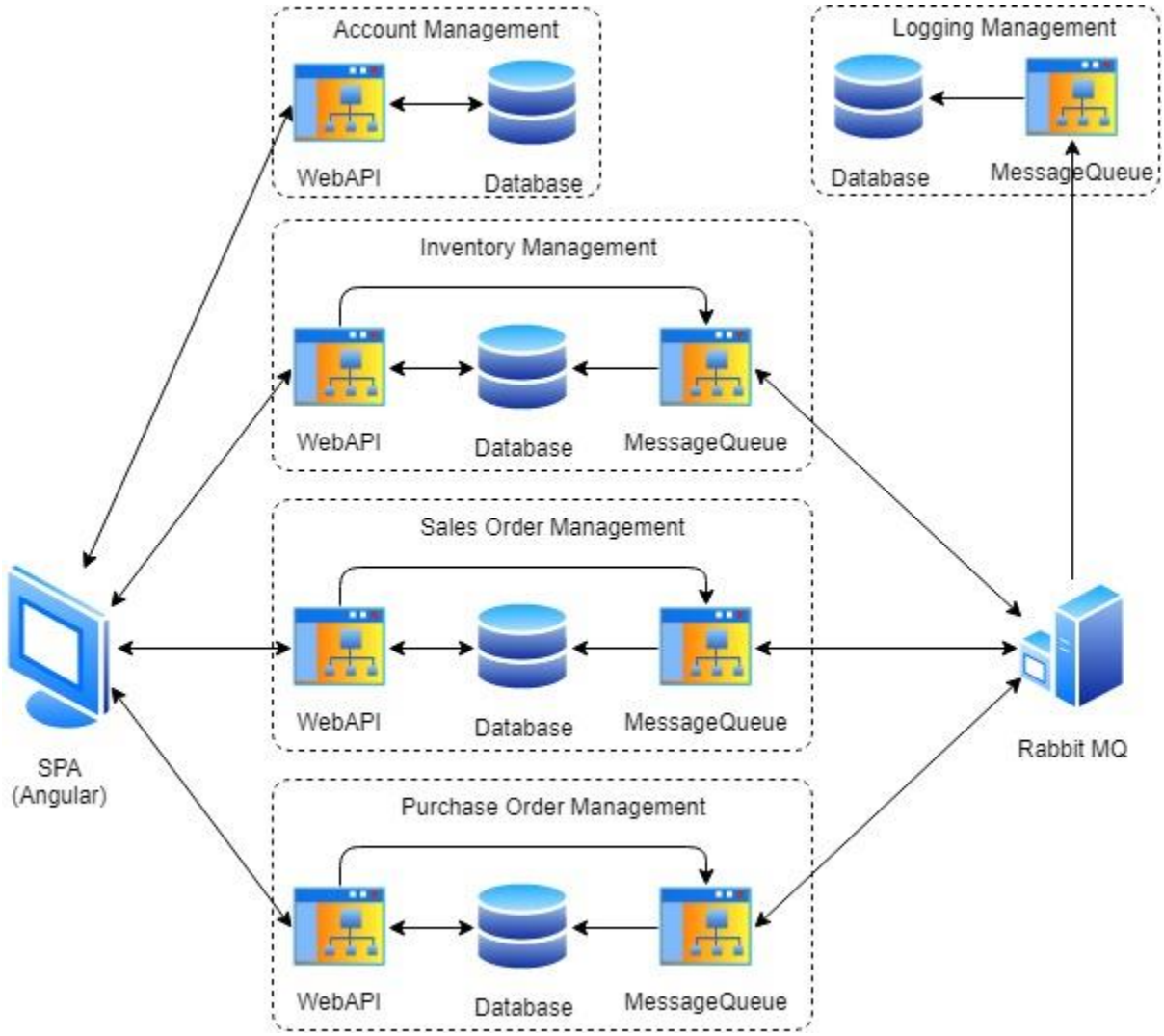
3.5.6. Nhược điểm

- Không an toàn: được phát triển từ javascript cho nên ứng dụng được viết bởi AngularJS không an toàn. Nên có sự bảo mật và xác thực phía server sẽ giúp ứng dụng trở nên an toàn hơn.
- Nếu người sử dụng ứng dụng của vô hiệu hóa JavaScript thì sẽ chỉ nhìn thấy trang cơ bản.

CHƯƠNG 4: PHÁT TRIỂN VÀ TRIỂN KHAI ỨNG DỤNG

4.1. Thiết kế kiến trúc

4.1.1. Lựa chọn kiến trúc phần mềm và thiết kế tổng quan



Hình 4.1: Thiết kế tổng quan của ứng dụng

Ứng dụng trong khuôn khổ đồ án được thiết kế và xây dựng dựa trên kiến trúc microservice, mô hình client – server và cơ sở dữ liệu được thiết kế tách biệt nhau.

Cụ thể ứng dụng sẽ bao gồm 2 thành phần chính là frontend và backend (mô hình client – server). Frontend là một ứng dụng single page application cung cấp giao diện cho người sử dụng, được xây dựng bằng AngularJS. Backend được xây dựng dựa trên kiến trúc microservice, được chia thành năm khối chức năng chính là quản lý tài khoản (AccountManagement), quản lý kho hàng (InventoryManagement), quản lý đơn nhập hàng (PurchaseOrderManagement), quản lý đơn bán hàng (SalesOrderManagement), quản lý ghi nhật ký (LoggingManagement).

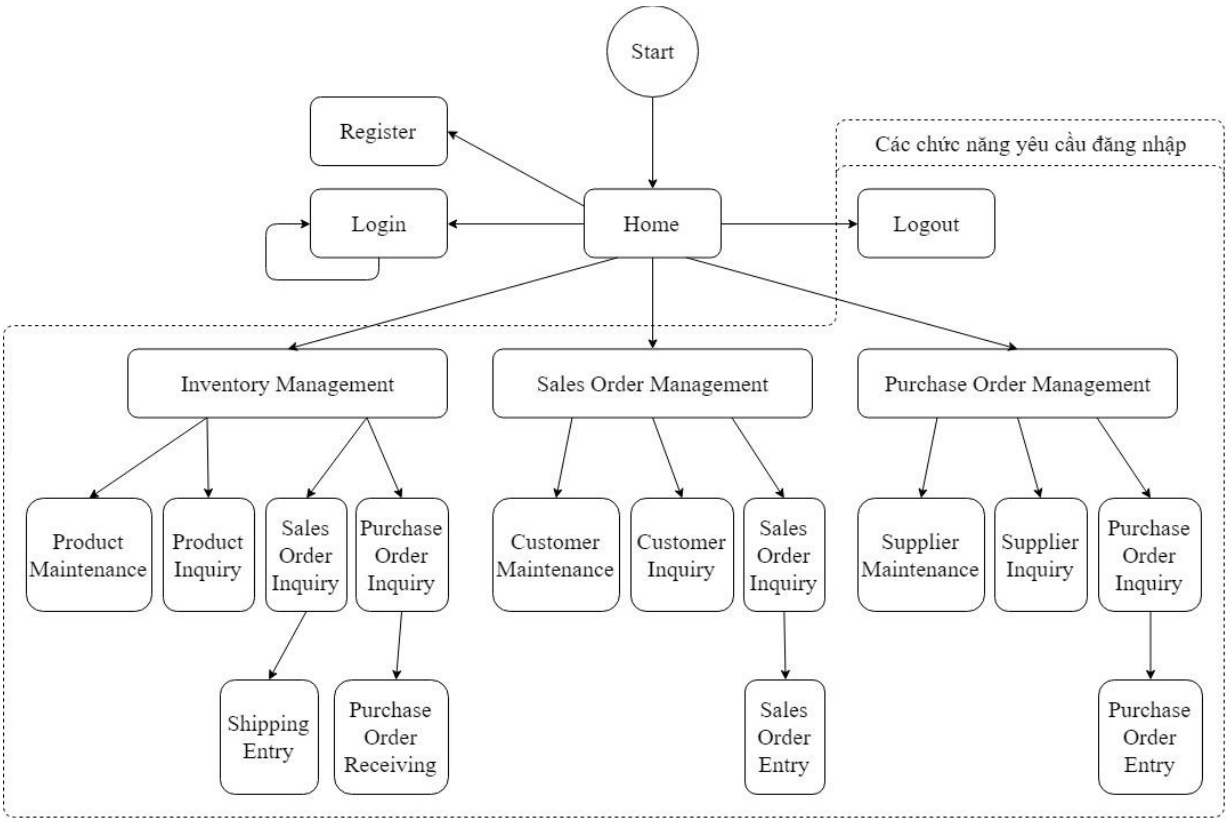
Tại mỗi khối chức năng có thể sẽ bao gồm 2 dịch vụ nhỏ là WebAPI để giao tiếp với ứng dụng frontend và MessageQueue để kết nối với RabbitMQ và giao tiếp với các dịch vụ nhỏ khác tại các khối chức năng khác. Mỗi khối chức năng sẽ sử dụng một database riêng và không kết nối, sử dụng bất kỳ database nào khác.

4.1.2. Thiết kế giao diện

a) Yêu cầu về thiết kế

- Giao diện phải được hiển thị và co dãn phù hợp trên các loại màn hình khác nhau, đặc biệt không bị nhảy giao diện gây khó chịu cho người dùng.
- Màu sắc sử dụng hài hòa, icon và hình ảnh mang tính gợi ý, tránh sử dụng mà không liên quan đến chức năng.
- Các giao diện sử dụng phải đồng dạng về thiết kế, các thông báo thiết kế dùng loại, cùng vị trí để người dùng có thể làm quen với giao diện một cách nhanh nhất
- Đảm bảo tính phản hồi chính xác, có hiệu ứng đơn giản khi click vào các button hoặc click vào thành phần trên giao diện để người dùng có thể kiểm soát được thao tác.

b) Thiết kế sơ đồ chuyển màn hình



Hình 4.2: Sơ đồ chuyển màn hình của ứng dụng

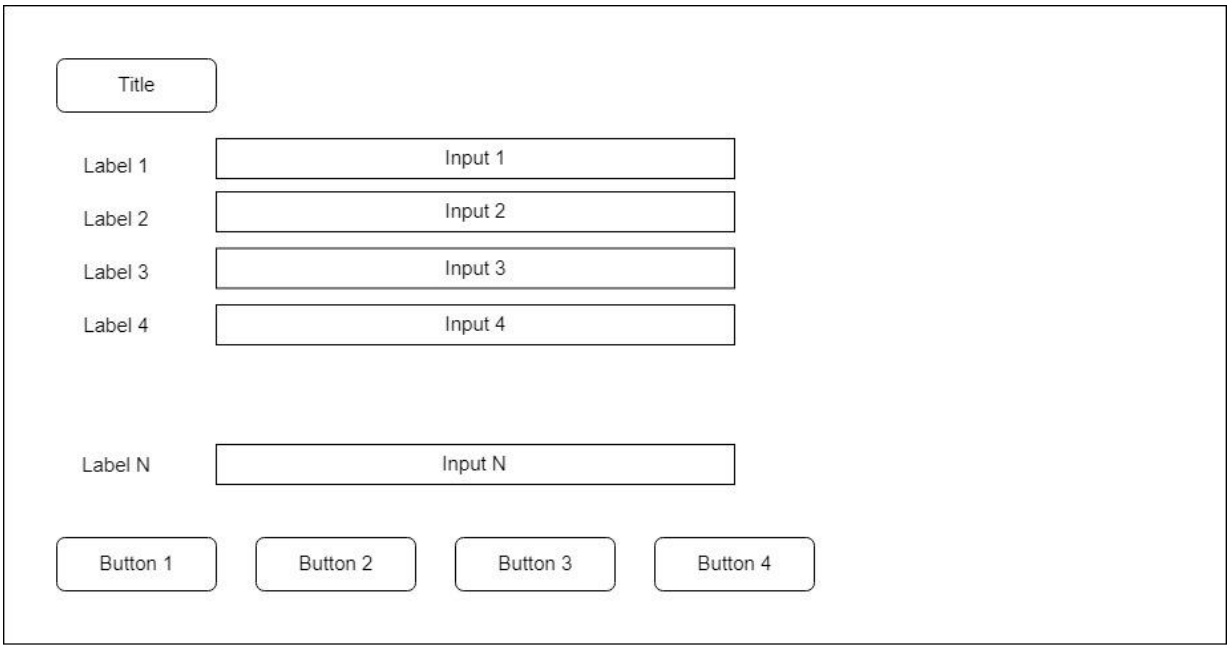
Hình 4.2 mô tả sơ đồ dịch chuyển màn hình của ứng dụng từ màn hình bắt đầu truy cập ứng dụng đến màn hình các chức năng cụ thể của ứng dụng.

c) Thiết kế màn hình

- Layout: chứa các thành phần dùng chung giữa các màn hình, ví dụ như các button Login/Logout, Logo, Menu

Hình 4.3: Thiết kế Layout

- Form: mẫu form sẽ có các thành phần chính như title (tên đối tượng), các label (tên thuộc tính của đối tượng), ô nhập dữ liệu, các button (Create, Update,…)

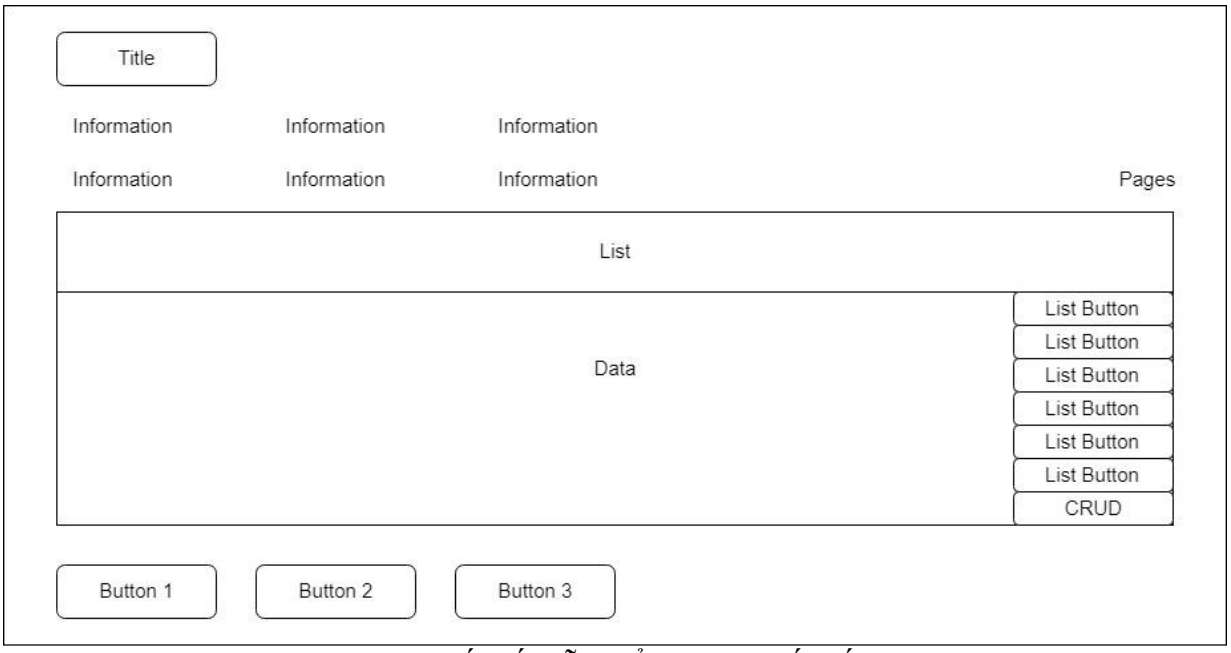


Hình 4.4: Thiết kế mẫu Form

- Danh sách dữ liệu: chứa danh sách dữ liệu của đối tượng (customer, supplier, product,..) được mô tả bởi title, có ô nhập dữ liệu để tìm kiếm theo các thuộc tính, có hiển thị số lượng item (trang này đang hiển thị bao nhiêu item, item có số thứ tự từ bao nhiêu đến bao nhiêu)

Hình 4.5: Thiết kế mẫu danh sách dữ liệu

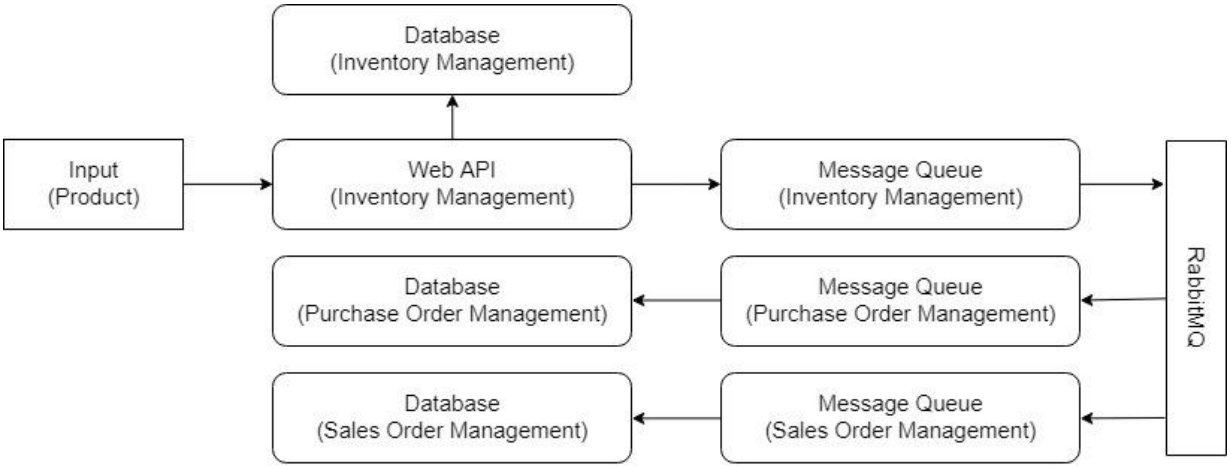
- Chi tiết đối tượng (Purchase Order Detail / Sales Order Detail): đối tượng chi tiết đơn hàng sẽ cần hiển thị các thông tin của đơn hàng và danh sách các chi tiết của đơn hàng, có thể thêm sửa xóa các chi tiết và cập nhật dữ liệu



Hình 4.6: Thiết kế mẫu hiển thị chi tiết đối tượng
(Purchase Order Detail/ Sales Order Detail)

4.1.3. Thiết kế luồng dữ liệu

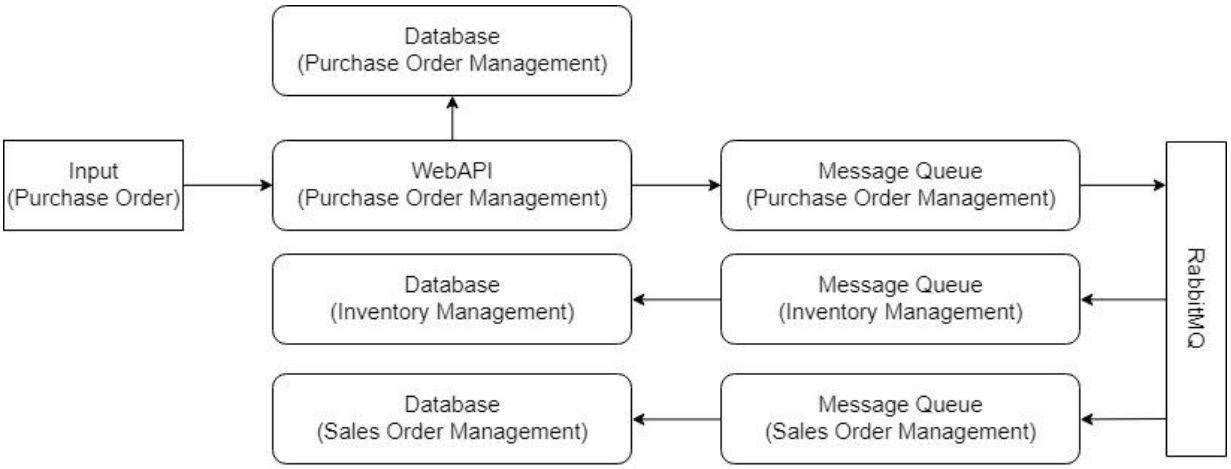
a) Luồng dữ liệu cho quy trình quản lý sản phẩm



Hình 4.7: Luồng dữ liệu của quy trình quản lý sản phẩm

Sau khi sản phẩm được tạo mới hoặc được cập nhật tại giao diện, thông tin sản phẩm sẽ được chuyển đến dịch vụ WebAPI, sản phẩm sẽ được cập nhật vào database, sau đó nó gửi tín hiệu đến dịch vụ MessageQueue để gửi thông tin cập nhật đến RabbitMQ. Các dịch vụ MessageQueue của quản lý đơn nhập hàng và quản lý đơn bán hàng sẽ chủ động lấy thông tin trên RabbitMQ để tự cập nhật vào database của dịch vụ. Đây chính là cơ chế đồng bộ hóa dữ liệu giữa các dịch vụ nhỏ trong ứng dụng quản lý bán hàng lớn.

b) Luồng dữ liệu cho quy trình quản lý đơn nhập hàng

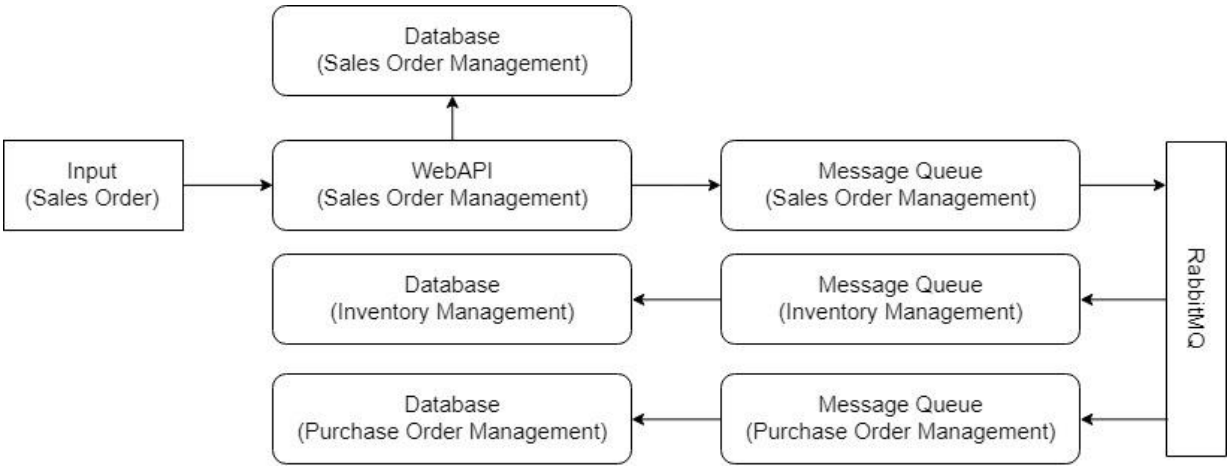


Hình 4.8: Luồng dữ liệu của quy trình quản lý đơn nhập hàng

Sau khi đơn nhập hàng được tạo mới hoặc được cập nhật tại giao diện, thông tin đơn sẽ được chuyển đến dịch vụ WebAPI, đơn hàng sẽ được cập nhật vào database, sau đó nó gửi tín hiệu đến dịch vụ MessageQueue để gửi thông tin cập nhật đến RabbitMQ.

Các dịch vụ MessageQueue của quản lý kho hàng và quản lý đơn bán hàng sẽ chủ động lấy thông tin trên RabbitMQ để tự cập nhật vào database của dịch vụ.

c) Luồng dữ liệu cho quy trình quản lý đơn bán hàng



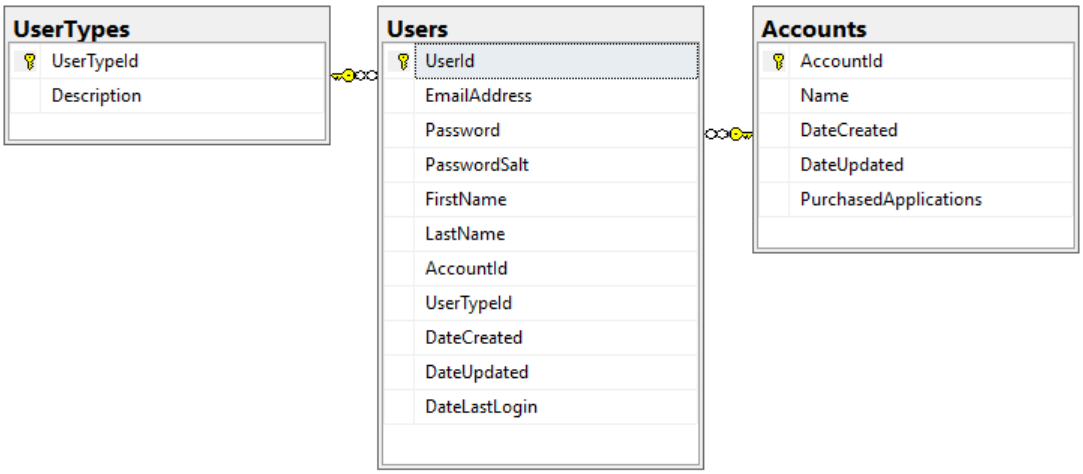
Hình 4.9: Luồng dữ liệu của quy trình quản lý đơn bán hàng

Sau khi đơn bán hàng được tạo mới hoặc được cập nhật tại giao diện, thông tin đơn sẽ được chuyển đến dịch vụ WebAPI, đơn hàng sẽ được cập nhật vào database, sau đó nó gửi tín hiệu đến dịch vụ MessageQueue để gửi thông tin cập nhật đến RabbitMQ. Các dịch vụ MessageQueue của quản lý kho hàng và quản lý đơn nhập hàng sẽ chủ động lấy thông tin trên RabbitMQ để tự cập nhật vào database của dịch vụ.

4.1.4. Thiết kế cơ sở dữ liệu

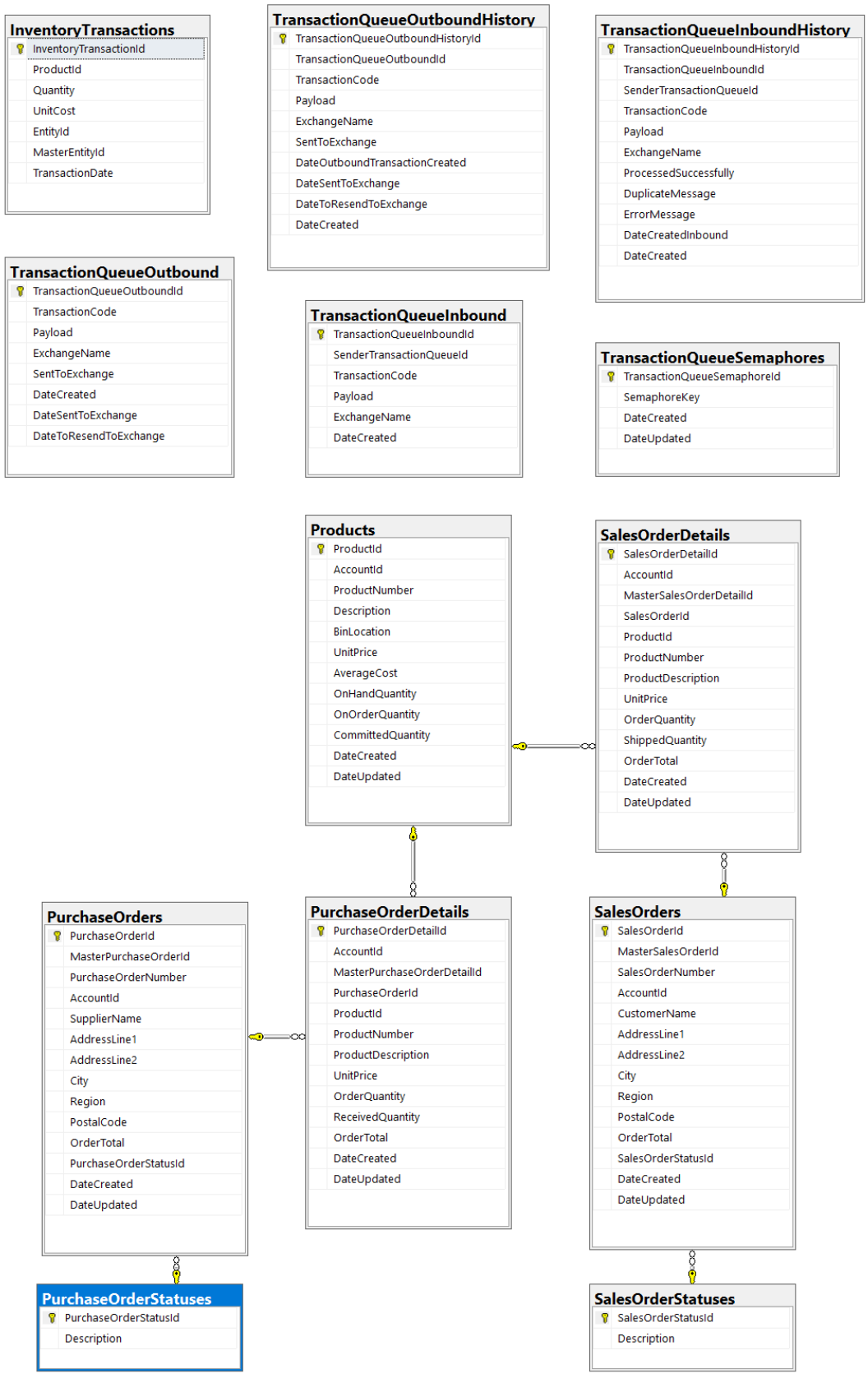
Lưu ý: Chi tiết kiểu dữ liệu và ý nghĩa của từng thuộc tính đã được chú thích tại thư mục Model trong source code nên sẽ không mô tả ở đây để giảm độ dài của báo cáo.

a) Thiết kế cơ sở dữ liệu dịch vụ quản lý tài khoản (AccountManagement)



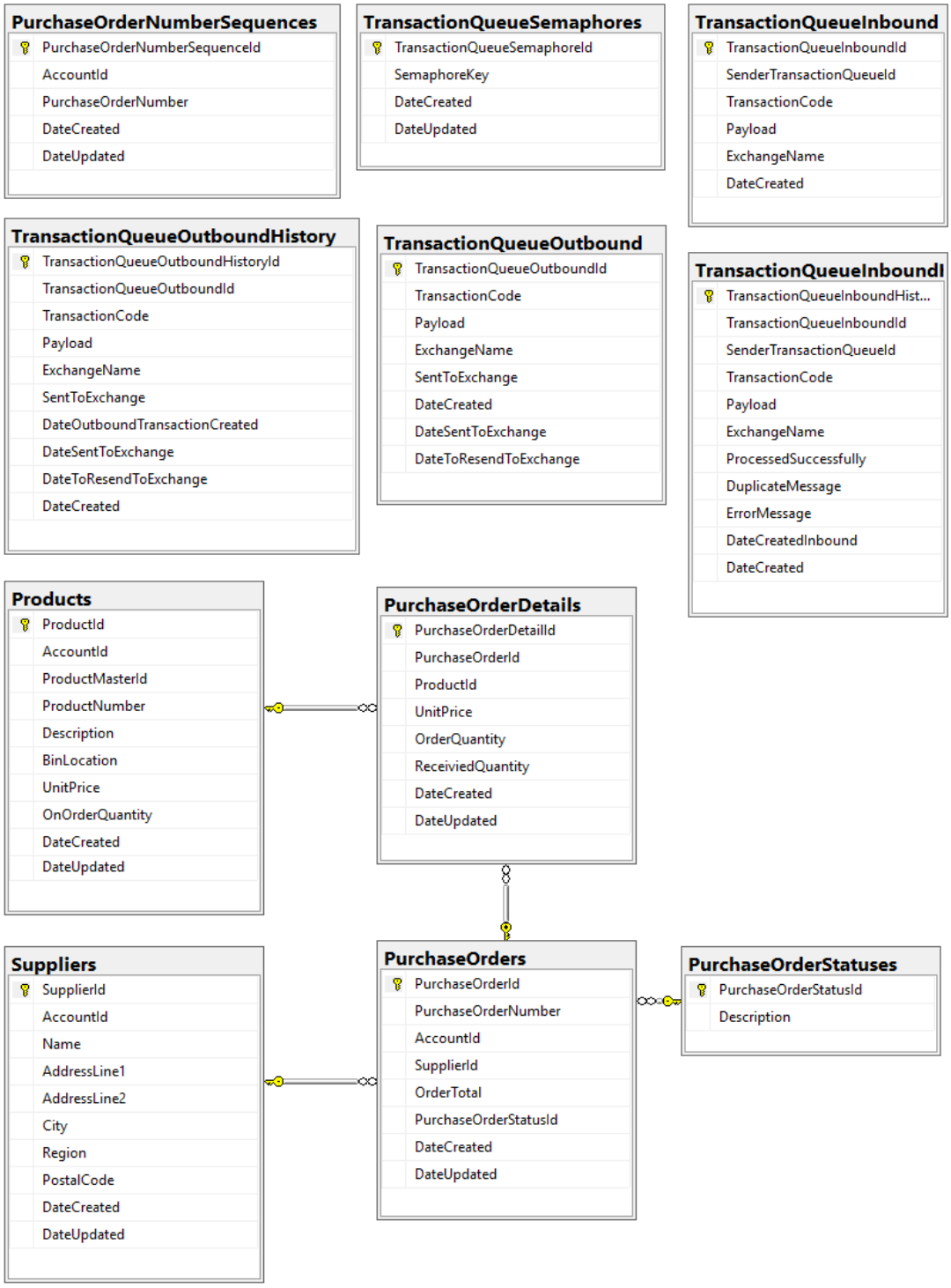
Hình 4.10: Diagram cơ sở dữ liệu dịch vụ quản lý tài khoản

b) Thiết kế cơ sở dữ liệu dịch vụ quản lý kho hàng (InventoryManagement)



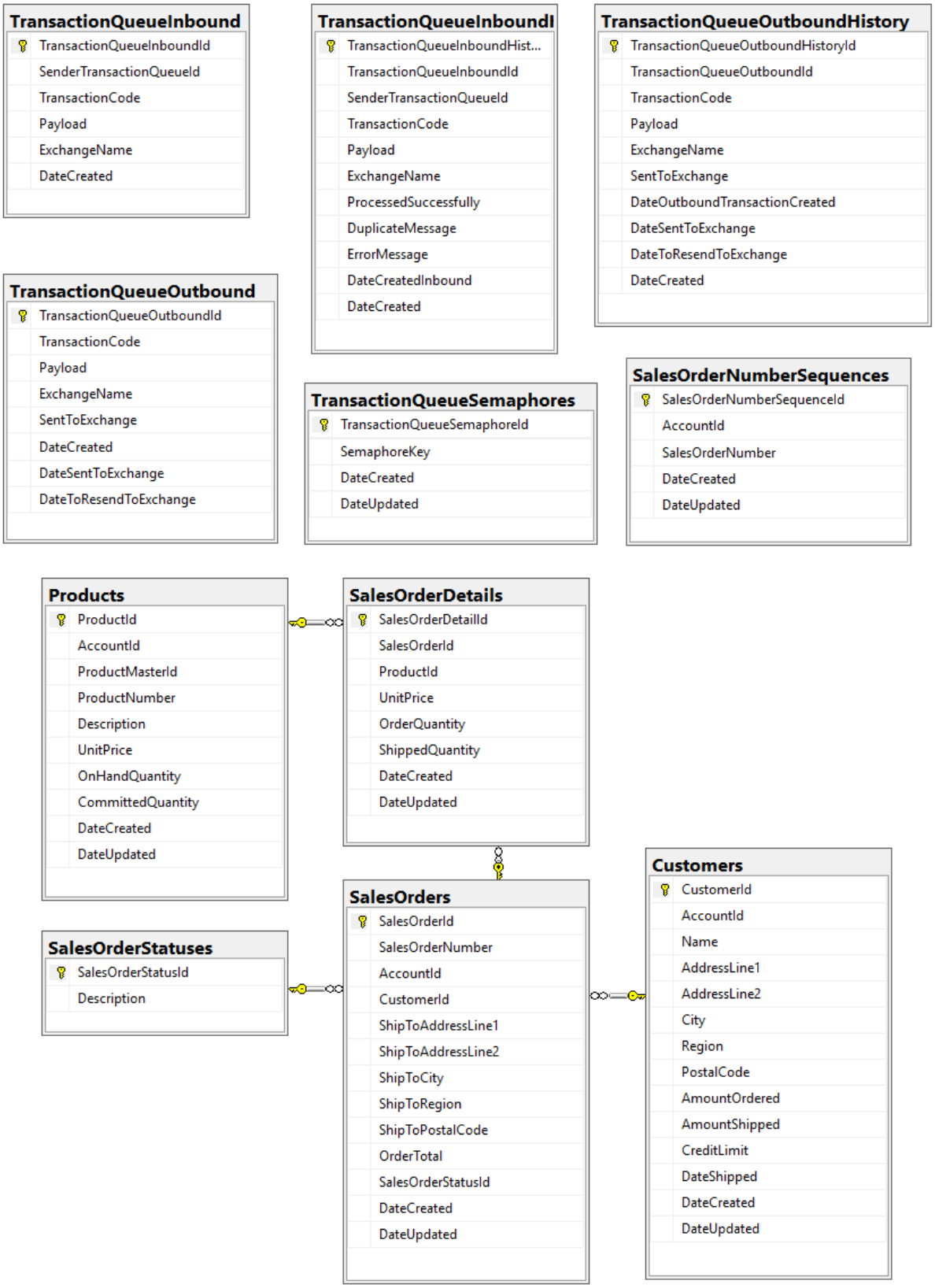
Hình 4.11: Diagram cơ sở dữ liệu dịch vụ quản lý kho hàng

c) Thiết kế cơ sở dữ liệu dịch vụ quản lý đơn nhập hàng (PurchaseOrderManagement)



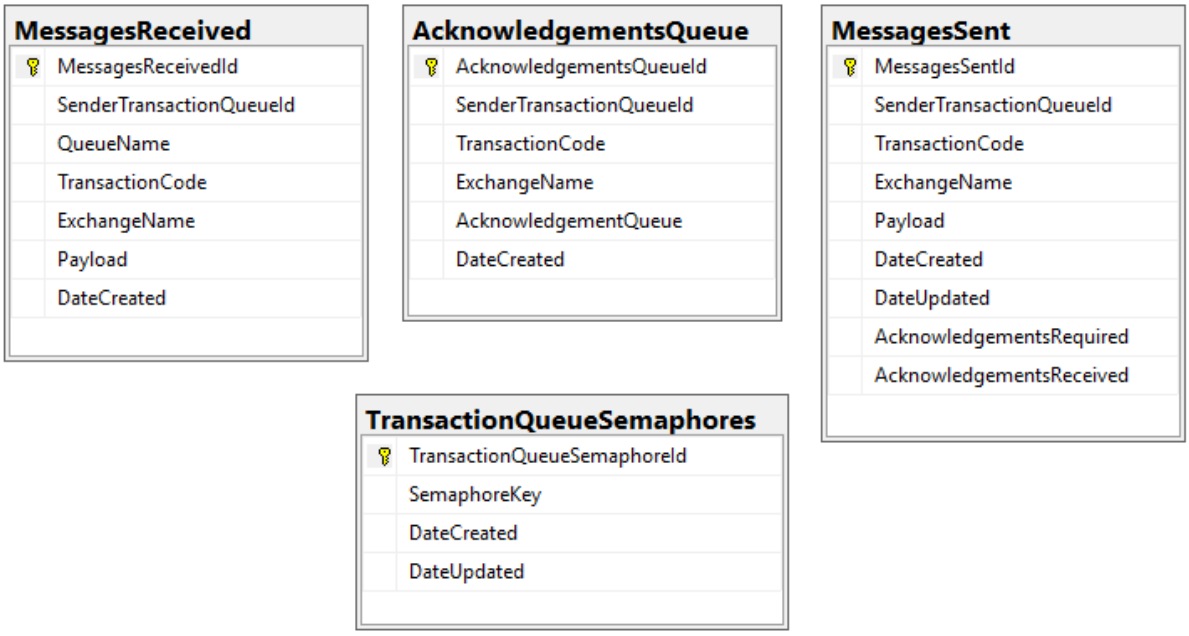
Hình 4.12: Diagram cơ sở dữ liệu dịch vụ quản lý đơn nhập hàng

d) Thiết kế cơ sở dữ liệu dịch vụ quản lý đơn bán hàng (SalesOrderManagement)



Hình 4.13: Diagram cơ sở dữ liệu dịch vụ quản lý đơn bán hàng

e) Thiết kế cơ sở dữ liệu dịch vụ ghi nhật ký hệ thống (LoggingManagement)



Hình 4.14: Diagram cơ sở dữ liệu dịch vụ ghi nhật ký hệ thống

4.2. Thiết kế chi tiết ứng dụng

4.2.1. Thiết kế dịch vụ quản lý tài khoản (Account Management)

Kiến trúc của dịch vụ quản lý tài khoản sẽ được chia thành ba tầng:

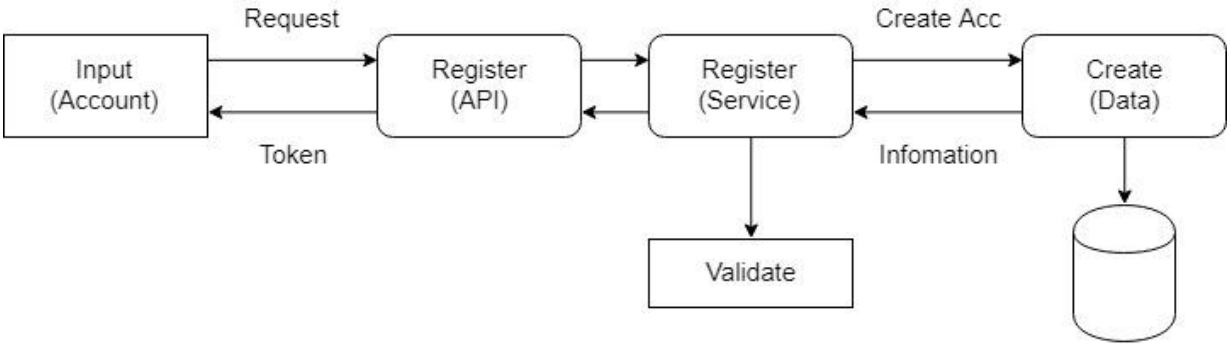
- Tầng WebAPI: nhiệm vụ giao tiếp với frontend để đáp ứng các hành động mà người dùng thực hiện trên giao diện thông qua controller, các chức năng mà dịch vụ cần có là đăng ký tài khoản và đăng nhập.
- Tầng Service: là tầng trung gian giữa tầng WebAPI và tầng Data, nhiệm vụ là kiểm tra dữ liệu đầu vào, vận chuyển dữ liệu từ tầng WebAPI đến tầng Data để ghi dữ liệu vào database và vận chuyển dữ liệu từ tầng Data đến tầng WebAPI để trả dữ liệu về cho người dùng.
- Tầng Data: là tầng làm nhiệm vụ thao tác với cơ sở dữ liệu, nó nhận yêu cầu, thông tin từ tầng Service và thực hiện các thao tác tương ứng với cơ sở dữ liệu cụ thể là đọc, thêm, sửa, xóa dữ liệu và trả về kết quả cho tầng Service.

Thiết kế luồng hoạt động cho các chức năng của dịch vụ:

➢ *Luồng hoạt động của chức năng đăng ký*:

B1: Người dùng nhập thông tin tài khoản tại giao diện
B2: Dữ liệu được chuyển đến API Register, và chuyển đến Service để kiểm tra dữ liệu sau đó chuyển đến Data để cập nhật dữ liệu vào database.

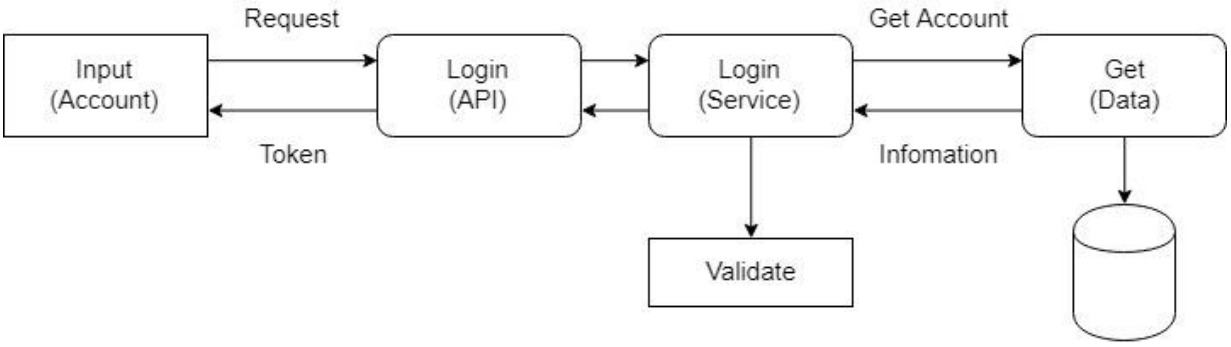
B3: Sau khi dữ liệu cập nhật thành công, thông tin tài khoản mới tạo được trả về cho API Register, tại đây nó thực hiện tạo token và gửi lại cho người dùng.



Hình 4.15: Luồng hoạt động chức năng đăng ký

- *Luồng hoạt động của chức năng đăng nhập:*

B1: Người dùng nhập thông tin tài khoản tại giao diện
B2: Dữ liệu được chuyển đến API Login, và chuyển đến Service để kiểm tra dữ liệu sau đó chuyển đến Data để lấy dữ liệu từ database và thực hiện kiểm tra tài khoản và mật khẩu tại Service.
B3: Sau khi xác thực thông tin, thông tin tài khoản được lấy từ database được trả về cho API Register, tại đây nó thực hiện tạo token và gửi lại cho người dùng.



Hình 4.16: Luồng hoạt động chức năng đăng nhập

4.2.2. Thiết kế dịch vụ quản lý kho hàng (Inventory Management)

Kiến trúc của dịch vụ quản lý kho hàng sẽ gồm 2 dịch vụ nhỏ là WebAPI và MessageQueue.

- *Dịch vụ WebAPI* gồm 3 tầng:
  - Tầng WebAPI: làm nhiệm vụ giao tiếp với frontend để đáp ứng các hành động mà người dùng thực hiện trên giao diện thông qua controller, các chức năng mà dịch vụ cần có là quản lý sản phẩm, cập nhật chi tiết đơn bán hàng, đơn nhập hàng.
  - Tầng Service: là tầng trung gian giữa tầng WebAPI và tầng Data, nhiệm vụ là kiểm tra dữ liệu đầu vào, vận chuyển dữ liệu từ tầng WebAPI đến tầng Data để

ghi dữ liệu vào database và vận chuyển dữ liệu từ tầng Data đến tầng WebAPI để trả dữ liệu về cho người dùng.

- Tầng Data: làm nhiệm vụ thao tác với cơ sở dữ liệu, nó nhận yêu cầu, thông tin từ tầng Service và thực hiện các thao tác tương ứng với cơ sở dữ liệu cụ thể là đọc, thêm, sửa, xóa dữ liệu và trả về kết quả cho tầng Service.

➢ *Dịch vụ Message Queue* gồm 2 tầng:

- Tầng Message Queue: làm nhiệm vụ giao tiếp với RabbitMQ, nhận tín hiệu từ WebAPI rồi gửi message lên RabbitMQ, chủ động lấy tin nhắn hiện có trên hàng đợi của RabbitMQ về và cập nhật vào database thông qua tầng Data.
- Tầng Data: làm nhiệm vụ thao tác với cơ sở dữ liệu, nó nhận yêu cầu, thông tin từ tầng MessageQueue và thực hiện các thao tác tương ứng với cơ sở dữ liệu cụ thể là đọc, thêm, sửa, xóa dữ liệu và trả về kết quả.

Thiết kế luồng hoạt động cho các chức năng của dịch vụ:

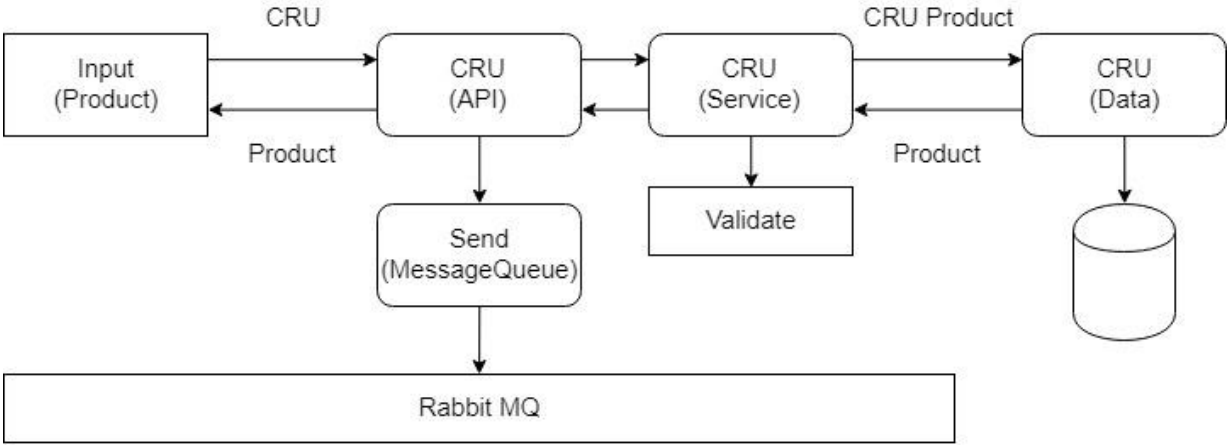
➢ *Luồng hoạt động của chức năng quản lý sản phẩm*:

B1: Người dùng nhập vào thông tin sản phẩm tại giao diện

B2: Dữ liệu và yêu cầu được gửi đến API sau đó gửi đến Service và Data để thực hiện các hành động tương ứng.

B3: Sau khi dữ liệu được cập nhật, tại API sẽ gửi tín hiệu đến MessageQueue để thực hiện gửi message lên RabbitMQ cho các dịch vụ khác lấy về để tự cập nhật tại cơ sở dữ liệu của dịch vụ đó

B4: Cuối cùng thông tin phản hồi sẽ được gửi trả về cho client



Hình 4.17: Luồng hoạt động chức năng quản lý sản phẩm

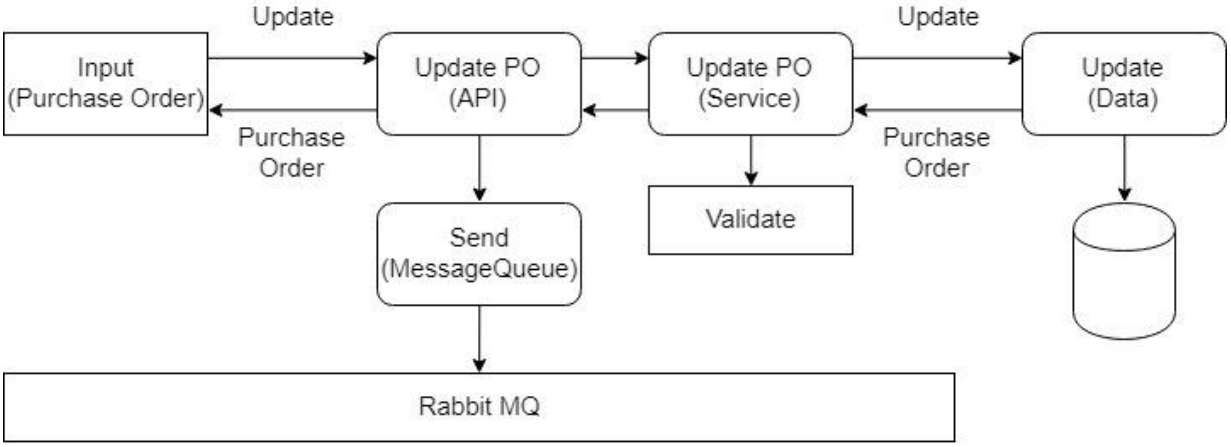
➢ *Luồng hoạt động của chức năng cập nhật chi tiết đơn nhập hàng:*

B1: Tại giao diện, người dùng sẽ được cập nhật số lượng sản phẩm được vận chuyển đến vào đơn nhập hàng.

B2: Dữ liệu được chuyển đến API, sau đó đến Service và Data để cập nhật vào cơ sở dữ liệu.

B3: Sau khi dữ liệu được cập nhật, tại API sẽ gửi tín hiệu đến MessageQueue để thực hiện gửi message lên RabbitMQ cho các dịch vụ khác lấy về để tự cập nhật tại cơ sở dữ liệu của dịch vụ đó

B4: Cuối cùng thông tin phản hồi sẽ được gửi trả về cho client



Hình 4.18: Luồng hoạt động chức năng cập nhật chi tiết đơn nhập hàng

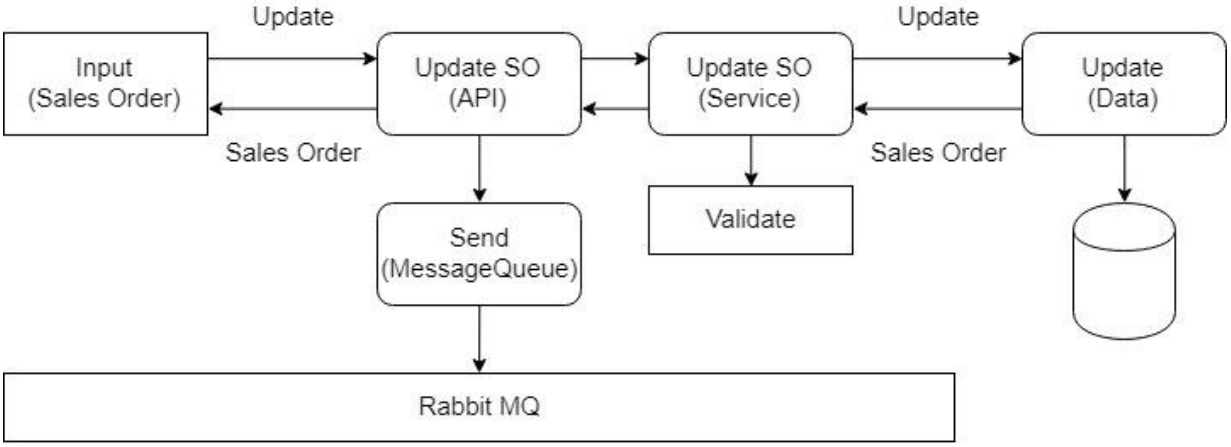
- *Luồng hoạt động của chức năng cập nhật chi tiết đơn bán hàng*:

B1: Tại giao diện, người dùng sẽ được cập nhật số lượng sản phẩm hiện có, có thể vận chuyển vào đơn bán hàng.

B2: Dữ liệu được chuyển đến API, sau đó đến Service và Data để cập nhật vào cơ sở dữ liệu.

B3: Sau khi dữ liệu được cập nhật, tại API sẽ gửi tín hiệu đến MessageQueue để thực hiện gửi message lên RabbitMQ cho các dịch vụ khác lấy về để tự cập nhật tại cơ sở dữ liệu của dịch vụ đó

B4: Cuối cùng thông tin phản hồi sẽ được gửi trả về cho client



Hình 4.19: Luồng hoạt động chức năng cập nhật chi tiết đơn bán hàng

4.2.3. Thiết kế dịch vụ quản lý đơn nhập hàng (Purchase Order Management)

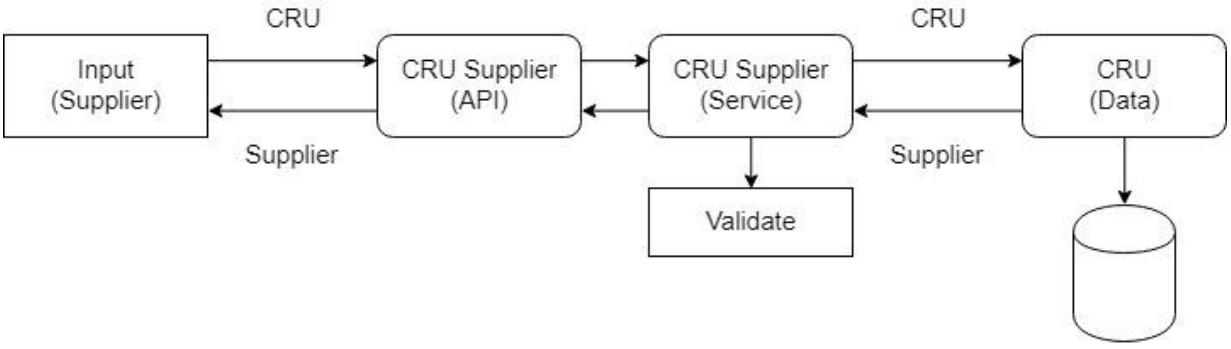
Kiến trúc của dịch vụ quản lý đơn nhập hàng sẽ gồm 2 dịch vụ nhỏ là WebAPI và MessageQueue.

- *Dịch vụ WebAPI* gồm 3 tầng:
  - Tầng WebAPI: làm nhiệm vụ giao tiếp với frontend để đáp ứng các hành động mà người dùng thực hiện trên giao diện thông qua controller, các chức năng mà dịch vụ cần có là quản lý nhà cung cấp, quản lý đơn nhập hàng, quản lý chi tiết đơn nhập hàng.
  - Tầng Service: là tầng trung gian giữa tầng WebAPI và tầng Data, nhiệm vụ là kiểm tra dữ liệu đầu vào, vận chuyển dữ liệu từ tầng WebAPI đến tầng Data để ghi dữ liệu vào database và vận chuyển dữ liệu từ tầng Data đến tầng WebAPI để trả dữ liệu về cho người dùng.
  - Tầng Data: làm nhiệm vụ thao tác với cơ sở dữ liệu, nó nhận yêu cầu, thông tin từ tầng Service và thực hiện các thao tác tương ứng với cơ sở dữ liệu cụ thể là đọc, thêm, sửa, xóa dữ liệu và trả về kết quả cho tầng Service.
- *Dịch vụ Message Queue* gồm 2 tầng:
  - Tầng Message Queue: làm nhiệm vụ giao tiếp với RabbitMQ, nhận tín hiệu từ WebAPI rồi gửi message lên RabbitMQ, chủ động lấy tin nhắn hiện có trên hàng đợi của RabbitMQ về và cập nhật vào database thông qua tầng Data.
  - Tầng Data: làm nhiệm vụ thao tác với cơ sở dữ liệu, nó nhận yêu cầu, thông tin từ tầng MessageQueue và thực hiện các thao tác tương ứng với cơ sở dữ liệu cụ thể là đọc, thêm, sửa, xóa dữ liệu và trả về kết quả.

Thiết kế luồng hoạt động cho các chức năng của dịch vụ:

- *Luồng hoạt động của chức năng quản lý nhà cung cấp:*

B1: Người dùng nhập thông tin nhà cung cấp tại giao diện
B2: Dữ liệu được chuyển đến API CRU Supplier, và chuyển đến Service để kiểm tra dữ liệu sau đó chuyển đến Data để cập nhật dữ liệu vào cơ sở dữ liệu.
B3: Sau khi xác thực thông tin, thông tin tài khoản được lấy từ database được trả về cho API Register, và API trả về thông tin supplier đã cập nhật cho client.



Hình 4.20: Luồng hoạt động chức năng quản lý nhà cung cấp

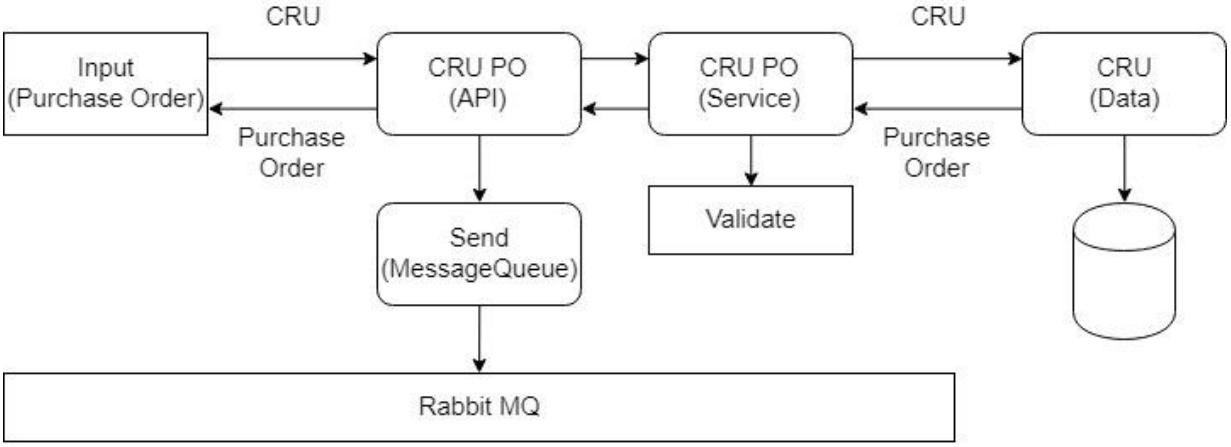
➢ *Luồng hoạt động của chức năng quản lý đơn nhập hàng:*

B1: Người dùng nhập dữ liệu đơn hàng khi thực hiện tạo mới hoặc chọn một đơn hàng trong danh sách để cập nhật.

B2: Dữ liệu được chuyển đến API, sau đó đến Service và Data để cập nhật vào cơ sở dữ liệu.

B3: Sau khi dữ liệu được cập nhật, tại API sẽ gửi tín hiệu đến MessageQueue để thực hiện gửi message lên RabbitMQ cho các dịch vụ khác lấy về để tự cập nhật tại cơ sở dữ liệu của dịch vụ đó

B4: Cuối cùng thông tin phản hồi sẽ được gửi trả về cho client



Hình 4.21: Luồng hoạt động chức năng quản lý đơn nhập hàng

4.2.4. Thiết kế dịch vụ quản lý đơn bán hàng (Sales Order Management)

Kiến trúc của dịch vụ quản lý đơn bán hàng sẽ gồm 2 dịch vụ nhỏ là WebAPI và MessageQueue.

➢ *Dịch vụ WebAPI* gồm 3 tầng:

- Tầng WebAPI: làm nhiệm vụ giao tiếp với frontend để đáp ứng các hành động mà người dùng thực hiện trên giao diện thông qua controller, các chức năng mà dịch vụ cần có là quản lý khách hàng, quản lý đơn bán hàng, quản lý chi tiết đơn bán hàng.
- Tầng Service: là tầng trung gian giữa tầng WebAPI và tầng Data, nhiệm vụ là kiểm tra dữ liệu đầu vào, vận chuyển dữ liệu từ tầng WebAPI đến tầng Data để ghi dữ liệu vào database và vận chuyển dữ liệu từ tầng Data đến tầng WebAPI để trả dữ liệu về cho người dùng.
- Tầng Data: làm nhiệm vụ thao tác với cơ sở dữ liệu, nó nhận yêu cầu, thông tin từ tầng Service và thực hiện các thao tác tương ứng với cơ sở dữ liệu cụ thể là đọc, thêm, sửa, xóa dữ liệu và trả về kết quả cho tầng Service.

➢ *Dịch vụ Message Queue* gồm 2 tầng:

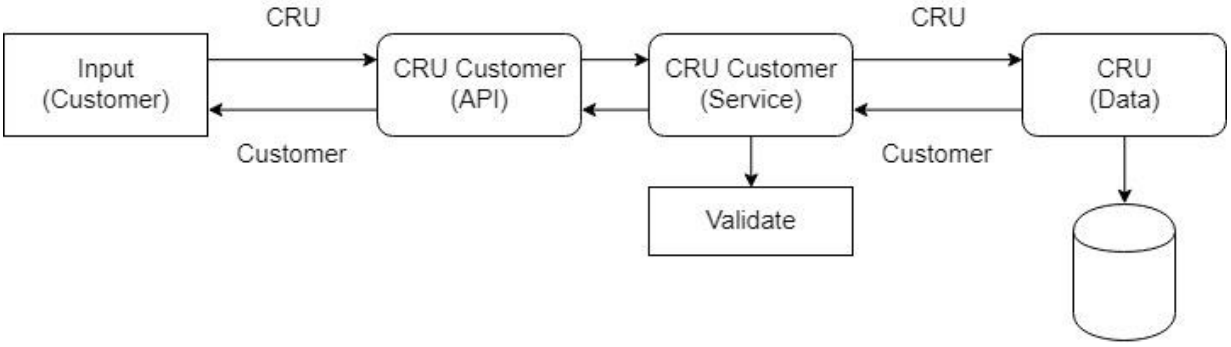
- Tầng Message Queue: làm nhiệm vụ giao tiếp với RabbitMQ, nhận tín hiệu từ WebAPI rồi gửi message lên RabbitMQ, chủ động lấy tin nhắn hiện có trên hàng đợi của RabbitMQ về và cập nhật vào database thông qua tầng Data.

- Tầng Data: làm nhiệm vụ thao tác với cơ sở dữ liệu, nó nhận yêu cầu, thông tin từ tầng MessageQueue và thực hiện các thao tác tương ứng với cơ sở dữ liệu cụ thể là đọc, thêm, sửa, xóa dữ liệu và trả về kết quả.

Thiết kế luồng hoạt động cho các chức năng của dịch vụ:

➢ *Luồng hoạt động chức năng quản lý khách hàng*:

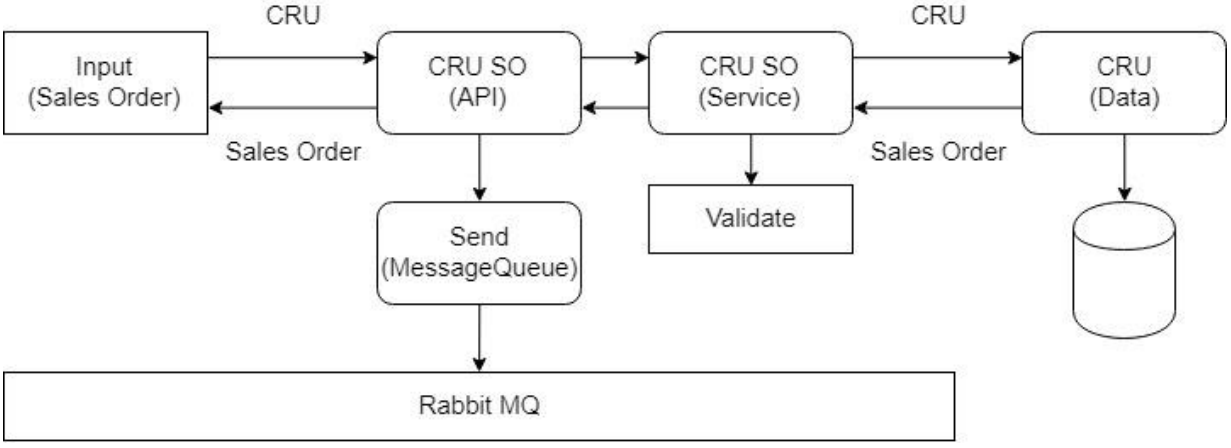
B1: Người dùng nhập thông tin khách hàng tại giao diện
B2: Dữ liệu được chuyển đến API CRU Customer, và chuyển đến Service để kiểm tra dữ liệu sau đó chuyển đến Data để cập nhật dữ liệu vào cơ sở dữ liệu.
B3: Sau khi xác thực thông tin, thông tin tài khoản được lấy từ database được trả về cho API Register, và API trả về thông tin customer đã cập nhật cho client



Hình 4.22: Luồng hoạt động chức năng quản lý khách hàng

➢ *Luồng hoạt động của chức năng quản lý đơn bán hàng*:

B1: Người dùng nhập dữ liệu đơn hàng khi thực hiện tạo mới hoặc chọn một đơn hàng trong danh sách để cập nhật.
B2: Dữ liệu chuyển đến API, sau đó đến Service và Data để cập nhật vào cơ sở dữ liệu.
B3: Sau khi dữ liệu được cập nhật, tại API sẽ gửi tín hiệu đến MessageQueue để thực hiện gửi message lên RabbitMQ cho các dịch vụ khác lấy về để tự cập nhật tại cơ sở dữ liệu của dịch vụ đó
B4: Cuối cùng thông tin phản hồi sẽ được gửi trả về cho client



Hình 4.23: Luồng hoạt động chức năng quản lý đơn bán hàng

4.3. Xây dựng ứng dụng

4.3.1. Thư viện và công cụ sử dụng

Mục đích	Công cụ	Nguồn
IDE lập trình	Visual Studio 2017 Community	https://visualstudio.microsoft.com/vs/
Nền tảng phát triển	.NetCore 2.2	https://dotnet.microsoft.com/download/dotnet-core/2.2
Cơ sở dữ liệu	SQL Server Express 2017	https://www.microsoft.com/en-us/download/details.aspx?id=55994
Framework frontend	Angular 6	https://angular.io/
Thư viện hỗ trợ giao tiếp giữa các dịch vụ trong một khối chức năng	Signal R	https://www.nuget.org/packages/Microsoft.AspNet.SignalR/
Thư viện hỗ trợ thao tác, làm việc với database	EntityFramework Core	https://www.nuget.org/packages/Microsoft.EntityFrameworkCore/
Nền tảng chạy Angular	Node JS 10.15.3	https://nodejs.org/en/
Test API	Postman 7.1.1	https://www.getpostman.com/
Message Broker (giao tiếp giữa các microservice)	RabbitMQ	https://www.rabbitmq.com/
Hỗ trợ khởi chạy ứng dụng	Command Prompt	Tích hợp trên Window
Kiểm thử ứng dụng	Google Chrome	https://www.google.com/intl/vi/chrome/

Bảng 4.1: Danh sách các thư viện và công cụ sử dụng

4.3.2. Một số chức năng và đặc điểm đáng chú ý

a) Cấu hình xác thực đăng nhập và cấp quyền

Mỗi microservice cho ứng dụng sẽ được bảo mật và bảo vệ bằng JSON web token. JSON web token (JWT) là một tiêu chuẩn mở (RFC 7419) xác định một cách nhỏ gọn và khép kín để truyền thông tin an toàn giữa các bên dưới dạng đối tượng JSON. Thông tin này có thể được xác minh và tin cậy vì nó được ký điện tử. JWT có thể được ký bằng cách sử dụng một khóa bí mật (với thuật toán HMAC) hoặc với cặp khóa công khai/riêng tư bằng RSA hoặc ECDSA.

Để đăng nhập vào ứng dụng, phương thức login trong controller của Account Management Web API sẽ được thực thi với thông tin đăng nhập của người dùng (địa chỉ

email và mật khẩu) được truyền từ request của client và phương thức login sẽ tiến hành gọi service để xác thực người dùng dựa trên cơ sở dữ liệu Account Management.

Sau khi đăng nhập thành công, JSON web token sẽ được tạo và trả lại ứng dụng client, nó sẽ được lưu trong bộ nhớ cục bộ của client. JSON web token sẽ được gửi kèm trong header của mỗi request HTTP của máy client được thực hiện cho bất kỳ Web API end point nào của ứng dụng.

Microsoft .NET Core hỗ trợ tạo và xác thực JSON web token. Phương thức CreatToken bên dưới lấy thông tin tài khoản của người dùng và tạo quyền truy cập, và sẽ được lưu trữ trong token. Thông tin này sẽ được sử dụng để xác thực người dùng trên mỗi request HTTP. Khi thông tin quyền truy cập được tạo, token có thể được ký và trả lại dưới dạng chuỗi được mã hóa trong response Web API.

```
/// <summary>
/// Create Token
/// </summary>
/// <param name="userId"></param>
/// <param name="firstName"></param>
/// <param name="lastName"></param>
/// <param name="emailAddress"></param>
/// <param name="companyName"></param>
/// <returns></returns>
public static string CreateToken(int userId,
                                 string firstName,
                                 string lastName,
                                 string emailAddress,
                                 int accountId,
                                 string companyName)
{
    var sharedKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(
                                    "CodeProject.Shared.Common.TokenManagement"));
    List<Claim> claims = new List<Claim>
    {
        new Claim(ClaimTypes.Email, emailAddress),
        new Claim(ClaimTypes.NameIdentifier, lastName),
        new Claim(ClaimTypes.GivenName, firstName),
        new Claim(ClaimTypes.Name, companyName),
        new Claim(ClaimTypes.PrimarySid, userId.ToString()),
        new Claim(ClaimTypes.PrimaryGroupSid, accountId.ToString())
    };

    var signinCredentials = new SigningCredentials(sharedKey,
                                    SecurityAlgorithms.HmacSha512Signature);
    var tokenDescription = new SecurityTokenDescriptor
    {
        Subject = new ClaimsIdentity(claims),
        NotBefore = DateTime.Now,
        Expires = DateTime.Now.AddMinutes(60),
        SigningCredentials = signinCredentials
    };
    var tokenHandler = new JwtSecurityTokenHandler();
    var token = tokenHandler.CreateToken(tokenDescription);
    string tokenString = tokenHandler.WriteToken(token);
    return tokenString;
}
```

Các ứng dụng ASP.NET Core sử dụng class StartUp để cấu hình các dịch vụ ứng dụng và đó là đường ống xử lý request HTTP. Kiến trúc ASP.NET Core có một hệ thống phần mềm trung gian (Middleware), là những đoạn code xử lý các request và response. Các middleware được liên kết với nhau để tạo thành một đường ống. Các request đến được chuyển qua đường ống, trong đó mỗi middleware có cơ hội thực hiện điều gì đó với request trước khi chuyển nó đến middleware tiếp theo. Response trả về được chuyển qua đường ống theo thứ tự ngược lại.

```
public class Startup
{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }
    public IConfiguration Configuration { get; }
    /// <summary>
    /// This method gets called by the runtime. Use this method to add services to the
container.
    /// </summary>
    /// <param name="services"></param>
    public void ConfigureServices(IServiceCollection services)
    {
        CorsPolicyBuilder corsBuilder = new CorsPolicyBuilder();

        corsBuilder.AllowAnyHeader();
        corsBuilder.AllowAnyMethod();
        corsBuilder.AllowAnyOrigin();
        corsBuilder.AllowCredentials();

        services.AddCors(options =>
        {
            options.AddPolicy("SiteCorsPolicy", corsBuilder.Build());
        });

        ConnectionStrings connectionStrings = new ConnectionStrings();
        Configuration.GetSection("ConnectionStrings").Bind(connectionStrings);

        services.AddDbContext<AccountManagementDatabase>(
                              options => options.UseSqlServer(
                    Configuration.GetConnectionString("PrimaryDatabaseConnectionString")));
        //
        //    Built-In Dependency Injection
        //
        services.AddTransient<IAccountManagementDataService,
AccountManagementDataService>();

        services.AddTransient<IAccountManagementBusinessService>(provider =>
        new AccountManagementBusinessService(provider
            .GetRequiredService<IAccountManagementDataService>(), connectionStrings));
        services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
                .AddJwtBearer(options =>
        {
            options.TokenValidationParameters = new TokenValidationParameters
            {
                ValidateIssuer = false,
                ValidateAudience = false,
                ValidateLifetime = true,
                ValidateIssuerSigningKey = true,
                ValidIssuer = "https://codeproject.microservices.com",
```

```
                ValidAudience = "https://codeproject.microservices.com",
                IssuerSigningKey = new SymmetricSecurityKey(Encoding.ASCII.GetBytes(
                                        "CodeProject.Shared.Common.TokenManagement"))
            };
        });
        services.AddScoped<SecurityFilter>();
        services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
        services.AddSignalR();
    }
    // This method gets called by the runtime. Use this method to configure the HTTP request
pipeline.
    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        app.UseCors("SiteCorsPolicy");
        app.UseAuthentication();
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }
        else
        {
            app.UseHsts();
        }

        app.UseHttpsRedirection();
        app.UseMvc();
    }
}
```

Tại Account Management WebAPI ta sẽ cấu hình các thành phần:

- *CORS Policy Configuration* - Ứng dụng sẽ thực hiện các yêu cầu web có nguồn gốc chéo. Thành phần CORS Middleware được yêu cầu để xử lý các yêu cầu xuất xứ chéo cho ứng dụng. Chính sách nguồn gốc chéo có thể được chỉ định khi thêm thành phần phần mềm trung gian CORS.
- *Database Configuration* - Các chuỗi kết nối cơ sở dữ liệu có thể được truy xuất từ cài đặt cấu hình JSON của ứng dụng và được đưa vào đường ống khi khởi động đồng thời định cấu hình Entity Framework Core DbContext và nhà cung cấp cơ sở dữ liệu của nó (SQL Server).
- *Dependency Injection* - ASP.NET Core hỗ trợ mẫu thiết kế phần mềm tiêm phụ thuộc (DI), đây là một kỹ thuật để đạt được Inversion of Control (IoC) giữa các lớp và các phụ thuộc của chúng.
- *JWT Bearer Token Authentication* - Thêm xác thực token vào Web API của bạn trong ASP.NET Core nhờ thành phần middleware JwtBearerAuthentication có trong framework. Điều này cho phép bạn cấu hình cách JSON web token được xác thực và cấu hình.
- *Action Filters* - Bộ lọc trong ASP.NET Core 2.1 cho phép bạn chạy một khối code trước hoặc sau các giai đoạn cụ thể trong đường ống xử lý request. Đối với ứng dụng này, token của người dùng sẽ được phân tích cú pháp trên mỗi yêu cầu web để trích xuất thông tin xác thực về người dùng.

- *Add MVC* - Thêm MVC vào đường ống thực thi yêu cầu sẽ đảm bảo rằng tất cả các yêu cầu cho ứng dụng web của bạn có thể định tuyến theo khung MVC, nghĩa là bạn có thể sử dụng bộ điều khiển, khung nhìn và mọi thứ khác có trong triển khai MVC.
- *Add SignalR* - Thêm hỗ trợ cho khung SignalR giúp phát triển chức năng web real-time dễ dàng. SignalR cho phép giao tiếp hai chiều giữa server và Client. Server có thể đẩy nội dung đến các Client được kết nối ngay lập tức.

Lúc này, ta sẽ cấu hình các thành phần lên các dịch vụ khác của ứng dụng để sử dụng được JSONWebToken được khởi tạo từ dịch vụ AccountManagement, ta cấu hình tại controller (ví dụ tại SalesOrderController) các thành phần:

- *Action Filter* - Bộ lọc hành động SecurityFilter được thêm vào controller sẽ thực thi một đoạn mã trước khi thực hiện từng phương thức hành động của bộ điều khiển.
- *Authorzization* - Thuộc tính Authorize được thêm vào sẽ thực hiện xác thực JSON Web Token.
- *EnableCors* - Kích hoạt CORS sẽ thực hiện chính sách CORS khi nó được cấu hình trong lớp khởi động.
- *Dependency Injection* - Sẽ tự động tích hợp Inventory Management Business Service thông qua hàm khởi tạo của controller. Một SignalR cũng sẽ được đưa vào thông qua hàm khởi tạo.

```
[ServiceFilter(typeof(SecurityFilter))]
[Authorize]
[Route("api/[controller]")]
[EnableCors("SiteCorsPolicy")]
[ApiController]
public class SalesOrderController : ControllerBase
{
    private readonly IInventoryManagementBusinessService
_inventoryManagementBusinessService;

    private IHubContext<MessageQueueHub> _messageQueueContext;

    /// <summary>
    /// Sales Controller
    /// </summary>
    public SalesOrderController(IInventoryManagementBusinessService
inventoryManagementBusinessService,
                                IHubContext<MessageQueueHub> messageQueueContext)
    {
        _inventoryManagementBusinessService = inventoryManagementBusinessService;
        _messageQueueContext = messageQueueContext;
    }
}
```

Khi cấu hình Asp.Net Core để sử dụng Xác thực JWT Bearer Token, ta sẽ có quyền truy cập vào token trong mỗi request được cung cấp. Thông qua cấu hình trong startup và controller, ActionFilter dưới đây sẽ được thực thi trước khi thực hiện mỗi phương thức của WebAPI controller. Asp.Net Core hiển thị thuộc tính HttpContext.User dưới dạng

đối tượng ClaimsPrincipal. Đối tượng người dùng được tự động gán thông tin xác nhận quyền từ token.

ActionFilter bên dưới trích xuất các thông tin trong request trong đó có token được gửi trong header của request và gán chúng vào class SecurityModel. Lớp SecurityModel được thêm vào HttpContext để các phương thức trong controller có thể tham chiếu thông tin yêu cầu và chuyển thông tin này đến các thành phần truy cập dữ liệu để lọc và bảo mật dữ liệu ở cấp độ người dùng.

```
public class SecurityFilter : IAsyncActionFilter
{
    /// <summary>
    /// Action Filter
    /// </summary>
    /// <param name="context"></param>
    /// <param name="next"></param>
    /// <returns></returns>

public async Task OnActionExecutionAsync(ActionExecutingContext context,
                                         ActionExecutionDelegate next)
    {
        string firstName = context.HttpContext.User.FindFirst(ClaimTypes.GivenName).Value;
        string lastName =
                    context.HttpContext.User.FindFirst(ClaimTypes.NameIdentifier).Value;
        string emailAddress = context.HttpContext.User.FindFirst(ClaimTypes.Email).Value;
        string companyName = context.HttpContext.User.FindFirst(ClaimTypes.Name).Value
        int userId = int.Parse(context.HttpContext.User.
                                    FindFirst(ClaimTypes.PrimarySid).Value);

        int accountId = int.Parse(context.HttpContext.User.FindFirst(
                                         ClaimTypes.PrimaryGroupSid).Value);

        string token = TokenManagement.CreateToken(userId, firstName,
                                                   lastName, emailAddress,
                                                   accountId, companyName);
        SecurityModel securityModel = new SecurityModel();
        securityModel.EmailAddress = emailAddress;
        securityModel.FirstName = firstName;
        securityModel.LastName = lastName;
        securityModel.UserId = userId;
        securityModel.AccountId = accountId;
        securityModel.Token = token;
        context.HttpContext.Items["SecurityModel"] = securityModel;
        var resultContext = await next();
    }
}
```

b) Đảm bảo toàn vẹn dữ liệu

Các giao dịch cơ sở dữ liệu xác định một mức cô lập, xác định mức độ mà một giao dịch phải được cách ly khỏi các sửa đổi dữ liệu được thực hiện bởi các giao dịch khác.

Tiêu chuẩn SQL xác định bốn mức cô lập:

- Read Uncommitted (Đọc không cam kết) - Đọc không cam kết là mức cô lập thấp nhất. Ở cấp độ này, một giao dịch có thể đọc và chưa thực hiện thay đổi thì bị thực

hiện bởi các giao dịch khác, do đó cho phép đọc bẩn. Ở cấp độ này, các giao dịch không bị cô lập với nhau.

- Read Committed (Đọc cam kết) - Mức cô lập này đảm bảo rằng mọi dữ liệu đã đọc được cam kết tại thời điểm nó được đọc. Do đó, nó không cho phép đọc bẩn. Giao dịch giữ khóa đọc hoặc ghi trên hàng hiện tại và do đó ngăn các giao dịch khác đọc, cập nhật hoặc xóa.
- Repeatable Read (Đọc lặp lại) - Đây là mức cô lập hạn chế nhất. Giao dịch giữ các khóa đọc trên tất cả các hàng mà nó tham chiếu và ghi các khóa trên tất cả các hàng mà nó chèn, cập nhật hoặc xóa. Vì các giao dịch khác không thể đọc, cập nhật hoặc xóa các hàng này, do đó, nó tránh được các lần đọc không lặp lại.
- Serializable (Nối tiếp) - Đây là mức cô lập cao nhất. Một thực thi nối tiếp được đảm bảo để được tuần tự hóa. Thực thi tuần tự hóa được định nghĩa là một thực thi các hoạt động trong đó thực hiện đồng thời các giao dịch dường như được thực thi ser seri.

Theo mặc định, Entity Framework Core sử dụng mức cô lập ReadCommitted. Do ứng dụng này có thể được sử dụng bởi hàng trăm người dùng cập nhật đồng thời số lượng sản phẩm và cổ phiếu, nên có nhiều khả năng nhiều người dùng có thể yêu cầu cập nhật đồng thời các hàng của bảng cơ sở dữ liệu. Để đảm bảo tính toàn vẹn dữ liệu và để ngăn chặn cập nhật ảo và mất dữ liệu, ta sẽ cấu hình mức cô lập Serializable. Sử dụng giao dịch tuần tự hóa sẽ đảm bảo rằng các cập nhật cho cùng một hàng sản phẩm sẽ được thực hiện theo thứ tự tuần tự, trong đó mỗi giao dịch SQL sẽ thực hiện để hoàn thành trước khi giao dịch SQL tiếp theo bắt đầu.

```
//
//    Begin a Serializable Transaction
//
_inventoryManagementDataService.OpenConnection(
                                    _connectionStrings.PrimaryDatabaseConnectionString);
_inventoryManagementDataService.BeginTransaction((int)IsolationLevel.Serializable);
```

Tuy nhiên, chỉ cần tạo một giao dịch tuần tự hóa là không đủ để đảm bảo tính toàn vẹn dữ liệu trong khi nhiều cập nhật đồng thời đang được thực hiện trên cùng một hàng của bảng cơ sở dữ liệu. Ngoài ra, bạn cần có được một khóa cập nhật cấp hàng khi chọn một hàng sẽ được cập nhật. Áp dụng SQL Server UPDLOCK cho câu lệnh SELECT sẽ giúp thực hiện điều này. UPDLOCK chỉ định rằng các khóa cập nhật sẽ được thực hiện và giữ cho đến khi giao dịch hoàn tất.

Một trong những điều thú vị với phiên bản mới nhất của Entity Framework Core là có thể ghi đè câu lệnh SELECT mà Entity Framework Core thường tạo. Entity Framework Core cho phép bạn thực hiện các truy vấn SQL thô khi làm việc với cơ sở dữ liệu quan hệ. Điều này có thể hữu ích nếu truy vấn bạn muốn thực hiện không thể được thể hiện bằng LINQ. Bởi vì chúng ta có thể tạo một câu lệnh SQL với UPDLOCK và sử dụng phương thức Entity Framework Core FromQuery để thực thi câu lệnh SQL với khóa cập nhật cấp hàng.

Điều quan trọng bây giờ là tham số hóa bất kỳ đầu vào nào của người dùng để bảo vệ chống lại tấn công SQL Injection. Entity Framework Core cũng hỗ trợ các truy vấn tham số. Bạn có thể đặt tham số trong chuỗi truy vấn SQL và sau đó cung cấp các giá trị tham số dưới dạng đối số bổ sung. Mọi giá trị tham số bạn cung cấp sẽ tự động được chuyển đổi thành đối tượng DbParameter.

Ví dụ, trong phương thức GetProductInformationUpdate của Inventory Management Repository, id sản phẩm đang được cung cấp dưới dạng đối số được tham số hóa và hàng đã chọn được trả về Inventory Management Service trong khi SQL Server giữ khóa trên hàng đó.

```
/// <summary>
/// Get Product Information For Update with exclusive row lock
/// </summary>
/// <param name="productId"></param>
/// <returns></returns>
public async Task<Product> GetProductInformationForUpdate(int productId)
{
    string sqlStatement =
                    "SELECT * FROM PRODUCTS WITH (UPDLOCK) WHERE PRODUCTID = @ProductId";
    DbParameter productIdParameter = new SqlParameter("ProductId", productId);
    Product product = await dbConnection.Products.FromSql(sqlStatement, productIdParameter)
                                                 .FirstOrDefaultAsync();
    return product;
}
```

c) Tách rời giao tiếp với Rabbit MQ khỏi Web API

Quan khảo sát và khuyến nghị của nhà phát hành thì nên triển khai RabbitMQ bên ngoài ứng dụng WebAPI. Vì bộ nhớ và tài nguyên của máy chủ web nên được coi là tài nguyên hạn chế, các ứng dụng WebAPI được thiết kế để trở thành bất đồng bộ với luồng được khởi tạo liên tục và hủy theo yêu cầu web. Triển khai RabbitMQ ta cần tạo ra các kết nối riêng trong ứng dụng WebAPI, việc tạo quá nhiều luồng đơn lẻ như vậy làm giảm đáng kể hiệu năng. Do đó các kết nối RabbitMQ nên được thực thi mà không cần mở và đóng nhiều lần và nên được triển khai bên ngoài ứng dụng WebAPI.

Để làm điều này, tôi tạo thêm một ứng dụng console Asp.Net Core 2.2 đa luồng tách biệt WebAPI, với nhiệm vụ là một dịch vụ MessageQueue cho mỗi khối chức năng, nó sẽ quản lý và xử lý tất cả các kết nối, kênh RabbitMQ và xử lý message. Bây giờ ứng dụng cần chủ động gửi message theo thời gian thực, vì vậy cần một giải pháp để gửi message đến dịch vụ MessageQueue (console) để nó chủ động xử lý tin nhắn.

Asp.Net Core SignalR là một thư viện mã nguồn mở giúp đơn giản hóa việc thêm chức năng web thời gian thực vào các ứng dụng, cho phép ứng dụng WebAPI chủ động đẩy dữ liệu đến MessageQueue. SignalR sử dụng các Hub để giao tiếp giữa máy chủ và máy khách. Hub là một đường ống cho phép máy khách gọi các phương thức trên máy chủ và ngược lại.

Để tạo Hub, ta chỉ cần thêm một lớp kế thừa Microsoft.AspNetCore.SignalR.Hub và xác định các phương thức trong lớp Hub có thể thực hiện bởi máy khách. Ví dụ:

```
namespace CodeProject.InventoryManagement.WebApi.SignalRHub
{
    public class MessageQueueHub : Hub
    {

    }
}
```

Trong Asp.Net Core Signal, ta có thể truy cập một instance của IHubContext thông qua injection dependency (tiêm các phụ thuộc). Một phiên bản của IHubContext được cấu hình trong lớp khởi động và được đưa vào controller và có thể được sử dụng để gửi tin nhắn đến máy khách. Ví dụ, câu lệnh sau thực hiện gửi một thông điệp tới tất cả các máy khách đang lắng nghe sự kiện xảy ra trên url “http://localhost:44302/MessageQueueHub”.

```
await _messageQueueContext.Clients.All.SendAsync(MessageQueueEndpoints.InventoryQueue,
                                                 string.Empty);
```

Để lắng nghe các tin nhắn ASP.NET Core SignalR, dịch vụ MessageQueue của quản lý kho thực hiện sử dụng gói Microsoft.AspNetCore.SignalR.Client. Thư viện máy khách ASP.NET Core SignalR .NET cho phép bạn giao tiếp với các trung tâm SignalR từ các ứng dụng .NET.

Dịch vụ hàng đợi tin nhắn sẽ khởi động các tác vụ luồng riêng biệt để gửi, nhận và xử lý tin nhắn trong hàng đợi tin nhắn. Khi bắt đầu chuỗi tác vụ SendMessages, một kết nối đến SignalR được thiết lập dựa trên URL SignalR của API quản lý kho lưu trữ của "https://localhost:44302/MessageQueueHub".

Trong trường hợp hub không hoạt động và chạy khi cố gắng kết nối với nó, logic kết nối lại được thêm vào để thử lại kết nối. Sau khi được kết nối với trung tâm, dịch vụ hàng đợi tin nhắn sẽ lắng nghe các sự kiện On và trên mỗi sự kiện được đưa lên, dịch vụ xếp hàng sẽ gọi phương thức GetMessgaesInQueue để lấy tin nhắn và gửi chúng đến RabbitMQ.

```
/// <summary>
/// Start Process Interval
/// </summary>
/// <param name="cancellationToken"></param>
/// <returns></returns>
public Task StartAsync(CancellationToken cancellationToken)
{
    StartSignalRConnection();
    _timer = new Timer(GetMessagesInQueue, null, TimeSpan.Zero,
                       TimeSpan.FromSeconds(_appConfig.SendingIntervalSeconds));
    return Task.CompletedTask;
}
/// <summary>
/// Start SignalR Connection
/// </summary>
private async void StartSignalRConnection()
{
```

```
    if (string.IsNullOrEmpty(_appConfig.SignalRHubUrl))
    {
        return;
    }
    string url = _appConfig.SignalRHubUrl; /// "https://localhost:44340/MessageQueueHub",
    //
    //  Build Hub Connection
    //
    Boolean buildHubConnection = false;
    while (buildHubConnection  == false)
    {
        try
        {
            _signalRHubConnection = new HubConnectionBuilder().WithUrl(url).Build();
            buildHubConnection  = true;
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
            await Task.Delay(5000);
        }
    }
    //
    //   Listen for SignalR messages
    //
    _signalRHubConnection.On<string>(_signalRQueue, (message) =>
    {
        this.GetMessagesInQueue(null);
    });
    //
    //   Listen for Hub Connection Closed Event
    //
    _signalRHubConnection.Closed += async (error) =>
    {
        Console.WriteLine("SignalR Connection Closed");
        await Task.Delay(10000);
        await _signalRHubConnection.StartAsync();
        Console.WriteLine("Restart SignalR");
    };
    //
    //  Start Hub Connection
    //
    connected = false;
    while (connected == false)
    {
        try
        {
              await _signalRHubConnection.StartAsync();
              connected = true;
        }
        catch (Exception ex)
        {
              await Task.Delay(10000);
        }
    }
}
```

4.3.3. Kết quả đạt được

Ứng dụng hoàn thành với tên gọi Quản lý bán hàng (Quản lý kinh doanh – MS.BusinessManagement). Thực hiện 5 khối chức năng chính:

- Quản lý tài khoản (MS.AccountManagement) cung cấp các chức năng đăng ký, đăng nhập, cung cấp token để sử dụng các dịch vụ khác
- Quản lý kho hàng (MS.InventoryManagement) cung cấp các chức năng quản lý sản phẩm (CRU), cập nhật chi tiết đơn hàng từ ứng dụng web, tự động cập nhật thông tin đơn hàng từ RabbitMQ
- Quản lý đơn nhập hàng (MS.PurchaseOrderManagement) cung cấp các chức năng quản lý nhà cung cấp (CRU), quản lý đơn nhập hàng (CRU), tự động cập nhật thông tin đơn hàng, thông tin sản phẩm từ RabbitMQ
- Quản lý đơn bán hàng (MS.SalesOrderManagement) cung cấp các chức năng quản lý khách hàng (CRU), quản lý đơn bán hàng (CRU), tự động cập nhật thông tin đơn hàng, thông tin sản phẩm từ RabbitMQ
- Quản lý ghi nhật ký hệ thống (MS.LoggingManagement) cung cấp chức năng tự động lấy thông tin từ RabbitMQ để ghi nhật ký tương ứng theo hành động.

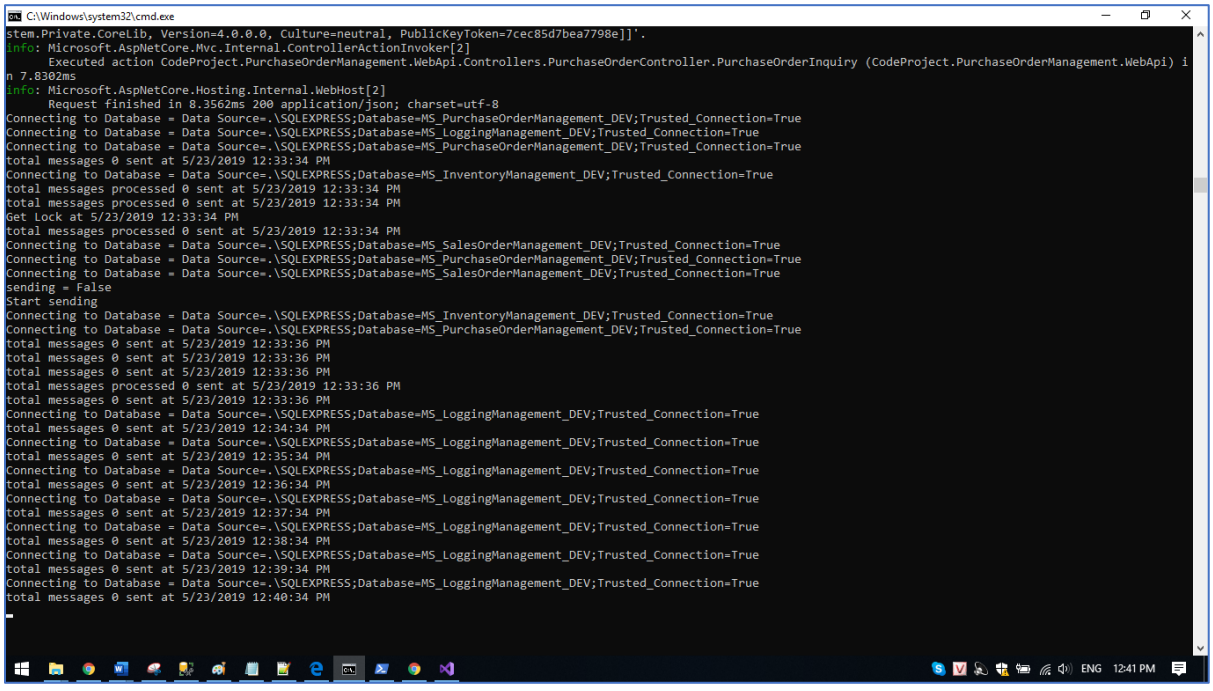
Thống kê thông tin mã nguồn ứng dụng:

<table>
<tr><td>Thông tin</td><td colspan="2">Số liệu thống kê</td></tr>
<tr><td>Số lượng project trong mã nguồn</td><td colspan="2">5 project Web Appication
MS.AccountManagement.WebAPI
MS.InventoryManagement.WebAPI
MS.PurchaseOrderManagement.WebAPI
MS.SalesOrderManagement.WebAPI
MS.Portal
4 project Console Application
MS.InventoryManagement.MessageQueues
MS.PurchaseOrderManagement.MessageQueues
MS.SalesOrderManagement.MessageQueues
MS.LoggingManagement.MessageQueues
MS.Support
11 project ClassLibrary
MS.Common
MS.AccountManagement.Service
MS.InventoryManagement.Service
MS.PurchaseOrderManagement.Service
MS.SalesOrderManagement.Service
MS.AccountManagement.Data
MS.InventoryManagement.Data
MS.PurchaseOrderManagement.Data
MS.SalesOrderManagement.Data
MS.LoggingManagement.Data</td></tr>
<tr><td>Số class trong mã nguồn</td><td>MS.Common
MS.AccountManagement.WebAPI
MS.AccountManagement.Service
MS.AccountManagement.Data</td><td>42
4
3
9</td></tr>
</table>

	MS.InventoryManagement.WebAPI MS.InventoryManagement.MessageQueues MS.InventoryManagement.Service MS.InventoryManagement.Data MS.PurchaseOrderManagement.WebAPI MS.PurchaseOrderManagement.MessageQueues MS.PurchaseOrderManagement.Service MS.PurchaseOrderManagement.Data MS.SalesOrderManagement.WebAPI MS.SalesOrderManagement.MessageQueues MS.SalesOrderManagement.Service MS.SalesOrderManagement.Data MS.LoggingManagement.MessageQueues MS.LoggingManagement.Data MS.Support MS.Portal	7 3 3 25 6 3 3 21 6 3 3 21 3 8 2
Tổng số class trong mã nguồn	175 class	
Dung lượng toàn bộ mã nguồn	357 MB (bao gồm cả mã nguồn đã biên dịch)	

Bảng 4.2: Thống kê thông tin mã nguồn ứng dụng

4.3.4. Minh họa các chức năng của ứng dụng

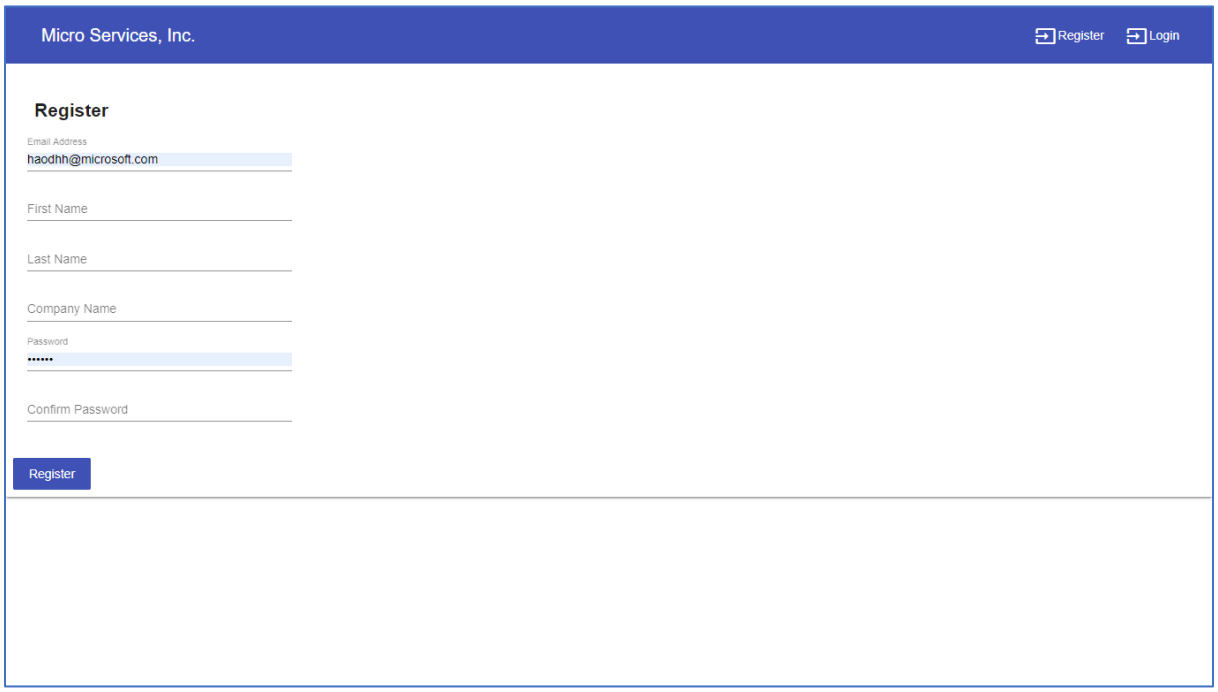


Hình 4.24: Giao diện ứng dụng hỗ trợ khởi chạy ứng dụng

Hình 4.24 mô tả ứng dụng SpawnProcesses, hiển thị thông tin các kết nối giữa các dịch vụ, kết nối với RabbitMQ, kết nối với SQLServer, số lượng message được trao đổi.

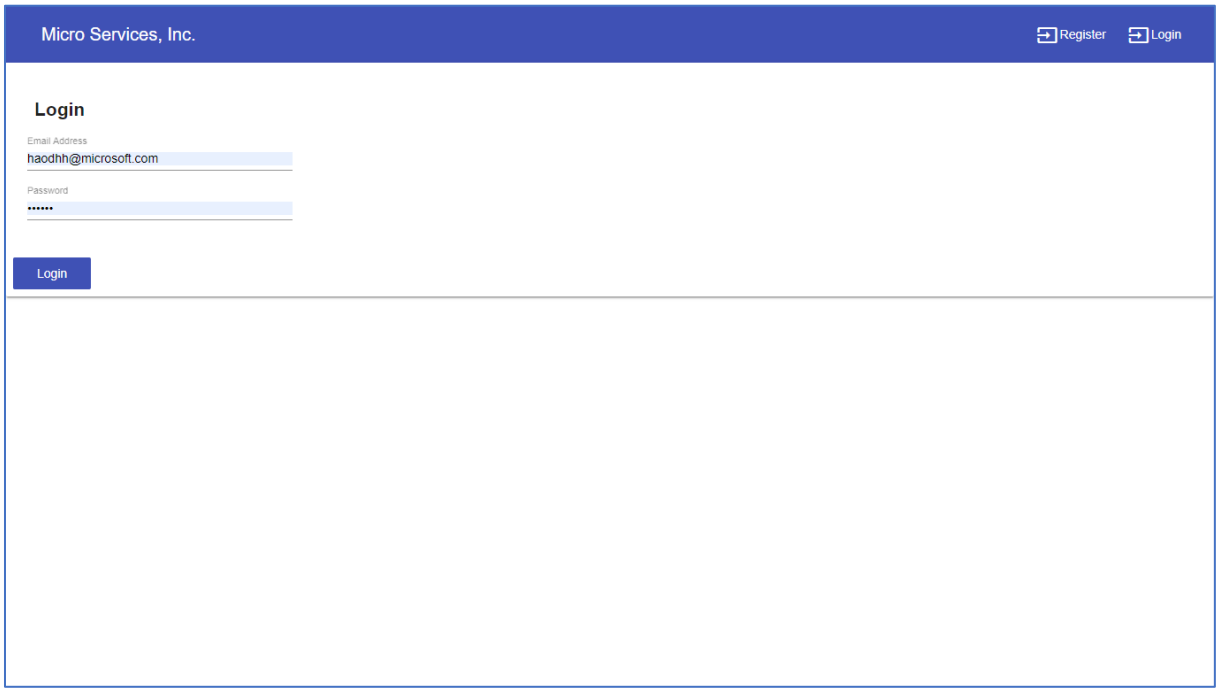
Hình 4.25: Giao diện trang chủ ứng dụng

Hình 4.25 mô tả giao diện trang chủ khi truy cập vào trang chủ ứng dụng, tại đây người dùng có thể đăng nhập hoặc đăng ký để sử dụng các dịch vụ khác của ứng dụng.



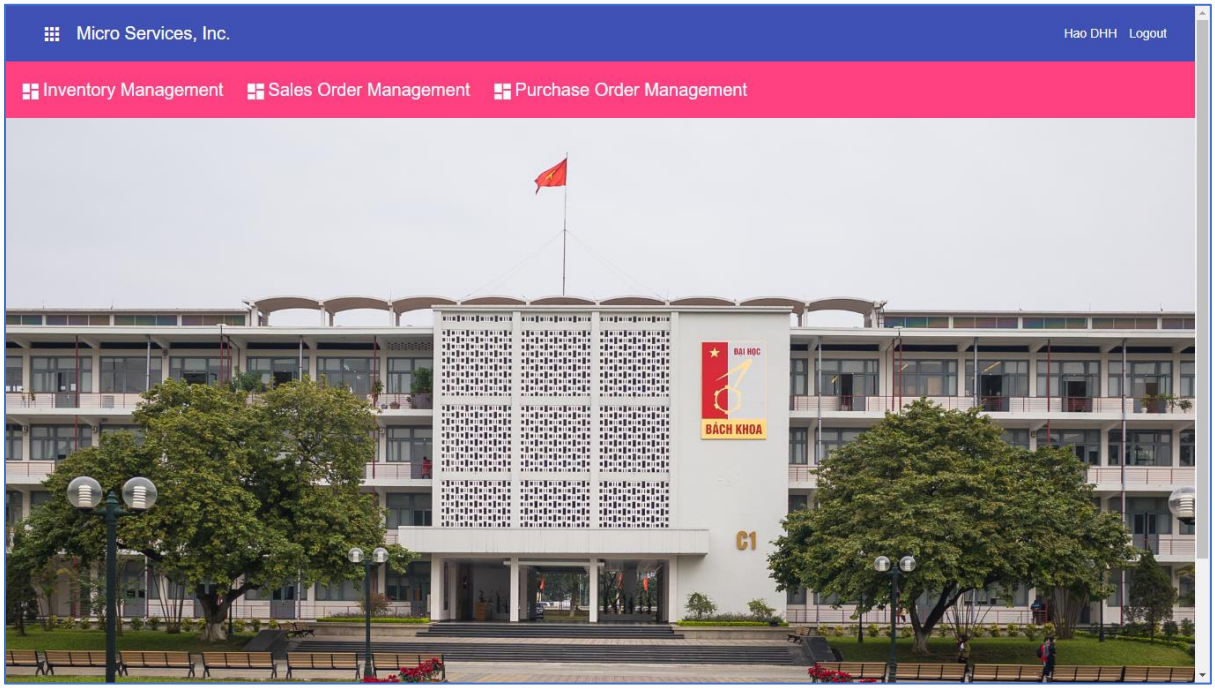
Hình 4.26: Giao diện chức năng đăng ký

Hình 4.26 mô tả chức năng đăng ký, cho phép người dùng nhập vào các thông tin tài khoản như Email Address, First Name, Last Name, Company Name, Password, Confirm Password và thực hiện đăng ký tài khoản bằng nhấn vào button “Register”.



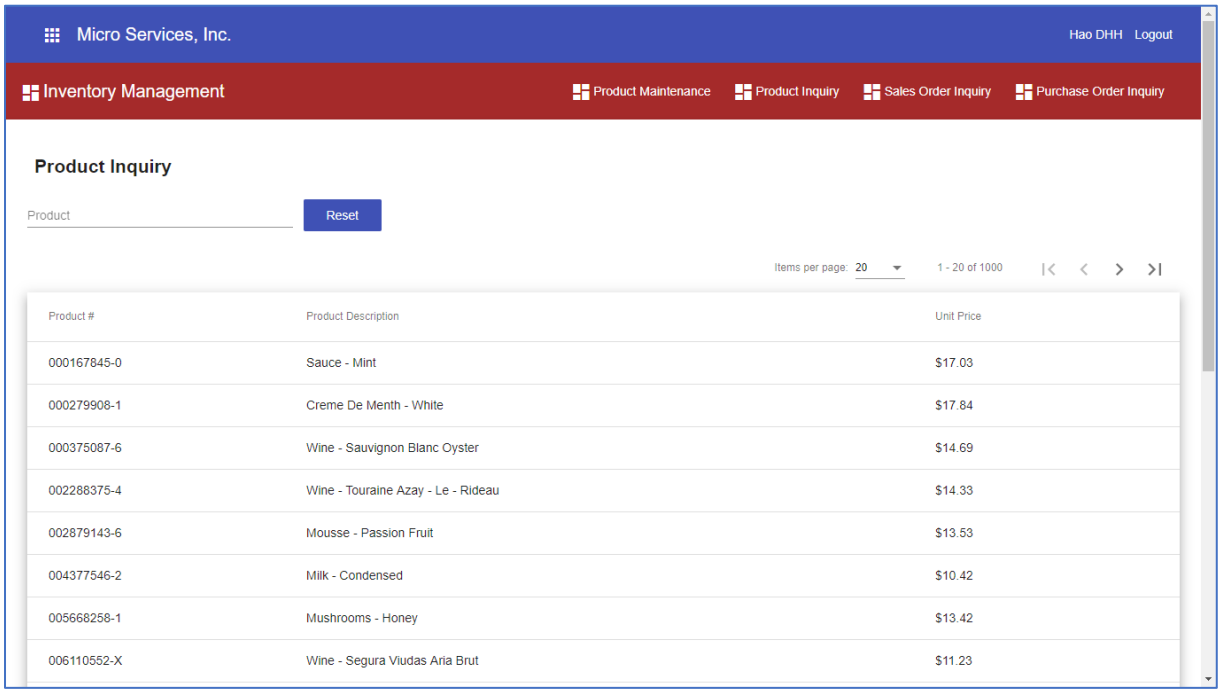
Hình 4.27: Giao diện chức năng đăng nhập

Hình 4.27 mô tả chức năng đăng nhập, cho phép người dùng nhập vào Email Address, Password và thực hiện đăng nhập vào ứng dụng bằng cách nhấn button "Login".



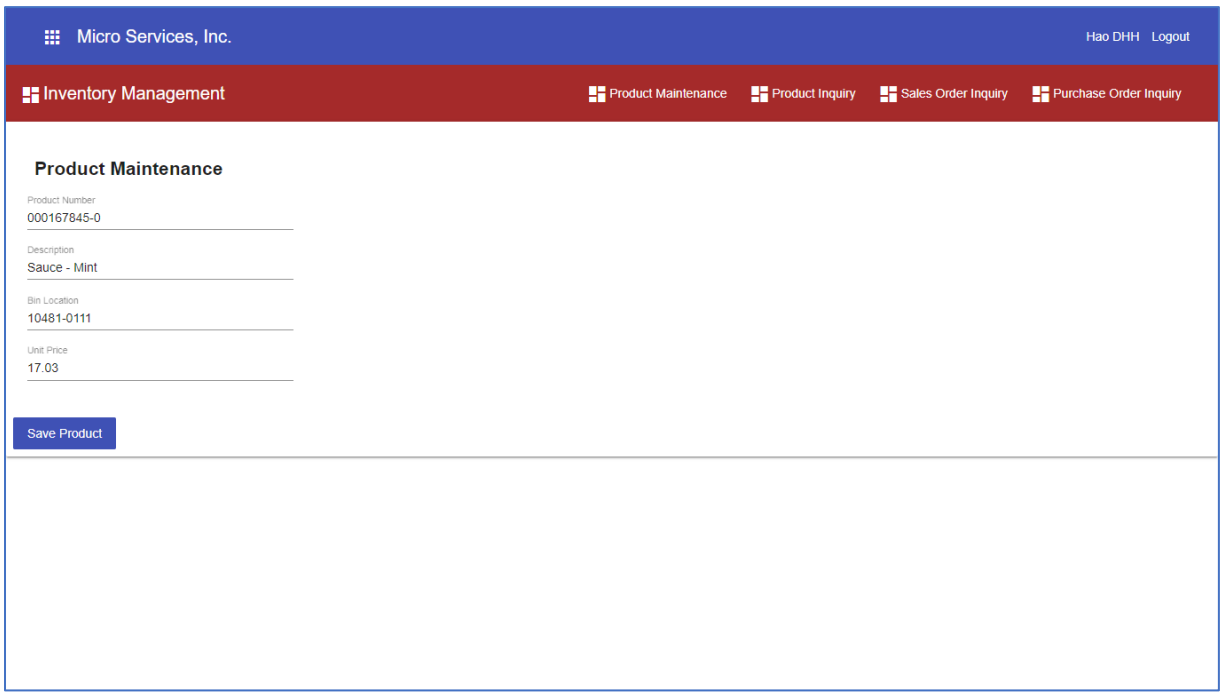
Hình 4.28: Giao diện trang chủ sau khi đăng nhập

Hình 4.28 mô tả giao diện trang chủ ứng dụng sau khi đăng nhập, tại đây người dùng có thể sử dụng các dịch vụ được hiển thị trên Menu.



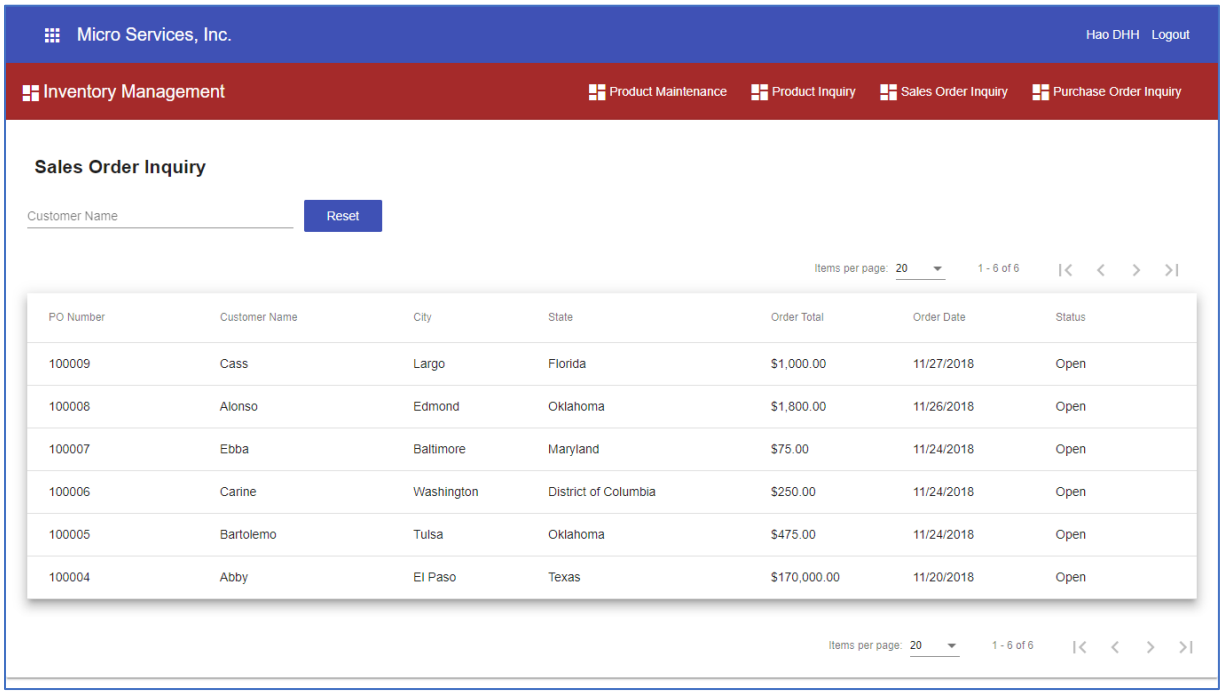
Hình 4.29: Giao diện chức năng hiển thị danh sách sản phẩm

Hình 4.29 mô tả chức năng hiển thị danh sách sản phẩm, tại đây người dùng có thể tìm kiếm sản phẩm và nhấn vào sản phẩm bất kỳ để xem chi tiết.



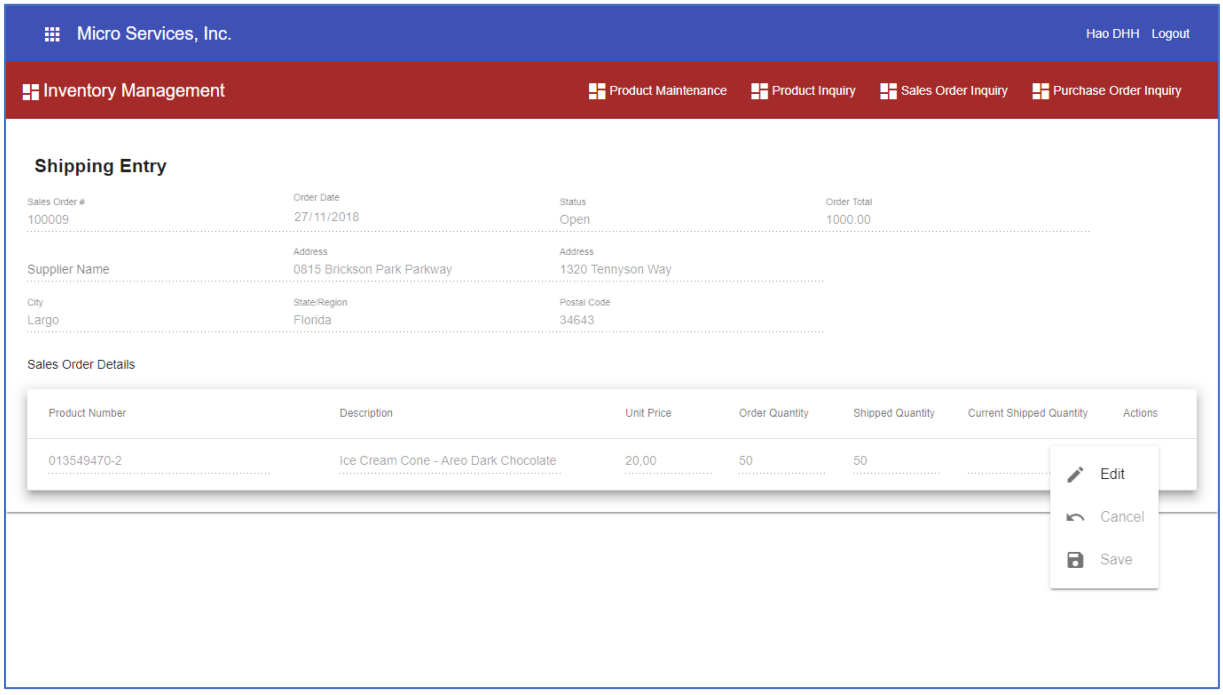
Hình 4.30: Giao diện chức năng tạo mới hoặc cập nhật sản phẩm

Hình 4.30 mô tả chức năng tạo mới hoặc cập nhật sản phẩm, cụ thể người dùng nhập thông tin sản phẩm và dựa vào mã sản phẩm mà ứng dụng sẽ thực hiện tạo mới nếu chưa có hoặc cập nhật thông tin vào sản phẩm đã có với mã số này.



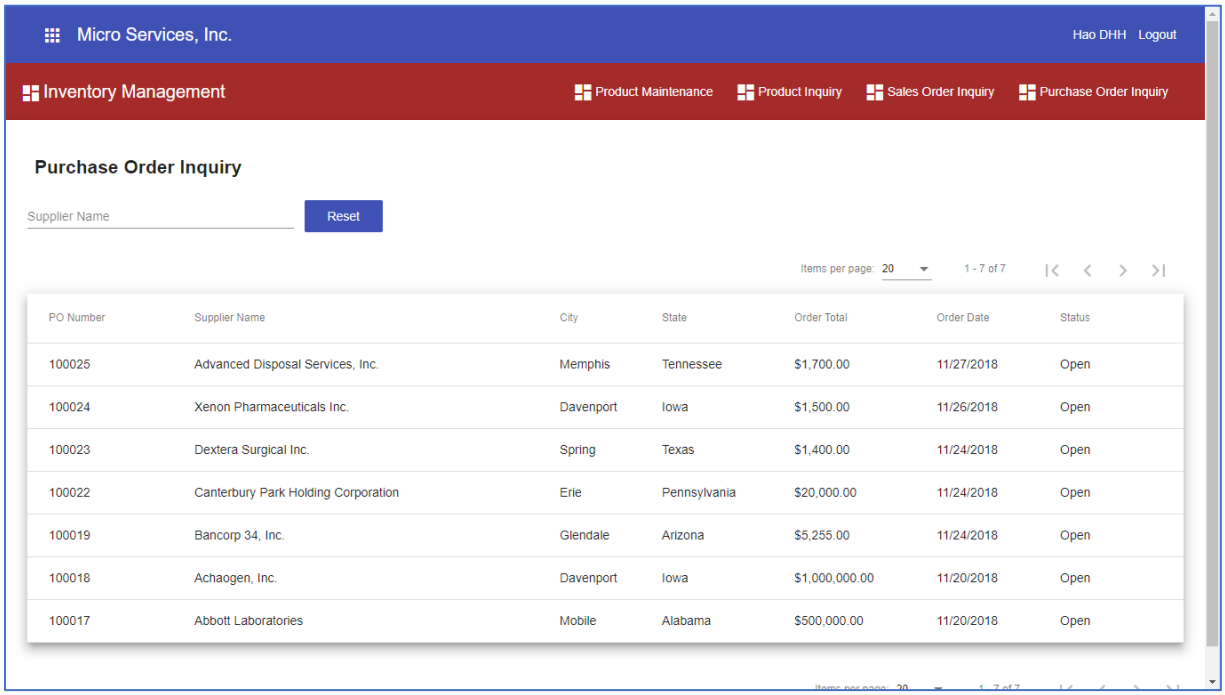
Hình 4.31: Giao diện chức năng hiển thị danh sách đơn bán hàng

Hình 4.31 mô tả chức năng hiển thị danh sách đơn bán hàng của dịch vụ quản lý kho, tại đây người dùng tìm kiếm và có thể nhấn vào đơn hàng bất kỳ để xem chi tiết..



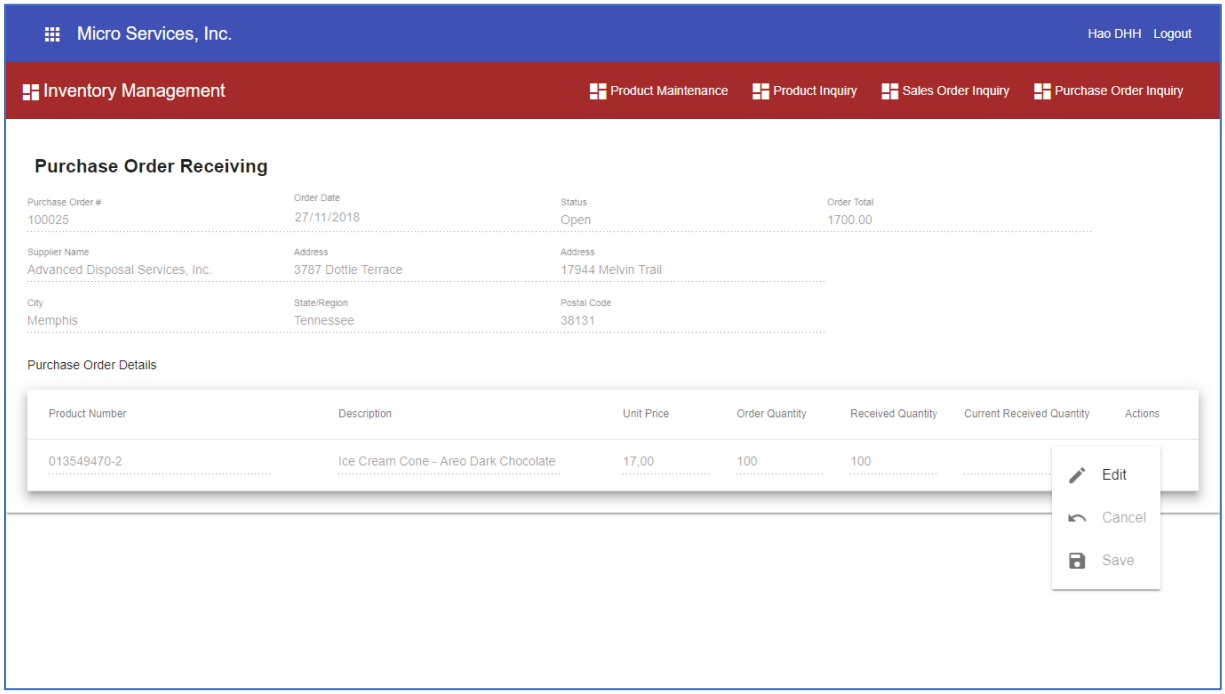
Hình 4.32: Giao diện chức năng cập nhật chi tiết đơn bán hàng

Hình 4.32 mô tả chức năng cập nhật chi tiết đơn bán hàng của dịch vụ quản lý kho, tại đây người dùng có thể cập nhật số lượng sản phẩm có thể bán vào đơn hàng.



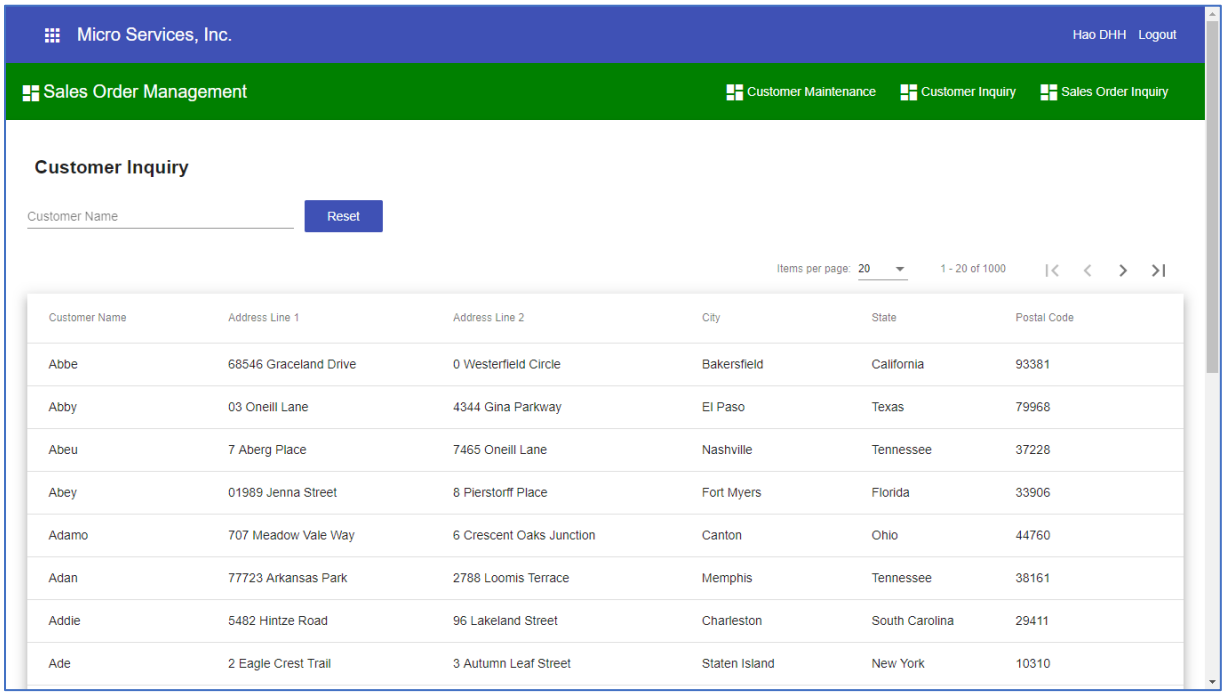
Hình 4.33: Giao diện chức năng hiển thị danh sách đơn nhập hàng

Hình 4.33 mô tả chức năng hiển thị danh sách đơn nhập hàng của dịch vụ quản lý kho, tại đây người dùng có thể tìm kiếm và nhấn vào đơn hàng bất kỳ để xem chi tiết.



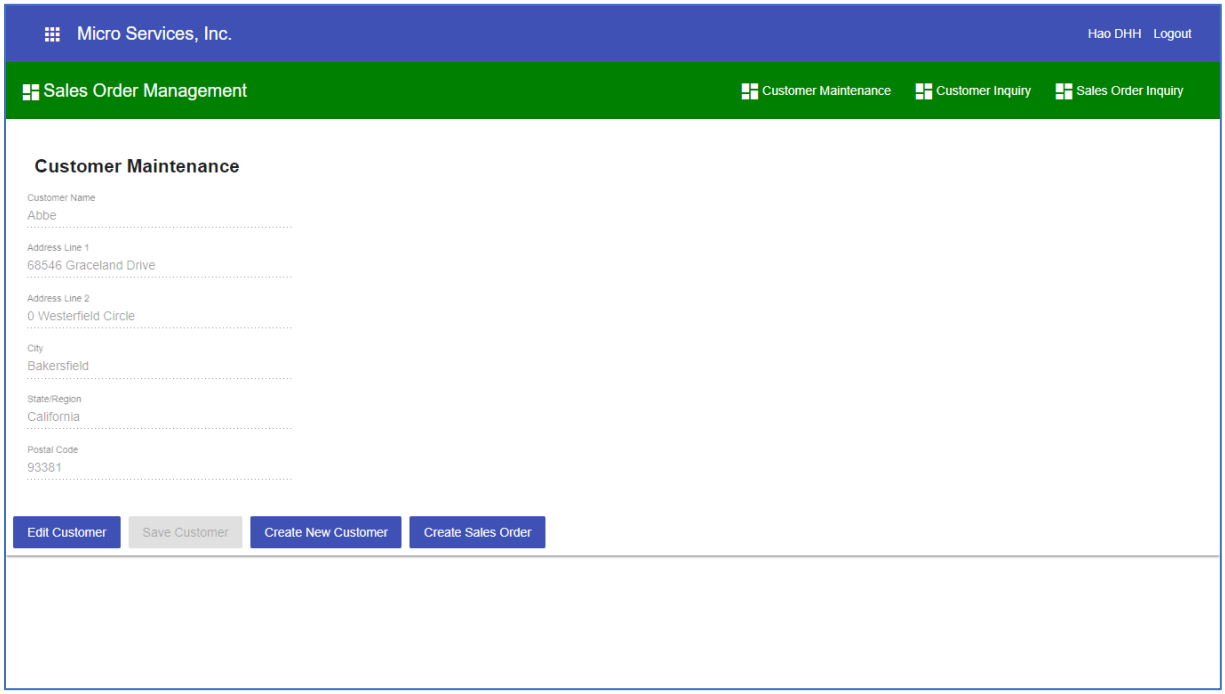
Hình 4.34: Giao diện chức năng cập nhật chi tiết đơn nhập hàng

Hình 4.34 mô tả chức năng cập nhật chi tiết đơn nhập hàng của dịch vụ quản lý kho, tại đây người dùng có thể cập nhật số lượng sản phẩm được nhập vào đơn hàng.



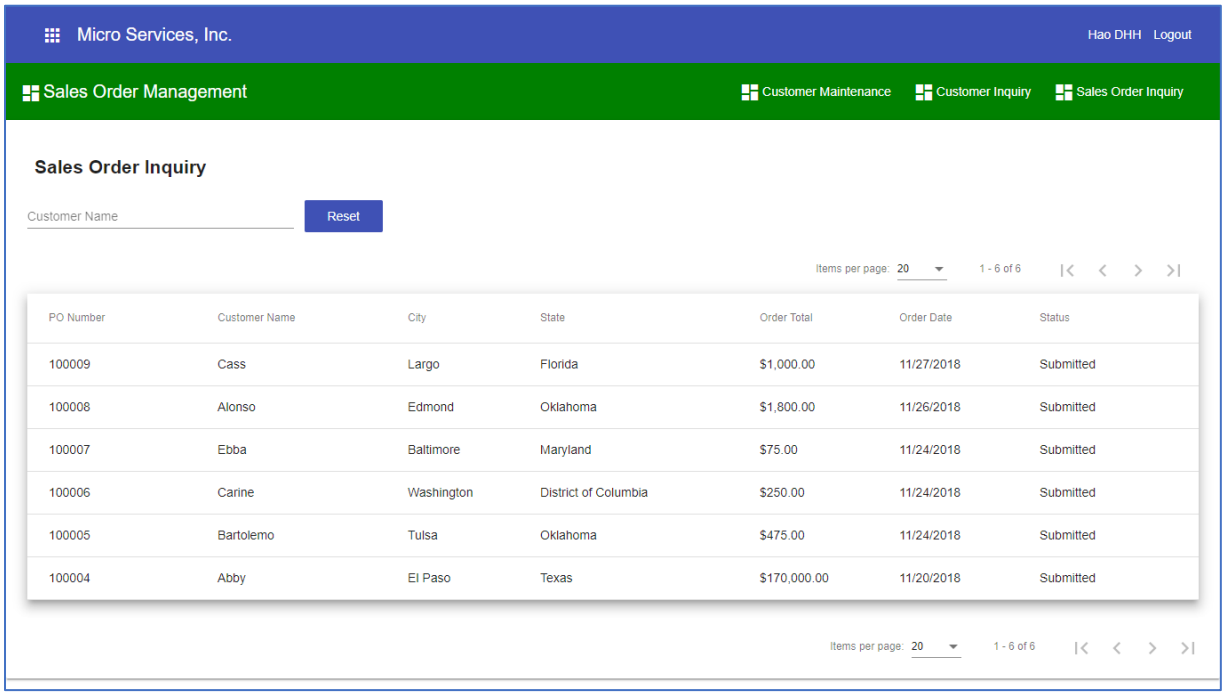
Hình 4.35: Giao diện chức năng hiển thị danh sách khách hàng

Hình 4.35 mô tả chức năng hiển thị danh sách khách hàng, tại đây người dùng có thể tìm kiếm và nhấn vào khách hàng bất kỳ dể xem thông tin chi tiết.



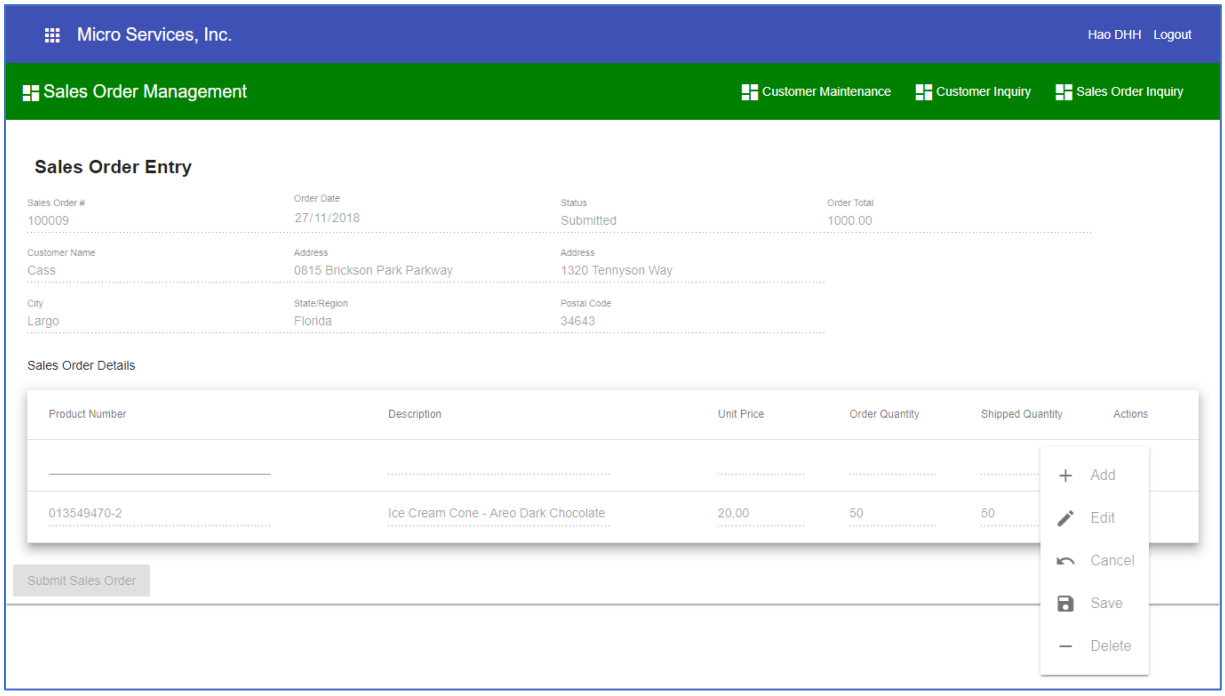
Hình 4.36: Giao diện chức năng quản lý khách hàng

Hình 4.36 mô tả chức năng quản lý khách hàng bao gồm tạo mới khách hàng, cập nhật thông tin khách hàng, tạo đơn hàng mới cho khách hàng.



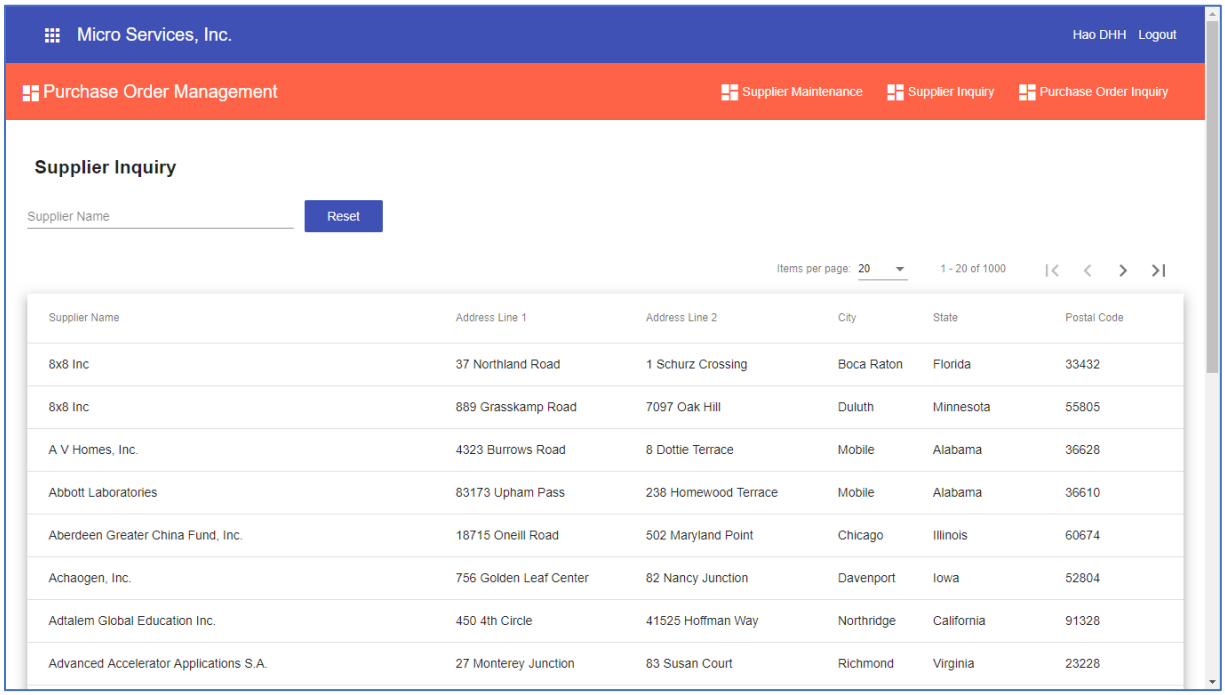
Hình 4.37: Giao diện chức năng hiển thị dánh sách đơn bán hàng

Hình 4.37 mô tả chức năng hiển thị danh sách đơn bán hàng của dịch vụ quản lý đơn bán hàng, người dùng có thể tìm kiếm và nhấn vào đơn hàng bất kỳ để xem chi tiết.



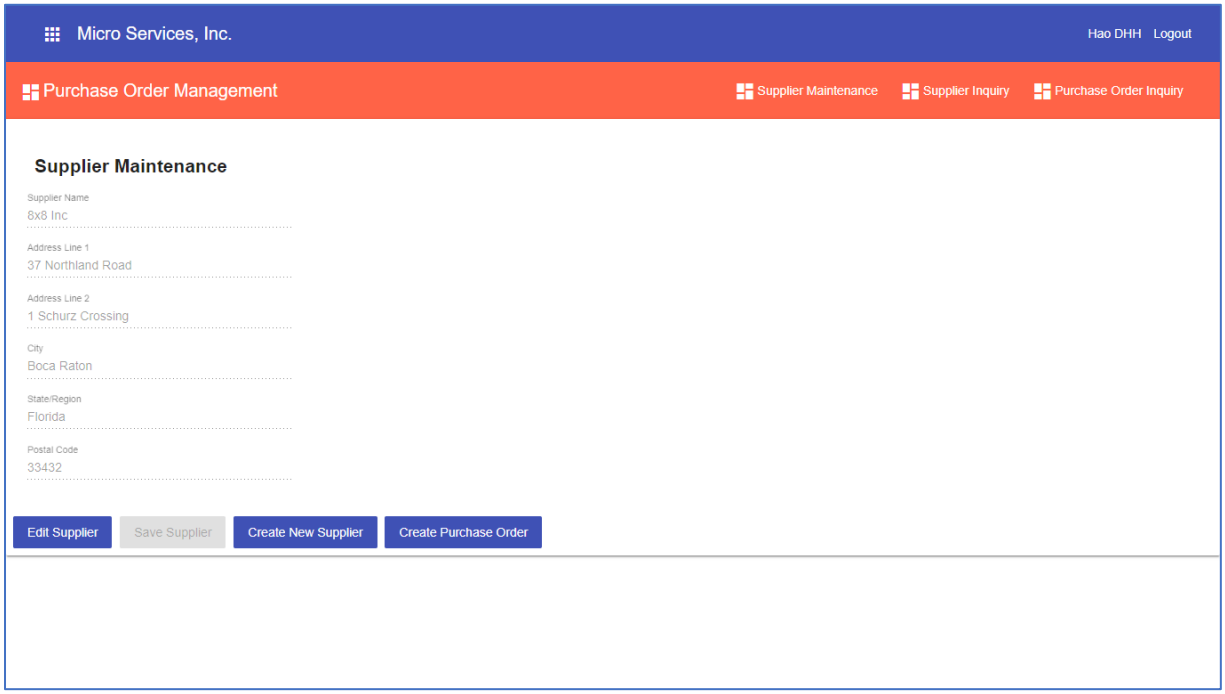
Hình 4.38: Giao diện chức năng quản lý chi tiết đơn bán hàng

Hình 4.38 mô tả chức năng quản lý chi tiết đơn bán hàng của dịch vụ quản lý đơn bán hàng, bao gồm hiển thị thông tin đơn hàng, cho phép thêm sửa xóa các chi tiết của đơn hàng như sản phẩm, số lượng, đơn giá.



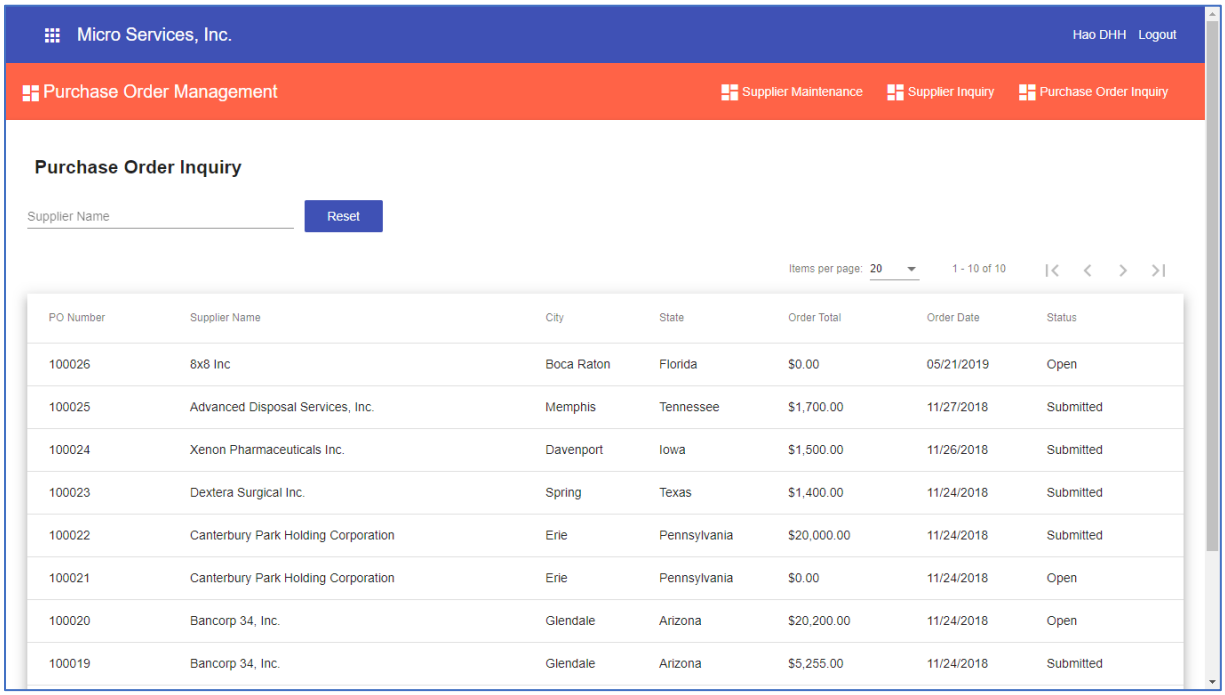
Hình 4.39: Giao diện chức năng hiển thị danh sách nhà cung cấp

Hình 4.39 mô tả chức năng hiển thị danh sách nhà cung cấp, người dùng có thể tìm kiếm và nhấn vào bất kỳ đơn hàng nào để xem chi tiết đơn hàng.



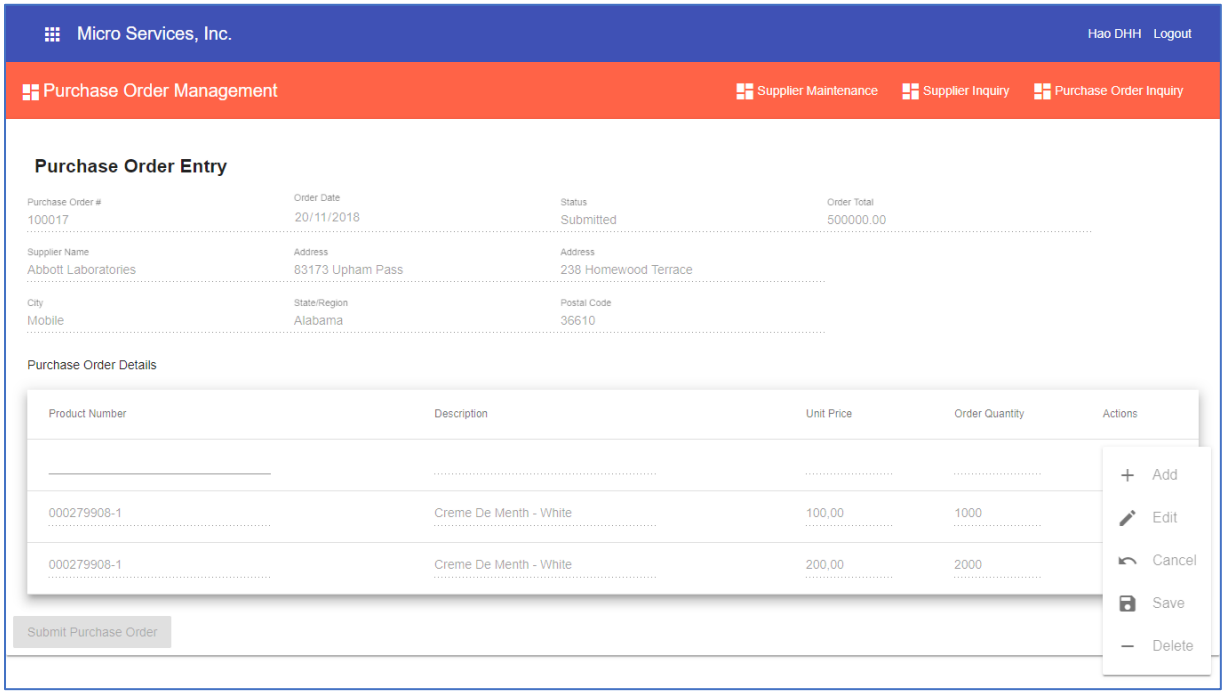
Hình 4.40: Giao diện chức năng quản lý nhà cung cấp

Hình 4.40 mô tả chức năng quản lý nhà cung cấp bao gồm tạo mới nhà cung cấp, cập nhật thông tin nhà cung cấp, tạo đơn hàng mới với nhà cung cấp.



Hình 4.41: Giao diện chức năng hiển thị danh sách đơn nhập hàng

Hình 4.41 mô tả chức năng hiển thị danh sách đơn nhập hàng của dịch vụ quản lý đơn nhập hàng,người dùng có thể tìm kiếm và nhấn vào đơn hàng bất kỳ để xem chi tiết.



Hình 4.42: Giao diện chức năng quản lý chi tiết đơn nhập hàng

Hình 4.42 mô tả chức năng quản lý chi tiết đơn nhập hàng của dịch vụ quản lý đơn nhập hàng, bao gồm hiển thị thông tin đơn hàng, cho phép thêm sửa xóa các chi tiết của đơn hàng như sản phẩm, số lượng, đơn giá.

4.4. Kiểm thử

4.4.1. Xây dựng kịch bản

Các loại kịch bản cần xây dựng kiểm thử:

- Kiểm thử vận hành của ứng dụng: khả năng build, run của ứng dụng.
- Kiểm thử khả năng đáp ứng của giao diện: khả năng co dãn theo kích thước màn hình, màu sắc, độ tương phản, kích thước các thành phần
- Kiểm thử các kiểm tra giá trị đầu vào: khi tạo mới hoặc cập nhật các đối tượng như sản phẩm, nhà cung cấp, khách hàng, đơn hàng.
- Kiểm thử các kết nối giữa của dịch vụ: kết nối SignalR giữa 2 dịch vụ cùng khối chức năng, kết nối từ client đến server, kết nối đến RabbitMQ, kết nối đến database.

4.4.2. Kết quả kiểm thử

- Ứng dụng có thể build và run trên Linux và MacOS, ứng dụng có thể chạy trên hầu hết các trình duyệt hiện tại hỗ trợ Javascript.
- Giao diện của ứng dụng có thể co dãn theo kích thước màn hình, kích thước và vị trí các thành phần không bị chồng chéo, màu sắc hiển thị không bị thay đổi.
- Các kết nối giữa các dịch vụ, kết nối với RabbitMQ, kết nối với SQL Server hoạt động ổn định

Chức năng	Đầu vào	Đầu ra	Kết quả
Tạo mới / Cập nhật sản phẩm (Product Maintenance)	*Product Number*: “000167845-0” *Description*: “Sauce – Mint” *Bin Location*: “10481-0111” *Unit Price*: “17.03”	successfully	Đạt
Tạo mới / Cập nhật sản phẩm (Product Maintenance)	*Product Number*: “” *Description*: “Sauce – Mint” *Bin Location*: “10481-0111” *Unit Price*: “17.03”	error	Đạt
Tạo mới / Cập nhật sản phẩm (Product Maintenance)	*Product Number*: “000167845-0” *Description*: “” *Bin Location*: “10481-0111” *Unit Price*: “17.03”	error	Đạt
Tạo mới / Cập nhật sản phẩm (Product Maintenance)	*Product Number*: “000167845-0” *Description*: “Sauce – Mint” *Bin Location*: “” *Unit Price*: “17.03”	error	Đạt
Tạo mới / Cập nhật sản phẩm (Product Maintenance)	*Product Number*: “000167845-0” *Description*: “Sauce – Mint” *Bin Location*: “10481-0111” *Unit Price*: “”	error	Đạt

Tạo mới / Cập nhật khách hàng (Customer Maintenance)	*Customer Name*: “Abbe” *Address Line 1*: “68546 Graceland Drive” *Address Line 2*: “0 Westerfield Circle” *City*: “Bakersfield” *State/Region*: “California” *Postal Code*: “93381”	successfully	Đạt
Tạo mới / Cập nhật khách hàng (Customer Maintenance)	*Customer Name*: “” *Address Line 1*: “68546 Graceland Drive” *Address Line 2*: “0 Westerfield Circle” *City*: “Bakersfield” *State/Region*: “California” *Postal Code*: “93381”	error	Đạt
Tạo mới / Cập nhật khách hàng (Customer Maintenance)	*Customer Name*: “Abbe” *Address Line 1*: “” *Address Line 2*: “0 Westerfield Circle” *City*: “Bakersfield” *State/Region*: “California” *Postal Code*: “93381”	error	Đạt
Tạo mới / Cập nhật khách hàng (Customer Maintenance)	*Customer Name*: “Abbe” *Address Line 1*: “68546 Graceland Drive” *Address Line 2*: “” *City*: “Bakersfield” *State/Region*: “California” *Postal Code*: “93381”	error	Đạt
Tạo mới / Cập nhật khách hàng (Customer Maintenance)	*Customer Name*: “Abbe” *Address Line 1*: “68546 Graceland Drive” *Address Line 2*: “0 Westerfield Circle” *City*: “” *State/Region*: “California” *Postal Code*: “93381”	error	Đạt
Tạo mới / Cập nhật khách hàng (Customer Maintenance)	*Customer Name*: “Abbe” *Address Line 1*: “68546 Graceland Drive” *Address Line 2*: “0 Westerfield Circle” *City*: “Bakersfield” *State/Region*: “” *Postal Code*: “93381”	error	Đạt
Tạo mới / Cập nhật khách hàng (Customer Maintenance)	*Customer Name*: “Abbe” *Address Line 1*: “68546 Graceland Drive” *Address Line 2*: “0 Westerfield Circle” *City*: “Bakersfield” *State/Region*: “California” *Postal Code*: “”	error	Đạt

Tạo mới / Cập nhật nhà cung cấp (Supplier Maintenance)	*Supplier Name:* "A V Homes, Inc." *Address Line 1:* "4323 Burrows Road" *Address Line 2:* "8 Dottie Terrace" *City:* "Mobile" *State/Region:* "Alabama" *Postal Code:* "36628"	successfully	Đạt
Tạo mới / Cập nhật nhà cung cấp (Supplier Maintenance)	*Supplier Name:* "" *Address Line 1:* "4323 Burrows Road" *Address Line 2:* "8 Dottie Terrace" *City:* "Mobile" *State/Region:* "Alabama" *Postal Code:* "36628"	error	Đạt
Tạo mới / Cập nhật nhà cung cấp (Supplier Maintenance)	*Supplier Name:* "A V Homes, Inc." *Address Line 1:* "" *Address Line 2:* "8 Dottie Terrace" *City:* "Mobile" *State/Region:* "Alabama" *Postal Code:* "36628"	error	Đạt
Tạo mới / Cập nhật nhà cung cấp (Supplier Maintenance)	*Supplier Name:* "A V Homes, Inc." *Address Line 1:* "4323 Burrows Road" *Address Line 2:* "" *City:* "Mobile" *State/Region:* "Alabama" *Postal Code:* "36628"	error	Đạt
Tạo mới / Cập nhật nhà cung cấp (Supplier Maintenance)	*Supplier Name:* "A V Homes, Inc." *Address Line 1:* "4323 Burrows Road" *Address Line 2:* "8 Dottie Terrace" *City:* "" *State/Region:* "Alabama" *Postal Code:* "36628"	error	Đạt
Tạo mới / Cập nhật nhà cung cấp (Supplier Maintenance)	*Supplier Name:* "A V Homes, Inc." *Address Line 1:* "4323 Burrows Road" *Address Line 2:* "8 Dottie Terrace" *City:* "Mobile" *State/Region:* "" *Postal Code:* "36628"	error	Đạt
Tạo mới / Cập nhật nhà cung cấp (Supplier Maintenance)	*Supplier Name:* "A V Homes, Inc." *Address Line 1:* "4323 Burrows Road" *Address Line 2:* "8 Dottie Terrace" *City:* "Mobile" *State/Region:* "Alabama" *Postal Code:* ""	error	Đạt

RUD chi tiết đơn bán hàng (Sales Order Entry)	*Product Number*: "013549470-2" *Description*: "Ice Cream Cone - Areo Dark Chocolate" *Unit Price*: "20,00" *Order Quantity*: "50"	successfully	Đạt
RUD chi tiết đơn bán hàng (Sales Order Entry)	*Product Number*: "" *Description*: "Ice Cream Cone - Areo Dark Chocolate" *Unit Price*: "20,00" *Order Quantity*: "50"	error	Đạt
RUD chi tiết đơn bán hàng (Sales Order Entry)	*Product Number*: "013549470-2" *Description*: "Ice Cream Cone - Areo Dark Chocolate" *Unit Price*: "" *Order Quantity*: "50"	error	Đạt
RUD chi tiết đơn bán hàng (Sales Order Entry)	*Product Number*: "013549470-2" *Description*: "Ice Cream Cone - Areo Dark Chocolate" *Unit Price*: "20,00" Order Quantity: ""	error	Đạt
RUD chi tiết đơn nhập hàng (Purchase Order Entry)	*Product Number*: "013549470-2" *Description*: "Ice Cream Cone - Areo Dark Chocolate" *Unit Price*: "17,00" *Order Quantity*: "100"	successfully	Đạt
RUD chi tiết đơn nhập hàng (Purchase Order Entry)	*Product Number*: "" *Description*: "Ice Cream Cone - Areo Dark Chocolate" *Unit Price*: "17,00" *Order Quantity*: "100"	error	Đạt
RUD chi tiết đơn nhập hàng (Purchase Order Entry)	*Product Number*: "013549470-2" *Description*: "Ice Cream Cone - Areo Dark Chocolate" *Unit Price*: "" *Order Quantity*: "100"	error	Đạt
RUD chi tiết đơn nhập hàng (Purchase Order Entry)	*Product Number*: "013549470-2" *Description*: "Ice Cream Cone - Areo Dark Chocolate" *Unit Price*: "17,00" *Order Quantity*: ""	error	Đạt

Bảng 4.3: Kết quả kiểm thử

4.5. Triển khai

Phần hướng dẫn này dành cho những ai đang có mã nguồn và muốn chạy thử nghiệm ứng dụng. Đầu tiên ta cần thiết lập môi trường hoạt động cho ứng dụng, hệ điều hành mà bạn nên sử dụng ở đây là Windows 10 để có được sự tương thích tốt nhất vì đây cũng là hệ điều hành mà ứng dụng này được thiết kế và xây dựng. Tuy nhiên bạn vẫn có thể biên dịch và chạy ứng dụng trên hệ điều hành Linux hoặc MacOS nếu cài đặt đủ các công cụ, phần mềm hỗ trợ vì ứng dụng được phát triển trên nền tảng .NetCore.

Các công cụ, phần mềm mà cần cài đặt là SQL Server Management Studio và SQL Server Express (2014 trở lên), Visual Studio 2017 hoặc 2019 phiên bản Professional hoặc Community Edition, ASP.NET Core 2.2, RabbitMQ 3.8 hoặc mới hơn, NodeJS 10.15.3 hoặc mới hơn và Angular Cli 6 hoặc mới hơn. Để có thể quan sát các message trên RabbitMQ một cách trực quan, ta cài đặt RabbitMQ Web UI Management tool.

Tiếp theo ta sẽ biên dịch lại ứng dụng, tôi đã viết một file .bat để hỗ trợ cho việc này. Thực hiện build toàn bộ ứng dụng bằng chạy file "BuildAllProject.bat" trong thư mục MS.Support. Để đảm bảo rằng không có lỗi xảy ra trong toàn bộ quá trình này, tôi khuyên bạn sử dụng Visual Studio và rebuild lại toàn bộ các project để có thể tự cập nhật lại các dependencies cần thiết. Sau đó ta build ứng dụng frontend là MS.Portal, việc cần làm trước tiên là ta cần restore lại file package.json để ứng dụng cập nhật lại toàn bộ các package tránh trường hợp build lỗi do thiếu package, và build bằng lệnh "ng build" trên command line. Đến đây ta thực hiện xong việc build ứng dụng.

Để ứng dụng chạy được thì ta cần cài đặt database cho nó, cách một là sử dụng migration có thể thực hiện dưới dạng command line để tự động tạo database thông qua các model đã tạo sẵn, cách hai là attach các file database được đính kèm trong thư mục MS.Database có trong mã nguồn.

Cuối cùng là chạy thử nghiệm ứng dụng, bước một là khởi chạy dịch vụ message queue của RabbitMQ bằng cách Start RabbitMQ Service, bước hai khởi chạy các dịch vụ phía backend bằng cách chạy file "StartServer.bat" trong thư mục MS.Support, bước ba là khởi chạy ứng dụng web phía frontend thông qua commandline bằng lệnh "ng serve", và bây giờ ta có thể truy cập "http://localhost:4200" để sử dụng ứng dụng.

Với một số trường hợp đặc biệt là các cổng mà ứng dụng sử dụng không khả dụng trên máy của bạn thì bạn có thể cấu hình lại tại file "Properties/launchSettings.json" đối với các ứng dụng "WebAPI", tại file "AppConfig.cs" đối với các dịch vụ MessageQueue (ví dụ: InventoryAppConfig.cs), và tại "src\app\app.component.ts" cho ứng dụng frontend MS.Portal.

CHƯƠNG 5: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

5.1. Kết luận

Kiến trúc microservice ra đời đã giải quyết được rất nhiều các nhược điểm mà kiến trúc nguyên khối đem lại. Rất nhiều tập đoàn lớn như Amazon, eBay, Netflix,… triển khai kiến trúc này và đem lại hiệu quả tốt. Tuy kiến trúc microservice giải quyết được rất nhiều nhược điểm của kiến trúc nguyên khối nhưng nó cũng vẫn còn và phát sinh thêm nhiều vấn đề khác cần giải quyết, vì thế kiến trúc này vẫn còn phải tiếp tục nghiên cứu và hoàn thiện.

Đối với quy mô đồ án này, ứng dụng xây dựng được với tên gọi "Quản lý bán hàng" cung cấp các chức năng cơ bản của một ứng dụng bình thường như quản lý sản phẩm, quản lý khách hàng, quản lý nhà cung cấp, quản lý đơn bán hàng, quản lý đơn nhập hàng và ứng dụng được thiết kế và xây dựng trên kiến trúc microservice.

Qua quá trình làm đồ án tôi thu được rất nhiều kiến thức về việc thiết kế, xây dựng và triển khai một ứng dụng theo kiến trúc microservice, cũng như kinh nghiệm áp dụng các công nghệ hữu ích vào ứng dụng, nó giúp ích rất nhiều vào việc triển khai các dự án thực tế. Bên cạnh đó, đồ án đã giúp tôi hoàn thiện thêm các kỹ năng tìm kiếm thông tin, làm việc nhóm và viết báo cáo,…

5.2. Hướng phát triển

Do ứng dụng được thiết kế và xây dựng dưới quy mô đồ án nên chỉ dừng lại ở mức tổng quát để học tập và thực hành việc triển khai một ứng dụng microservice. Ứng dụng này có số lượng chức năng còn hạn chế cũng như giao diện và cấu trúc của ứng dụng có thể chưa phải là tối ưu nhất. Và ứng dụng ở dạng tổng quát nên hoàn toàn có thể vận dụng để phát triển lên một dự án thực tế. Vì thế sau kết quả của đồ án này, em sẽ tiếp tục nâng cấp, bổ sung thêm các chức năng cho hệ thống như cho phép nhập dữ liệu từ file, thiết kế thêm giao diện và chức năng để khách hàng và nhà cung cấp sử dụng, cũng như tiếp tục nghiên cứu để thiết kế một cấu trúc mã nguồn có thể tối ưu hơn.

Cuối cùng, một lần nữa xin gửi lời cảm ơn chân thành tới TS. Hoàng Văn Hiệp đã gợi ý đề tài, đồng hành và giúp đỡ trong quá trình hoàn thiện đồ án để có thể đạt được kết quả tốt nhất. Xin chân thành cảm ơn thày!

TÀI LIỆU THAM KHẢO

[1] Cesar de la Torre and Bill Wagner and Mike Rousos, .NET Microservices: Architecture for Containerized .NET Applications, Mike Rousos, Principal Software Engineer, DevDiv CAT team, Microsoft, 2019

[2] David Dossot, RabbitMQ Essentials, Packt Publishing Ltd., 2014

[3] José M. Aguilar, SignalR Programming in Microsoft ASP.NET, Microsoft Press, 2014

[4] Entity Framework Core, https://docs.microsoft.com/en-us/ef/core/, last visited May 2019

[5] What is Angular?, https://angular.io/docs, last visited May 2019

[6] Goalkicker.com, Angular 2+ Notes for Professionals, Stack Overflow Documentation, 2018