

**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI
PHÂN HIỆU TẠI TP. HỒ CHÍ MINH
BỘ MÔN CÔNG NGHỆ THÔNG TIN**



ĐỒ ÁN TỐT NGHIỆP

**ĐỀ TÀI: TÌM HIỂU KIẾN TRÚC XÂY DỰNG PHẦN MỀM
MICROSERVICES**

Giảng viên hướng dẫn: PHẠM THI VƯƠNG

Sinh viên thực hiện: PHẠM VĂN TỊNH

Lớp : CÔNG NGHỆ THÔNG TIN

Khoá : K57

Tp. Hồ Chí Minh, năm 2020

**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI
PHÂN HIỆU TẠI TP. HỒ CHÍ MINH
BỘ MÔN CÔNG NGHỆ THÔNG TIN**



ĐỒ ÁN TỐT NGHIỆP

**ĐỀ TÀI: TÌM HIỂU KIẾN TRÚC XÂY DỰNG PHẦN MỀM
MICROSERVICES**

Giảng viên hướng dẫn: PHẠM THI VƯƠNG

Sinh viên thực hiện: PHẠM VĂN TỊNH

Lớp : CÔNG NGHỆ THÔNG TIN

Khoá : K57

Tp. Hồ Chí Minh, năm 2020

ĐỒ ÁN TỐT NGHIỆP

Họ và tên sinh viên: PHẠM VĂN TỊNH

MSSV: 5751071043

Chuyên ngành: Công nghệ thông tin

Lớp: CQ.57.CNTT

1. Tên đề tài đồ án tốt nghiệp:

 Tìm hiểu kiến trúc xây dựng phần mềm Microservices

2. Nhiệm vụ thực tập tốt nghiệp:

- Tìm hiểu kiến trúc xây dựng phần mềm Microservices.
- Tìm hiểu và xây máy chủ nhận dạng (Identity Server) .

3. Ngày bắt đầu thực hiện đồ án tốt nghiệp: ngày ... tháng năm

4. Ngày hoàn thành báo cáo đồ án tốt nghiệp: ngày tháng năm

5. Họ tên giáo viên hướng dẫn: PHẠM THI VƯƠNG

Trưởng bộ môn

Tp. Hồ Chí Minh, ngày Thángnăm 2020
Giáo viên hướng dẫn

PHẠM THI VƯƠNG

LỜI CẢM ƠN

Trước tiên với tình cảm sâu sắc và chân thành nhất, cho phép em tỏ lòng biết ơn thầy PHẠM THI VƯƠNG (giảng viên trường Đại Học Sài Gòn) đã tạo điều kiện hỗ trợ, giúp đỡ em trong suốt quá trình học tập và nghiên cứu đề tài Xác Thực và Phân Quyền Trong Hệ Thống Microservices. Trong suốt thời gian từ khi bắt đầu thực hiện đồ án thực tập tốt nghiệp đến nay, em nhận được rất nhiều sự quan tâm giúp đỡ của thầy Phạm Thi Vương.

Với lòng biết ơn sâu sắc nhất, em xin gửi đến quý Thầy Cô ở bộ môn Công Nghệ Thông Tin trường Đại Học Giao Thông Vận Tải phân hiệu TPHCM đã truyền đạt vốn kiến thức cho chúng em trong suốt quá trình học tập tại trường. Nhờ có những lời hướng dẫn, dạy bảo của thầy cô nên đề tài nghiên cứu của em mới có thể hoàn thành tốt đẹp.

Một lần nữa, em xin chân thành cảm ơn thầy cô đã trực tiếp giúp đỡ, quan tâm, hướng dẫn em hoàn thành tốt bài báo cáo này trong thời gian qua.

Bài báo cáo đồ án thực hiện trong khoảng thời gian gần 3 tháng. Bước đầu đi vào thực tế của em còn nhiều hạn chế và còn nhiều ngỡ ngàng nên không tránh khỏi thiếu sót, em rất mong nhận được những ý kiến đóng góp quý báu của quý Thầy Cô để kiến thức của em trong lĩnh vực này được hoàn thiện hơn đồng thời có điều kiện bổ sung, nâng cao ý thức của mình.

Em xin chân thành cảm ơn.

Tp. Hồ Chí Minh, ngày Thángnăm 2020
Sinh viên

PHẠM VĂN TỊNH

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Tp. Hồ Chí Minh, ngày Tháng năm 2020
Giáo viên hướng dẫn

PHẠM THI VƯƠNG

MỤC LỤC

DANH MỤC CHỮ VIẾT TẮT	1
BẢNG BIỂU, SƠ ĐỒ, HÌNH VẼ	4
CHƯƠNG 1 – GIỚI THIỆU MICROSERVICES	7
1.1 Tổng quan	7
1.2 Kiến trúc nguyên khối (Monolithic architecture)	8
1.3 Giới thiệu Microservices (Microservices Architecture)	9
CHƯƠNG 2 – KIẾN TRÚC MICROSERVICES HƯỚNG API.....	12
2.1 Đặc điểm chính	12
2.2 Kiến trúc Microservices: Lợi ích và hạn chế.....	13
2.2.1 Những lợi ích.....	13
2.2.2 Hạn chế.....	14
2.3 Áp dụng MSA	15
2.4 Giao tiếp từ ứng dụng phía client đến microservice	16
2.4.1 API Gateway	17
2.4.2 Tích hợp Microservices.....	18
2.4.3 Decentralized Gateway	20
2.4.4 API Gateway với Service Meshes.....	21
2.5 Kiến trúc Microservices – Kiến trúc Tham Chiếu.....	22
2.5.1 Kiến trúc phân lớp	22
2.5.2 Chuyển từ kiến trúc Monolithic sang kiến trúc Microservices.....	23
2.6 Kiến trúc Microservices – Kiến trúc Phân Đoạn	24
2.7 Kiến trúc Microservices – Kiến trúc Tế Bào	25
2.8 Cài đặt	26
2.8.1 Tích hợp và triển khai liên tục	26
2.8.2 Các công nghệ hữu ích cho MSA	28
2.9 MSA trong thực tế.....	30

2.9.1	Sử dụng Microservices vào thực tế	30
2.9.2	Tóm tắt	31
CHƯƠNG 3 – GIỚI THIỆU MÁY CHỦ NHẬN DẠNG (Identity Server)		32
3.1	Giới thiệu.....	32
3.1.1	Xác thực.....	33
3.1.2	Ủy quyền truy cập API	33
3.1.3	Kết hợp OpenID Connect và OAuth 2.0	34
3.2	Identity Server	34
3.2.1	Đăng nhập một lần (Single Sign-On - SSO).....	34
3.2.2	Xác thực (Authentication)	36
3.2.3	Kiểm soát truy cập - ủy quyền (Authorization).....	37
3.3	Identity Server dựa trên kiến trúc một khối	37
3.4	Máy chủ nhận dạng dựa trên kiến trúc Microservices.....	40
3.5	Xác định, xây dựng Identity Server dựa trên kiến trúc Microservices	42
3.5.1	Microservices Core.....	42
3.5.2	API Gateway	43
3.5.3	Các ứng dụng phía client	44
CHƯƠNG 4 – SỬ DỤNG MÁY CHỦ NHẬN DẠNG TRONG THỰC TẾ.....		46
4.1	Cài đặt	46
4.2	Đăng nhập vào ứng dụng của hệ thống thông qua Identity Server.....	51
4.3	Cấu hình ứng dụng client mới.....	52
4.4	Bảo vệ API bằng Identity Server	60
4.4.1	Bảo vệ tài nguyên API.....	60
4.4.2	Các quy định về phạm vi truy cập tài nguyên API	62
4.5	Tài nguyên nhận dạng	63
4.6	Quản lý người dùng.....	65
4.7	Quản lý phân quyền người dùng	67
4.8	Người dùng tự quản lý thông tin cá nhân của chính mình	68
CHƯƠNG 5 – TỔNG KẾT		69

5.1	Kết quả đạt được	69
5.2	Vấn đề chưa được giải quyết.....	69
CHƯƠNG 6 – TÀI LIỆU THAM KHẢO		70

DANH MỤC CHỮ VIẾT TẮT

STT	Từ viết tắt	Danh từ	Mô tả	Ghi chú
1		Microservice Architecture	Hệ thống phần mềm sử dụng kiến trúc phân tán	
2		Monolithic Architecture	Hệ thống phần mềm sử dụng kiến trúc nguyên khối	
3	SOA	Service-Oriented Architecture	Hệ thống phần mềm sử dụng kiến trúc hướng dịch vụ	
4	ERP	Enterprise Resource Planning	Phần mềm quản trị doanh nghiệp	
5	CRM	Customer Relationship Management	Phần mềm quản lý khách hàng	
6		Runtime	Thời gian chạy ứng dụng	
7		Service	Dịch vụ	
8	HTTP/HTTPS	Hypertext Transfer Protocol/ Hypertext Transfer Protocol Security	Là giao thức chuẩn của mạng internet	
9		microservice	Dịch vụ siêu nhỏ	
10	ESB	Enterprise Service Bus	Là một công cụ phần mềm trung gian phức tạp giúp tích hợp các thành phần, dịch vụ riêng lẻ thành một hệ thống và phân phối công việc giữa các thành phần	
11		Container	Nơi lưu trữ các ứng dụng	
12	REST	Representational State Transfer	Là một chuẩn thiết kế giao tiếp dựa trên HTTP	

13	gRPC		Là một chuẩn thiết kế giao tiếp dựa trên HTTP2	
14	SOILD		Là từ viết tắt của 5 từ đầu tiên trong nguyên tắc thiết kế hướng đối tượng	
15	SOAP	Simple Object Access Protocol	Là một chuẩn thiết kế giao tiếp	
16	JMS	Java Message Service		
17	AMQP	Advanced Message Queuing Protocol	Giao thức xếp hàng tin nhắn	
18	STOMP	Streaming Text Oriented Messaging Protocol	Giao thức tin nhắn hướng văn bản đơn giản	
19	MQTT	Message Queuing Telemetry Transport	Là giao thức mạng nhẹ, vận chuyển tin nhắn giữa các thiết bị	
20	JSON	JavaScript Object Notation	Là một kiểu dữ liệu mở trong Javascript	
21	RAML	RESTful API Modeling Language	Mô tả các API không tuân theo tất cả ràng buộc của REST	
22	WSDL	Web Services Description Language	Ngôn ngữ mô tả dịch vụ web	
23		Service Registry	Đăng ký dịch vụ	
24		Service Discovery	Khám phá dịch vụ	
25		Deployment	Triển khai	
26		Package	Gói	
27		OAuth2.0	Là tiêu chuẩn mở cho ủy quyền truy cập	

28		OpenID Connect	Là lớp xác thực trên OAuth2.0	
29	SSO	Single Sign-On	Đăng nhập một lần	
30	IAM	Identity Account Manager	Quản lý truy cập và nhận dạng	
31		Portable	Phần mềm xách tay, không cần phải cài đặt	
32		Load balancing	Cân bằng tải	
33	TLS/SSL	Transport Layer Security/ Secure Sockets Layer.	Là các giao thức mật mã được thiết kế để cung cấp giao tiếp an toàn qua một mạng máy tính	
34	SPA	Single Page Application	Ví dụ như Vue.js, React js, Angular	

BẢNG BIỂU, SƠ ĐỒ, HÌNH VẼ

Bảng 1.1: Các vùng kiến trúc MSA.

Bảng 2.1: Các công nghệ hữu ích cho MSA.

Bảng 3.1: So sánh Identity Server dựa trên hai kiến trúc.

Bảng 3.2: Xác định các microservices và chức năng.

Bảng 4.1: Điểm cuối truy cập của Identity Server.

Bảng 4.2: Thông số của ứng dụng client.

Bảng 4.3: Thông số kỹ thuật bảo vệ API bằng.

Bảng 4.4: Các thông số xác định phạm vi truy cập API.

Bảng 4.5: Thông số chi tiết danh tính xác thực và ủy quyền.

Hình 1.1: Kiến trúc kỹ thuật số điển hình.

Hình 1.2: Kiến trúc nguyên khối.

Hình 1.3: Kiến trúc kỹ thuật số hỗ trợ MSA.

Hình 2.1: Kiến trúc Microservices.

Hình 2.2: Một ví dụ MSA.

Hình 2.3: Kiến trúc Microservice với API Gateway.

Hình 2.4: Tích hợp trong MSA.

Hình 2.5: Micro-Gateway trong MSA.

Hình 2.6: Control Plane và Data Plane trong lưới dịch vụ.

Hình 2.7: Kiến trúc MSA phân lớp.

Hình 2.8: Kiến trúc MSA: Strangler Facade trong Strangler Pattern.

Hình 2.9: Kiến trúc MSA phân đoạn.

Hình 2.10: Kiến trúc MSA dựa trên Cell-Based .

Hình 2.11: Tích hợp và triển khai trong MSA 1.

Hình 2.12: Tích hợp và triển khai trong MSA 2.

Hình 3.1: Tổng quát Identity Server.

Hình 3.2: Single Sign-On.

Hình 3.3: Xác thực

Hình 3.4: Identity Server dựa trên kiến trúc nguyên khối.

Hình 3.5: Cấu trúc dự án Identity Server dựa theo kiến trúc nguyên khối.

Hình 3.6: Identity Server dựa trên kiến trúc Microservices.

Hình 3.7: Cấu hình một dịch vụ khi sử dụng Nginx làm chức năng cân bằng tải.

Hình 3.8: Đóng gói một dịch vụ thành Docker Image sử dụng Docker Compose.

Hình 3.9: API Gateway Ocelot cho ASP.NET Core.

Hình 3.10: Khai báo một API với API Gateway Ocelot bằng file Configuration.json.

Hình 4.1: Phiên bản Docker được cài đặt trên Ubuntu.

Hình 4.2: Tiện ích Docker trên VS Code.

Hình 4.3: Docker Image sau khi đóng gói.

Hình 4.4: Docker Container sau khi được khởi tạo từ Docker Image

Hình 4.5: Trang chủ Identity Server.

Hình 4.6: Trang chủ ứng dụng Admin Dashboard.

Hình 4.7: Trang chủ ứng dụng User Profile.

Hình 4.8: Swagger quản lý API Identity Microservice.

Hình 4.9: Đăng nhập vào ứng dụng client.

Hình 4.10: Đăng nhập vào ứng dụng thông qua Identity Server.

Hình 4.11: Mã token được gửi về ứng dụng client.

Hình 4.12: Tạo ứng dụng client mới với thông tin cơ bản.

Hình 4.13: Quy trình đăng nhập.

Hình 4.14: Thêm các nút chức năng đăng nhập, đăng xuất.

Hình 4.15: Sử dụng lớp UserManager từ thư viện oidc-client để quản trị các giao thức OpenID Connect.

Hình 4.16: Method đăng nhập và đăng xuất .

Hình 4.17: Nhấn vào nút đăng nhập để đăng nhập cho người dùng.

Hình 4.18: Nhấn vào nút đăng xuất để đăng xuất khỏi ứng dụng.

Hình 4.19: Bảo vệ API bằng Identity Server.

Hình 4.20: Trình xử lý mã token xác thực.

Hình 4.21: Trình xử lý mã token ủy quyền.

Hình 4.22: Xác định phạm vi API được bảo vệ.

Hình 4.23: Xác định danh tích xác thực và ủy quyền.

Hình 4.24: Quản lý danh sách thông tin người dùng.

Bảng 4.25: Thêm mới người dùng.

Bảng 4.26: Chính sửa thông tin chi tiết và xóa tài khoản người dùng.

Bảng 4.27: Quản lý thông tin các nhóm phân quyền.

Bảng 4.28: Chính sửa thông tin chi tiết ủy quyền truy cập API.

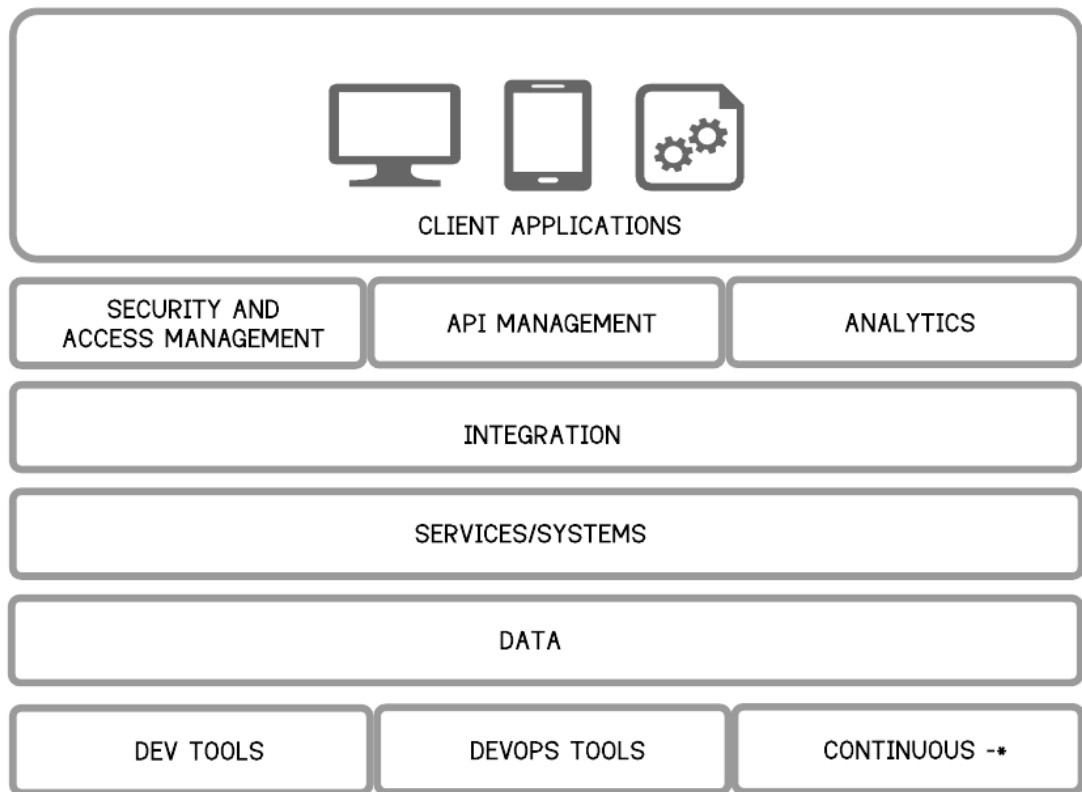
Bảng 4.29: Chính sửa thông tin chi tiết ủy quyền truy cập ứng dụng client.

Bảng 4.30: Trang thông tin cá nhân người dùng.

CHƯƠNG 1 – GIỚI THIỆU MICROSERVICES

1.1 Tổng quan

Trong thời đại mà việc cung cấp trải nghiệm kỹ thuật số cho người dùng là quan trọng hơn bao giờ hết, thành công trong kinh doanh nằm ở việc cung cấp các dịch vụ kỹ thuật số (website, mobile app, v.v.) nhanh với sự hài lòng cao của khách hàng. Vì thế, cần có sự liên kết giữa các chiến lược tổng thể của công ty và các sáng kiến phát triển kỹ thuật số để chuyển đổi kiến trúc kinh doanh cốt lõi sang kiến trúc kỹ thuật số. Kiến trúc kỹ thuật số là kiến trúc thúc đẩy tích hợp nhanh chóng các công nghệ mới để thúc đẩy chuyển đổi. Nói một cách chi tiết, kiến trúc kỹ thuật số bao gồm tất cả những gì hỗ trợ phát triển chức năng trong kinh doanh. Nền tảng của kỹ thuật số là tất cả các công nghệ, ứng dụng, quản lý API, bảo mật, phân tích, tích hợp, dịch vụ, cơ sở dữ liệu cũng như cơ sở hạ tầng.



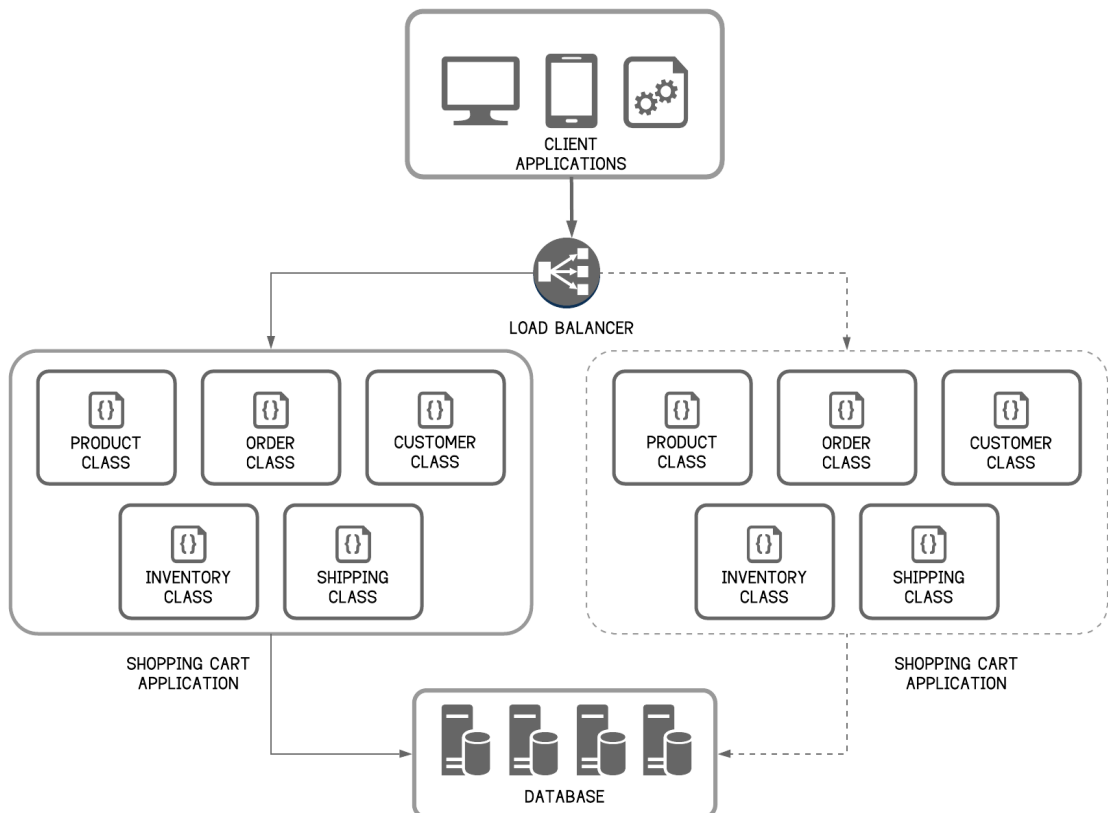
Hình 1.1 Kiến trúc kỹ thuật số điển hình

1.2 Kiến trúc nguyên khối (Monolithic architecture)

Trong một ứng dụng nguyên khối cổ điển, mọi thứ được kết hợp chặt chẽ và được triển khai thành một khối lớn.

Các ứng dụng phần mềm doanh nghiệp được thiết kế để hỗ trợ nhiều yêu cầu kinh doanh. Do đó, một ứng dụng phần mềm nhất định cung cấp hàng trăm khả năng kinh doanh và tất cả các chức năng như vậy thường được xếp thành một ứng dụng nguyên khối duy nhất. Phần mềm ERP (Enterprise Resource Planning), phần mềm CRM (Customer Relationship Management) và các hệ thống phần mềm khác là những ví dụ điển hình – chúng được xây dựng như một khối nguyên khối với hàng trăm khả năng kinh doanh.

Vì thế, việc triển khai, khắc phục sự cố, mở rộng và nâng cấp các ứng dụng phần mềm thật sự khó khăn.



Hình 1.2 Kiến trúc nguyên khối.

Kiến trúc hướng dịch vụ (Service Oriented Architecture - SOA) được thiết kế để khắc phục các vấn đề phát sinh từ các ứng dụng có kiến trúc nguyên khối bằng cách đưa ra khái niệm “Service”. Do đó, với SOA một ứng dụng phần mềm được thiết kế như một tổ hợp các service. Khái niệm SOA không yêu cầu triển khai các service thành một khối nhưng cách triển khai phổ biến nhất của nó: các dịch vụ web – web services thúc đẩy một ứng dụng phần mềm được triển khai dưới dạng một khối. Tương tự như các ứng dụng phần mềm có kiến trúc nguyên khối, các service sẽ phát triển theo thời gian bằng cách tích lũy các khả năng khác nhau. Sự tăng trưởng này sớm biến những ứng dụng đó thành những quả cầu nguyên khối, không khác gì những ứng dụng có kiến trúc nguyên khối thông thường.

Hình 1.2 cho thấy một phần mềm bán lẻ bao gồm nhiều dịch vụ. Tất cả các dịch vụ này được triển khai vào cùng một khối và có cùng một thời gian chạy. Do đó, nó cho thấy một số đặc điểm của ứng dụng nguyên khối:

- Phức tạp và được thiết kế, phát triển, triển khai như một đơn vị duy nhất. Dẫn đến sự khó khăn để thực hiện một số phương pháp phát triển nhanh và phân phối.
- Cập nhật, sửa chữa một phần của ứng dụng bắt buộc triển khai lại toàn bộ hệ thống.
- Một service không ổn định có thể khiến toàn bộ ứng dụng sụp đổ và nói chung, thật khó khăn để đổi mới và áp dụng các công nghệ mới.
- Khó mở rộng quy mô ứng dụng phần mềm với các yêu cầu tài nguyên xung đột. Ví dụ: khi một service yêu cầu nhiều CPU hơn, trong khi dịch vụ kia lại yêu cầu nhiều bộ nhớ hơn.

Những đặc điểm này đã dẫn đến sự ra đời của kiến trúc Microservices.

1.3 Giới thiệu Microservices (Microservices Architecture)

Kiến trúc Microservices (MSA) là một phương pháp phát triển các ứng dụng phần mềm bằng cách xây dựng chúng dưới dạng các microservice (dịch vụ siêu nhỏ) và độc lập. MSA có thể được triển khai để tạo thành một kiến trúc kỹ thuật số linh hoạt hơn. Các nhóm nhỏ hơn có thể làm việc riêng trên các microservices và chia sẻ chúng với các nhà phát triển khác thông qua API để tái sử dụng lại. Trên thực tế, các

microservices này có thể được triển khai, mở rộng và cập nhật mà không ảnh hưởng đến phần còn lại của hệ thống. Khi kết hợp với Container (thường với Docker), MSA là lựa chọn lý tưởng cho các công ty phân phối và mở rộng quy mô nhanh, đồng thời cho phép hỗ trợ nhiều nền tảng và thiết bị. Khi áp dụng MSA, các microservice của nó có thể phân loại thành một số loại.

Đồng thời, hầu hết các tổ chức chưa hoàn toàn sẵn sàng để áp dụng MSA thuần túy. Mặc dù phương pháp tiếp cận MSA được tuân theo để phát triển mới hơn so với nhiều hệ thống thông thường. Chẳng hạn: hệ thống SOA, ESB (phần mềm tích hợp – Enterprise Service Bus), ...hay bất kỳ kiến trúc được triển khai theo chiều dọc nào khác của ứng dụng kế thừa sẽ vẫn là một phần của cấu trúc liên kết tổng thể trong ứng dụng phần mềm doanh nghiệp. Các hệ thống này sẽ không thể bị bỏ qua và sẽ được các service mô-đun truy cập (ít nhất là trong giai đoạn chuyển tiếp). Các hệ thống và service này sẽ được hiển thị thông qua các API và được quản lý truy cập nội bộ lẫn bên ngoài.

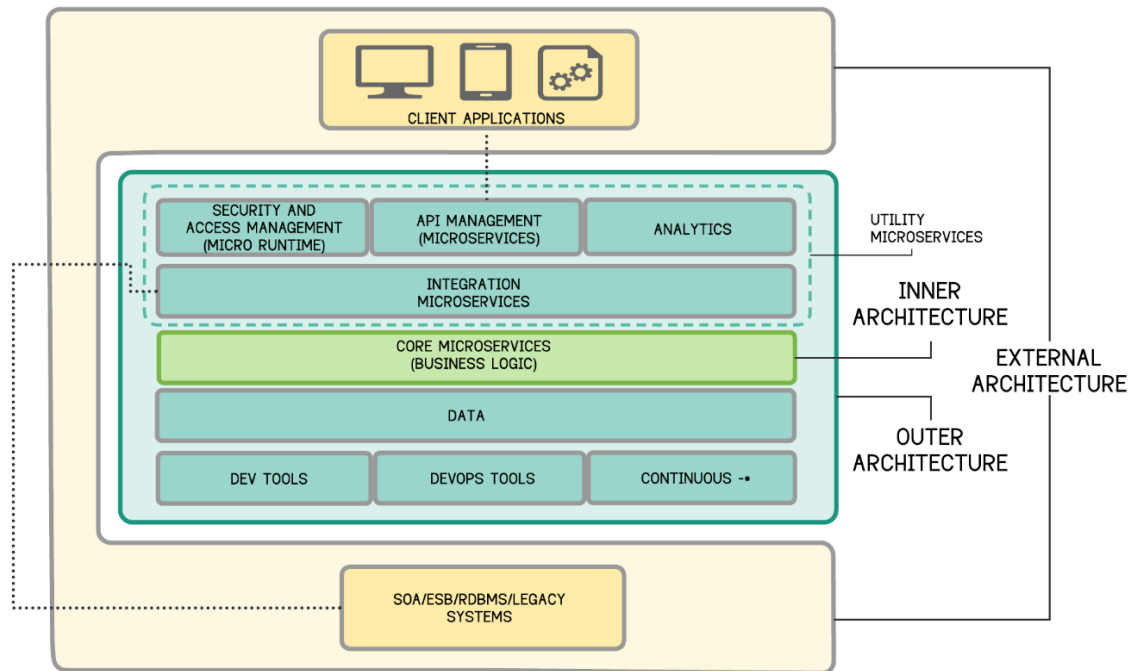
Bằng cách xem xét các thực tế ở trên, cái nhìn tổng thể của kiến trúc kỹ thuật số hỗ trợ MSA có thể được chia làm ba khu vực:

- Vùng 1: kiến trúc bên trong.
- Vùng 2: kiến trúc bên ngoài một phần.
- Vùng 3: kiến trúc bên ngoài toàn bộ

Bảng 1.1 Các vùng kiến trúc MSA.

Kiến trúc bên trong	Core Microservices – bao gồm tất cả microservice chứa các dịch vụ dữ liệu và logic nghiệp vụ.
Kiến trúc bên ngoài một phần	Tiện ích Microservices – Bao gồm các dịch vụ tích hợp, gateway, security token, v.v.
Kiến trúc bên ngoài toàn bộ	Ứng dụng người dùng cuối (web app, mobile app, v.v.)

Do đó, kiến trúc kỹ thuật số hỗ trợ MSA sẽ gần giống với kiến trúc được mô tả như hình 1.3:



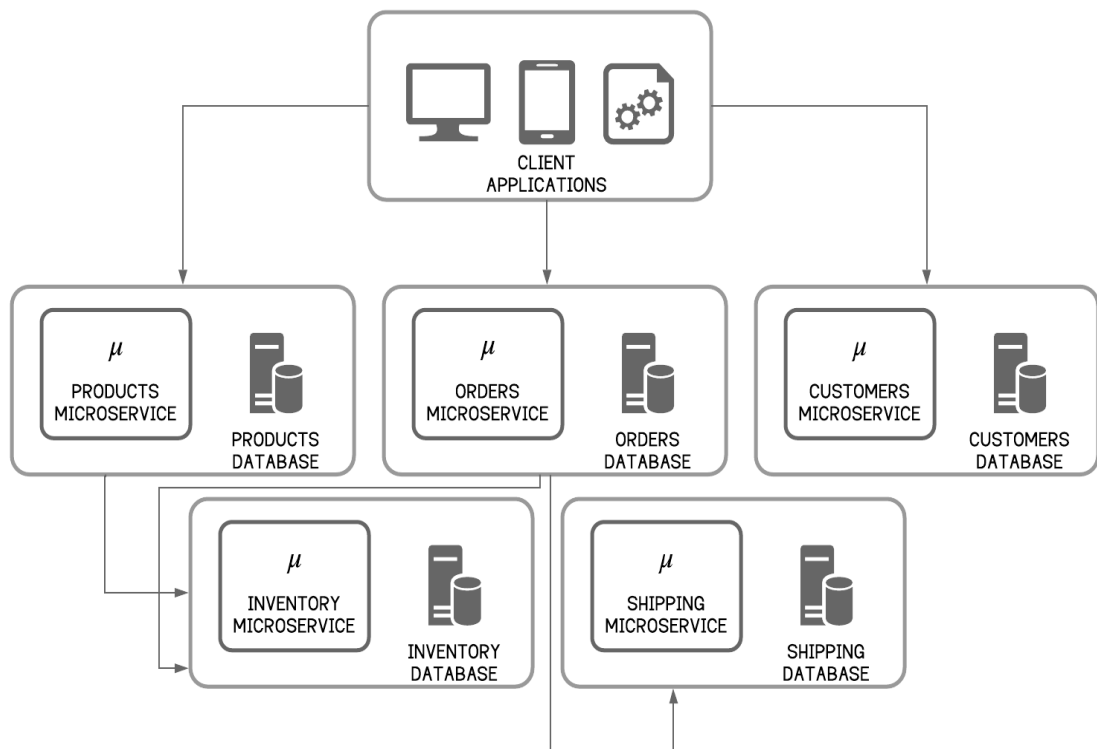
Hình 1.3 Kiến trúc kỹ thuật số hỗ trợ MSA.

CHƯƠNG 2 – KIẾN TRÚC MICROSERVICES HƯỚNG API

2.1 Đặc điểm chính

Kiến trúc Microservices (MSA) là một mẫu kiến trúc ứng dụng trong đó một ứng dụng lớn được chia nhỏ thành nhiều microservice được ghép nối lỏng lẻo. Mục tiêu của kiến trúc này là phân rã ứng dụng một cách đầy đủ để tạo điều kiện cho việc triển khai và phát triển ứng dụng được nhanh chóng.

Trái ngược với ứng dụng nguyên khối cổ điển, trong đó mọi thứ được kết hợp chặt chẽ và được triển khai thành một khối lớn, MSA cố gắng tách tất cả các dịch vụ chức năng thành các microservice có phạm vi duy nhất, được xác định rõ ràng và chạy trong quy trình riêng biệt của nó. Mỗi microservice chỉ nên sở hữu một trách nhiệm cho một chức năng duy nhất. Bằng cách cho phép các nhóm nhỏ tự chủ phát triển, triển khai và mở rộng quy mô một cách độc lập, kiến trúc này còn cho phép phát triển song song các microservice, do đó tăng tốc đáng kể chu kỳ sản xuất và triển khai.



Hình 2.1 Kiến trúc Microservices.

MSA là sự phát triển của kiến trúc hướng dịch vụ (SOA) và việc áp dụng MSA có tương quan chặt chẽ với việc sử dụng DevOps cũng như tích hợp và phân phối liên tục (CI/CD). SOA cổ điển thường được thực hiện bên trong các khối triển khai và hướng đến nền tảng nhiều hơn, trong khi các microservice của MSA có thể được triển khai độc lập và do đó cung cấp tính linh hoạt hơn ở mọi tình huống. Các microservice khai thác lợi thế của điện toán đám mây, đóng gói và quản lý động đã đưa khái niệm SOA lên một cấp độ mới, nơi cơ sở hạ tầng đám mây cho phép các microservice được triển khai và quản lý trên quy mô lớn. Với Docker và Kubernetes cung cấp các cách hiệu quả để phát triển, triển khai và quản lý các microservice, các nhà phát triển có thể tạo ra vô số dịch vụ cùng một lúc, giám sát các dịch vụ và cung cấp tài nguyên cho từng dịch vụ. Mỗi microservice có thể được triển khai, nâng cấp, mở rộng quy mô và khởi động lại độc lập với tất cả các thành phần còn lại trong ứng dụng phần mềm.

2.2 Kiến trúc Microservices: Lợi ích và hạn chế

2.2.1 Những lợi ích

- Các dịch vụ riêng lẻ nhanh hơn để phát triển, thử nghiệm và triển khai, đồng thời dễ hiểu, dễ bảo trì và dễ nâng cấp. Chúng cũng có thể được tái sử dụng.
- Mỗi dịch vụ được phát triển độc lập bởi một nhóm và chỉ tập trung vào dịch vụ đó và được tự lựa chọn bất kỳ công nghệ nào phù hợp. Các nhà phát triển hiếm khi cần phối hợp triển khai các thay đổi cục bộ cho dịch vụ của họ. Những loại thay đổi này có thể được triển khai ngay sau khi chúng được thử nghiệm.
- MSA cho phép phân phối và triển khai liên tục các ứng dụng lớn, phức tạp.
- MSA cho phép mỗi dịch vụ được mở rộng một cách độc lập. Các nhà phát triển chỉ cần triển khai số lượng phiên bản của mỗi dịch vụ đáp ứng các hạn chế về khả năng và tính khả dụng của nó. Hơn nữa, họ có thể sử dụng cơ sở hạ tầng triển khai đã được tối ưu hóa phù hợp nhất với yêu cầu tài nguyên của từng microservice.

- MSA tạo ra sự chắc chắn của hệ thống. Ví dụ, nếu có một microservice nào đó bị hỏng thì các microservice còn lại vẫn tiếp tục hoạt động. Trong khi đó, một thành phần hoạt động trong ứng dụng phần mềm có kiến trúc nguyên khối có thể làm hỏng toàn bộ hệ thống.

2.2.2 Hạn chế

Giống như mọi công nghệ khác, kiến trúc Microservices không phải hoàn hảo về mọi mặt mà còn có những nhược điểm:

- Việc phân tích dữ liệu và xử lý chúng thành các microservice độc lập có thể làm tăng độ phức tạp của thiết kế
 - Một ứng dụng khách (web app, mobile app, v.v.) phải biết một số điểm cuối để thực hiện chức năng và mỗi microservice phải biết về ít nhất một hoặc hai microservice khác, ví dụ như cách tương tác, địa chỉ IP, v.v.
 - Các nhà phát triển phân biệt giữa các dịch vụ bên ngoài và bên trong. Để từ đó thực hiện cơ chế giao tiếp giữa chúng.
 - Các nhà phát triển cũng phải viết mã để xử lý lỗi kết nối chậm hoặc không khả dụng. Họ cũng cần xử lý các giao tiếp phân tán để triển khai các trường hợp sử dụng trải dài nhiều dịch vụ.
- Các ứng dụng phía client (web app, mobile app, v.v.) sẽ gặp khó khăn trong việc tích hợp các microservice do sự khác nhau về giao diện và giao thức bảo mật do các nhóm khác nhau triển khai độc lập chúng.
- MSA sử dụng rất nhiều giao tiếp thông qua mạng và do đó, ứng dụng phần mềm một khối với cùng một phần cứng thường hoạt động tốt hơn.
- Công cụ cho nhà phát triển (Integrated Development Environment - IDE) được định hướng vào việc xây dựng các ứng dụng nguyên khối và không cung cấp công cụ hỗ trợ rõ ràng cho việc phát triển các ứng dụng phần mềm dựa trên kiến trúc Microservices.

2.3 Áp dụng MSA

Bất chấp sự phức tạp của kiến trúc Microservices, chúng thường là một lựa chọn tốt để phát triển ứng dụng nhanh chóng theo định hướng của người dùng và chúng góp phần xác định ranh giới chức năng rõ ràng và các nhóm phát triển. Họ tạo ra sự phát triển bền vững, do đó thúc đẩy sự nhanh nhẹn. Điều này có nghĩa là hợp tác phát triển, vận chuyển và gắn kết thường xuyên với người dùng. Mỗi khu vực chức năng trong một ứng dụng được thực hiện bởi một microservice của riêng nó. Các nhà phát triển có thể bắt đầu bằng cách phân tách ứng dụng thành microservice theo khả năng kinh doanh. Tốt nhất, mỗi microservice nên có một chức năng duy nhất hoặc một nhóm chức năng nhỏ.

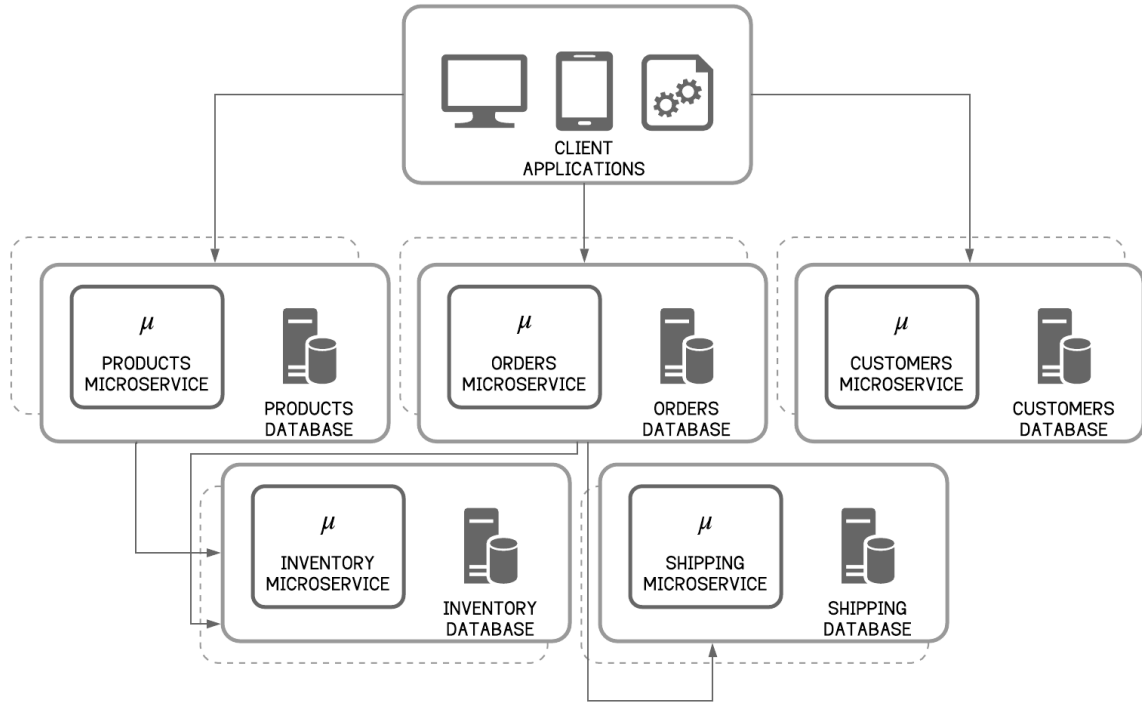
Đối với giao tiếp giữa các microservice, các microservice có thể sử dụng các cơ chế giao tiếp dựa trên yêu cầu/phản hồi (request/response) đồng bộ như REST dựa trên HTTP, Thrift hoặc gRPC. Ngoài ra, có thể sử dụng cơ chế giao tiếp dựa trên thông điệp không đồng bộ như AMQP, MQTT hoặc STOMP. Có thể chọn định dạng thông báo như JSON hoặc XML mà con người có thể đọc được hoặc định dạng nhị phân như Avro. Để đảm bảo các kết nối lỏng lẻo, mỗi microservice phải có cơ sở dữ liệu riêng của nó chứ không phải chia sẻ một cơ sở dữ liệu duy nhất với các microservice khác.

MSA có thể được triển khai với bất kỳ ngôn ngữ lập trình nào hỗ trợ các service dựa trên REST và RPC. Trong bài báo cáo này sẽ đặc biệt tập trung vào ASP.NET Core. ASP.NET Core là một dự án mã nguồn mở và thư viện đa nền tảng (cross-platform) cho việc xây dựng những ứng dụng dựa trên kết nối đám mây, web app, IoT và backend.

Sau khi các microservice được xác định và triển khai, chúng phải được đóng gói. Docker là một công nghệ được ưa thích để đóng gói các microservice và là công nghệ hỗ trợ chính cho MSA. Triển khai nhanh nhưng không thay đổi bên trong, sử dụng tài nguyên tối ưu một cách tối đa là một số lợi thế của việc sử dụng Docker. Các Docker Container nên được sắp xếp động bằng cách sử dụng trình quản lý cụm trên đám mây, ví dụ như Kubernetes. Các nhà phát triển có thể triển khai và mở rộng một

cách độc lập từng microservice, điều phối việc triển khai thông qua Kubernetes và cộng tác giữa các nhóm phát triển bao gồm các công cụ để quản lý.

2.4 Giao tiếp từ ứng dụng phía client đến microservice



Hình 2.2 Một ví dụ MSA.

Kiến trúc Microservices rất chi tiết, có nghĩa là ứng dụng phía client không cần tương tác nhiều với các microservice. Trong ứng dụng bán lẻ hình 2.2, ứng dụng client trực tiếp sử dụng một số microservice. Nếu ứng dụng phía client muốn liệt kê các sản phẩm có sẵn, nó sẽ gọi Products Microservice và nếu muốn đặt hàng, nó sẽ gọi Orders Microservice. Khi một đơn đặt hàng được đặt, chức năng của Orders Microservice sẽ phải kiểm tra tình trạng còn hàng của sản phẩm từ Inventory Microservice kiểm kê và tiếp đến giao tiếp với Shipping Microservice để bắt đầu quá trình vận chuyển. Về lý thuyết, khách hàng có thể yêu cầu trực tiếp đến từng microservice vì mỗi microservice đều có một điểm cuối truy cập công khai. Tuy nhiên cách tiếp cận này sẽ tạo ra một số vấn đề như sau:

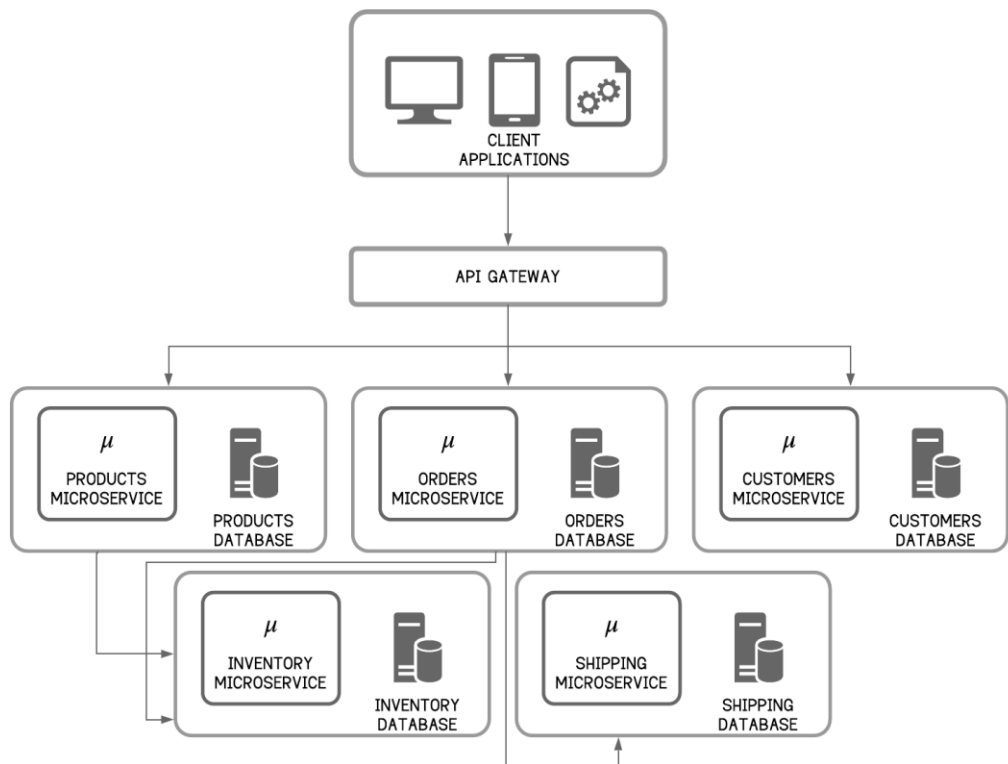
- Có quá nhiều điểm truy cập cuối được đưa ra mà SOA đã cố gắng hạn chế bằng việc sử dụng ESB.

- Nếu có sự không phù hợp giữa nhu cầu khách hàng và các API, client sẽ phải thực hiện một số giao tiếp để yêu cầu những gì nó cần. Trong một ứng dụng phức tạp, điều này có nghĩa là sẽ có hàng trăm giao tiếp qua mạng.
- Việc cấu trúc lại các microservice sẽ là một thách thức. Theo thời gian, microservice sẽ phải được cấu trúc lại.
- Việc thực hiện xác thực, ủy quyền, điều chỉnh, giám sát cho từng microservice sẽ tạo ra sự trùng lặp và không cần thiết.

Vì những vấn đề này, hiếm khi nào các ứng dụng client giao tiếp trực tiếp với microservice. Khi thiết kế và xây dựng các ứng dụng dựa trên kiến trúc Microservices lớn hoặc phức tạp, một cách tiếp cận tốt cần xem xét đó là API Gateway.

2.4.1 API Gateway

Việc triển khai API Gateway làm điểm duy nhất cho tất cả các ứng dụng client sẽ giải quyết các vấn đề gặp phải trong giao tiếp trực tiếp từ ứng dụng client đến các microservice.



Hình 2.3 Kiến trúc Microservice với API Gateway.

Với API Gateway, trước tiên tất cả các yêu cầu từ ứng dụng client đều đi qua các API được xác định làm cổng truy cập (còn gọi là API Gateway) thông qua REST/HTTP. Các API này định tuyến các yêu cầu đến microservice. Cổng kết nối chịu trách nhiệm chính về định tuyến các yêu cầu, thành phần và giao thức. Các cổng API thường sẽ xử lý một yêu cầu duy nhất bằng cách gọi nhiều microservice và tổng hợp kết quả.

Về cơ bản, một cổng API cung cấp cho nhà phát triển dễ dàng triển khai các microservice dưới dạng API được quản lý và thường là một phần của dịch vụ quản lý API. Nền tảng quản lý API phải cung cấp các cách để tích hợp và quản lý các nhà phát triển, tạo danh mục API và tài liệu, báo cáo sử dụng kèm theo, thực thi các biện pháp điều chỉnh, biện pháp phòng ngừa về độ tin cậy và bảo mật. Nền tảng này cung cấp các công cụ, khả năng kiểm soát và khả năng hiển thị để mở rộng quy mô microservice thông qua API cho các nhà phát triển mới và kết nối chúng với các hệ thống khác.

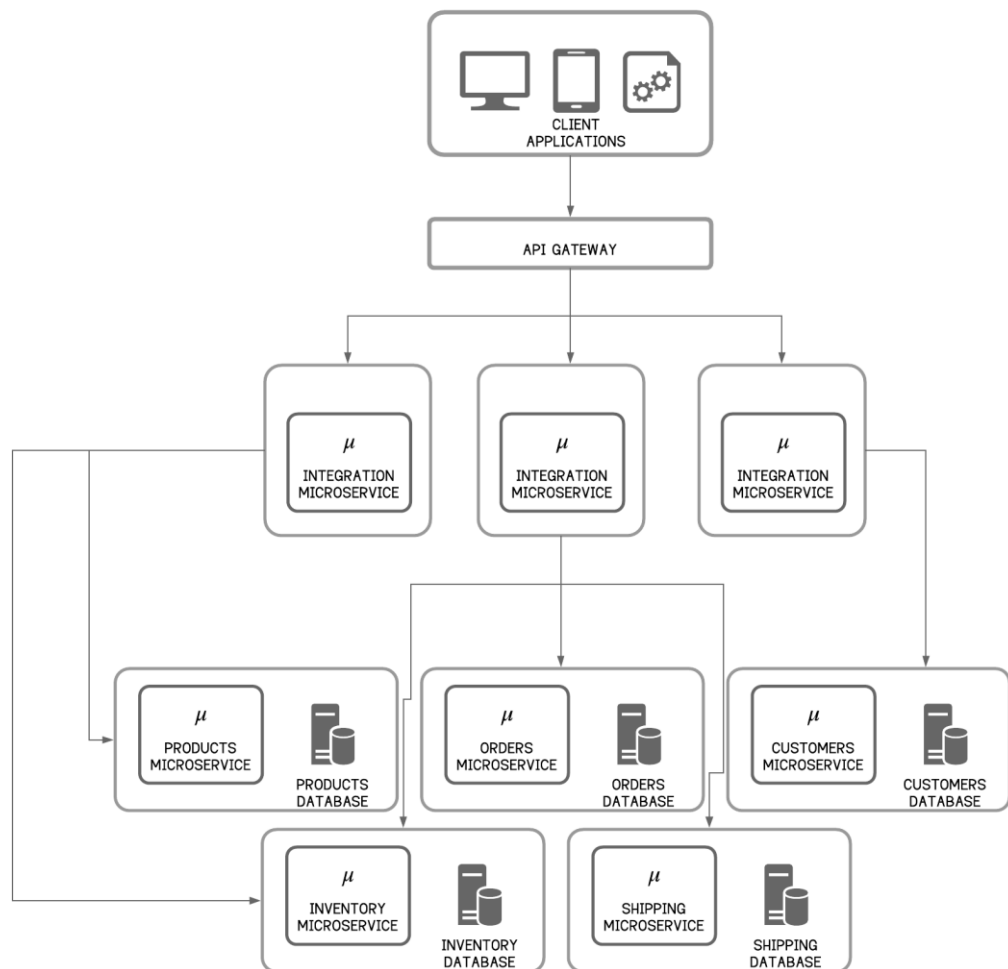
2.4.2 Tích hợp Microservices

Việc triển khai các microservice và để chúng hiển thị thông qua một lớp API Gateway dường như là một lựa chọn phù hợp khi triển khai một ứng dụng phần mềm MSA. Mặc dù sự phổ biến của các microservice càng ngày càng tăng, nhưng điều này không có nghĩa là các ứng dụng nguyên khối đã lỗi thời. Có những ứng dụng phần mềm vẫn phù hợp với kiến trúc nguyên khối và do đó nó vẫn còn tồn tại. Vì vậy, trong hầu hết các kịch bản kinh doanh ngày càng phát triển, đòi hỏi tích hợp nhiều kiến trúc phần mềm khác. Các khả năng của ESB trong SOA, chẳng hạn như định tuyến, thông báo, kết hợp, giao thức và chuyển đổi định dạng thông báo.

Trong một SOA điển hình, các dịch vụ tổng hợp được kết hợp với ESB để tích hợp nhiều dịch vụ và hệ thống khác. ESB được sử dụng làm nơi tập trung tất cả các kết nối của các dịch vụ và hệ thống. Tuy nhiên, đây là một mô hình chống lại MSA bởi vì bất kỳ hệ thống nào đòi hỏi nhiều quản lý trung tâm đều có thể trở thành vấn đề như bất kỳ hệ thống nguyên khối nào. Do đó, ESB không phù hợp với MSA.

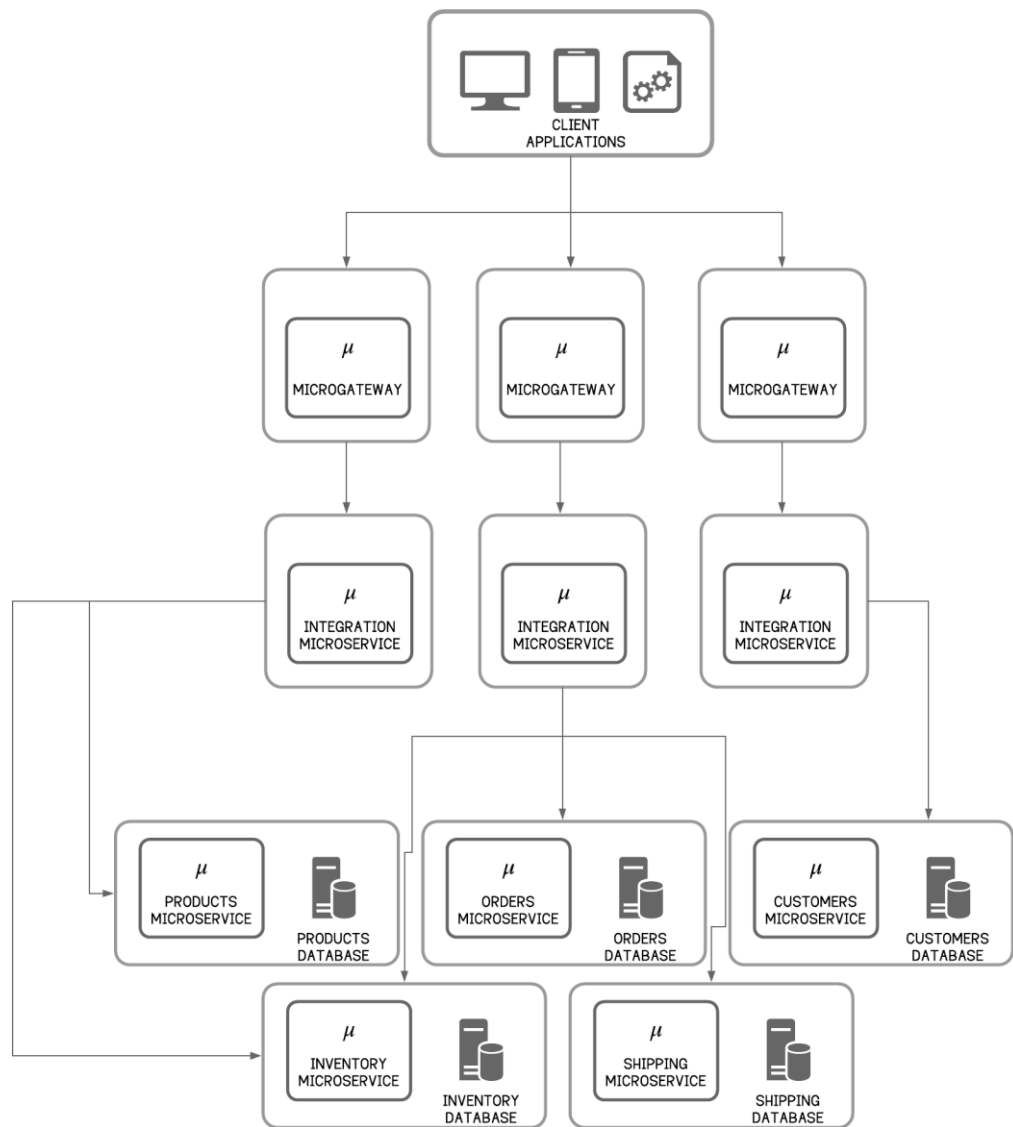
Là một giải pháp thay thế cho ESB, việc triển khai các dịch vụ tích hợp là lựa chọn tốt nhất theo quan điểm của MSA. Theo cách tiếp cận này, các dịch vụ tích hợp sẽ được triển khai dưới dạng lớp microservice thứ 2. Các dịch vụ này thường phải hỗ trợ chức năng ESB như định tuyến, biến đổi, điều phối, khả năng phục hồi, v.v. Nếu lấy ví dụ bán lẻ, có thể xây dựng một microservice tích hợp để xử lý các đơn đặt hàng và vận chuyển như hình 2.4. Trái ngược với cách tiếp cận ESB nguyên khối, tất cả các chức năng ESB được tách biệt và phân tán.

Như hình 2.4, cả logic nghiệp vụ và logic tích hợp đều có thể được viết dưới dạng microservice và sẽ nằm trong một vùng chạy riêng biệt. Các dịch vụ tích hợp cũng như các microservice có thể được giao tiếp với các ứng dụng khách thông qua API Gateway.



Hình 2.4 Tích hợp trong MSA.

2.4.3 Decentralized Gateway



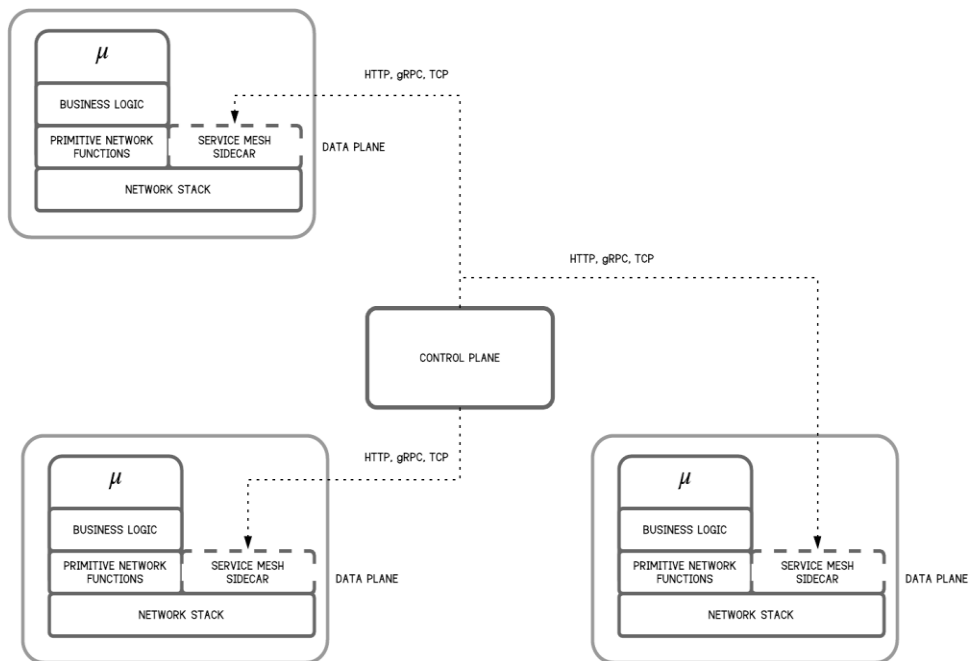
Hình 2.5 Micro-Gateway trong MSA.

Các cổng phân cấp (Decentralized Gateway) cực kỳ hữu ích khi nói đến việc mở rộng quy mô. Ví dụ: Giả sử một tình huống khi yêu cầu nhiều phiên bản microservice của sản phẩm nhưng chỉ một hoặc một vài phiên bản microservice là cần thiết. Đây là một kịch bản thực tế trong đó nhiều yêu cầu sẽ được nhận bởi một nhóm microservice cụ thể. Hãy tưởng tượng một kiến trúc nguyên khối có sử dụng API Gateway duy nhất trong đó tất cả các API nằm trên đó và các service nằm phía sau cổng đó. Để mở rộng quy mô microservice, API Gateway cũng sẽ phải

được mở rộng. Mở rộng một phần của hệ thống dẫn đến việc mở rộng toàn bộ việc triển khai API Gateway. Để tránh tình trạng xảy ra như vậy, một API Gateway được phân cấp là lựa chọn tốt nhất khi có một microgateway cho mỗi microservice. Bất cứ khi nào microservice nào cần được mở rộng quy mô, chỉ microgateway nối tiếp với nó mới phải mở rộng quy mô theo.

2.4.4 API Gateway với Service Meshes

Một trong những câu hỏi thường gặp liên quan đến cổng API là nó khác với lưới dịch vụ như thế nào. Một lưới dịch vụ xử lý các tương tác từ microservice đến microservice và được sử dụng để định tuyến phức tạp. Nó có thể thực hiện các tác vụ như cân bằng tải, xác thực dịch vụ, khám phá dịch vụ, định tuyến và thực thi chính sách. Lưới dịch vụ cũng có thể cung cấp thông tin chi tiết về môi trường microservices, kiểm soát lưu lượng truy cập, triển khai bảo mật và thực thi các chính sách cho tất cả các dịch vụ trong lưới. Có hai thành phần logic tạo ra lưới dịch vụ: mặt phẳng dữ liệu và mặt phẳng điều khiển.



Hình 2.6 Control Plane và Data Plane trong lưới dịch vụ.

Mặc dù cổng API và lưới dịch vụ có các khả năng chồng chéo, sự khác biệt chính giữa chúng là cổng API là cơ bản trong việc hiển thị các dịch vụ vi mô dưới dạng các API được quản lý cho các bên bên ngoài (bên ngoài MSA) trong khi lưới

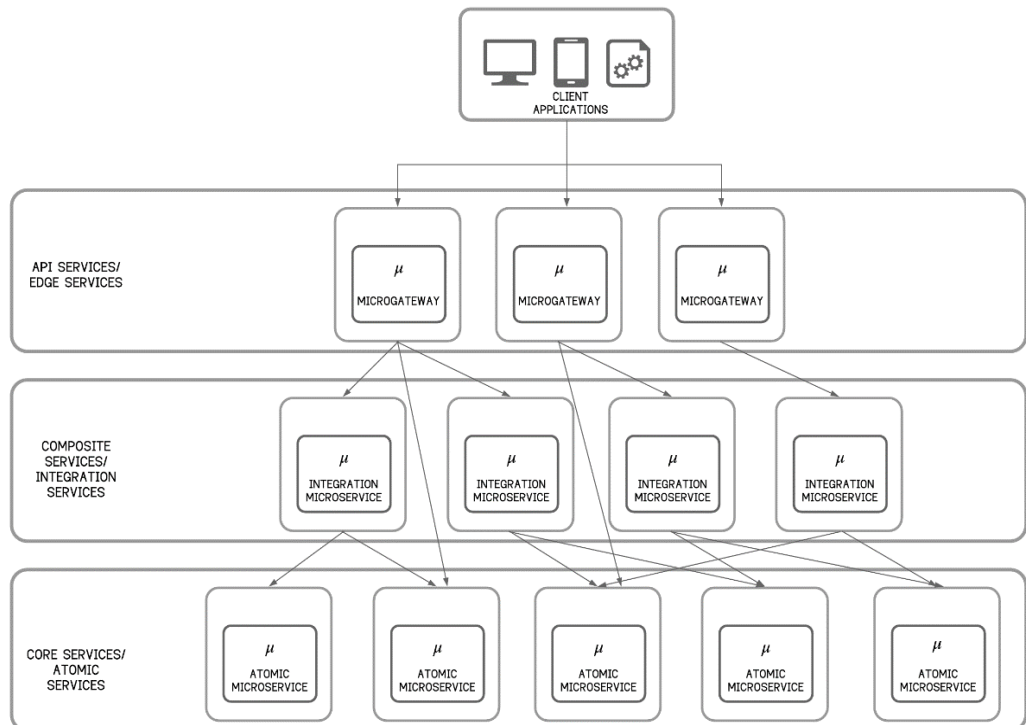
dịch vụ chỉ là giao tiếp giữa các dịch vụ cơ sở hạ tầng không có bất kỳ khái niệm kinh doanh nào về toàn bộ giải pháp. Các lưới dịch vụ ở dạng gốc của chúng có một lỗ hổng quản lý API cần được lấp đầy. Những điều này liên quan đến việc cung cấp dịch vụ cho người tiêu dùng bên ngoài (bảo mật nâng cao, khám phá, quản trị, v.v.), thông tin chi tiết về doanh nghiệp, thực thi chính sách và kiểm tiền.

Lưới dịch vụ và cổng API bổ sung để chúng tồn tại ở các cấp độ khác nhau và giải quyết các vấn đề khác nhau. Chúng có thể tồn tại độc lập nhưng cùng tồn tại bổ sung cho nhau.

2.5 Kiến trúc Microservices – Kiến trúc Tham Chiếu

2.5.1 Kiến trúc phân lớp

Hầu hết các doanh nghiệp tuân theo kiến trúc phân lớp với cả nguyên tắc SOA và khái niệm MSA bằng cách nhóm các service hoặc microservice thành các lớp trong kiến trúc tổng thể của doanh nghiệp. Cách tiếp cận này làm cho mỗi lớp kiến trúc trở thành một tập hợp các thành phần được chia sẻ tập trung một cách hợp lý.



Hình 2.7 Kiến trúc MSA phân lớp.

Trong kiến trúc MSA này, các khả năng được nhóm thành các lớp bằng cách tuân theo yêu cầu của hệ thống. Nó là một hệ thống tập trung, nơi mà dữ liệu di chuyển từ lớp này sang lớp khác. Với API Gateway cũng được phân lớp, một MSA thuần túy có thể được triển khai với các loại dịch vụ khác nhau có thể được phân loại thành một vài lớp khác nhau như thể hiện ở hình 2.6.

Các Core Services/ Atomic Service được đặt ở lớp dưới cùng. Chúng là các service khép kín và cực kỳ chi tiết (không có phụ thuộc vào bên ngoài) bao gồm logic nghiệp vụ với ít hoặc không có giao tiếp qua mạng.

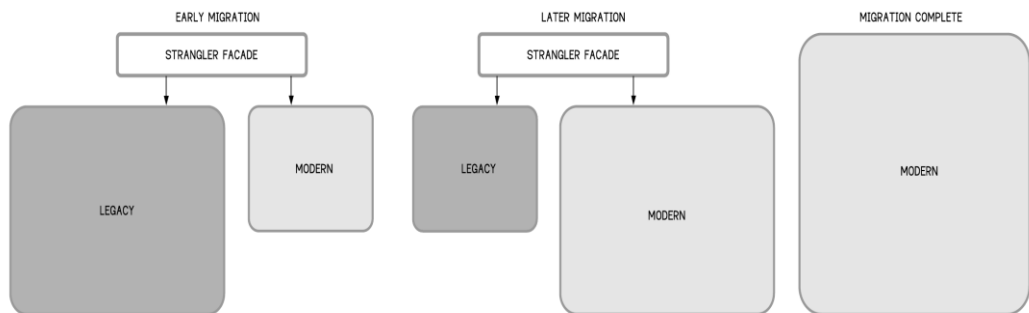
Các Composite/Integration microservices bao gồm các dịch vụ tích hợp và tổng hợp. Các dịch vụ này thường phải hỗ trợ một phần đáng kể chức năng của ESB như định tuyến, chuyển đổi, điều phối, khả năng phục hồi và các mẫu ổn định. Chúng có chức năng định tuyến, logic ánh xạ dữ liệu và logic truyền thông mạng xử lý giao tiếp giữa các dịch vụ thông qua các giao thức khác nhau. Các dịch vụ này cũng có thể là cầu nối giữa các hệ thống kế thừa và độc quyền khác, các API bên ngoài và cơ sở dữ liệu dùng chung.

API/Edge microservices là loại dịch vụ bên thứ ba và nằm trên cùng trong kiến trúc MSA phân lớp. Các dịch vụ này là một loại dịch vụ tổng hợp đặc biệt áp dụng các khả năng định tuyến cơ bản, bảo mật API, điều chỉnh, v.v.

2.5.2 Chuyển từ kiến trúc Monolithic sang kiến trúc Microservices

Một lợi thế của việc giới thiệu một Gateway để truy cập vào MSA là khả năng chuyển một ứng dụng phần mềm nguyên khối sang MSA một cách trơn tru và liên tục. Viết lại một ứng dụng nguyên khối lớn từ đầu là một nỗ lực lớn và có rất nhiều rủi ro đi kèm theo với nó. Điều này cũng ngăn người dùng sử dụng hệ thống mới cho đến khi hoàn tất. Sẽ có rất nhiều điều không chắc chắn liên quan cho đến khi hệ thống mới được phát triển và hoạt động như mong đợi. Do đó, sẽ có những cải tiến tối thiểu hoặc các tính năng mới được cung cấp trên nền tảng hiện tại, vì vậy doanh nghiệp sẽ phải chờ đợi để có bất kỳ tính năng nào được phát triển và phát hành.

Được gọi là Strangler Pattern, đây là một mẫu thiết kế để từng bước ứng dụng phần mềm nguyên khối thành MSA bằng cách thay thế một chức năng cụ thể bằng một microservice mới. Khi chức năng mới đã sẵn sàng, thành phần cũ bị loại bỏ, dịch vụ mới được đưa vào sử dụng và cuối cùng thành phần cũ ngừng hoạt động hoàn toàn. Tuy nhiên, khi loại bỏ dần, sẽ có rất nhiều thay đổi. Các thiết kế microservice sẽ thay đổi và làm điều này theo thời gian sẽ dẫn đến sự khó khăn trong việc triển khai. Để làm cho các ứng dụng client không có nhiều thay đổi thường xuyên như vậy, cổng API có thể được sử dụng thành Strangler Facade.

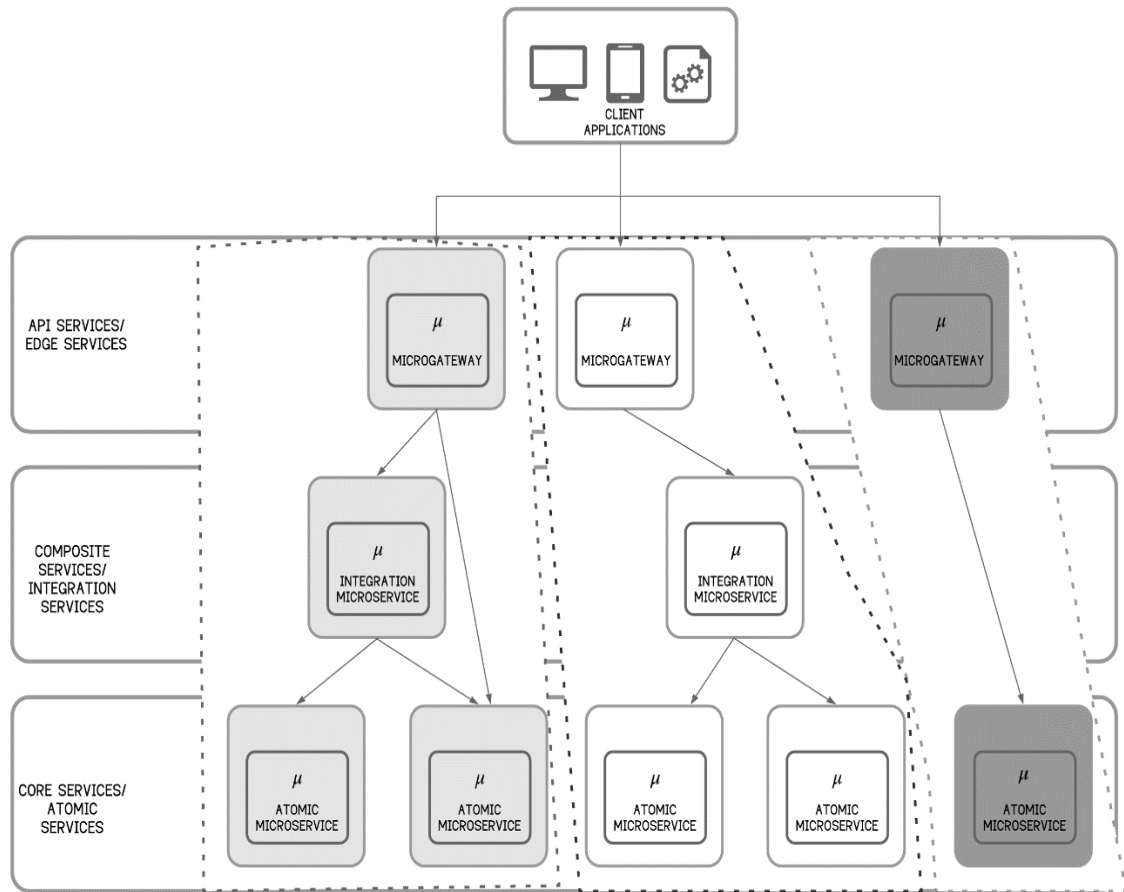


Hình 2.8 Kiến trúc MSA: Strangler Facade trong Strangler Pattern.

Khi sử dụng Strangler Facade, API Gateway sẽ hoạt động như một lớp proxy với các giao diện tĩnh hoặc được xác định rõ ràng tương ứng với các microservice. Những thay đổi tiếp theo trong microservice sẽ không làm ảnh hưởng đến các ứng dụng client vì chúng chỉ xử lý ở API Gateway.

2.6 Kiến trúc Microservices – Kiến trúc Phân Đoạn

Kiến trúc phân đoạn là một phân đoạn con của kiến trúc phân lớp, được tạo ra bằng cách chia kiến trúc phân lớp thành các phân đoạn nhỏ dựa trên các khả năng chức năng trong mỗi lớp kiến trúc. Nó cũng là một hệ thống tập trung, nơi luồng dữ liệu có thể di chuyển từ lớp này sang lớp khác. Mẫu này tách biệt các lớp của mỗi phân khu và mỗi phân khu chịu trách nhiệm về chức năng tổng thể của nó ở mỗi lớp.



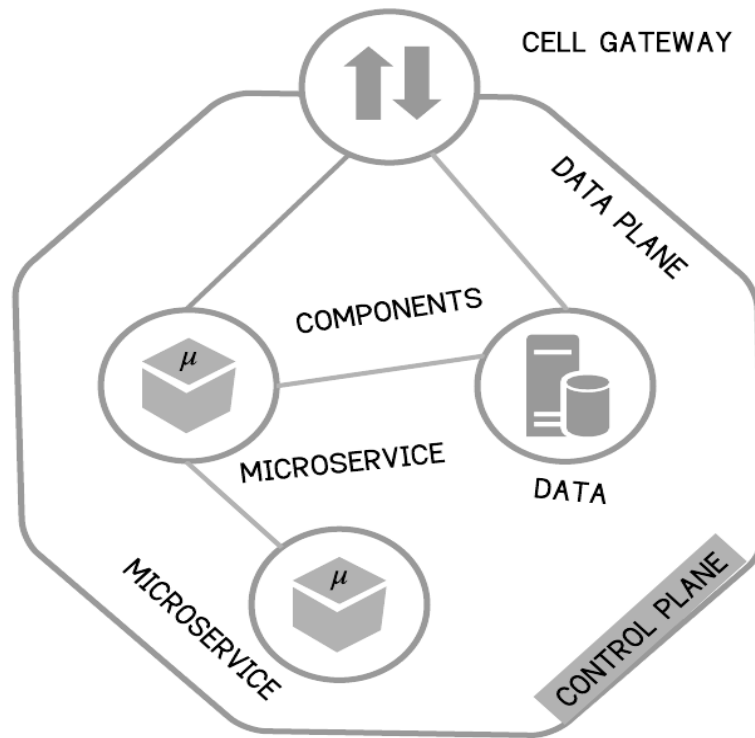
Hình 2.9 Kiến trúc MSA phân đoạn.

2.7 Kiến trúc Microservices – Kiến trúc Tế Bào

Một kiến trúc phân đoạn là cấp quá cao để thực thi một đơn vị kiến trúc độc lập, phi tập trung. Đồng thời, các microservice thường quá nhỏ để được coi như một đơn vị kiến trúc. Trong kiến trúc dựa trên cell, các khả năng chức năng được nhóm lại trong một đơn vị kiến trúc, được gọi là cell, dựa trên phạm vi sở hữu. Một cell có thể triển khai độc lập, có thể quản lý và có thể quan sát được. Các thành phần bên trong cell có thể giao tiếp được với nhau bằng cách sử dụng các phương thức giao tiếp được hỗ trợ. Tất cả các giao tiếp bên ngoài phải diễn ra thông qua API Gateway hoặc proxy, cung cấp API, sự kiện hoặc luồng thông qua điểm cuối mạng được quản lý bằng cách sử dụng các giao thức mạng tiêu chuẩn. Các đội có thể tự tổ chức để tạo ra các đơn vị kiến trúc được triển khai liên tục và cập nhật từng bước.

Gateway là điểm kiểm soát, cung cấp giao diện được xác định rõ ràng cho một tập hợp con trong các API, sự kiện và luồng. Trong mẫu này, Gateway trở thành điểm truy

cập (điểm cuối) duy nhất. Kiến trúc MSA dựa trên cell-based có thể hoạt động trên mô hình bảo mật cục bộ trong cell hoặc mở rộng sang mô hình bảo mật khác bằng cách kết nối ngoài ranh giới của cell.



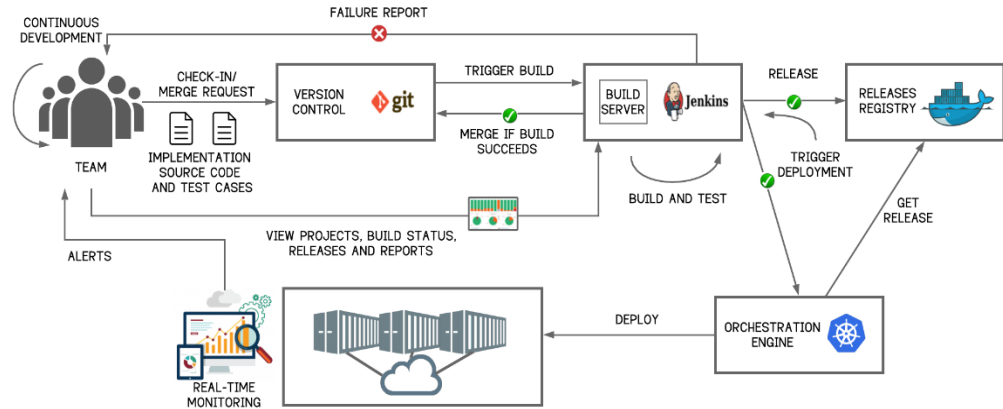
Hình 2.10 Kiến trúc MSA dựa trên Cell-Based.

2.8 Cài đặt

2.8.1 Tích hợp và triển khai liên tục

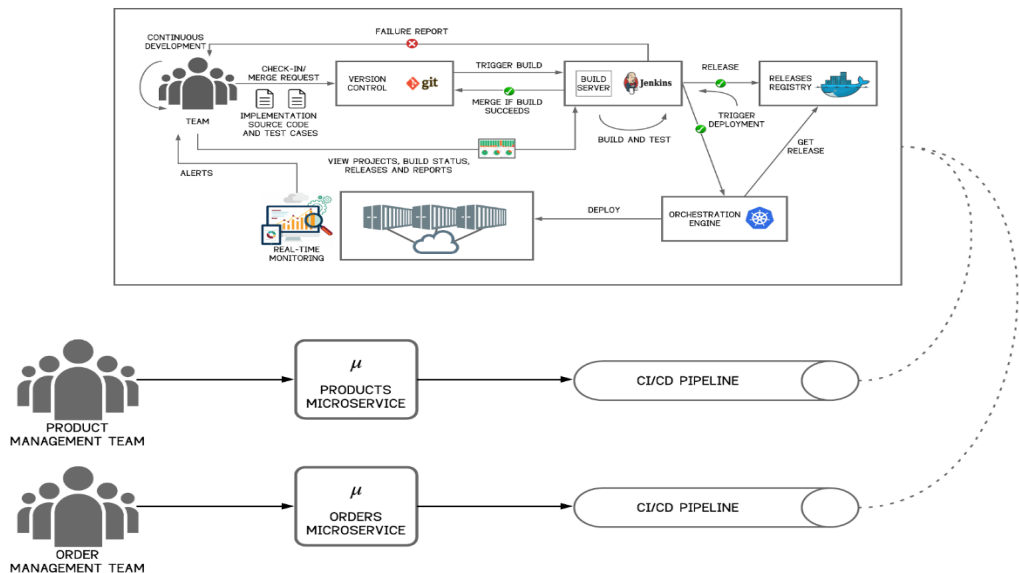
Để tạo ra trải nghiệm tuyệt vời, việc sử dụng “liên tục” không còn là điều “tốt nhất có thể có” vì tốc độ tiếp cận thị trường có thể phá vỡ một công ty hoặc giúp công ty tồn tại và phát triển trong tương lai. Ở nhiều doanh nghiệp hàng đầu, sự chuyển đổi từ một phát triển đơn lẻ sang phát triển song song và từ phát triển thác nước sang các quy trình tích hợp liên tục tự động/phân phối liên tục (CI/CD) tạo điều kiện lập cho phát triển ứng dụng phần mềm. Bản chất phân tán và độc lập

của MSA cho phép tích hợp và triển khai liên tục. Cơ sở hạ tầng cần thiết để nhóm tích hợp, kiểm tra và triển khai bất kỳ thay đổi nào phải hoàn toàn tự động. Điều này thúc đẩy trải nghiệm người dùng phản hồi cao.



Hình 2.11 Tích hợp và triển khai trong MSA 1.

Mỗi microservice sẽ có đường dẫn CI/CD riêng. Các kênh CI/CD cần được thiết lập để các thay đổi đối với mã nguồn tự động dẫn đến một vùng chứa mới được xây dựng, thử nghiệm và triển khai theo giai đoạn và cuối cùng được đưa vào sản xuất. Các nhóm tự quản sẽ sở hữu toàn bộ vòng đời của một microservice. Nhóm sẽ liên tục làm việc để nâng cao các tính năng của microservice và khắc phục sự cố gặp phải trong quá trình sản xuất.



Hình 2.12 Tích hợp và triển khai trong MSA 2.

- Các nhóm cam kết các thay đổi đối với kho lưu trữ kiểm soát phiên bản chẳng hạn như Git.
- Hệ thống CI/CD có thể được sử dụng để xây dựng các thay đổi và chạy các bài kiểm tra từng microservice mà thay đổi được cam kết.
- Nếu có bất kỳ lỗi kiểm tra nào, hệ thống CI/CD sẽ hợp nhất các thay đổi và chuẩn bị một bản phát hành mới cho microservice đó.
- Các ứng dụng phần mềm MSA được đóng gói dưới dạng Docker Container có thể dễ dàng triển khai bởi một công cụ điều phối như Kubernetes.
- Sau đó, hệ thống CI/CD xuất bản bản phát hành lên kho lưu trữ trung tâm và hướng dẫn công cụ điều phối chọn phiên bản mới nhất của microservice chứa các thay đổi mới nhất.
- Sau đó, công cụ điều phối lấy bản phát hành mới nhất từ kho lưu trữ là triển khai nó trong quá trình sử dụng.
- Tất cả các trường hợp trong quá trình sản xuất đều được giám sát bởi các công cụ tự động tạo ra cảnh báo cho nhóm phát triển nếu có bất kỳ vấn đề nào. Sau khi được cảnh báo, nhóm phát triển sẽ khắc phục sự cố và gửi yêu cầu thay đổi tới hệ thống kiểm soát phiên bản. Toàn bộ quy trình sau đó được lặp lại để đẩy những thay đổi vào sản xuất.

2.8.2 Các công nghệ hữu ích cho MSA

Bảng 2.1 Các công nghệ hữu ích cho MSA.

IDE	<ul style="list-style-type: none"> • Eclipse • IntelliJ • VSCode
Code Repository	<ul style="list-style-type: none"> • Github • Azure DevOps • Gitlab
Binary Repository	<ul style="list-style-type: none"> • Maven • JFrog

	<ul style="list-style-type: none"> • Ballerina Central
CI/CD	<ul style="list-style-type: none"> • Jenkins • Travis • Codefresh
Test Frameworks	<ul style="list-style-type: none"> • SOAP UI • JMeter • TestNG
Configuration Management Tools	<ul style="list-style-type: none"> • Puppet • Chef • Ansible
Container Orchestration	<ul style="list-style-type: none"> • Kubernetes • OpenShift • Mesosphere • DC/OS • Hashicorp • Nomad • Docker Swarm
Service Mesh	<ul style="list-style-type: none"> • Istio • Linkerd • Conduit • nginMesh
Observability	<p>Monitoring</p> <ul style="list-style-type: none"> • Application Metrics • Prometheus/Grafana • Splunk • AppDynamics <p>System Metrics</p> <ul style="list-style-type: none"> • Icinga

	<ul style="list-style-type: none"> • AWS Cloud Watch <p>Logging</p> <ul style="list-style-type: none"> • Fluentd • ELK Stack: Beats, Logstash, Elasticsearch, Kibana <p>Distributed Tracing</p> <ul style="list-style-type: none"> • Jaeger • Zipkin • AppDash • AppDynamics • WSO2 Analytics Server
--	--

2.9 MSA trong thực tế

2.9.1 Sử dụng Microservices vào thực tế

Kiến trúc Microservice đã phát triển từ kiến trúc doanh nghiệp tập trung, bao gồm các đặc điểm chính của nó và thảo luận những ưu và nhược điểm của việc sử dụng MSA. Tuy nhiên, cần có một bộ nguyên tắc vững chắc về thời điểm sử dụng kiến trúc Microservices và khi nào thì không nên dùng đến.

- Kiến trúc Microservices là kiến trúc vô cùng lý tưởng trong doanh nghiệp yêu cầu tính mô-đun.
- Nếu vấn đề kinh doanh mà công ty đang giải quyết bằng phần mềm ứng dụng khá đơn giản, có thể không cần đến MSA (có một ứng dụng web nguyên khối đơn giản và một cơ sở dữ liệu thường là đủ).
- Ứng dụng phần mềm phải được triển khai trên Container.
- Nếu hệ thống quá phức tạp để có thể tách biệt thành các microservice, nên xác định các chức năng có thể sử dụng microservices với tác động đến hệ thống tối thiểu. Sau đó, bắt đầu triển khai trường hợp sử dụng MSA và xây dựng hệ sinh thái cần thiết.
- Hiểu rõ khả năng kinh doanh là điều khá quan trọng để thiết kế Microservices. Hiểu được các kỹ thuật thiết kế Microservices.

2.9.2 Tóm tắt

Kiến trúc Microservices là sự lựa chọn lý tưởng cho việc xây dựng các ứng dụng phức tạp và đang phát triển. Chúng tạo ra sự phát triển bền vững, do đó thúc đẩy sự nhanh nhẹn và khả năng mở rộng. MSA cũng không dành cho tất cả mọi người – các điều kiện tiên quyết và tác động phá vỡ của nó phải được hiểu rõ trước khi bắt đầu. Ngoài ra, các kiến trúc nguyên khối cũ hơn sẽ vẫn là một phần của cấu trúc liên kết tổng thể của doanh nghiệp chủ yếu vì việc loại bỏ chúng hoàn toàn sẽ rất tốn kém. Một cách để quản lý tốc độ và tính linh hoạt mà MSA cung cấp giải pháp giải quyết sự phức tạp là sử dụng API. Xây dựng API một cách phù hợp giúp các microservice dễ dàng quản lý và cho phép chúng cùng tồn tại với các hệ thống cũ hiện có.

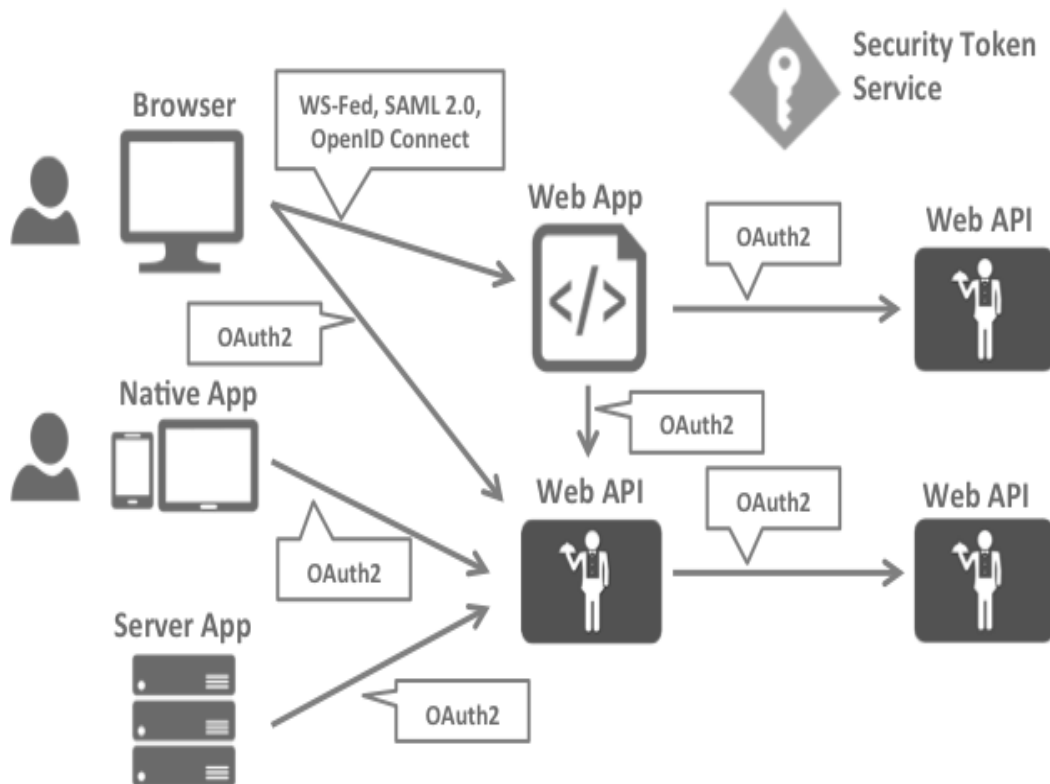
Đây là lý do tại sao việc triển khai MSA theo hướng API là rất quan trọng để làm cho các microservice trở nên hiệu quả đối với hầu hết các doanh nghiệp. API đã được phát triển, triển khai rộng rãi, tuân thủ các tiêu chuẩn như REST, giúp chúng thân thiện với nhà phát triển trong việc xây dựng và quản lý. MSA với API Gateway cho các microservice giải quyết tất cả các khía cạnh bằng quản lý API, cung cấp khả năng kiểm soát, khả năng hiển thị, v.v.

CHƯƠNG 3 – GIỚI THIỆU MÁY CHỦ NHẬN DẠNG (Identity Server)

3.1 Giới thiệu

Khi các ứng dụng, dịch vụ và việc áp dụng API của các doanh nghiệp, việc quản lý danh tính (chẳng hạn như nhân viên, nhà cung cấp, đối tác, khách hàng, v.v.) trên các ứng dụng phần mềm hay các dịch vụ nội bộ dùng chung trở thành một thách thức. Máy chủ nhận dạng (Identity Server) là một sản phẩm có thể đáp ứng nhiệm vụ này và cũng cung cấp tùy chọn để giải quyết vấn đề về nhận dạng trong tương lai.

Identity Server là một sản phẩm IAM (Identity and Access Management) mã nguồn mở hướng API được thiết kế giúp xây dựng các giải pháp CIAM (Customer Identity Access Management) hiệu quả. Nó dựa trên các tiêu chuẩn mở như OAuth, OpenID, OIDC. Identity Server được xây dựng qua các công cụ mã nguồn mở hoàn toàn ASPNET Core, Identity Server, Angular, v.v.



Hình 3.1 Tổng quát Identity Server.

Các tương tác phổ biến trong Identity Server:

- Trình duyệt web truy cập với các ứng dụng web.
- Các ứng dụng web giao tiếp với các API.
- Các ứng dụng dựa trên trình duyệt giao tiếp với các API.
- Các ứng dụng native (desktop app, mobile app, v.v.) giao tiếp với các API.
- Các ứng dụng dựa trên server giao tiếp với các API.
- Các API giao tiếp với nhau

Thông thường, mỗi lớp (frontend, middle, backend) phải bảo vệ tài nguyên và triển khai xác thực/ủy quyền. Các mối quan tâm về bảo mật được chia làm hai phần: xác thực và ủy quyền.

3.1.1 Xác thực

Xác thực là cần thiết khi một ứng dụng cần biết danh tính của người dùng hiện tại. Thông thường, các ứng dụng này quản lý dữ liệu thay mặt cho người dùng đó và cần đảm bảo rằng người dùng rằng người dùng này chỉ có thể truy cập vào dữ liệu mà họ được phép. Ví dụ phổ biến nhất cho điều đó là các ứng dụng web – nhưng các ứng dụng trên các nền tảng khác cũng có nhu cầu sử dụng.

Các giao thức xác thực phổ biến nhất là SAML2p, OpenID Connect, v.v. SAML2p là giao thức phổ biến nhất và được triển khai rộng rãi nhất.

OpenID Connect là phần mềm mới nhất trong ba phần mềm, nhưng được xem là có tiềm năng nhất cho các ứng dụng hiện tại. Nó được thiết kế để thân thiện với API.

3.1.2 Ủy quyền truy cập API

Các ứng dụng có hai cách cơ bản mà chúng giao tiếp với API – sử dụng danh tính ứng dụng hoặc ủy quyền danh tính của người dùng. Đôi khi cần kết hợp cả hai phương pháp.

OAuth2 là một giao thức cho phép các ứng dụng yêu cầu token từ dịch vụ cung cấp token bảo mật và sử dụng chúng để giao tiếp với các API. Việc ủy quyền

này làm giảm sự phức tạp trong cả ứng dụng khách cũng như các API vì xác thực và ủy quyền có thể được thực hiện tập trung.

3.1.3 Kết hợp OpenID Connect và OAuth 2.0

OpenID Connect và OAuth 2.0 rất giống nhau – trên thực tế OpenID Connect là một tiện ích mở rộng trên OAuth 2.0. Hai mối quan tâm bảo mật cơ bản, xác thực và truy cập API, được kết hợp thành một giao thức duy nhất.

Sự kết hợp giữa OpenID Connect và OAuth 2.0 là cách tiếp cận tốt nhất để bảo mật các ứng dụng hiện đại trong tương lai gần. Identity Server là một triển khai của hai giao thức này và được tối ưu hóa cao để giải quyết các vấn đề bảo mật điển hình của ứng dụng web, di động, v.v.

Identity Server là một nhà cung cấp OpenID Connect – nó triển khai các giao thức OpenID Connect và OAuth 2.0. Là một phần mềm phát hành token cho các ứng dụng client.

Identity Server có một số tính năng:

- Bảo vệ tài nguyên.
- Xác thực người dùng bằng cách sử dụng tài khoản cục bộ hoặc thông qua nhà cung cấp danh tính bên thứ ba.
- Đăng nhập một lần (Single Sign-On).
- Quản lý và xác thực cho ứng dụng client (web app, SPA, mobile app, v.v.).
- Quản lý thông tin người dùng.
- Cung cấp token nhận dạng và truy cập cho ứng dụng client.
- Xác thực token.

3.2 Identity Server

3.2.1 Đăng nhập một lần (Single Sign-On - SSO)

Đăng nhập một lần (SSO) là một trong những tính năng chính Identity Server cho phép người dùng cung cấp thông tin đăng nhập của họ một lần và có quyền truy cập vào nhiều ứng dụng. Người dùng không được nhắc nhập thông tin đăng nhập của họ khi truy cập từng ứng dụng cho đến khi phiên của họ bị chấm

dứt. Ngoài ra, người dùng có thể truy cập tất cả các ứng dụng mà không cần đăng nhập vào từng ứng dụng riêng lẻ. Vì vậy, nếu người dùng đăng nhập vào ứng dụng A, chẳng hạn, họ sẽ tự động có quyền truy cập vào ứng dụng B trong suốt thời gian của phiên đó mà không cần phải nhập lại thông tin đăng nhập.

Ngoài ra, nó chịu trách nhiệm xác thực người dùng. Kiểm tra thông tin đăng nhập của nó và phát hành mã token cho ứng dụng client. Xác thực là cần thiết khi một ứng dụng cần biết danh tính của người dùng hiện tại. Thông thường, các ứng dụng quản lý này quản lý dữ liệu thay mặt cho người dùng đó và cần đảm bảo rằng người dùng này chỉ có thể truy cập vào cơ sở dữ liệu mà họ được phép.



Hình 3.2 Single Sign-On.

Single Sign-On, là một thuộc tính của kiểm soát truy cập cho các hệ thống phần mềm độc lập có nhiều liên quan. Với thuộc tính này, người dùng có thể truy cập vào hệ thống cục bộ hoặc hệ thống kết nối bằng một tên người dùng và mật khẩu mà không cần sử dụng tên người dùng hoặc mật khẩu khác.

Trong một hệ thống Single Sign-On, có hai vai trò: Nhà cung cấp dịch vụ (Services Provider) và Nhà cung cấp danh tính (Identity Provider). Đặc điểm quan trọng của hệ thống này là mối quan hệ tin cậy được xác định trước giữa nhà cung

cấp dịch vụ và nhà cung cấp danh tính. Các nhà cung cấp dịch vụ tin tưởng các xác nhận do nhà cung cấp danh tính đưa ra và các tuyên bố xác thực.

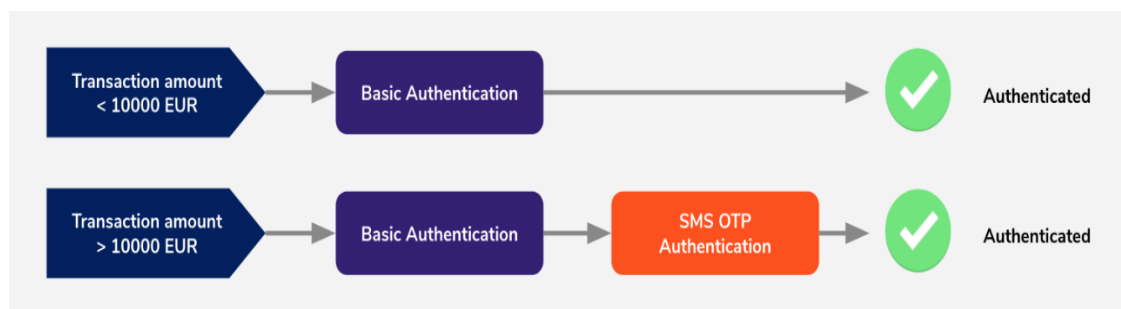
Một số lợi ích khi sử dụng Single Sign-On:

- Người dùng chỉ cần một cặp tên người dùng/mật khẩu duy nhất để truy cập nhiều dịch vụ. Do đó, người dùng không gặp vấn đề về việc ghi nhớ nhiều cặp tên người dùng/mật khẩu.
- Người dùng chỉ được xác thực một lần tại nhà cung cấp danh tính và sau đó họ được tự động đăng nhập vào tất cả các dịch vụ được đăng ký trước đó. Quá trình này thuận tiện hơn cho người dùng vì họ không phải cung cấp tên người dùng/mật khẩu.
- Danh tính người dùng được quản lý tại một điểm trung tâm. Điều này an toàn hơn, ít phức tạp hơn và dễ quản lý.

3.2.2 Xác thực (*Authentication*)

Xu hướng hiện tại yêu cầu sử dụng dịch vụ hàng trăm trang web trong một thế giới nhiều kết nối. Hầu hết các trang web này cần người dùng tạo tài khoản bằng cách cung cấp địa chỉ email và mật khẩu hợp lệ. Việc ghi nhớ tất cả các ID người dùng và mật khẩu khác nhau mà người dùng có thể khó khăn và phức tạp.

Để dễ dàng hơn, hầu hết các trang web hiện tại cung cấp cho người dùng tùy chọn đăng nhập bằng tài khoản Facebook, Google, v.v. Vì hầu hết người dùng internet đều có ít nhất một trong những tài khoản này, nên việc tạo tài khoản mới trở thành một hành động tức thì.

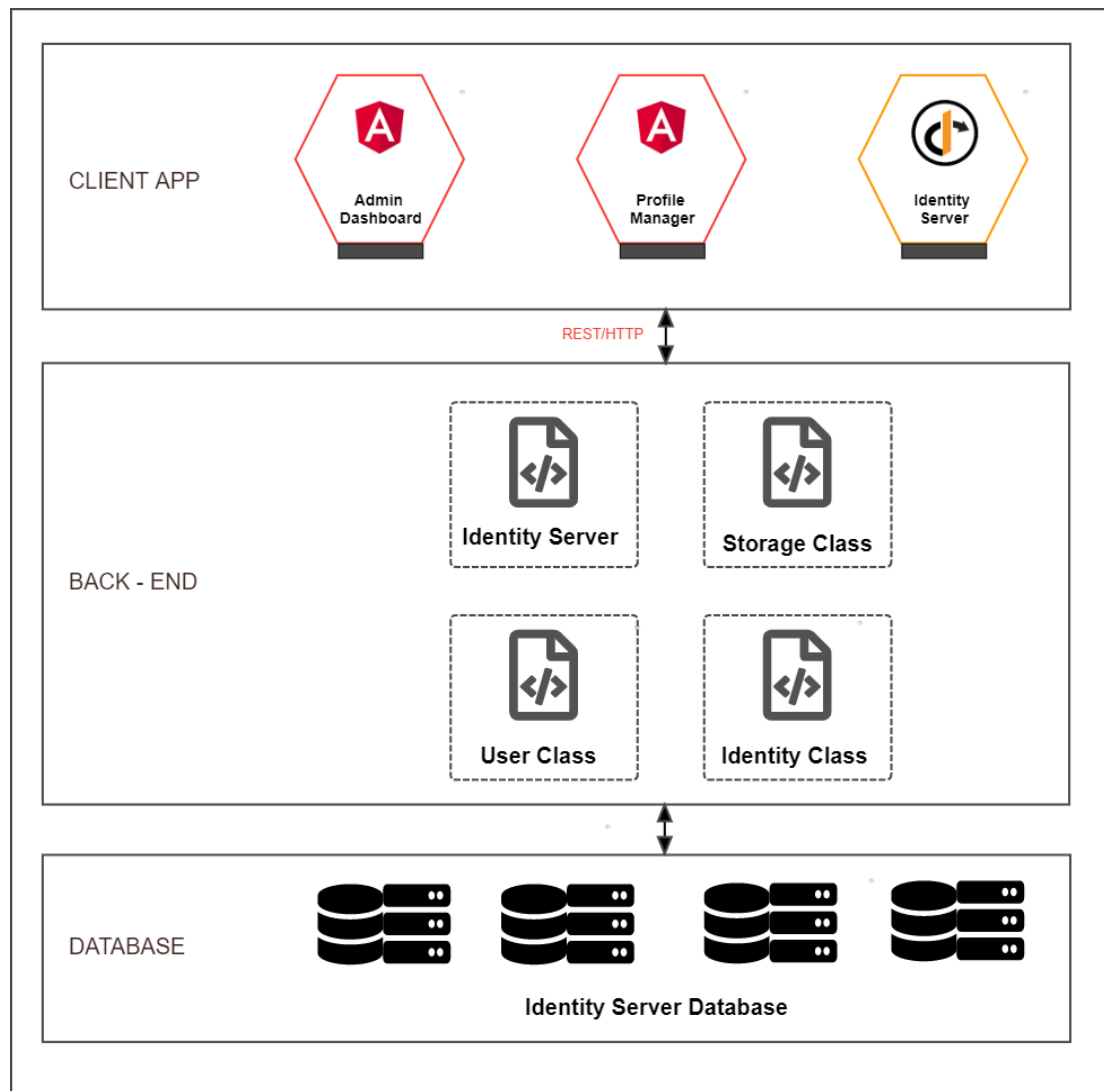


Hình 3.3 Xác thực.

3.2.3 Kiểm soát truy cập - ủy quyền (Authorization)

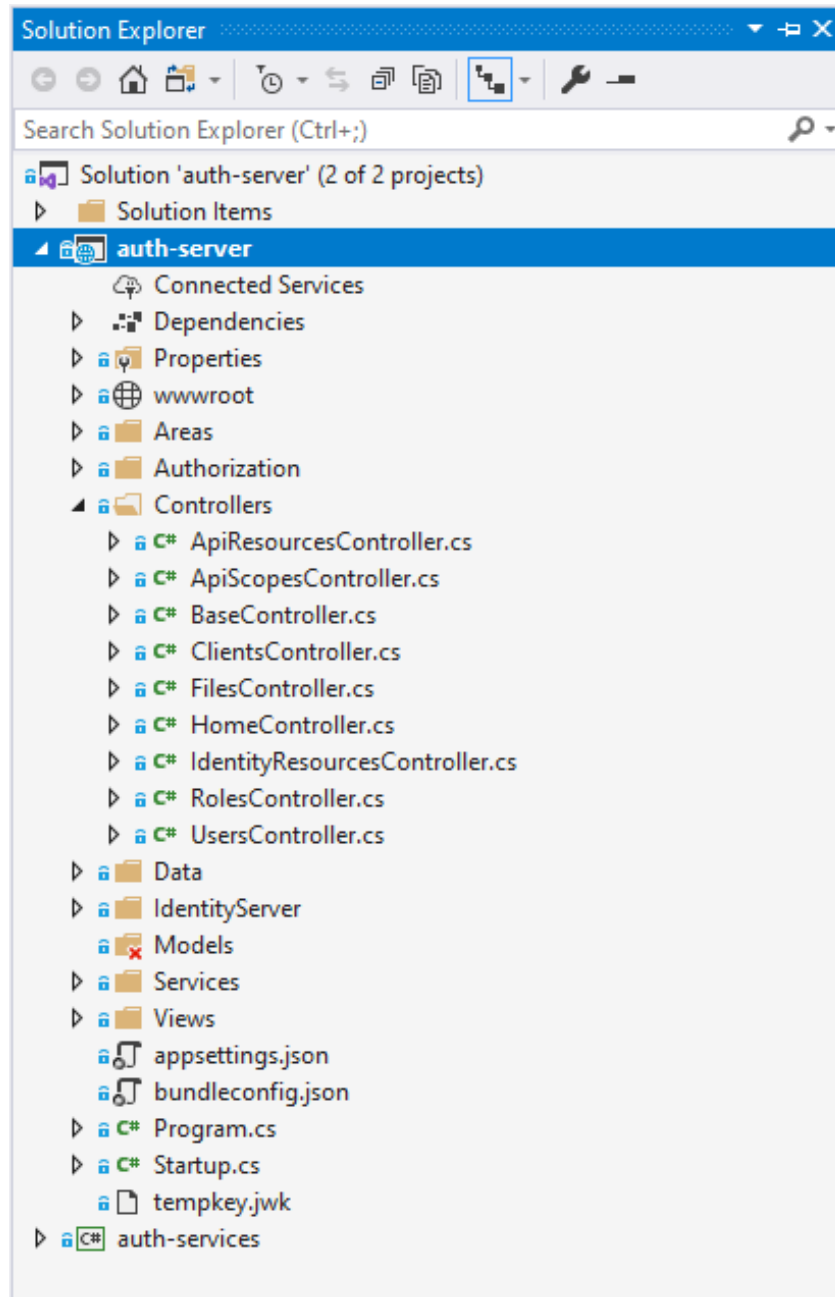
Kiểm soát truy cập là một cách giới hạn quyền truy cập vào hệ thống hoặc các tài nguyên hệ thống. Trong máy tính, kiểm soát truy cập là một quá trình mà người dùng được cấp quyền truy cập và các đặc quyền nhất định đối với hệ thống, tài nguyên hoặc thông tin. Trong các hệ thống kiểm soát truy cập cũ hơn, người dùng phải xuất trình thông tin xác thực trước khi họ có thể được cấp quyền truy cập. Trong các hệ thống vật lý hiện tại, những thông tin xác thực này có thể nhiều dạng, nhưng những thông tin xác thực này không thể được chuyển giao để cung cấp tính bảo mật cao.

3.3 Identity Server dựa trên kiến trúc một khối



Hình 3.4 Identity Server dựa trên kiến trúc nguyên khối.

Identity Server ban đầu được xây dựng dựa trên kiến trúc một khối. Toàn bộ các chức năng được đóng gói thành một khối lớn bao gồm: quản lý ứng dụng client, bảo vệ tài nguyên API, tài nguyên nhận dạng, quản lý danh tính người dùng như hình 3.6 phía dưới. Identity Server được triển khai bằng ASP.NET Core 3.1, ASP.NET Identity và Identity Server 4, v.v.



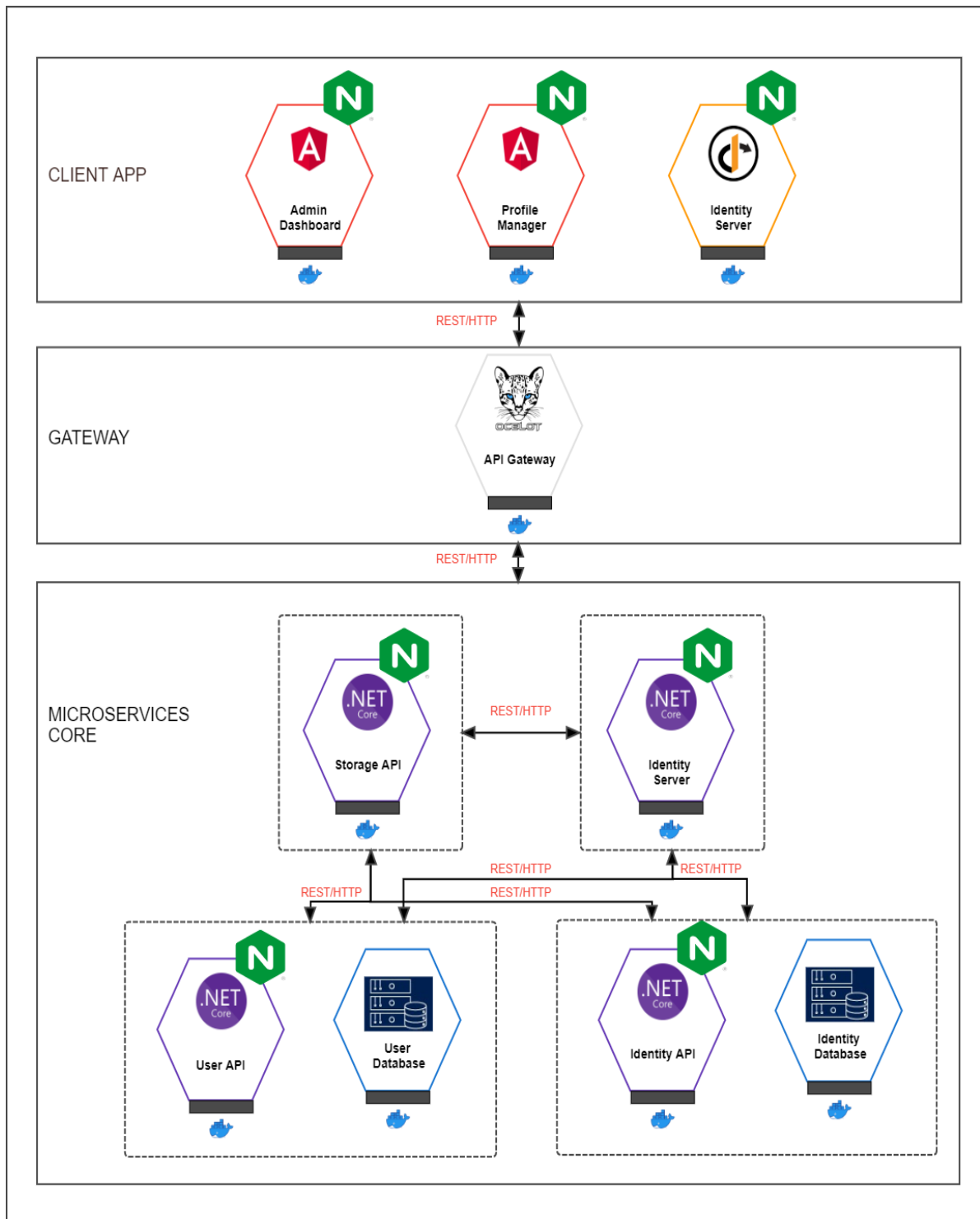
Hình 3.5 Cấu trúc dự án Identity Server dựa theo kiến trúc nguyên khối.

Như bao vấn đề mà các ứng dụng phần mềm nguyên khối thường gặp khác. Khi đưa Identity Server vào trong sản xuất có khá nhiều vấn đề hạn chế:

- Identity Server sử dụng Identity Server 4 và ASP.NET Identity, các thư viện sẽ liên tục cập nhật các phiên bản mới bao gồm các bản vá lỗi và cũng như các chức năng mới về quản lý ứng dụng client, quản lý danh tính người dùng, bảo vệ tài nguyên, v.v. cũng như database. Vì thế, việc liên tục cập nhật và sửa đổi và triển khai sẽ trở nên khó khăn.
- Việc Identity Server sử dụng cả Identity Server 4 và ASP.NET Identity. Kiến trúc ứng dụng nguyên khối không cho phép mở rộng quy mô một dịch vụ một cách độc lập. Ví dụ: yêu cầu mở rộng quy mô ở phần danh tính người dùng (ASP.NET Identity) thì dịch vụ Identity Server 4 không có thay đổi cũng cần phải triển khai lại.
- Identity Server được xây dựng từ nhiều thành phần khác nhau. Gồm có các phần: dịch vụ liên quan đến Identity Server 4 (client, tài nguyên cần được bảo vệ, danh tính, v.v.), dịch vụ máy chủ cấp token và xác thực token, dịch vụ lưu trữ tệp tin, dịch vụ liên quan đến ASP.NET Identity (quản lý người dùng và quyền truy cập tài nguyên). Nếu một trong các dịch vụ không hoạt động như mong muốn có thể làm ảnh hưởng đến cả hệ thống.
- Các dịch vụ chỉ có thể được xây dựng từ ASP.NET Core bởi vì cả dự án được đóng gói thành một khối. Khó tiếp cận các công nghệ mới phù hợp cho từng dịch vụ.
- Mặc dù thời gian triển khai Identity Server sử dụng trong thực tế nhanh nhưng nếu có thay đổi nhỏ ở một dịch vụ nào đó, bắt buộc cả hệ thống phải khởi động lại. Do đó, thời gian bảo trì hệ thống sẽ bị kéo dài.

Các hạn chế này là khởi đầu dẫn đến việc xây dựng Identity Server dựa trên kiến trúc Microservices từ những dịch vụ có sẵn ở kiến trúc một khối bằng việc tách các thành phần thành các microservices.

3.4 Máy chủ nhận dạng dựa trên kiến trúc Microservices.



Hình 3.6 Identity Server dựa trên kiến trúc Microservices.

Identity Server dựa trên kiến trúc Microservices khắc phục được một số hạn chế khi triển khai ứng dụng phần mềm dựa trên kiến trúc nguyên khối. Chẳng hạn:

Bảng 3.1 So sánh Identity Server dựa trên hai kiến trúc..

Identity Server dựa trên kiến trúc nguyên khối (hình 3.6)	Identity Server dựa trên kiến trúc Microservice (hình 3.8)
Đóng gói 4 dịch vụ thành một khối duy nhất. Hạn chế cho khả năng triển khai, bảo trì và nâng cấp	Tách biệt 4 dịch vụ này thành các microservice nhỏ như hình 3.8. Thuận lợi cho việc triển khai, bảo trì và nâng cấp
Triển khai lại toàn bộ hệ thống khi có một thay đổi nhỏ.	Chỉ triển khai một microservice bị thay đổi.
Các dịch vụ được đóng gói thành một khối duy nhất, khi một dịch vụ nào đó gặp lỗi thì cả hệ thống bị ảnh hưởng.	Các microservice được triển khai độc lập với nhau. Khi xảy ra lỗi ở một microservice nào đó thì chỉ ảnh hưởng đến chức năng mà microservice đó đảm nhiệm và kéo theo một số microservice khác có giao tiếp đến nó. Hệ thống vẫn hoạt động bình thường khi mất microservice đó thay vì cả hệ thống bị ảnh hưởng.
Khó áp dụng công nghệ mới.	Có thể áp dụng công nghệ mới để phù hợp cho từng microservice.
Tất cả các dịch vụ được triển khai thành một khối duy nhất, dẫn đến việc quản lý Logs trở nên khó khăn. Vì tất cả các dịch vụ đều ghi Logs của chính mình vào một file chung duy nhất.	Mỗi microservice có một file Logs riêng, giúp dễ dàng quản lý.
Tương tự với dịch vụ, cơ sở dữ liệu tập trung nên khi gặp lỗi sẽ ảnh hưởng toàn bộ hệ thống.	Cơ sở dữ liệu phi tập trung.

3.5 Xác định, xây dựng Identity Server dựa trên kiến trúc Microservices

3.5.1 Microservices Core

Bảng 3.2 Xác định các microservices và chức năng.

Tên microservice	Chức năng	Ngôn ngữ/Framework
Identity Server	Cung cấp token, xác nhận token. Còn có giao diện phục vụ cho dịch vụ Single Sign-On	ASP.NET Core 3.1, Identity Server 4, ASP.NET Identity, ASP.NET Core Blazor
Storage API	Cung cấp API có chức năng lưu trữ và hiển thị các tệp tin do người dùng tải lên	ASP.NET Core 3.1, Swagger
User API	Cung cấp API có chức năng quản lý các tác vụ liên quan đến người dùng	ASP.NET Core 3.1, ASP.NET Identity, Swagger
Identity API	Cung cấp các API có chức năng quản lý các tác vụ liên quan đến client, bảo vệ API, danh tính	ASP.NET Core 3.1, Identity Server 4, Swagger

Ngoài ra, các microservice được tích hợp với một ứng dụng web server Nginx. Có chức năng load balancing, cân bằng tải, proxy, bảo mật v.v. Nginx là phần mềm mã nguồn mở và có hiệu năng vô cùng ấn tượng.

```

nginx
Copy
server {
    listen      80;
    server_name example.com *.example.com;
    location / {
        proxy_pass          http://localhost:5000;
        proxy_http_version  1.1;
        proxy_set_header    Upgrade $http_upgrade;
        proxy_set_header    Connection keep-alive;
        proxy_set_header    Host $host;
        proxy_cache_bypass  $http_upgrade;
        proxy_set_header    X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header    X-Forwarded-Proto $scheme;
    }
}

```

Hình 3.7 Cấu hình một dịch vụ khi sử dụng Nginx làm chức năng cân bằng tải.

Tất cả microservice được đóng gói thành Docker Image để triển khai thành các Docker Container thông qua Docker Compose trên cả nền tảng Window, Linux và Mac. Vì thế, việc triển khai các microservice trên tất cả các nền tảng là việc vô cùng dễ dàng. Ngoài ra, microservice được đóng gói và triển khai trên một Container riêng biệt và độc lập so với các microservice khác, tạo điều kiện cho việc sửa lỗi, nâng cấp và bảo trì.

JSON Copy

```

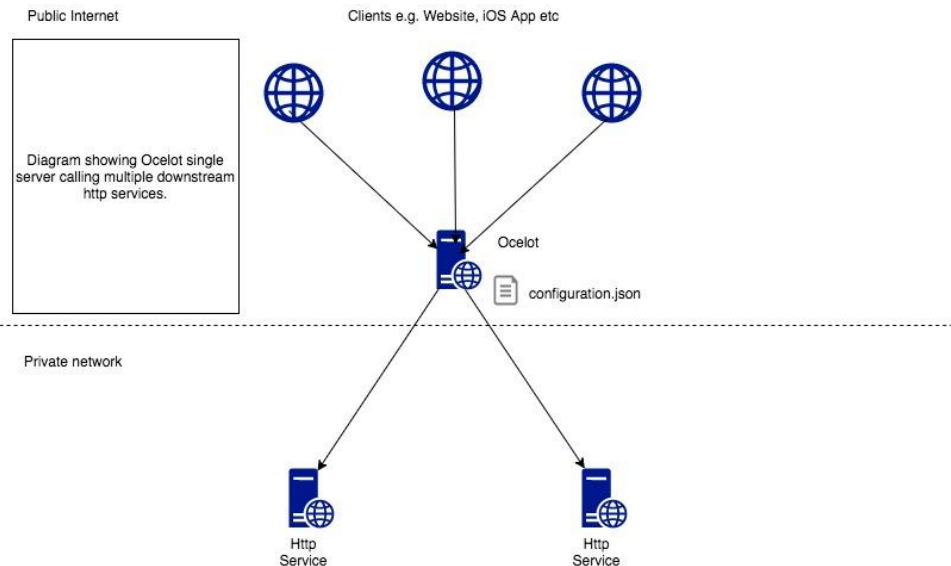
version: '3.4'

services:
  webapp:
    image: mcr.microsoft.com/dotnet/core/samples:aspnetapp
    ports:
      - 80
      - 443
    environment:
      - ASPNETCORE_ENVIRONMENT=Development
      - ASPNETCORE_URLS=https://+:443;http://+:80
      - ASPNETCORE_Kestrel__Certificates__Default__Password=password
      - ASPNETCORE_Kestrel__Certificates__Default__Path=/https/aspnetapp.pfx
    volumes:
      - ~/.aspnet/https:/https:ro

```

Hình 3.8 Đóng gói một dịch vụ thành Docker Image sử dụng Docker Compose.

3.5.2 API Gateway



Hình 3.9 API Gateway Ocelot cho ASP.NET Core.

Như hình 3.8, Identity Server có một API Gateway làm nhiệm vụ định tuyến các yêu cầu/phản hồi giữa các microservice và ứng dụng client. Như hình 3.8, có bốn microservice tức là sẽ có bốn điểm đầu cuối mà các ứng dụng client

cần biết để có thể giao tiếp với chúng. Vì thế sử dụng API Gateway là một giải pháp vô cùng hữu ích.

Ocelot hướng đến những người sử dụng .NET chạy ứng dụng có kiến trúc hướng dịch vụ SOA hay kiến trúc Microservices cần một điểm vào thống nhất trong hệ thống.

Ocelot là một loạt các phần mềm trung gian theo một thứ tự cụ thể.

Ocelot điều khiển đối tượng `HttpRequest` thành một trạng thái được chỉ định bởi một cấu hình (khai báo) trước đó của nó cho đến khi nó đạt đến phần mềm trung gian của trình tạo đối tượng `HttpRequestMessage` được sử dụng để thực hiện yêu cầu đối với các dịch vụ bên dưới (microservice). Phần mềm trung gian thực hiện các yêu cầu là điểm cuối cùng trong đường dẫn Ocelot. Nó không gọi phần mềm trung gian tiếp theo. Có một phần mềm trung gian ánh xạ `HttpResponseMessage` vào đối tượng `HttpResponse` và được trả lại cho máy khách.



```

JSON
Copy

{
  "DownstreamPathTemplate": "/api/{version}/{everything}",
  "DownstreamScheme": "http",
  "DownstreamHostAndPorts": [
    {
      "Host": "basket-api",
      "Port": 80
    }
  ],
  "UpstreamPathTemplate": "/api/{version}/b/{everything}",
  "UpstreamHttpMethod": [ "POST", "PUT", "GET" ],
  "AuthenticationOptions": {
    "AuthenticationProviderKey": "IdentityApiKey",
    "AllowedScopes": []
  }
}

```

Hình 3.10 Khai báo một API với API Gateway Ocelot bằng file Configuration.json..

3.5.3 Các ứng dụng phía client

- Angular - Admin Dashboard: Giao diện quản lý tất cả các dịch vụ liên quan đến xác thực và ủy quyền. Bao gồm: ứng dụng client, tài nguyên API được bảo vệ, các quy định của các tài nguyên, danh tính, thông tin người dùng, các chức năng liên quan đến ủy quyền người dùng, v.v.

- Angular – User Profile: Giao diện cung cấp các chức năng cho từng người dùng riêng biệt có thể xem và chỉnh sửa thông tin cá nhân của chính mình. Ngoài ra còn có thể xóa tài khoản khi không cần dùng đến.
- Blazor – Identity Server: Giao diện đăng nhập và đăng xuất phục vụ chung cho tất cả các ứng dụng client (Single Sign-On).

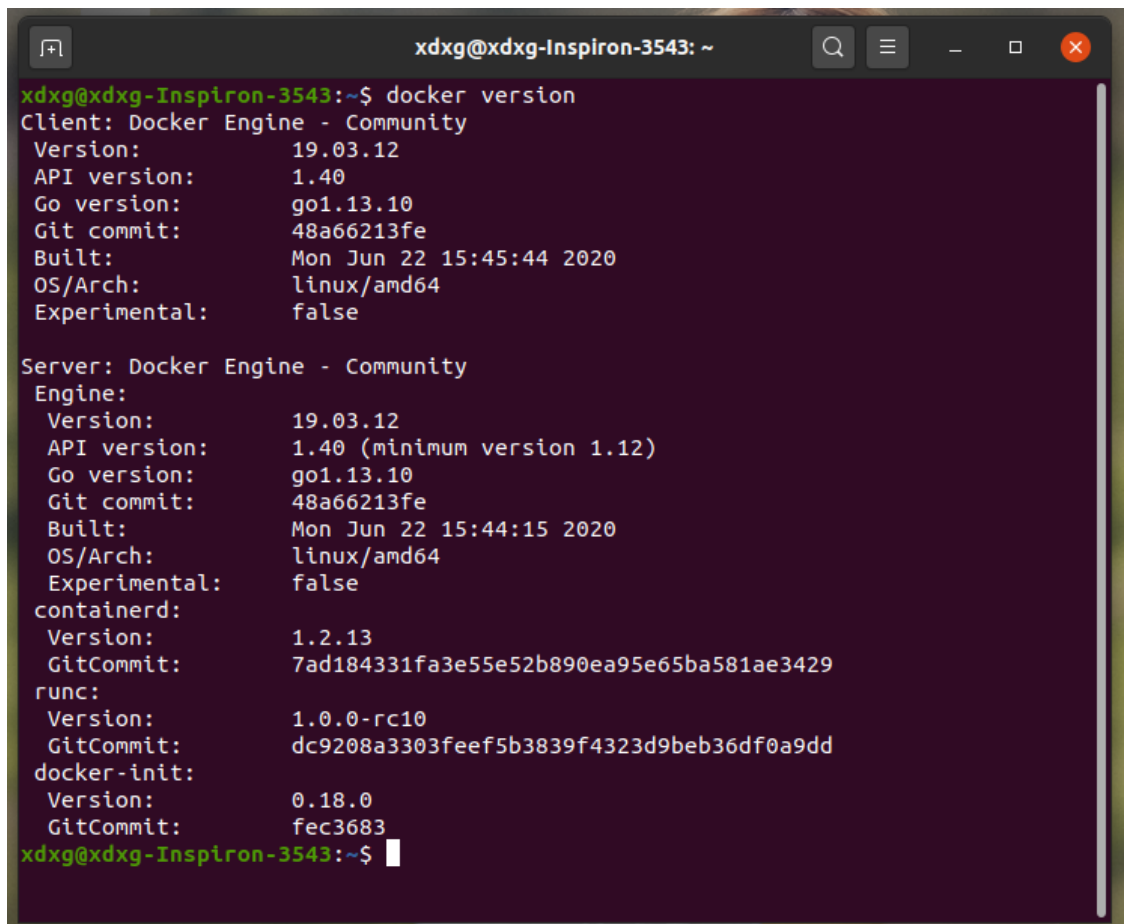
Các ứng dụng client đều được đóng gói với Docker Image và Nginx. Vì thế chúng cũng có khả năng triển khai nhanh gọn và trên tất cả các nền tảng hệ điều hành.

CHƯƠNG 4 – SỬ DỤNG MÁY CHỦ NHẬN DẠNG TRONG THỰC TẾ

4.1 Cài đặt

1. Cài đặt Identity Server được khuyến cáo sử dụng Docker. Các microservice được viết mã để đóng gói thành các Docker Image để thuận lợi cho việc di chuyển và triển khai nhanh chóng. Trong hướng dẫn cài đặt này sử dụng Docker cho Ubuntu 20.04 (LTS). Yêu cầu hệ thống

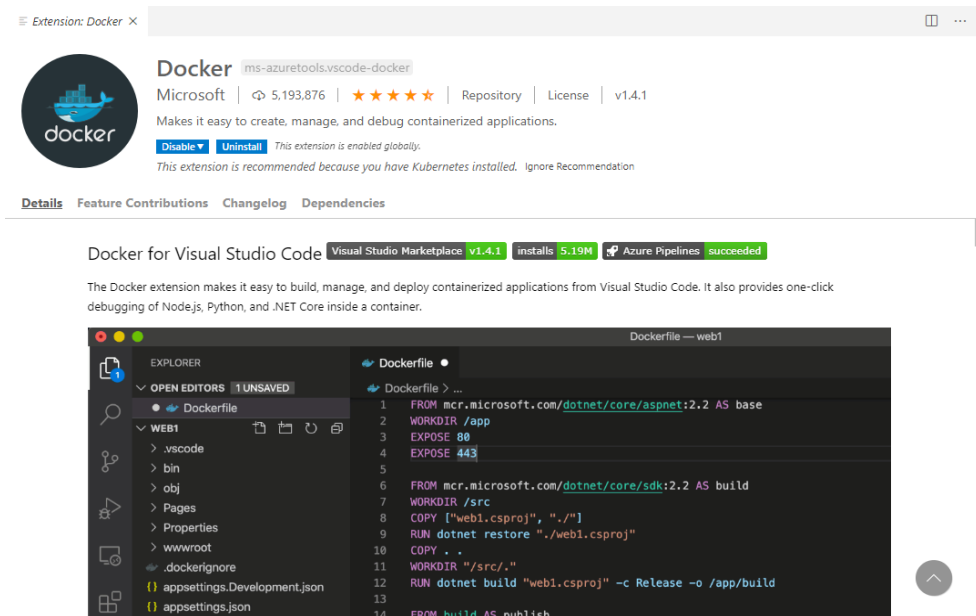
- Ubuntu 20.04 (LTS) hoặc các phiên bản thấp hơn bao gồm 19.10 (LTS), 18.04 (LTS), 16.04 (LTS)
- Docker Engine được hỗ trợ trên vi xử lý x86_64, amd64, armhf và các arm64.
- Ram tối thiểu 4GB. Yêu cầu 8GB để chạy tốt.



```
xdxg@xdxg-Inspiron-3543: ~  
xdxg@xdxg-Inspiron-3543:~$ docker version  
Client: Docker Engine - Community  
Version:           19.03.12  
API version:       1.40  
Go version:        go1.13.10  
Git commit:        48a66213fe  
Built:             Mon Jun 22 15:45:44 2020  
OS/Arch:           linux/amd64  
Experimental:      false  
  
Server: Docker Engine - Community  
Engine:  
Version:           19.03.12  
API version:       1.40 (minimum version 1.12)  
Go version:        go1.13.10  
Git commit:        48a66213fe  
Built:             Mon Jun 22 15:44:15 2020  
OS/Arch:           linux/amd64  
Experimental:      false  
containerd:  
Version:           1.2.13  
GitCommit:         7ad184331fa3e55e52b890ea95e65ba581ae3429  
runc:  
Version:           1.0.0-rc10  
GitCommit:         dc9208a3303feef5b3839f4323d9beb36df0a9dd  
docker-init:  
Version:           0.18.0  
GitCommit:         fec3683  
xdxg@xdxg-Inspiron-3543:~$
```

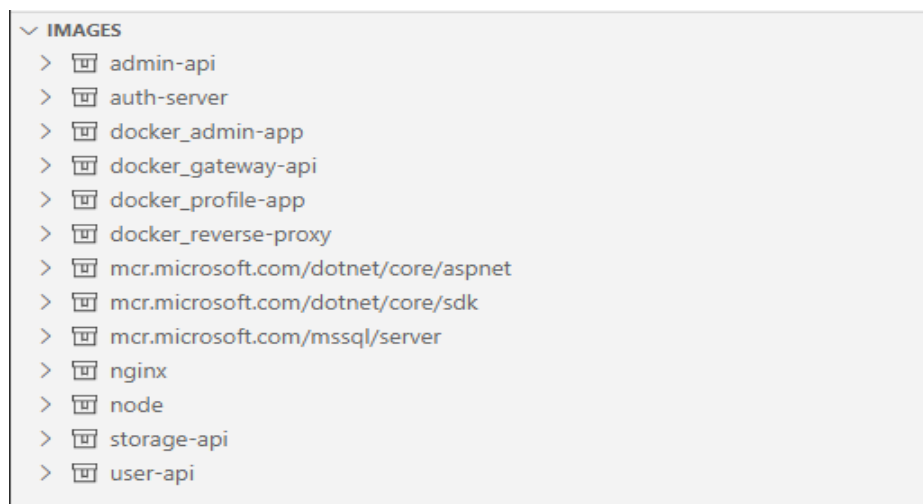
Hình 4.1 Phiên bản Docker được cài đặt trên Ubuntu.

2. Có thể quản lý các image, container, v.v. thông qua các lệnh terminal. Nhưng để quản lý khách quan hơn, có thể sử dụng các tiện ích có sẵn trên Visual Studio Code. Tiện ích Docker trên VS Code là một ví dụ:



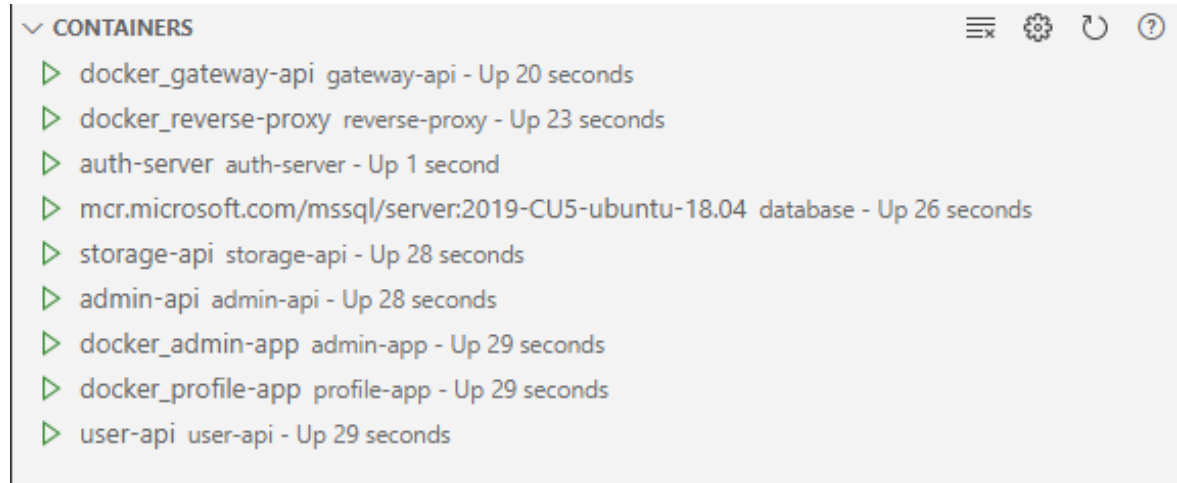
Hình 4.2 Tiện ích Docker trên VS Code.

3. Tải về file ZIP hay clone dự án từ github. Địa chỉ dự án được lưu trên github: <https://github.com/tinhpham76/Microservices>.
4. Từ thư mục gốc Microservice, di chuyển đến `Microservices/src/Docker`. Thực thi lệnh `docker-compose build`. Các dịch vụ trong Identity Server sẽ được đóng gói thành các Docker Image bao gồm các tài nguyên kèm theo (môi trường, thư viện các microservice cần sử dụng) như hình 4.3.



Hình 4.3 Docker Image sau khi đóng gói.

5. Sau khi đóng gói tất cả các microservice và các tài nguyên kèm theo. Thực thi lệnh `docker-compose up -d` bắt đầu khởi chạy các Docker Image thành các Docker Container. Các microservice sẽ bắt đầu được triển khai.



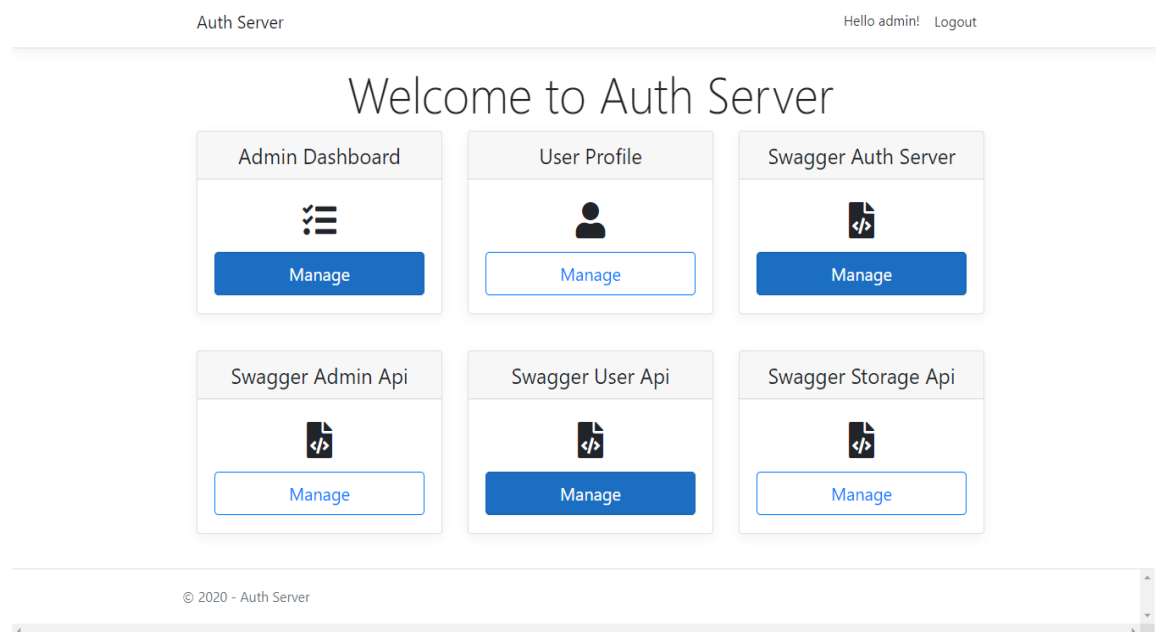
Hình 4.4 Docker Container sau khi được khởi tạo từ Docker Image.

6. Tất cả các ứng dụng client truy cập vào hệ thống thông qua API Gateway, nhưng các microservice có một cổng truy cập riêng để đưa ra các API và quản lý chúng thông qua Swagger. Các ứng dụng client và điểm cuối API:

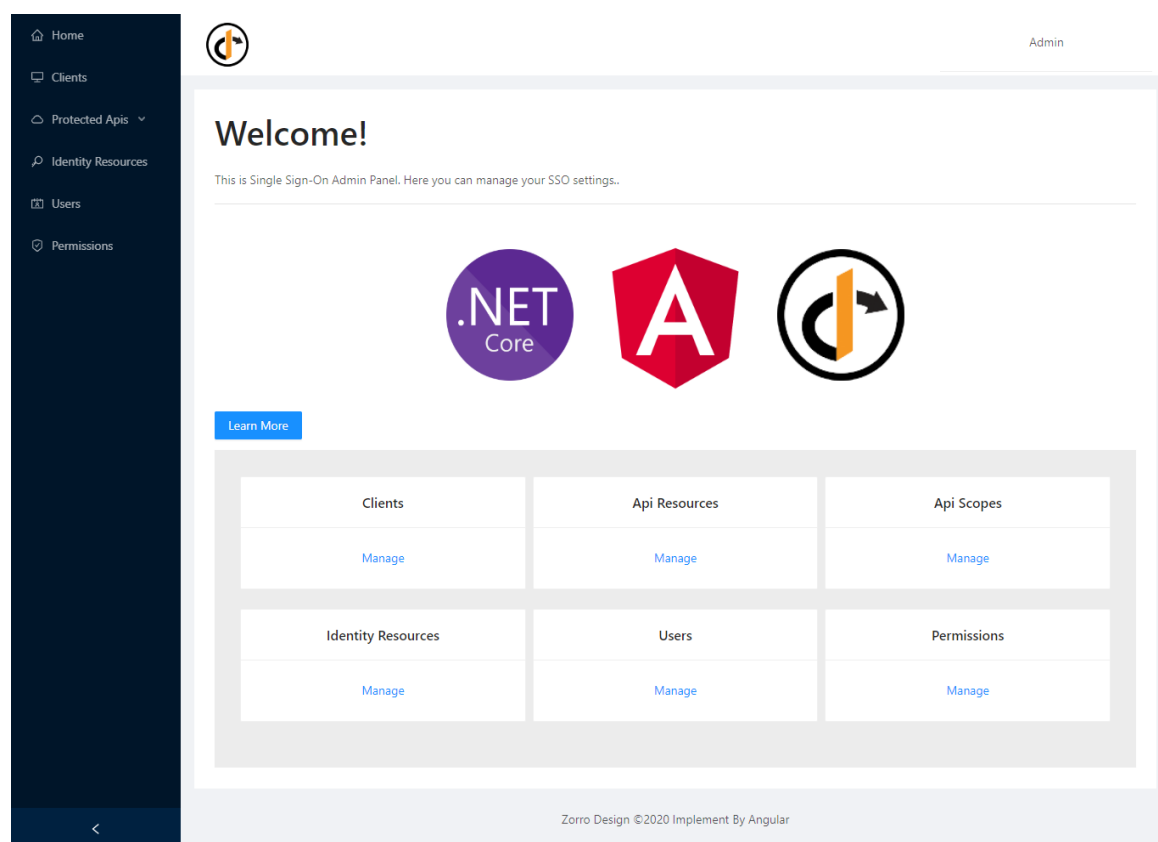
Bảng 4.1 Điểm cuối truy cập của Identity Server.

Tên	Điểm truy cập
Identity Server	http://localhost:5001
Angular – Admin Dashboard	http://localhost:4200
Angular – User Profile	http://localhost:4300
API Gateway	http://localhost:8001
Swagger Identity API	http://localhost:5001/swagger
Swagger Admin API	http://localhost:5003/swagger
Swagger User API	http://localhost:5005/swagger
Swagger Storage API	http://localhost:5007/swagger

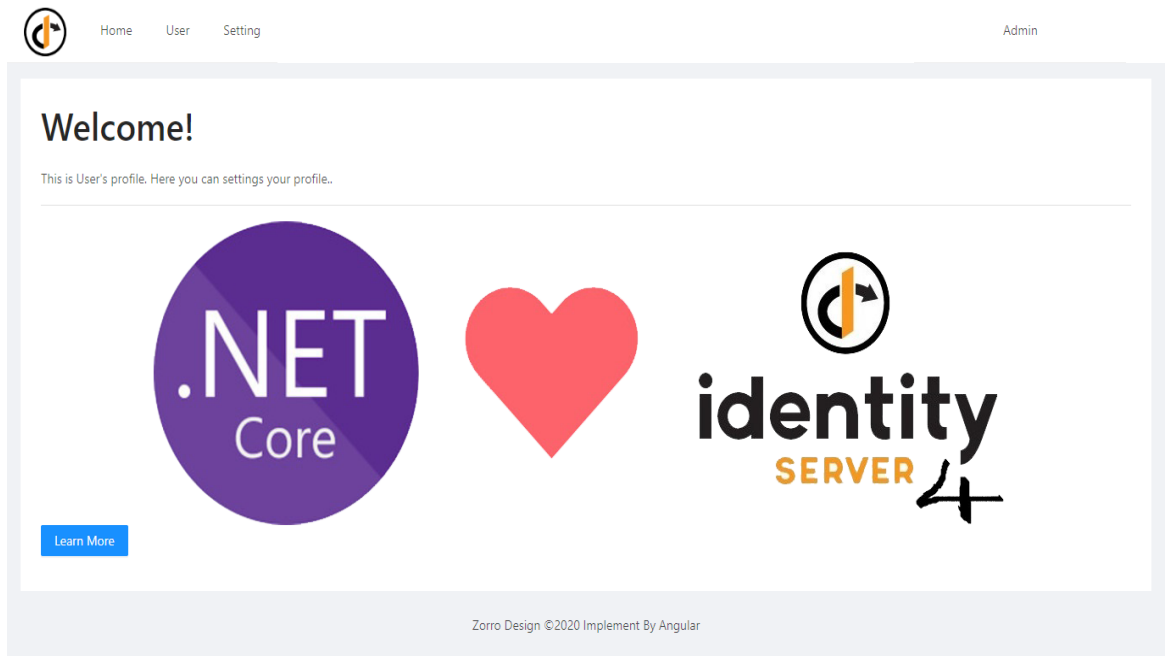
7. Một số hình ảnh giao diện của hệ thống sau khi cài đặt:



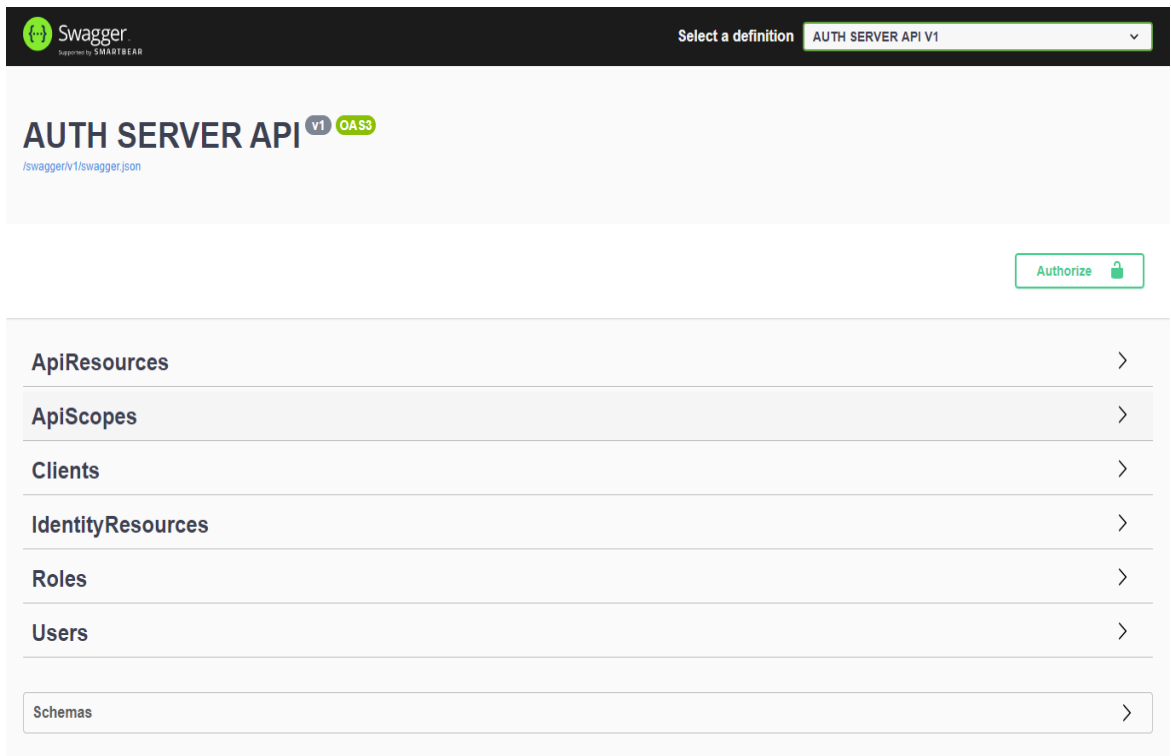
Hình 4.5 Trang chủ Identity Sever.



Hình 4.6 Trang chủ ứng dụng Admin Dashboard.



Hình 4.7 Trang chủ ứng dụng User Profile.



Hình 4.8 Swagger quản lý API Identity Microservice.

4.2 Đăng nhập vào ứng dụng của hệ thống thông qua Identity Server




1. Truy cập các ứng dụng client (Angular – Admin Dashboard, Angular – User Profile) và đăng nhập thông qua Identity Server.


Login to continue

☐ Remember me [Forgot Password?](#)

LOGIN

or sign up using

 Đăng nhập bằng Identity Server

Hình 4.9 Đăng nhập vào ứng dụng client.

2. Hệ thống sẽ tự động chuyển tiếp đến trang đăng nhập của Identity Server (trang đăng nhập SSO), và người dùng cần đăng nhập bằng tài khoản được cấp trước đó hoặc phải tự tạo tài khoản mới.

Log in

Use a local account to log in.

UserName

Password

☐ Remember me?

Log in

[Forgot your password?](#)

[Register as a new user](#)

[Resend email confirmation](#)

Use another service to log in.

There are no external authentication services configured. See [this article](#) for details on setting up this ASP.NET application to support logging in via external services.

Hình 4.10 Đăng nhập vào ứng dụng thông qua Identity Server .

3. Sau khi đăng nhập thành công, hệ thống tự động quay về ứng dụng trước đó và gửi kèm theo một mã token, các ứng dụng client sử dụng các token này để có quyền truy cập vào các tài nguyên API của hệ thống. Mã token có thể chứa các thông cơ bản và các thông tin bổ sung do nhà phát triển cài đặt thêm.

```
{, ...}
  access_token: "66F60F1E591208B3926149BBCC6B5D2C1A2B970225BAD529C104301626B5C4A4"
  expires_at: 1597311206
  id_token: "eyJhbGciOiJSUzI1NiIsImtpZCI6IjRjQWJjZjQ3OTAwQkQZFQZFRkVhcnVNTQZ3Q3QzQ3MUMzMzU4RUFGIiwidHlwIjoisIld"
  profile: {s_hash: "ebHEocmAe0nfoGV_ZqJdJg", sid: "D240A18C78EE93C795A397D9D305C5DA", ...}
    Avatar: "https://github.githubassets.com/images/modules/logos_page/GitHub-Mark.png"
    Email: "admin@admin.com"
    FullName: "Admin"
    Permissions: ["AUTH_SERVER_VIEW", "AUTH_SERVER_CREATE", "AUTH_SERVER_UPDATE", "AUTH_SERVER_DELETE"]
    Role: "Admin"
    Username: "admin"
  ▶ amr: ["pwd"]
    auth_time: 1597307604
    idp: "local"
    s_hash: "ebHEocmAe0nfoGV_ZqJdJg"
    sid: "D240A18C78EE93C795A397D9D305C5DA"
    sub: "38fd894b-d89e-4994-bcb7-a3ff86a6ef32"
  scope: "AUTH_SERVER ADMIN_API USER_API openid profile"
  session_state: "fApnKSS5uOvHuzMnat8r1Fjgy2tN70A39PkvvupicFU.8B2F758871611875FC610D8B6F7937F3"
  token_type: "Bearer"
```

Hình 4.11 Mã token được gửi về ứng dụng client.

4.3 Cấu hình ứng dụng client mới

Yêu cầu người dùng có quyền truy cập tất cả các API (mặc định người dùng được phân quyền Admin). Sau khi đăng nhập thành công vào ứng dụng Angular – Admin Dashboard:

1. Truy cập vào dịch vụ Clients/Create new Client và tiến hành tạo mới một ứng dụng client với các thông tin cơ bản bao gồm:

* Client Name:

test-client-spa

Description:

đăng nhập một lần với ứng dụng single page application

* Client Uri:

http://localhost:4400

Logo Uri:

http://localhost:8001/multi-media/angular-logo.png

⬇️ Click to Upload

angular-logo.png

Client Type:

Empty

Default

Hình 4.12 Tao ứng dụng client mới với thông tin cơ bản.

- Tên, mô tả, địa chỉ url
 - Thêm logo cho ứng dụng để dễ phân biệt (được tải lên tệp hình ảnh không qua 60Mb hoặc có thể dán đường dẫn) hoặc có thể bỏ qua.
 - Chọn loại ứng dụng phía client
2. Sau khi tạo thông tin cơ bản thành công. Ứng dụng sẽ chuyển tiếp đến trang cài đặt các thông số chi tiết cho ứng dụng. Để ứng dụng client có thể kết nối được với Identity Server. Người dùng cần cài đặt một số thông số riêng tương ứng với loại ứng dụng đó. Trong bài này sẽ ví dụ đăng ký ứng dụng SPA (Angular) sử dụng OIDC.

Bảng 4.12 Thông số của ứng dụng client.

Thông tin cơ bản	
Client id	Id duy nhất của ứng dụng client.
Enable	Chỉ định ứng dụng được bật hay tắt.
Client Secrets	Danh sách khóa bí mật của ứng dụng khách – dùng để truy cập vào máy chủ cung cấp mã token.
Require Client Secret (mặc định: true)	Chỉ định xem ứng dụng client này có cần yêu cầu mã bí mật từ máy chủ cung cấp token hay không.
Allowed Grant Types	Chỉ định các loại kết nối mà ứng dụng client có thể sử dụng.
Required PKCE (mặc định: true)	Chỉ định ứng dụng client dựa trên Code Flow có phải gửi khóa bằng chứng hay không.
Allow Plain Text PKCE (mặc định: false)	Chỉ định ứng dụng client sử dụng PKCE có thể sử dụng văn bản thuần túy hay không.
Redirect Uris	Chỉ định các URL được phép nhận mã token truy cập hoặc mã token ủy quyền.
Allowed Scopes	Theo mặc định, ứng dụng client không được phép truy cập bất cứ tài nguyên nào – chỉ định

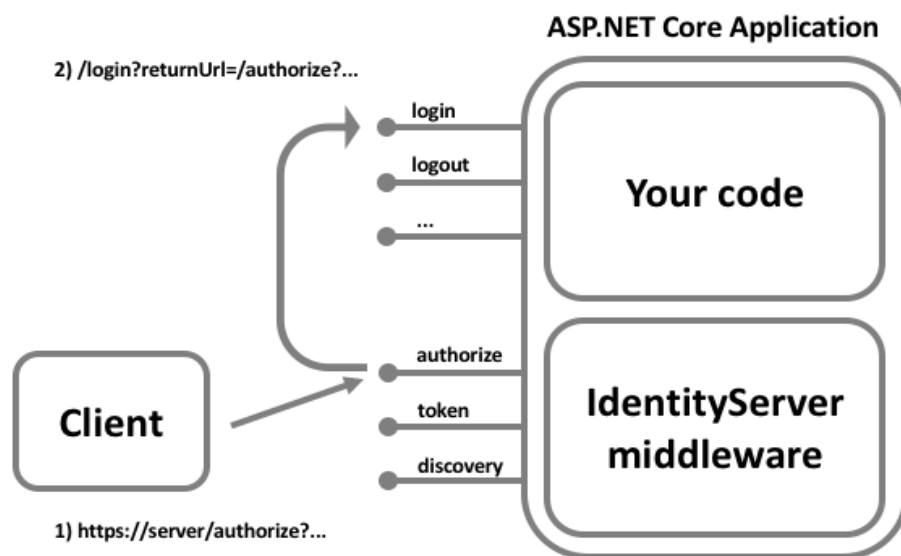
	tài nguyên được phép truy cập bằng cách thêm tên tài nguyên tương ứng.
Allow Offline Access	Chỉ định ứng dụng client có thể yêu cầu làm mới mã token hay không.
Allow Access Tokens Via Browser	Chỉ định ứng dụng client có thể nhận mã token mới thông qua trình duyệt hay không.
Properties	Lưu trữ mọi giá trị dành riêng cho ứng dụng khách.
Xác thực/Đăng xuất	
Post Logout Redirect Uris	Chỉ định các URL được chuyển hướng đến sau khi đăng xuất.
Front Channel Logout Uri	Chỉ định URL chỉ đăng xuất tại ứng dụng client.
Front Channel Logout Session Required (mặc định: true)	Chỉ định xem Id phiên sử dụng của người dùng.
Back Channel Logout Uri	Chỉ định URL đăng xuất tại ứng dụng client.
Back Channel Logout Session Required (mặc định: true)	Chỉ định xem Id phiên của người dùng.
Enable Local Login (mặc định là true)	Chỉ định ứng dụng client này có thể sử dụng tài khoản cục bộ hay bên thứ ba.
Identity Provider Restrictions	Chỉ định dịch vụ bên thứ ba nào có thể được sử dụng với ứng dụng client này
User Sso Lifetime	Thời lượng tối đa (tính bằng giây) kể từ lần cuối cùng người dùng xác thực (mặc định là null) xác định thời gian tồn tại của phiên đăng nhập.
Token	

Identity Token Lifetime (mặc định: 300 giây / 5 phút)	Thời gian tồn tại của mã token nhận dạng tính bằng giây.
Allowed Identity Token Signing Algorithms	Danh sách các thuật toán được phép ký cho mã token nhận dạng. Nếu trống, sẽ sử dụng thuật toán ký mặc định của máy chủ.
Access Token Lifetime (mặc định: 3600 giây / 1 giờ)	Thời gian tồn tại của mã token truy cập tính bằng giây.
Authorization Code Lifetime (mặc định: 300 giây / 5 phút)	Thời gian tồn tại của mã token ủy quyền tính bằng giây).
Absolute Refresh Token Lifetime	Thời gian tồn tại tối đa của mã token làm mới tính bằng giây.
Sliding Refresh Token Lifetime	Thời gian trượt của mã token làm mới trong vài giây.
Refresh Token Usage	<ul style="list-style-type: none"> • ReUse xử lý mã token làm mới sẽ giữ nguyên khi làm mới mã token • OneTime xử lý mã token làm mới sẽ được cập nhật khi làm mới mã token. Đây là mặc định.
Refresh Token Expiration	<ul style="list-style-type: none"> • Absolute mã token làm mới sẽ hết hạn vào một thời điểm cố định (được chỉ định bởi Absolute Refresh Token Lifetime) • Sliding khi làm mới mã token, thời gian tồn tại của mã token làm mới sẽ được gia hạn (theo số lượng được chỉ định trong Sliding RefreshToken Lifetime). Thời gian tồn tại sẽ không vượt quá Absolute Refresh Token Lifetime.
Update Access Token Claims	Nhận hoặc đặt một giá trị cho biết liệu mã

On Refresh	token truy cập (và các xác nhận quyền sở hữu của nó) có được cập nhật theo yêu cầu mã token làm mới hay không.
Access Token Type (mặc định: Jwt)	Chỉ định xem mã token truy cập là mã thông báo tham chiếu hay mã token JWT.
Include Jwt Id (mặc định: true)	Chỉ định xem mã token truy cập JWT có nên có một ID duy nhất được nhúng hay không.
Allowed Cors Origins	Nếu được chỉ định, sẽ được sử dụng bởi các triển khai dịch vụ chính sách CORS mặc định để xây dựng chính sách CORS cho các ứng dụng khách JavaScript.
Claims	Cho phép xác nhận quyền sở hữu cài đặt cho khách hàng (sẽ được bao gồm trong mã token truy cập).
Always Send Client Claims (mặc định: false)	Nếu được đặt, yêu cầu của khách hàng sẽ được gửi cho mọi luồng. Nếu không, chỉ dành cho luồng thông tin xác thực khách hàng.
Always Include User Claims In Id Token (Mặc định: false).	Khi yêu cầu cả mã thông báo id và mã thông báo truy cập, nếu yêu cầu của người dùng luôn được thêm vào mã thông báo id thay vì yêu cầu khách hàng sử dụng điểm cuối userinfo.
Client Claims Prefix	Nếu được đặt, các loại xác nhận quyền sở hữu khách hàng tiền tố sẽ được bắt đầu bằng. Mặc định cho client. Mục đích là để đảm bảo họ không vô tình va chạm với các xác nhận quyền sở hữu của người dùng.
Pair Wise Subject Salt	Giá trị được sử dụng trong tạo subjectId theo cặp cho người dùng của ứng dụng client này.

Require Consent	Tên hiển thị của ứng dụng client
Allow Remember Consent (mặc định: true)	Chỉ định xem người dùng có thể chọn lưu trữ các quyết định về sự đồng ý hay không.
Consent Lifetime (mặc định: null).	Thời gian tồn tại của sự đồng ý của người dùng trong vài giây.
Client Name	Tên hiển thị của ứng dụng client
Client Uri	URI để biết thêm thông tin về ứng dụng client
Logo Uri	URI cho logo ứng dụng client
User Code Type	Chỉ định loại mã người dùng để sử dụng cho máy khách
Device Code Lifetime (mặc định: 300 giây / 5 phút)	Thời gian tồn tại của mã thiết bị tính bằng giây

3. Sau khi cài đặt các thông số cần thiết, cần thiết lập kết nối từ ứng dụng client với Identity Server sử dụng OIDC Client. Sử dụng tài liệu OpenID Connect.



Hình 4.13 Quy trình đăng nhập.

4. Tạo giao diện đăng nhập cho ứng dụng với nút chức năng đăng nhập bằng Identity Server. Nó sẽ giống như thế này:

```

<div class="login100-form-social flex-c-m">
  <a (click)="loginWithIS4()" class="login100-form-social-item flex-c-m bg4 m-r-5">
    
  </a>
  <a (click)="createNotification('error', 'Facebook', 'Unregistered or unsupported applications!')"
    class="login100-form-social-item flex-c-m bg1 m-r-5">
    <i class="fa fa-facebook-f" aria-hidden="true"></i>
  </a>

  <a (click)="createNotification('error', 'Twitter', 'Unregistered or unsupported applications!')"
    class="login100-form-social-item flex-c-m bg2 m-r-5">
    <i class="fa fa-twitter" aria-hidden="true"></i>
  </a>
</div>

```

Hình 4.14 Thêm các nút chức năng đăng nhập, đăng xuất.

```

export function getClientSettings(): UserManagerSettings {
  return {
    authority: 'https://localhost:5000',
    client_id: 'angular_user_profile',
    redirect_uri: 'http://localhost:4300/auth-callback',
    post_logout_redirect_uri: 'http://localhost:4300/',
    scope: 'SSO_SERVER openid profile',
    silent_redirect_uri: 'http://localhost:4300/silent-refresh.html',
    response_type: 'code',
    filterProtocolClaims: true,
    loadUserInfo: true,
    automaticSilentRenew: true,
  };
}

```

Hình 4.15 Sử dụng lớp UserManager từ thư viện oidc-client để quản trị các giao thức OpenID Connect.

```

login() {
  return this.manager.signinRedirect();
}

async signOut() {
  await this.manager.signoutRedirect();
}

```


Hình 4.16 Method đăng nhập và đăng xuất .

Login to continue

☐ Remember me [Forgot Password?](#)

LOGIN


or sign up using



Đăng nhập bằng Identity Server

Hình 4.17 Nhấn vào nút đăng nhập để đăng nhập cho người dùng.

Admin



admin

admin@admin.com

Logout

Hình 4.18 Nhấn vào nút đăng xuất để đăng xuất khỏi ứng dụng.

4.4 Bảo vệ API bằng Identity Server

4.4.1 Bảo vệ tài nguyên API

* Api Name:

Display Name:

Description:

Enable: ☒

Show In Discovery Document: ☒

Allowed Access Token Signing Algorithms:

Scopes:

Add new scope:

Api Claims:

Add new claim:

Hình 4.19 Bảo vệ API bằng Identity Server.

Sử dụng Angular – Admin Dashboard để xác thực các tài nguyên API cần được bảo vệ. Từ dịch vụ Protected APIs/ API Resources có thể thêm mới, xóa phạm vi và chỉnh sửa các thông số chi tiết của API cần bảo vệ. Các thông số kỹ thuật khi bảo vệ API bằng Identity Server:

Bảng 4.3 Thông số kỹ thuật bảo vệ API bằng .

Enabled (mặc định: true)	Cho biết nếu tài nguyên này được kích hoạt và có thể được yêu cầu.
Name	Tên duy nhất của API. Giá trị này được sử dụng để xác

	thực với sự xem xét nội tâm và sẽ được thêm vào đối tượng của mã thông báo truy cập gửi đi.
Display Name	Giá trị này có thể được sử dụng.
Description	Giá trị này có thể được sử dụng.
Api Secrets	Bí mật API được sử dụng cho điểm cuối nội quan. API có thể xác thực bằng cách sử dụng tên và bí mật API.
Allowed Access Token Signing Algorithms	Danh sách các thuật toán ký được phép cho mã thông báo truy cập. Nếu trống, sẽ sử dụng thuật toán ký mặc định của máy chủ.
User Claims	Danh sách các loại xác nhận quyền sở hữu của người dùng được liên kết sẽ được bao gồm trong mã thông báo truy cập.
Scopes	Danh sách tên phạm vi API.

Identity Server phát hành mã token truy cập ở định dạng JWT (Json Web Token) theo mặc định.

Mọi nền tảng liên quan ngày nay đều có hỗ trợ xác thực mã JWT. Việc bảo vệ API dựa trên ASP.NET Core chỉ là vấn đề thêm trình xử lý xác thực JWT:

```
services.AddAuthentication(options =>
{
    options.DefaultAuthenticateScheme = IdentityServerAuthenticationDefaults.AuthenticationScheme;
})
.AddIdentityServerAuthentication(options =>
{
    options.Authority = "https://localhost:5000";
    options.ApiSecret = "secret";
    options.ApiName = "ADMIN_API";
    options.ApiName = "AUTH_SERVER";
});
```

Hình 4.20 Trình xử lý mã token xác thực.

```

services.AddAuthorization(options =>
{
    options.AddPolicy("bearer", builder =>
    {
        // require scope1
        builder.RequireScope("ADMIN_API");
        builder.RequireScope("AUTH_SERVER");
    });
});

```

Hình 4.21 Trình xử lý mã token ủy quyền.

4.4.2 Các quy định về phạm vi truy cập tài nguyên API

Enable: ☒

* Name:

Display Name:

Description:

Required: ☐

Emphasize: ☐

Show In Discovery Document: ☒

User Claims:

Add new claim:

+ sub	+ name	+ given_name	+ family_name	+ middle_name	+ nickname
+ preferred_username	+ profile	+ picture	+ website	+ email	+ email_verified
+ gender	+ birthdate	+ zoneinfo	+ locale	+ phone_number	+ phone_number_verified
+ address	+ updated_at				

Hình 4.22 Xác định phạm vi API được bảo vệ.

Sử dụng Angular – Admin Dashboard để xác thực các tài nguyên API cần được bảo vệ. Từ dịch vụ Protected APIs/ API Scopes có thể thêm mới, xóa phạm vi và chỉnh sửa các thông số chi tiết của phạm vi API cần bảo vệ.

Bảng 4.4 Các thông số xác định phạm vi truy cập API .

Enabled (Mặc định: true)	Cho biết tài nguyên này được kích hoạt và có thể truy cập.
Name	Tên duy nhất của API. Giá trị này được sử dụng để xác thực với sự xem xét nội tâm và sẽ được thêm vào đối tượng của mã token truy cập gửi đi.
Display Name	Giá trị này có thể được sử dụng, ví dụ như trên màn hình chấp thuận.
Description	Giá trị này có thể được sử dụng, ví dụ như trên màn hình chấp thuận.
User Claims	Danh sách các loại xác nhận quyền sở hữu của người dùng được liên kết sẽ được bao gồm trong mã thông báo truy cập.

4.5 Tài nguyên nhận dạng

Tài nguyên nhận dạng là một nhóm xác nhận quyền sở hữu được đặt tên có thể được yêu cầu bằng cách sử dụng các tham số liên quan đến phạm vi.

Thông số kỹ thuật OpenID Connect gợi ý một vài tên phạm vi tiêu chuẩn để xác nhận loại ảnh xạ có thể hữu ích hoặc có thể tự do thiết kế chúng.

Một trong số chúng thực sự bắt buộc, phạm vi “openid”, yêu cầu nhà cung cấp trả lại các yêu cầu phụ trong mã token:

Enable: ☒

* Name:

* Display Name:

Description:

Required: ☐

Emphasize: ☐

Show In Discovery Document: ☒

User Claims:

Add new claim:

Hình 4.23 Xác định danh tính xác thực và ủy quyền.

Sử dụng Angular – Admin Dashboard để xác thực các tài nguyên API cần được bảo vệ. Từ dịch vụ Identity Resources có thể thêm mới, xóa phạm vi và chỉnh sửa các thông số chi tiết của các tài nguyên danh tính dùng để xác thực và ủy quyền. Bao gồm các thông số kỹ thuật:



Bảng 4.5 Thông số chi tiết danh tính xác thực và ủy quyền.

Enabled (mặc định: true)	Cho biết nếu tài nguyên này được kích hoạt và có thể được yêu cầu.
Name	Tên duy nhất của tài nguyên nhận dạng. Đây là giá trị mà khách hàng sẽ sử dụng cho tham số phạm vi trong yêu cầu ủy quyền.

Display Name	Giá trị này sẽ được sử dụng.
Description	Giá trị này sẽ được sử dụng.
Required	Chỉ định xem người dùng có thể bỏ chọn phạm vi trên màn hình đồng ý hay không.
Emphasize	Chỉ định xem màn hình đồng ý có nhấn mạnh phạm vi này hay không.
Show In Discovery Document	Chỉ định xem phạm vi này có được hiển thị trong tài liệu khám phá hay không.
User Claims	Danh sách các loại xác nhận quyền sở hữu của người dùng được liên kết cần được bao gồm trong mã thông báo nhận dạng.

4.6 Quản lý người dùng

Angular – Admin Dashboard cung cấp khả năng quản lý tất cả các tác vụ liên quan đến người dùng. Identity Server sử dụng thư viện ASP.NET Identity, cung cấp mọi chức năng cơ bản mà một ứng dụng quản lý người dùng thường có và nhà phát triển có thể bỏ những yêu cầu riêng.

Search user with id, name,... <input type="text"/>					
Index	Avatar	User Name	Full Name	Email	Action
0		admin	Admin	admin@admin.com	Edit Delete
1		xdxg	Phạm Văn Tĩnh	tinh_pham@outlook.com	Edit Delete

Hình 4.24. Quản lý danh sách thông tin người dùng.

* User Name: * Email:

* First Name: * Last Name:

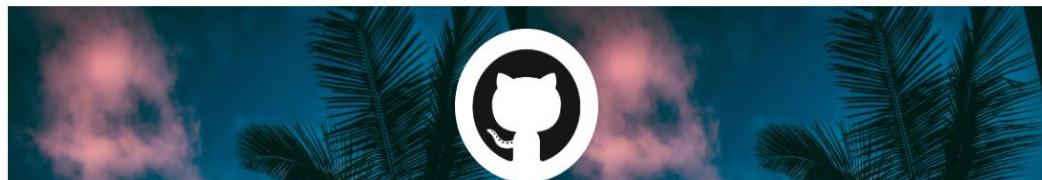
* Phone: * Date of Birth:

Avatar Uri:

* Password:

* Confirm Password:

Bảng 4.25 Thêm mới người dùng.



* User Id:

* User Name: * Email:

* First Name: * Last Name:

Phone: Date of Birth:

Create Date: Last Update:

Reset Password:

Avatar Uri:

User's Roles:

Bảng 4.26 Chỉnh sửa thông tin chi tiết và xóa tài khoản người dùng.

4.7 Quản lý phân quyền người dùng

Angular – Admin Dashboard cung cấp các chức năng quản lý ủy quyền cho từng người dùng truy cập vào các API, ứng dụng client. ASP.NET Identity cung cấp các khả năng có sẵn và có thể tùy biến thêm theo ý của nhà phát triển.

Search permission with id, name,...					
Index	Id	Name	User Permissions	Client Permissions	Delete
0	Admin	Admin	Edit	Edit	Delete
1	Member	Member	Edit	Edit	Delete

Bảng 4.27 Quản lý thông tin các nhóm phân quyền.

Search user permission with id, na...						
Index	Api Resource	View	Create	Update	Delete	Save
0	ADMIN_API	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Save
1	AUTH_SERVER	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Save
2	USER_API	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Save

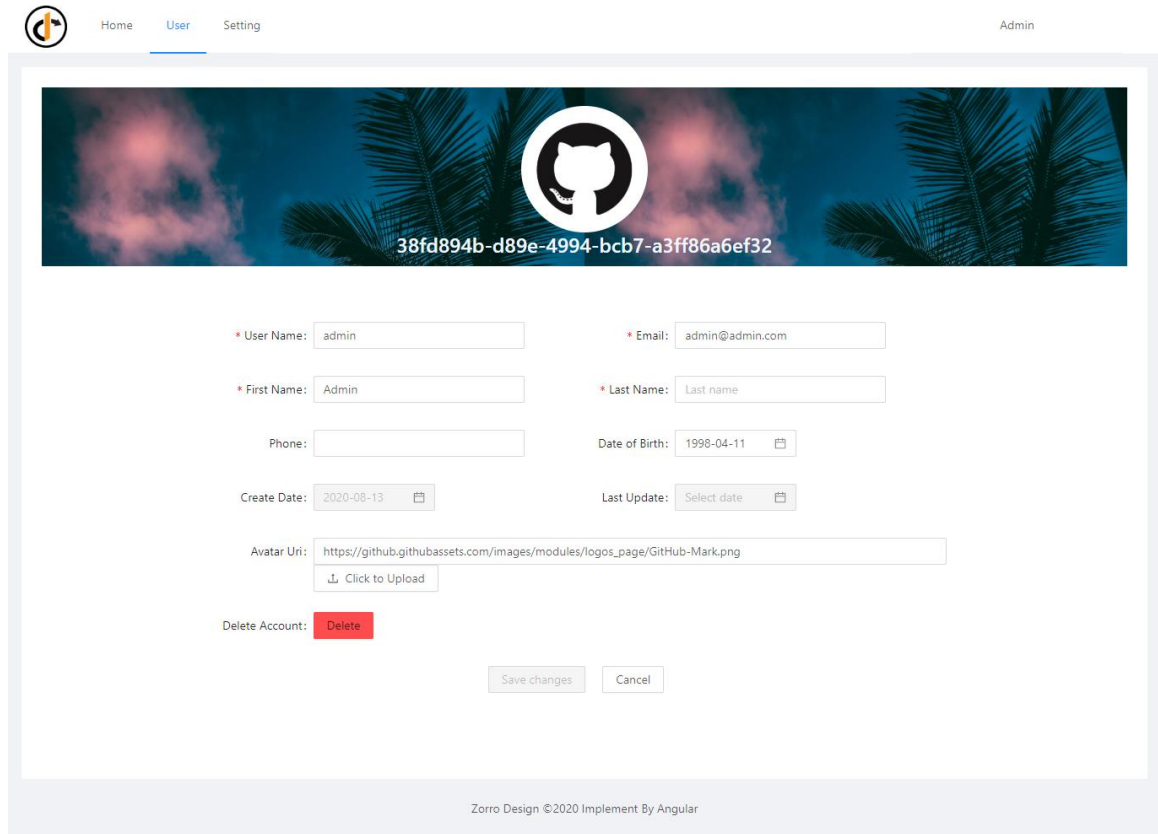
Bảng 4.28 Chỉnh sửa thông tin chi tiết ủy quyền truy cập API.

Search client permission with id, na...			
Index	Client Name	Enable	Save
0	Swagger Auth Server	<input type="checkbox"/>	Save
1	Swagger Admin Api	<input type="checkbox"/>	Save
2	Swagger User Api	<input type="checkbox"/>	Save
3	Angular Admin Dashboard	<input type="checkbox"/>	Save
4	Angular User Profile	<input type="checkbox"/>	Save
5	test-client-spa	<input type="checkbox"/>	Save

Bảng 4.29 Chỉnh sửa thông tin chi tiết ủy quyền truy cập ứng dụng client.

4.8 Người dùng tự quản lý thông tin cá nhân của chính mình

Angular – User Profile cung cấp khả năng quản lý thông tin của chính người dùng. Không yêu cầu quyền truy cập API. Có thể thay đổi thông tin cơ bản, avatar, mật khẩu và xóa tài khoản khi không sử dụng.



The screenshot displays the 'User' profile management interface. At the top, there is a navigation bar with 'Home', 'User' (active), and 'Setting' links, and a user role indicator 'Admin'. Below the navigation bar is a header banner featuring a GitHub logo and a unique identifier: '38fd894b-d89e-4994-bcb7-a3ff86a6ef32'. The main content area contains a form for updating user information. The form includes fields for 'User Name' (admin), 'Email' (admin@admin.com), 'First Name' (Admin), 'Last Name' (Last name), 'Phone', 'Date of Birth' (1998-04-11), 'Create Date' (2020-08-13), and 'Last Update' (Select date). There is also a field for 'Avatar Uri' with a URL and a 'Click to Upload' button. At the bottom of the form, there is a 'Delete Account' button labeled 'Delete', and 'Save changes' and 'Cancel' buttons. The footer of the page reads 'Zorro Design ©2020 Implement By Angular'.

Bảng 4.30 Trang thông tin cá nhân người dùng.

CHƯƠNG 5 – TỔNG KẾT

5.1 Kết quả đạt được

- Xác định được các ưu điểm và nhược điểm kiến trúc phần mềm nguyên khối, kiến trúc ứng dụng phần mềm hướng dịch vụ và kiến trúc Microservices.
- Áp dụng kiến trúc Microservices hướng API vào trong các ứng dụng hiện đại.
- Phân tách ứng dụng nguyên khối thành ứng dụng có kiến trúc Microservices.
- Thực hành với các công nghệ phổ biến của Microservice bao gồm: Docker, RESTful API, API Gateway, v.v.
- Thực hành với ứng dụng web Angular 10.
- Xây dựng máy chủ nhận dạng cung cấp dịch vụ xác thực và ủy quyền tập trung.

5.2 Vấn đề chưa được giải quyết

- Giao diện người dùng không được đẹp mắt người dùng. Chỉ tập trung vào các chức năng là chính.
- Xây dựng và thiết kế API ở một số dịch vụ không tuân thủ theo tiêu chuẩn REST
- Báo cáo còn nhiều sai sót.

CHƯƠNG 6 – TÀI LIỆU THAM KHẢO

- [1]. Docker: <https://docs.docker.com/>
- [2]. Ocelot: <https://ocelot.readthedocs.io/en/latest/introduction/gettingstarted.html>
- [3]. Nginx: <https://www.nginx.com/>
- [4]. Angular: <https://angular.io/>
- [5]. Ant Design of Angular: <https://ng.ant.design/docs/introduce/en>
- [6]. Swagger for Netcore: <https://docs.microsoft.com/en-us/aspnet/core/tutorials/getting-started-with-swashbuckle?view=aspnetcore-3.1&tabs=visual-studio>
- [7]. Serilog for Netcore: <https://serilog.net/>
- [8]. ASPNET Identity: <https://docs.microsoft.com/en-us/aspnet/identity/overview/getting-started/introduction-to-aspnet-identity#:~:text=ASP.NET%20Identity%20does%20not,users%20in%20the%20web%20site.>
- [9]. Identity Server 4: <https://identityserver4.readthedocs.io/en/latest/>
- [10]. Angular on Docker: <https://github.com/tieppt/angular-docker>
- [11]. JP Project for SSO: <https://github.com/brunohbrito/JPPProject.IdentityServer4.SSO>
- [12]. Host ASPNET Core on Linux with Nginx: <https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/linux-nginx?view=aspnetcore-3.1>
- [13]. Docker compose ASP.NET Core with SQL Server: <https://docs.docker.com/compose/aspnet-mssql-compose/>
- [14]. Bootstrap for Angular: <https://ng-bootstrap.github.io/#/home>
- [15]. vietname-dev: <https://github.com/vietnam-devs>
- [16]. OAuth 2.0: <https://www.oauth.com/>
- [17]. OpenID Connect: <https://openid.net/connect/>
- [18]. OIDC client: <https://github.com/IdentityModel/oidc-client-js>
- [19]. How Single Sign-On work: <https://www.onelogin.com/learn/how-single-sign-on-works>
- [20]. Microservices in Practice: <https://wso2.com/whitepapers/microservices-in-practice-key-architectural-concepts-of-an-msa/>
- [21]. Microservices architecture style: <https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices>

- [22]. Nguyên tắc thiết kế REST API: <https://medium.com/eway/nguy%C3%AAn-t%E1%BA%AFC-thi%E1%BA%BFt-k%E1%BA%BF-rest-api-23add16968d7>
- [23]. Monolithic vs Microservices architecture: <https://www.geeksforgeeks.org/monolithic-vs-microservices-architecture/?ref=rp>
- [24]. Fullstack mark blog: <https://fullstackmark.com/>
- [25]. Jason Watmore's Blog: <https://jasonwatmore.com/>
- [26]. CodAffection: <https://www.codaffection.com/>