BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC CẦN THƠ

**KHOA CÔNG NGHỆ THÔNG TIN & TRUYỀN THÔNG**



**LUẬN VĂN TỐT NGHIỆP ĐẠI HỌC**

**NGÀNH KỸ THUẬT PHẦN MỀM**

**ĐỀ TÀI**

**ỨNG DỤNG QUẢN LÝ LOÀI**

**DÙNG KIẾN TRÚC MICROSERVICE**

*Giảng viên hướng dẫn:*

**TS. Lâm Hoài Bảo**

*Sinh viên thực hiện:*

Họ và tên: **Vương Cẩm Thanh**

Mã số sinh viên: **B1805916**

Lớp: **Kỹ thuật phần mềm A3, K44**

**Học kỳ II, 2021-2022**

TRƯỜNG ĐẠI HỌC CẦN THƠ

**KHOA CÔNG NGHỆ THÔNG TIN & TRUYỀN THÔNG**

**BỘ MÔN CÔNG NGHỆ PHẦN MỀM**



**LUẬN VĂN TỐT NGHIỆP ĐẠI HỌC**

**NGÀNH KỸ THUẬT PHẦN MỀM**

**ĐỀ TÀI**

**ỨNG DỤNG QUẢN LÝ LOÀI**

**DÙNG KIẾN TRÚC MICROSERVICE**

**Giảng viên hướng dẫn:**

**TS. Lâm Hoài Bảo**

**Sinh viên thực hiện:**

Họ và tên: **Vương Cẩm Thanh**

Mã số sinh viên: **B1805916**

Khóa: **K44**

**Cần Thơ, 04/2022**

LỜI CẢM ƠN

Lời đầu tiên, em xin chân thành cảm ơn thầy Lâm Hoài Bảo đã hướng dẫn và giúp đỡ em rất nhiều để em có thể hoàn thành luận văn. Trong quá trình thực hiện luận văn đã gặp không ít những khó khăn, chậm trễ hơn so với tiến độ đặt ra nhưng nhờ có sự giúp đỡ, hướng dẫn và những lời khuyên, an ủi tận tình từ thầy, em đã có thể quản lý được tiến độ dự án và đảm bảo hoàn thành luận văn đúng thời gian quy định.

Ngoài ra, em cũng xin cảm ơn các Thầy Cô khoa Công nghệ thông tin và truyền thông đã tạo điều kiện cho em học tập và truyền đạt những kiến thức chuyên môn cũng như các kinh nghiệm trong suốt quá trình học tập tại trường để em có đủ khả năng nghiên cứu và thực hiện tốt đề tài này.

Mặc dù đã có nhiều cố gắng thực hiện đề tài một cách hoàn chỉnh nhất, nhưng do hạn chế về mặt thời gian nghiên cứu cũng như kiến thức và kinh nghiệm nên không thể tránh khỏi những thiếu sót nhất định. Em rất mong nhận được sự thông cảm cũng như sự góp ý của quý thầy, cô và các bạn để đề tài của em được hoàn chỉnh hơn.

Xin trân thành cảm ơn.

Cần Thơ, 27 tháng 04 năm 2022

Vương Cẩm Thanh

Mục lục

[A. Phần Giới Thiệu 8](#_Toc103153467)

[1. Đặt vấn đề 8](#_Toc103153468)

[2. Lịch sử giải quyết vấn đề 8](#_Toc103153469)

[3. Mục tiêu đề tài 11](#_Toc103153470)

[4. Đối tượng và phạm vi nghiên cứu 11](#_Toc103153471)

[5. Nội dung nghiên cứu 12](#_Toc103153472)

[6. Những đóng góp chính của đề tài 12](#_Toc103153473)

[7. Bố cục của quyển luận văn 13](#_Toc103153474)

[B. Phần Nội Dung 14](#_Toc103153475)

[Chương 1. Mô Tả Bài Toán 14](#_Toc103153476)

[1. Mô tả chi tiết bài toán 14](#_Toc103153477)

[2. Bối cảnh sản phẩm 14](#_Toc103153478)

[3. Các chức năng sản phẩm 15](#_Toc103153479)

[4. Đặc điểm của người sử dụng 16](#_Toc103153480)

[5. Môi trường vận hành 17](#_Toc103153481)

[6. Các ràng buộc thực thi và thiết kế 17](#_Toc103153482)

[7. Các giả định và phụ thuộc 18](#_Toc103153483)

[8. Các yêu cầu bên ngoài 18](#_Toc103153484)

[9. Các quy tắc nghiệp vụ 19](#_Toc103153485)

[10. Các tính năng của hệ thống 21](#_Toc103153486)

[11. Các yêu cầu phi chức năng 48](#_Toc103153487)

[Chương 2. Kiến Thức Nền 51](#_Toc103153488)

[1. Tổng quan về Microservices 51](#_Toc103153489)

[2. Kiến trúc Monolithic 52](#_Toc103153490)

[3. Định nghĩa về Microservices 53](#_Toc103153491)

[4. Đặc điểm của Microservices 54](#_Toc103153492)

[5. Mẫu thiết kế của kiến trúc Microservices 56](#_Toc103153493)

[6. .NET 6 LTS 62](#_Toc103153494)

[7. Blazor Web Assembly 63](#_Toc103153495)

[8. HealthChecks 63](#_Toc103153496)

[9. Seq 64](#_Toc103153497)

[10. RabbitMQ 64](#_Toc103153498)

[11. MongoDB 66](#_Toc103153499)

[12. Docker 68](#_Toc103153500)

[Chương 3. Phát Triển Phần Mềm Theo Kiến Trúc Microservices 70](#_Toc103153501)

[1. Mô hình hoá các microservices 70](#_Toc103153502)

[2. Triển khai 73](#_Toc103153503)

[3. Kiểm thử trong Microservices 74](#_Toc103153504)

[4. Sử dụng kiến trúc Microservices khi nào là hợp lý 75](#_Toc103153505)

[Chương 4. Identity Service 76](#_Toc103153506)

[1. Tổng quan service 76](#_Toc103153507)

[2. Microsoft Authentication 78](#_Toc103153508)

[3. Các hạn chế của Microsoft Authentication 78](#_Toc103153509)

[4. Xác thực bằng token: 79](#_Toc103153510)

[5. Các REST API quan trọng 80](#_Toc103153511)

[6. Cơ sở dữ liệu 98](#_Toc103153512)

[Chương 5. RabbitMQ 99](#_Toc103153513)

[Chương 6. MongoDB 99](#_Toc103153514)

[Chương 7. Kiểm thử và đánh giá kết quả 99](#_Toc103153515)

[1. Giới thiệu 99](#_Toc103153516)

[2. Kế hoạch kiểm thử 100](#_Toc103153517)

[3. Quản lý kiểm thử 102](#_Toc103153518)

danh mục hình

[Hình 1 Trang web của ứng dụng Netflix 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366893)

[Hình 2 Trang web của Amazon 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366894)

[Hình 3 Trang web của birdwatchingvietnam 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366895)

[Hình 4 Use case của Guest 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366896)

[Hình 5 Use case của Researcher 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366897)

[Hình 6 Use case của Expert 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366898)

[Hình 7 Các thành phần của kiến trúc Monolithic 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366899)

[Hình 8 Ví dụ kiến trúc Microservices 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366900)

[Hình 9 Mẫu kiến trúc SCS 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366901)

[Hình 10 Chia Frontend thành modules 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366902)

[Hình 11 Liên lạc không đồng bộ 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366903)

[Hình 12 Chức năng của nền tảng Microservices 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366904)

[Hình 13 Linh vật .NET 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366905)

[Hình 14 Logo Blazor 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366906)

[Hình 15 Logo watchdogs healthchecks 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366907)

[Hình 16 Logo Flask 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366908)

[Hình 17 Logo Seq 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366909)

[Hình 18 Logo SQL Server 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366910)

[Hình 19 RabbitMQ 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366911)

[Hình 20 MongoDb 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366912)

[Hình 21 Docker 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366913)

[Hình 22 Mô hình hoạt động của Docker 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366914)

[Hình 23 Kiến trúc tổng quan của hệ thống 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366915)

[Hình 24 Domain-Driven Design 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366916)

[Hình 25 JWT 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366917)

[Hình 26 Sequence Diagram Login 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366918)

[Hình 27 Sequence Diagram Get all users 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366919)

[Hình 28 Sequence Diagram Get user by id 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366920)

[Hình 29 Sequence Diagram Refresh token 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366921)

[Hình 30 Sequence Diagram Chage name 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366922)

[Hình 31 Sequence Diagram Change password 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366923)

[Hình 32 Sequence Diagram Create new user 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366924)

[Hình 33 Identity Class Diagram 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366925)

[Hình 34 Kiến trúc N-Layer 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366926)

[Hình 36 Sequence Diagram Create new species 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366927)

[Hình 37 Sequence Diagram Get all species 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366928)

[Hình 38 Sequence Diagram Get species by id 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366929)

[Hình 39 Sequence Diagram Get author species 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366930)

[Hình 40 Sequence Diagram Update species 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366931)

[Hình 41 Sequence Diagram Find species by name 9](file:///F:\Projects\C%23\LuanVan\documents\Luận%20Văn%20Tốt%20Nghiệp.docx#_Toc103366932)

Danh mục bảng

[Bảng 1 Bảng phân chia quyền sử dụng 8](#_Toc103366871)

[Bảng 2 Bảng so sánh Microservices và monolith 8](#_Toc103366872)

[Bảng 3 REST API Đăng Nhập 8](#_Toc103366873)

[Bảng 4 REST API Lấy tất cả người dùng 8](#_Toc103366874)

[Bảng 5 REST API Lấy người dùng theo id 8](#_Toc103366875)

[Bảng 6 REST API Refresh token 8](#_Toc103366876)

[Bảng 7 REST API Change name 8](#_Toc103366877)

[Bảng 8 REST API Change password 8](#_Toc103366878)

[Bảng 9 REST API Create user 8](#_Toc103366879)

[Bảng 10 Bảng người dùng 8](#_Toc103366880)

[Bảng 11 REST API Tạo loài mới 8](#_Toc103366881)

[Bảng 12 REST API Lấy tất cả loài 8](#_Toc103366882)

[Bảng 13 REST API Lấy loài theo id 8](#_Toc103366883)

[Bảng 14 REST API Lấy loài của tác giả 8](#_Toc103366884)

[Bảng 15 REST API Cập nhật loài 8](#_Toc103366885)

[Bảng 16 REST API Tìm loài theo tên 8](#_Toc103366886)

[Bảng 17 Trách nhiệm quyền hạn 9](#_Toc103366887)

[Bảng 18 : Kế hoạch dự đoán và chi phí 9](#_Toc103366888)

[Bảng 19 Các rủi ro 9](#_Toc103366889)

[Bảng 20 Kịch bản kiểm thử 9](#_Toc103366890)

BẢNG CHÚ GIẢI THUẬT NGỮ

|  |  |
| --- | --- |
| Thuật ngữ, khái niệm/từ viết tắt | Định nghĩa |
| HTTP | Hypertext Transfer Protocol |
| REST | Representational State Transfer |
| API | Application Programming Interface |
| URL | Uniform Resource Locator |
| SPA | Single Page Application |
| JWT | JSON Web Token |

TÓM LƯỢC

Với sự phát triển nền công nghiệp hiện đại đã làm biến đổi khí hậu, ô nhiễm môi trường,… Cùng với việc săn bắt trái phép, khai thác rừng,… dẫn đến sự sụt giảm nghiêm trọng của các loài động thực vật.

Do đó, các vườn quốc gia được thành lập nhằm đảm bảo sự đa dạng về sinh học, đóng góp về y học, lợi ích về nông nghiệp, điều tiết môi trường và nhiều lợi ích khác. Và để quản lý tốt hơn các loài vật ở đây cũng như là hội nhập cuộc cách mạng công nghiệp số hóa dữ liệu. Giải pháp là phát triển một phần mềm nhằm mục đích tra cứu quản lý thông tin về các loài vật, hỗ trợ công tác bảo tồn các loài ở đây sử dụng kiến trúc microservices.

Kiến trúc Microservices một kiểu kiến trúc phần mềm mới và đang rất phát triển hiện nay nhất là các nền tảng điện toán đám mây. Trong đó các dịch vụ được chia nhỏ để thực hiện một chức năng duy nhất của hệ thống. Việc chia nhỏ các dịch vụ trong kiến trúc Microservices giúp cho hệ thống, dễ mở rộng theo chiều ngang. Với kiến trúc cũ gọi là Monolithic thì tất các chức năng nằm chung một máy chủ và là kiến trúc mở rộng theo chiều sâu vì khi muốn mở rộng phải thêm ram hoặc cpu. Với Microservices thì khi mở rộng chỉ cần thêm một service khác là được. Kiến trúc Microservices được coi là lời giải ưu việt cho bài toán xây dựng và phát triển hệ thống dựa trên dịch vụ hiện nay.

Xuất phát từ những ý nghĩa thực tiễn như vậy, em đã thực hiện đề tài luận văn “Phát triển hệ thống quản lý loài dùng kiến trúc Microservice” để tìm hiểu và áp dụng kiến trúc Microservices trong việc xây dựng và phát triển một ứng dụng cụ thể - ứng dụng web “Species Wiki” giúp tra cứu quản lý thông tin về các loài vật, hỗ trợ công tác bảo tồn các loài ở đây sử dụng kiến trúc microservices.

# Phần Giới Thiệu

### Đặt vấn đề

Dịch vụ đám mây ngày càng phát triển đã giúp các nhà xây dựng và phát triển phần mềm tạo ra các hệ thống có khả năng thích ứng cao với nhiều môi trường khác nhau, tăng khả năng tái sử dụng. Các hệ thống được phát triển nhanh chóng và giảm được sự phức tạp, hạ giá thành khi xây dựng và triển khai.

Tuy nhiên việc không quan tâm đến kích thước của các dịch vụ trong hệ thống đang đặt ra bài toán khó cho các nhà xây dựng và phát triển phần mềm là làm sao giảm chi phí khi xây dựng hệ thống với các dịch vụ, làm sao tránh ảnh hưởng đến cả hệ thống khi muốn thay đổi một số chức năng của hệ thống,…

Phạm vi của dịch vụ càng lớn hệ thống càng trở lên phức tạp, khó phát triển, kiểm thử và bảo trì. Chính những điều này đang làm cho việc xây dựng và phát triển hệ thống phần mềm dựa trên dịch vụ đang vượt khỏi khả năng kiểm soát của các kiểu kiến trúc phần mềm hiện có và cần phải có một kiểu kiến trúc mới để giải quyết vấn đề này.

### Lịch sử giải quyết vấn đề

Sự phát triển của mạng internet, đã bắt đầu nhen nhóm về Microservices từ những năm 2004, nhưng chưa quá nổi bật. Nhưng đến những năm gần đây với sự bùng nổ của công nghệ với Smartphone, wifi vv… dẫn đến việc con người ngày càng tiếp cận với các dịch vụ công nghệ nhiều hơn, nhiều business công nghệ xuất hiện nhằm đáp ứng nhu cầu hiện nay. Thay vì trước đây chỉ có những hệ thống nhỏ, với những trang tin tức, thông tin, với lượng truy cập và tương tác không quá lớn, thì bây giờ những dịch vụ mạng xã hội, fintech phát triển đòi hỏi sự phát triển ngày càng nhanh để đáp ứng nhu cầu ngàng càng tăng đến người dùng. Do đó, sự bất lợi và nhược điểm của Monolithic bắt đầu xuất hiện.

Kiến trúc Microservices là một kiểu kiến trúc phần mềm mới và đang rất phát triển hiện nay. Trong đó các dịch vụ được chia nhỏ để thực hiện một chức năng duy nhất của hệ thống. Việc chia nhỏ các dịch vụ trong kiến trúc Microservices giúp cho hệ thống đơn giản hơn, dễ phát triển hơn, giảm chi phí xây dựng, tăng khả năng thích ứng công nghệ. Kiến trúc Microservices được coi là lời giải ưu việt cho bài toán xây dựng và phát triển hệ thống dựa trên dịch vụ hiện nay. Nó đã và đang được nghiên cứu và ứng dụng rộng rãi bởi các công ty lớn như Netflix, Ebay, Amazon, Twitter, Paypal, Gilt, Soundcloud,… Đặc biệt là hiện nay khi các sản phẩm phần mềm đóng gói đang dần được thay thế bởi các phần mềm dịch vụ thì kiến trúc Microservices sẽ là đề tài ngày càng được quan tâm.

Hiện nay, kiến trúc Microservices được coi là giải pháp tối ưu để các công ty coi như là lời giải ưu việc cho bài toán xây dựng và phát triển hệ thống dựa trên dịch vụ như hiện nay các công ty áp dụng kiến trúc này:

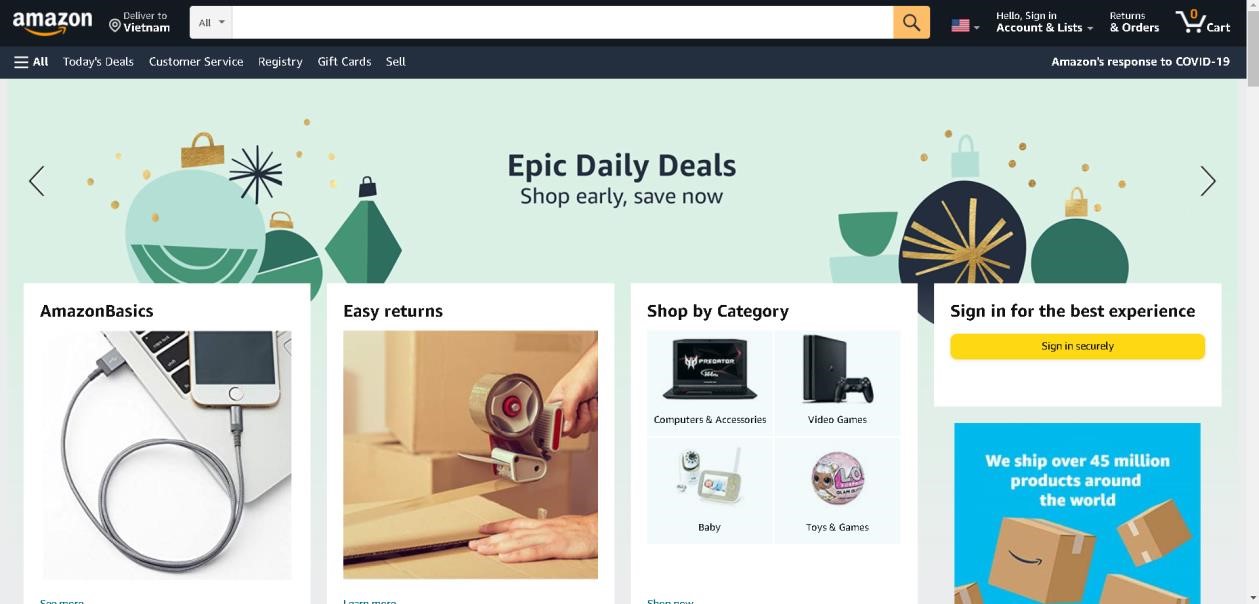
* <https://www.netflix.com/vn/>

Hình 1 Trang web của ứng dụng Netflix

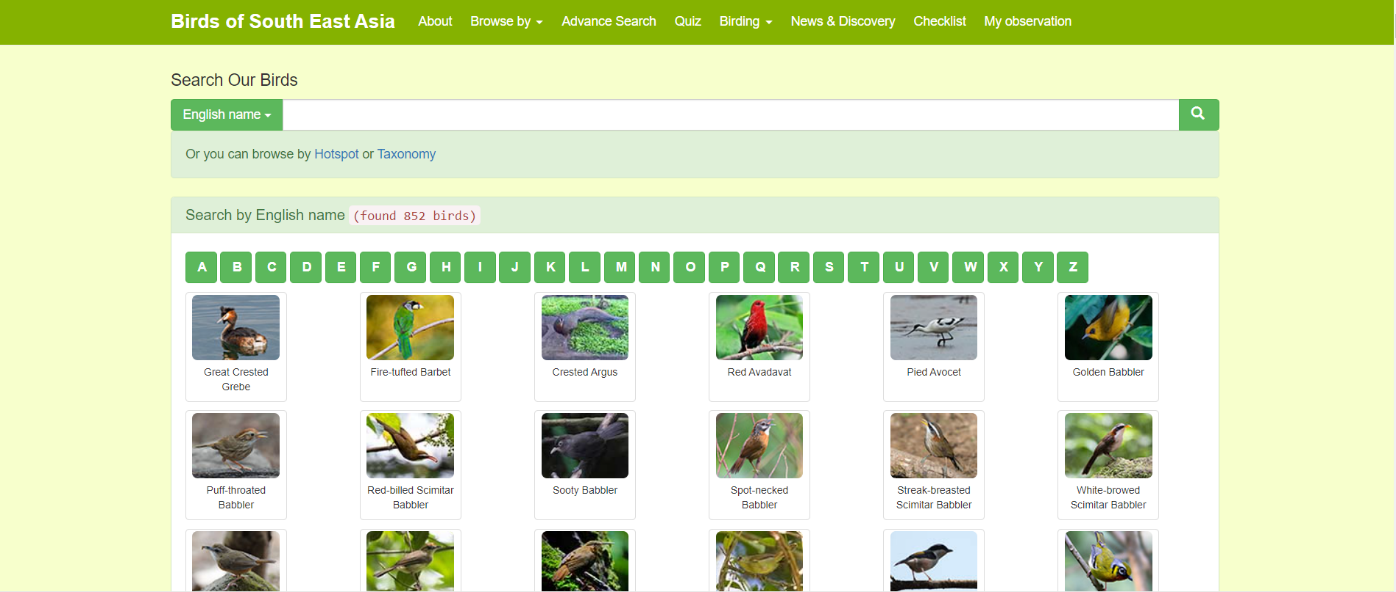
Netflix là một trong những công ty đi đầu trong việc phát triển Microservices, một công ty dịch vụ xem video trực tuyến của Mỹ, nội dung chủ yếu là phim và các chương trình truyền hình, rất phổ biến ở Mỹ và nhiều nước khác trên thế giới mang đến đa dạng các loại trương trình truyền hình, phim, phim tài liệu đoạt giải thưởng và nhiều nội dung khác trên hàng nghìn thiết bị có kết nối internet.

* <https://www.amazon.com/>

Amazon là công ty hàng đầu thế giới về lĩnh vực bán lẻ trực tuyến. Hơn thế, Amazon đang vươn lên mạnh mẽ với vai trò là nhà cung cấp dịch vụ điện toán đám mây. Xuất phát điểm của Amazon là trang bán sách. Về sau Amazon mở rộng với hoàng loạt hàng hóa tiêu dùng và phương tiện truyền thông kỹ thuật số cũng như các thiết bị điện tử của mình.



Hình 2 Trang web của Amazon

Và trang web cung cấp thông tin về loài vật, Bird watching Vietnam – website thông tin về các loài chim ở Việt Nam, Lào, Campuchia và Thái Lan. Ứng dụng cho phép xem danh sách các loài chim, lọc/tìm kiếm loài chim: <https://www.birdwatchingvietnam.net/>

Hình 3 Trang web của birdwatchingvietnam

### Mục tiêu đề tài

Xuất phát từ những ý nghĩa thực tiễn như vậy, em đã thực hiện đề tài luận văn “Phát triển hệ thống quản lý loài dùng kiến trúc Microservice” để tìm hiểu và áp dụng kiến trúc Microservices trong việc xây dựng và phát triển một ứng dụng cụ thể - ứng dụng web người dùng chia sẻ địa điểm, kinh nghiệm của họ về các loài vật cũng như giới thiệu địa điểm có thể quan sát sinh vật. Dựa trên việc áp dụng kiến trúc Microservices trong thực tế từ đó đưa ra các phân tích, đánh giá và rút ra các ưu nhược điểm của kiến trúc Microservices.

* Tìm hiểu nền tảng web đã phát triển để lấy dữ liệu và cập nhật tọa độ các loài vật.
* Tìm hiểu thêm một số framework và các kỹ thuật khác hiện đang được các công ty phần mềm sử dụng để tích hợp với Microservices.
* Tìm hiểu tích hợp Open Street Map vào web sử dụng ..
* Tìm hiểu MongoDb, SqlServer, RabbitMq, Seq, HealthChecks, Blazor, Open Street Map, Webassembly, SignalR, SPA, REST API, ...

Trang web có vài chức năng cụ thể sau:

* Sử dụng hai ngôn ngữ tiếng Anh và tiếng Việt. Tùy chỉnh màu sắc chủ đề.
* Hiển thị thông tin các loài vật hiện có. Tìm kiếm loài theo tên sử dụng machine learning, sắp xếp các loài theo bảng chữ cái.
* Hiển thị loài của một tác giả. Hiển thị thông tin tác giả.
* Tích hợp bản đồ vào hệ thống để hiển thị vị trí loài vật.
* Thông báo thời gian thực khi cập thông tin.
* Duyệt thông tin các loài vật mới.

### Đối tượng và phạm vi nghiên cứu

#### Đối tượng sử dụng:

* Người dùng bình thường (khách) xem thông tin và phân bố các loài động thực vật, định vị tọa độ bản thân, tìm kiếm nâng cao loài theo tên.
* Nhà nghiên cứu có quyền thêm thông tin, vị trí loài mới, sửa thông tin của loài.
* Chuyên gia duyệt bài viết của các nhà nghiên cứu.

#### Phạm vi nghiên cứu:

Mô hình kiến trúc Microservices và các công nghệ đi kèm. Mỗi phần em sẽ giới thiệu sơ lược và trình bày những nội dung cơ bản nhất, những điểm mạnh hay lợi ích mà nó mang lại cho các nhà phát triển phần mềm. Sau khi tìm hiểu em sẽ vận dụng kết quả tìm hiểu được vào việc phát triển hệ thống sử dụng kiến trúc Microservices nhằm mục đích minh họa cho phần lý thuyết đã trình bày.

### Nội dung nghiên cứu

Tìm hiểu và nghiên cứu các tài liệu về mô hình Microservices và các công nghệ có liên quan đến việc phát triển hệ thống như: .Net Framework, Docker, RabbitMq, MongoDb, Sql Server, Entity Framework,SignalR, Blazor WebAssembly, Seq, HealthChecks, … của tác giả trong và ngoài nước, các bài báo, thông tin trên mạng… sau đó chọn lọc và sắp xếp lại theo ý tưởng của mình. Dựa trên kết quả tìm hiểu được để phát triển hệ thống có áp dụng tất cả những nội dung đã nghiên cứu nhằm mục đích minh họa cho phần cơ sở lý thuyết sẽ trình bày trong nội dung luận văn này.

### Những đóng góp chính của đề tài

* Kết quả nghiên cứu có thể làm tài liệu tham khảo.
* Phần nghiên cứu lý thuyết sẽ cung cấp một cách nhìn tổng quát về quá trình phát triển phần mềm theo kiến trúc Microservices, ưu nhược điểm và các điểm liên quan.
* Phát triển thành công một trang web sử dụng kiến trúc Microservices, đầy đủ các chức năng cơ bản đã đề ra.
* Xây dựng giao diện hài hòa, thân thiện với người dùng.
* Giao diện hệ thống tương thích với nhiều thiết bị khác nhau, chạy tốt trên nhiều trình duyệt phổ biến.
* Tính bảo mật cao.
* Hiệu xuất ứng dụng cao, tốc độ load dữ liệu nhanh.
* Dễ mở rộng hệ thống.

### Bố cục của quyển luận văn

* **Phần giới thiệu**: nêu ra các vấn đề cần giải quyết, lý do tại sao phải thực hiện đề tài này, trong quá khứ đã có các hệ thống, các website nào tương tự được xây dựng để giải quyết các vấn đề đã đặt ra và chưa giải quyết được những vấn đề nào, qua đó xác định được mục tiêu của đề tài, đó cũng chính là những vấn đề trọng tâm mà đề tài đang thực hiện muốn giải quyết cũng như xác định được đối tượng và phạm vi nghiên cứu của đề tài.

# Phần Nội Dung

## Mô Tả Bài Toán

### Mô tả chi tiết bài toán

Hệ thống có những yêu cầu sau:

* Đối với người dùng bình thường (khách):
  + Xem danh sách loài
  + Tìm kiếm loài
  + Xem danh sách các tác giả
  + Xem danh sách các loài của tác giả
  + Xem chi tiết loài
  + Xem vị trí loài trên bản đồ
* Đối với nhà nghiên cứu (Researcher):
  + Xem thông tin cá nhân
  + Cập nhật thông tin
  + Xem danh sách loài của cá nhân
  + Tạo loài mới
  + Sửa thông tin loài
* Đối với chuyên gia (Expert):
  + Xem danh sách các loài cần duyệt
  + Sửa loài cần duyệt
  + Xem thống kê các loài
  + Quản lý các nhà nghiên cứu

### Bối cảnh sản phẩm

Với sự phát triển nền công nghiệp hiện đại đã làm biến đổi khí hậu, ô nhiễm môi trường,… Cùng với việc săn bắt trái phép, khai thác rừng,… dẫn đến sự sụt giảm nghiêm trọng của các loài động thực vật.

Do đó, các vườn quốc gia được thành lập nhằm đảm bảo sự đa dạng về sinh học, đóng góp về y học, lợi ích về nông nghiệp, điều tiết môi trường và nhiều lợi ích khác. Và để quản lý tốt hơn các loài vật ở đây cũng như là hội nhập cuộc cách mạng công nghiệp số hóa dữ liệu. Giải pháp là phát triển một phần mềm nhằm mục đích tra cứu quản lý thông tin về các loài vật, hỗ trợ công tác bảo tồn các loài ở đây sử dụng kiến trúc microservices.

Kiến trúc Microservices một kiểu kiến trúc phần mềm mới và đang rất phát triển hiện nay nhất là các nền tảng điện toán đám mây. Trong đó các dịch vụ được chia nhỏ để thực hiện một chức năng duy nhất của hệ thống. Việc chia nhỏ các dịch vụ trong kiến trúc Microservices giúp cho hệ thống, dễ mở rộng theo chiều ngang. Với kiến trúc cũ gọi là Monolithic thì tất các chức năng nằm chung một máy chủ và là kiến trúc mở rộng theo chiều sâu vì khi muốn mở rộng phải thêm ram hoặc cpu. Với Microservices thì khi mở rộng chỉ cần thêm một service khác là được. Kiến trúc Microservices được coi là lời giải ưu việt cho bài toán xây dựng và phát triển hệ thống dựa trên dịch vụ hiện nay.

Xuất phát từ những ý nghĩa thực tiễn như vậy, em đã thực hiện đề tài luận văn “Phát triển hệ thống quản lý loài dùng kiến trúc Microservice” để tìm hiểu và áp dụng kiến trúc Microservices trong việc xây dựng và phát triển một ứng dụng cụ thể - ứng dụng web “Species Wiki” giúp tra cứu quản lý thông tin về các loài vật, hỗ trợ công tác bảo tồn các loài ở đây sử dụng kiến trúc microservices.

### Các chức năng sản phẩm

* Các người dùng bình thường (khách):
* Xem danh sách loài
* Tìm kiếm loài theo tên
* Tìm kiếm loài theo hình
* Lọc loài
* Gom loài theo bảng chữ cái
* Xem danh sách các tác giả
* Xem danh sách các loài của tác giả
* Xem chi tiết loài
* Xem vị trí loài trên bản đồ
* Gợi ý loài tương tự
* Xem các hình của loài
* Chuyển đổi ngôn ngữ
* Chỉnh màu nền
* Đăng nhập
* Đối với nhà nghiên cứu (Researcher):
* Xem thông tin cá nhân
* Sửa tên
* Đổi mật khẩu
* Xem danh sách loài của cá nhân
* Tạo loài mới
* Sửa thông tin loài
* Đăng xuất
* Đối với chuyên gia (Expert):
* Xem danh sách các loài cần duyệt
* Sửa loài cần duyệt
* Xem thống kê các loài
* Tạo nhà nghiên cứu mới
* Sửa thông tin nhà nghiên cứu
* Khóa tài khoản nhà nghiên cứu
* Reset mật khẩu tài khoản

### Đặc điểm của người sử dụng

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nhóm**  **người sử dụng** | **Đặc trưng** | **Vai trò** | **Quyền hạn** | **Mức độ quan trọng** |
| Expert | Là người chịu trách nhiệm quản lý toàn bộ hệ thống | Expert | Expert | Rất quan trọng |
| Người dùng  chưa có  tài  khoản | Là người dùng có thể tạo tài khoản để sử dụng các chức năng của hệ thống | Người dùng bình thường | Người dùng bình thường | Quan trọng |
| Researcher | Là người dùng đã đăng ký tài khoản, người thực hiện quản lý các bài viết của hệ thống | Researcher | Researcher | Quan trọng |

Bảng 1 Bảng phân chia quyền sử dụng

### Môi trường vận hành

* Phần cứng:
  + Máy tính có kết nối internet.
  + Ram: 12GB
  + HDD: 500GB
  + CPU: Intel Core i5
* Hệ điều hành và phần mềm:
  + Hệ điều hành: Windows 11
  + Trình duyệt web: Google Chrome, Microsoft Edge
  + Phần mềm API: Postman

### Các ràng buộc thực thi và thiết kế

* Thực thi:
  + Máy chủ phải được đặt trong môi trường, nhiệt độ đủ tốt để hoạt động.
  + Phải có kết nối internet ổn định.
  + Thời gian downtime của các service ít nhất là 30s.
  + Thời gian phản hồi ngắn.
  + Hoạt động 24/7.
  + Đáp ứng được tối thiểu 1000 người dùng cùng lúc.
  + Website được tải ổn định trên các thiết bị, hạn chế tình trạng quá tải dẫn đến chậm lag gây mất thời gian của người dùng.
  + Có độ bảo mật cao, khó bị tấn công và sập bất ngờ
  + Đường truyền mạng của hệ thống ổn định, có đường truyền sẵn sàng thay thế
* Thiết kế:
  + Sử dụng mô hình Client-Server và chuẩn REST API.
  + Sử dụng JWT để xác thực không trạng thái (stateless) trên web.
  + Ngôn ngữ lập trình sử dụng: C#, Python.
  + Giao diện đơn giản, thân thiện với người sử dụng, giảm số bước thao tác khi thực hiện một chức năng.
  + Hệ quản trị CSDL: MongoDb, SQL Server.
  + Thông điệp giao tiếp giữa client và server bằng cách gửi nhận các gói tin dưới dạng JSON.

### Các giả định và phụ thuộc

* Tài khoản người dùng có thể bị tấn công.
* Máy chủ có thể bị nhiễm viruses hoặc hacker tấn công làm hỏng dữ liệu.
* Máy chủ có thể bị quá tải do vận hành trong thời gian dài.
* Đường truyền mạng không ổn định làm gián đoạn kết nối.
* Bàn giao sản phẩm không đúng thời hạn
* Tốc độ phản hồi của hệ thống phụ thuộc vào nhiều yếu tố: phần cứng của người dùng, tốc độ mạng, băng thông của nhà cung cấp dịch vụ mạng

### Các yêu cầu bên ngoài

* Giao diện người dùng:
  + Giao diện nền web: sử dụng Bootstrap, CSS, Material Design.
  + Giao diện thân thiện với người dùng.
  + Màu sắc: Đơn giản, dễ nhìn, không gây rối mắt
  + Thanh header đặt cố định trên cùng của trang để người dùng dễ tương tác
  + Các thành phần của giao diện phải thống nhất
  + Có popup thông báo.
* Giao tiếp phần cứng:
  + Services: giao tiếp qua mạng internet.
  + Client: giao tiếp với người dùng qua màn hình và các thiết bị nhập xuất (bàn phím, chuột, màn hình cảm ứng,…).
* Giao tiếp phần mềm:
  + Cơ sở dữ liệu: SQL Service, MongoDb.
  + Hệ điều hành: Windows/Linux(Ubuntu).
  + Các phần mềm khác: Seq, RabbitMQ.
* Giao tiếp truyền thông tin:
  + Trình duyệt web: Google Chrome, Microsoft Edge, FireFox (hỗ trợ tốt nhất cho Google Chrome).
  + Định dạng thông điệp: truyền theo dạng json.
  + Cần internet (wifi, dữ liệu di động,…) để làm cầu nối giữa client và các services
  + Sử dụng giao thức TCP/IP để truyền và nhận dữ liệu giữa service với client

### Các quy tắc nghiệp vụ

#### Thêm email

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Mật khẩu hợp lệ  Email đúng định dạng  Email không trùng | **Y**  **Y**  **Y** | **N**  **\***  **\*** | **\***  **N**  **\*** | **\***  **\***  **N** |
| Thêm thành công  Báo lỗi | **X** | **X** | **X** | **X** |

#### Đổi mật khẩu

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Mật khẩu cũ hợp lệ  Mật khẩu cũ chính xác  Mật khẩu mới hợp lệ | **Y**  **Y**  **Y** | **N**  **\***  **\*** | **\***  **N**  **\*** | **\***  **\***  **N** |
| Sửa thành công  Báo lỗi | **X** | **X** | **X** | **X** |

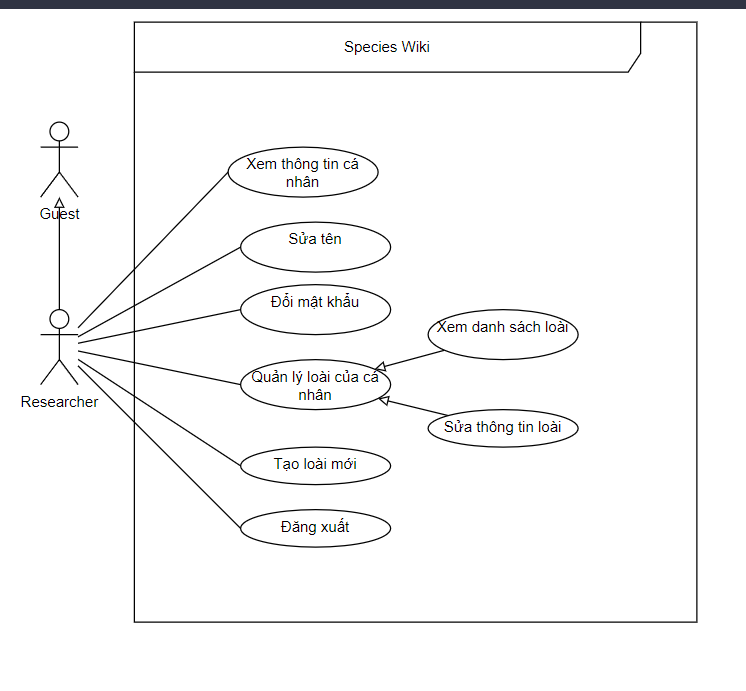
#### Đăng nhập

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Email hợp lệ  Mật khẩu hợp lệ  Đúng email  Đúng mật khẩu  Email không bị vô hiệu | **Y**  **Y**  **Y**  **Y**  **Y** | **N**  **\***  **\***  **\***  **\*** | **\***  **N**  **\***  **\***  **\*** | **Y**  **Y**  **N**  **\***  **\*** | **Y**  **Y**  **\***  **N**  **\*** | **Y**  **Y**  **Y**  **Y**  **N** |
| Đăng nhập thành công  Báo lỗi không hợp lệ  Báo lỗi sai thông tin đăng nhập | **X** | **X** | **X** | **X** | **X** | **X** |

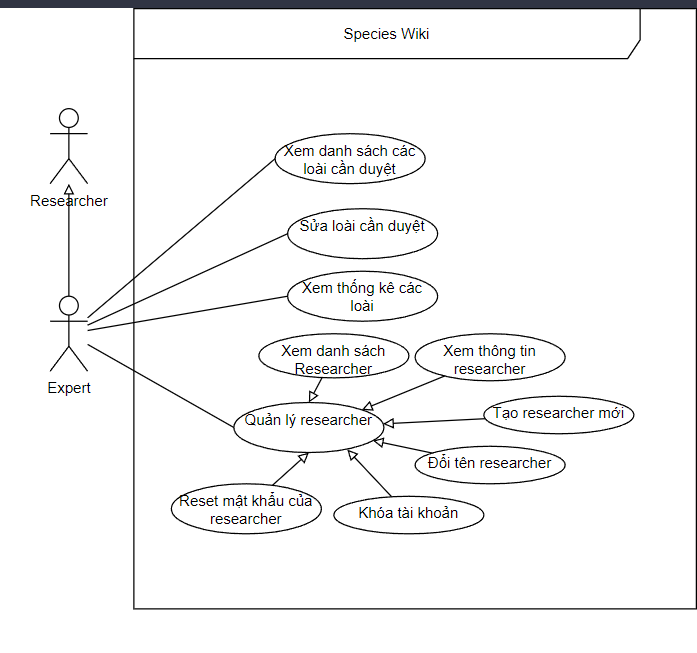
### Các tính năng của hệ thống



Hình 4 Use case của Guest



Hình 5 Use case của Researcher



Hình 6 Use case của Expert

#### Xem danh sách loài

|  |  |
| --- | --- |
| **Tên use case:** Xem danh sách loài | **ID:** UC-1 |
| **Actor chính:** Guest, Researcher, Expert | **Mức độ cần thiết:** Phải có |
| **Phân loại:** Cao |
| **Các thành phần tham gia và mối quan tâm*:***  **Xem danh sách chi tiết sản phẩm:** Khi tác nhân muốn xem danh sách các loài. | |
| **Mô tả tóm tắt:**  Tác nhân có nhu cầu xem danh sách các loài có trong hệ thống, hệ thống sẽ trả về danh sách loài có trong hệ thống. | |
| **Trigger:** Khi tác nhân nhấp vào xem danh sách loài.  **Type:** External | |
| **Các mối quan hệ:**  **+Association (kết hợp):** Guest  **+Include(bao gồm):** Không  **+Extend(mở rộng):** Không  **+Generalization(tổng quát hóa):** Không | |
| **Luồng xử lý bình thường của sự kiện:**   1. Tác nhân nhấp vào xem danh sách loài. 2. Hệ thống hiển thị danh sách các loài lên màn hình. 3. Kết thúc một sự kiện. | |
| **Các luồng sự kiện con:** | |
| **Luồng luân phiên/đặc biệt (Alternate/Exceptional flows)**: | |

#### Tìm kiếm loài theo tên

|  |  |
| --- | --- |
| **Tên use case:** Tìm kiếm loài theo tên | **ID:** UC-2 |
| **Actor chính:** Guest, Researcher, Expert | **Mức độ cần thiết:** Phải có |
| **Phân loại:** Cao |
| **Các thành phần tham gia và mối quan tâm*:***  **Xem danh sách chi tiết sản phẩm:** Khi tác nhân muốn tìm loài theo tên. | |
| **Mô tả tóm tắt:**  Khi tác nhân có nhu cầu tìm loài có trong hệ thống, hệ thống sẽ trả về danh sách loài có trong hệ thống. | |
| **Trigger:** Khi tác nhân nhấp vào tìm loài theo tên.  **Type:** External | |
| **Các mối quan hệ:**  **+Association (kết hợp):** Guest  **+Include(bao gồm):** Không  **+Extend(mở rộng):** Không  **+Generalization(tổng quát hóa):** Không | |
| **Luồng xử lý bình thường của sự kiện:**   1. Tác nhân nhập tên loài vào ô tìm kiếm tên loài 2. Tác nhân nhấn vào nút tìm kiếm. 3. Hệ thống hiển thị danh sách các loài lên màn hình. 4. Kết thúc một sự kiện. | |
| **Các luồng sự kiện con:** | |
| **Luồng luân phiên/đặc biệt (Alternate/Exceptional flows)**:  **E1:** Tác nhân nhập tên chưa chính xác thì hệ thống sẽ tìm tương đối theo tên. | |

#### Tìm kiếm loài theo hình

|  |  |
| --- | --- |
| **Tên use case:** Tìm kiếm loài theo hình | **ID:** UC-3 |
| **Actor chính:** Guest, Researcher, Expert | **Mức độ cần thiết:** Nên có |
| **Phân loại:** Cao |
| **Các thành phần tham gia và mối quan tâm*:***  **Xem danh sách chi tiết sản phẩm:** Khi tác nhân muốn tìm loài theo hình. | |
| **Mô tả tóm tắt:**  Khi tác nhân có nhu cầu tìm loài có trong hệ thống, hệ thống sẽ trả về danh sách loài có trong hệ thống. | |
| **Trigger:** Khi tác nhân nhấp vào tìm loài theo hình.  **Type:** External | |
| **Các mối quan hệ:**  **+Association (kết hợp):** Guest  **+Include(bao gồm):** Không  **+Extend(mở rộng):** Không  **+Generalization(tổng quát hóa):** Không | |
| **Luồng xử lý bình thường của sự kiện:**   1. Tác nhân chọn hình loài 2. Tác nhân nhấn vào nút tìm kiếm. 3. Hệ thống hiển thị danh sách các loài lên màn hình. 4. Kết thúc một sự kiện. | |
| **Các luồng sự kiện con:** | |
| **Luồng luân phiên/đặc biệt (Alternate/Exceptional flows)**: | |

#### Xem danh sách tác giả

|  |  |
| --- | --- |
| **Tên use case:** Xem danh sách các tác giả | **ID:** UC-4 |
| **Actor chính:** Guest, Researcher, Expert | **Mức độ cần thiết:** Nên có |
| **Phân loại:** Trung bình |
| **Các thành phần tham gia và mối quan tâm*:***  **Xem danh sách chi tiết sản phẩm:** Khi tác nhân muốn xem danh sách các tác giả. | |
| **Mô tả tóm tắt:**  Khi tác nhân có nhu cầu xem các tác giả trong hệ thống, hệ thống sẽ trả về danh sách có trong hệ thống. | |
| **Trigger:** Khi tác nhân nhấp vào xem danh sách tác giả.  **Type:** External | |
| **Các mối quan hệ:**  **+Association (kết hợp):** Guest  **+Include(bao gồm):** Không  **+Extend(mở rộng):** Không  **+Generalization(tổng quát hóa):** Không | |
| **Luồng xử lý bình thường của sự kiện:**   1. Tác nhân chọn xem danh sách tác giả 2. Hệ thống hiển thị danh sách lên màn hình. 3. Kết thúc một sự kiện. | |
| **Các luồng sự kiện con:** | |
| **Luồng luân phiên/đặc biệt (Alternate/Exceptional flows)**: | |

#### Xem danh sách loài của tác giả

|  |  |
| --- | --- |
| **Tên use case:** Xem danh sách các tác giả | **ID:** UC-5 |
| **Actor chính:** Guest, Researcher, Expert | **Mức độ cần thiết:** Nên có |
| **Phân loại:** Trung bình |
| **Các thành phần tham gia và mối quan tâm*:***  **Xem danh sách chi tiết sản phẩm:** Khi tác nhân muốn xem danh sách loài của các tác giả. | |
| **Mô tả tóm tắt:**  Khi tác nhân có nhu cầu xem loài của các tác giả trong hệ thống, hệ thống sẽ trả về danh sách có trong hệ thống. | |
| **Trigger:** Khi tác nhân nhấp vào xem danh sách loài của tác giả.  **Type:** External | |
| **Các mối quan hệ:**  **+Association (kết hợp):** Guest  **+Include(bao gồm):** Không  **+Extend(mở rộng):** Không  **+Generalization(tổng quát hóa):** Không | |
| **Luồng xử lý bình thường của sự kiện:**   1. Tác nhân chọn xem danh sách loài của tác giả 2. Hệ thống hiển thị danh sách lên màn hình. 3. Kết thúc một sự kiện. | |
| **Các luồng sự kiện con:** | |
| **Luồng luân phiên/đặc biệt (Alternate/Exceptional flows)**: | |

#### Xem chi tiết loài

|  |  |
| --- | --- |
| **Tên use case:** Xem chi tiết loài | **ID:** UC-6 |
| **Actor chính:** Guest, Researcher, Expert | **Mức độ cần thiết:** Phải có |
| **Phân loại:** Cao |
| **Các thành phần tham gia và mối quan tâm*:***  **Xem danh sách chi tiết sản phẩm:** Khi tác nhân muốn xem thông tin loài. | |
| **Mô tả tóm tắt:**  Khi tác nhân có nhu cầu xem thông tin loài trong hệ thống, hệ thống sẽ trả về thông tin chi tiết của loài trong hệ thống. | |
| **Trigger:** Khi tác nhân nhấp vào xem chi tiết loài.  **Type:** External | |
| **Các mối quan hệ:**  **+Association (kết hợp):** Guest  **+Include(bao gồm):** Không  **+Extend(mở rộng):** Không  **+Generalization(tổng quát hóa):** Không | |
| **Luồng xử lý bình thường của sự kiện:**   1. Tác nhân chọn xem chi tiết loài 2. Hệ thống hiển thị danh sách lên màn hình. 3. Kết thúc một sự kiện. | |
| **Các luồng sự kiện con:** | |
| **Luồng luân phiên/đặc biệt (Alternate/Exceptional flows)**: | |

#### Chuyển ngôn ngữ

|  |  |
| --- | --- |
| **Tên use case:** Chuyển ngôn ngữ | **ID:** UC-7 |
| **Actor chính:** Guest, Researcher, Expert | **Mức độ cần thiết:** Nên có |
| **Phân loại:** Cao |
| **Các thành phần tham gia và mối quan tâm*:***  **Xem danh sách chi tiết sản phẩm:** Khi tác nhân muốn chuyển ngôn ngữ của web. | |
| **Mô tả tóm tắt:**  Khi tác nhân có nhu cầu chuyển ngôn ngữ. | |
| **Trigger:** Khi tác nhân nhấp vào chuyển ngôn ngữ.  **Type:** External | |
| **Các mối quan hệ:**  **+Association (kết hợp):** Guest  **+Include(bao gồm):** Không  **+Extend(mở rộng):** Không  **+Generalization(tổng quát hóa):** Không | |
| **Luồng xử lý bình thường của sự kiện:**   1. Tác nhân chọn chuyển ngôn ngữ 2. Hệ thống hiển thị danh sách các ngôn ngữ. 3. Tác nhân chọn ngôn ngữ khác. 4. Hệ thống hiển thị ngôn ngữ mới. 5. Kết thúc một sự kiện. | |
| **Các luồng sự kiện con:** | |
| **Luồng luân phiên/đặc biệt (Alternate/Exceptional flows)**: | |

#### Đổi màu nền

|  |  |
| --- | --- |
| **Tên use case:** Đổi màu nền | **ID:** UC-8 |
| **Actor chính:** Guest, Researcher, Expert | **Mức độ cần thiết:** Nên có |
| **Phân loại:** Trung bình |
| **Các thành phần tham gia và mối quan tâm*:***  **Xem danh sách chi tiết sản phẩm:** Khi tác nhân muốn đổi màu nền của web. | |
| **Mô tả tóm tắt:**  Khi tác nhân có nhu cầu đổi màu nền. | |
| **Trigger:** Khi tác nhân nhấp vào đổi màu nền.  **Type:** External | |
| **Các mối quan hệ:**  **+Association (kết hợp):** Guest  **+Include(bao gồm):** Không  **+Extend(mở rộng):** Không  **+Generalization(tổng quát hóa):** Không | |
| **Luồng xử lý bình thường của sự kiện:**   1. Tác nhân chọn đổi màu nền 2. Hệ thống hiển thị màu mới. 3. Kết thúc một sự kiện. | |
| **Các luồng sự kiện con:** | |
| **Luồng luân phiên/đặc biệt (Alternate/Exceptional flows)**: | |

#### Đăng nhập

|  |  |
| --- | --- |
| **Tên use case:** Đăng nhập | **ID:** UC-9 |
| **Actor chính:** Guest | **Mức độ cần thiết:** Phải có |
| **Phân loại:** Cao |
| **Các thành phần tham gia và mối quan tâm*:***  **Xem danh sách chi tiết sản phẩm:** Khi tác nhân muốn đăng nhập. | |
| **Mô tả tóm tắt:**  Khi tác nhân muốn đăng nhập vào hệ thống để thực các chức năng của hệ thống. | |
| **Trigger:** Khi tác nhân nhấp vào đăng nhập.  **Type:** External | |
| **Các mối quan hệ:**  **+Association (kết hợp):** Guest  **+Include(bao gồm):** Không  **+Extend(mở rộng):** Không  **+Generalization(tổng quát hóa):** Không | |
| **Luồng xử lý bình thường của sự kiện:**   1. Tác nhân chọn đăng nhập 2. Hệ thống hiển thị ô nhập tài khoản và mật khẩu. 3. Tác nhân nhập vào tài khoản và mật khẩu. 4. Tác nhân nhấn nút đăng nhập 5. Kết thúc một sự kiện. | |
| **Các luồng sự kiện con:**  **S1:** Ở bước 3 nếu chiều dài của mật khẩu hoặc tài khoản không hợp lệ sẽ báo lỗi | |
| **Luồng luân phiên/đặc biệt (Alternate/Exceptional flows)**: | |

#### Xem thông tin cá nhân

|  |  |
| --- | --- |
| **Tên use case:** Xem thông tin cá nhân | **ID:** UC-10 |
| **Actor chính:** Researcher, Expert | **Mức độ cần thiết:** Phải có |
| **Phân loại:** Cao |
| **Các thành phần tham gia và mối quan tâm*:***  **Xem danh sách chi tiết sản phẩm:** Khi tác nhân muốn xem thông tin cá nhân. | |
| **Mô tả tóm tắt:**  Khi tác nhân muốn xem thông tin cá nhân của mình trong hệ thống. | |
| **Trigger:** Khi tác nhân nhấp vào xem thông tin cá nhân.  **Type:** External | |
| **Các mối quan hệ:**  **+Association (kết hợp):** Researcher  **+Include(bao gồm):** Không  **+Extend(mở rộng):** Không  **+Generalization(tổng quát hóa):** Không | |
| **Luồng xử lý bình thường của sự kiện:**   1. Tác nhân chọn xem thông tin cá nhân 2. Hệ thống hiển thị thông tin cá nhân 3. Kết thúc một sự kiện. | |
| **Các luồng sự kiện con:** | |
| **Luồng luân phiên/đặc biệt (Alternate/Exceptional flows)**: | |

#### Sửa thông tin cá nhân

|  |  |
| --- | --- |
| **Tên use case:** Sửa thông tin cá nhân | **ID:** UC-11 |
| **Actor chính:** Researcher, Expert | **Mức độ cần thiết:** Phải có |
| **Phân loại:** Cao |
| **Các thành phần tham gia và mối quan tâm*:***  **Xem danh sách chi tiết sản phẩm:** Khi tác nhân muốn sửa thông tin cá nhân. | |
| **Mô tả tóm tắt:**  Khi tác nhân muốn sửa thông tin cá nhân của mình trong hệ thống. | |
| **Trigger:** Khi tác nhân nhấp vào sửa thông tin cá nhân.  **Type:** External | |
| **Các mối quan hệ:**  **+Association (kết hợp):** Researcher  **+Include(bao gồm):** Không  **+Extend(mở rộng):** Không  **+Generalization(tổng quát hóa):** Không | |
| **Luồng xử lý bình thường của sự kiện:**   1. Tác nhân chọn sửa thông tin cá nhân 2. Hệ thống hiển thị thông tin cá nhân 3. Tác nhân sửa thông tin 4. Kết thúc một sự kiện. | |
| **Các luồng sự kiện con:** | |
| **Luồng luân phiên/đặc biệt (Alternate/Exceptional flows)**: | |

#### Đổi mật khẩu

|  |  |
| --- | --- |
| **Tên use case:** Đổi mật khẩu | **ID:** UC-12 |
| **Actor chính:** Researcher, Expert | **Mức độ cần thiết:** Phải có |
| **Phân loại:** Cao |
| **Các thành phần tham gia và mối quan tâm*:***  **Xem danh sách chi tiết sản phẩm:** Khi tác nhân muốn đổi mật khẩu. | |
| **Mô tả tóm tắt:**  Khi tác nhân muốn đổi mật khẩu của mình trong hệ thống. | |
| **Trigger:** Khi tác nhân nhấp vào đổi mật khẩu.  **Type:** External | |
| **Các mối quan hệ:**  **+Association (kết hợp):** Researcher  **+Include(bao gồm):** Không  **+Extend(mở rộng):** Không  **+Generalization(tổng quát hóa):** Không | |
| **Luồng xử lý bình thường của sự kiện:**   1. Tác nhân chọn đổi mật khẩu 2. Hệ thống hiển thị ô để nhập mật khẩu cũ và mới 3. Tác nhân cập nhật mật khẩu 4. Kết thúc một sự kiện. | |
| **Các luồng sự kiện con:** | |
| **Luồng luân phiên/đặc biệt (Alternate/Exceptional flows)**: | |

#### Xem danh sách loài cá nhân

|  |  |
| --- | --- |
| **Tên use case:** Xem danh sách loài cá nhân | **ID:** UC-13 |
| **Actor chính:** Researcher, Expert | **Mức độ cần thiết:** Phải có |
| **Phân loại:** Cao |
| **Các thành phần tham gia và mối quan tâm*:***  **Xem danh sách chi tiết sản phẩm:** Khi tác nhân muốn xem danh sách loài cá nhân. | |
| **Mô tả tóm tắt:**  Khi tác nhân muốn xem danh sách loài cá nhân của mình trong hệ thống. | |
| **Trigger:** Khi tác nhân nhấp vào xem danh sách loài cá nhân.  **Type:** External | |
| **Các mối quan hệ:**  **+Association (kết hợp):** Researcher  **+Include(bao gồm):** Không  **+Extend(mở rộng):** Không  **+Generalization(tổng quát hóa):** Không | |
| **Luồng xử lý bình thường của sự kiện:**   1. Tác nhân chọn xem danh sách loài cá nhân 2. Hệ thống hiển thị danh sách loài 3. Kết thúc một sự kiện. | |
| **Các luồng sự kiện con:** | |
| **Luồng luân phiên/đặc biệt (Alternate/Exceptional flows)**: | |

#### Sửa loài cá nhân

|  |  |
| --- | --- |
| **Tên use case:** Sửa loài cá nhân | **ID:** UC-14 |
| **Actor chính:** Researcher, Expert | **Mức độ cần thiết:** Phải có |
| **Phân loại:** Cao |
| **Các thành phần tham gia và mối quan tâm*:***  **Xem danh sách chi tiết sản phẩm:** Khi tác nhân muốn sửa loài cá nhân. | |
| **Mô tả tóm tắt:**  Khi tác nhân muốn sửa loài cá nhân của mình trong hệ thống. | |
| **Trigger:** Khi tác nhân nhấp vào sửa loài cá nhân.  **Type:** External | |
| **Các mối quan hệ:**  **+Association (kết hợp):** Researcher  **+Include(bao gồm):** Không  **+Extend(mở rộng):** Không  **+Generalization(tổng quát hóa):** Không | |
| **Luồng xử lý bình thường của sự kiện:**   1. Tác nhân chọn sửa loài cá nhân 2. Hệ thống hiển thị thông tin loài 3. Tác nhân sửa thông tin loài 4. Kết thúc một sự kiện. | |
| **Các luồng sự kiện con:** | |
| **Luồng luân phiên/đặc biệt (Alternate/Exceptional flows)**: | |

#### Tạo loài mới

|  |  |
| --- | --- |
| **Tên use case:** Tạo loài mới | **ID:** UC-15 |
| **Actor chính:** Researcher, Expert | **Mức độ cần thiết:** Phải có |
| **Phân loại:** Cao |
| **Các thành phần tham gia và mối quan tâm*:***  **Xem danh sách chi tiết sản phẩm:** Khi tác nhân muốn tạo loài mới. | |
| **Mô tả tóm tắt:**  Khi tác nhân muốn tạo loài mới của mình trong hệ thống. | |
| **Trigger:** Khi tác nhân nhấp vào tạo loài mới.  **Type:** External | |
| **Các mối quan hệ:**  **+Association (kết hợp):** Researcher  **+Include(bao gồm):** Không  **+Extend(mở rộng):** Không  **+Generalization(tổng quát hóa):** Không | |
| **Luồng xử lý bình thường của sự kiện:**   1. Tác nhân chọn tạo loài mới 2. Hệ thống hiển thị ô để nhập thông tin loài 3. Tác nhân nhập vào thông tin loài 4. Kết thúc một sự kiện. | |
| **Các luồng sự kiện con:** | |
| **Luồng luân phiên/đặc biệt (Alternate/Exceptional flows)**: | |

#### Đăng xuất

|  |  |
| --- | --- |
| **Tên use case:** Đăng xuất | **ID:** UC-16 |
| **Actor chính:** Researcher, Expert | **Mức độ cần thiết:** Phải có |
| **Phân loại:** Cao |
| **Các thành phần tham gia và mối quan tâm*:***  **Xem danh sách chi tiết sản phẩm:** Khi tác nhân muốn đăng xuất. | |
| **Mô tả tóm tắt:**  Khi tác nhân muốn đăng xuất của mình ra khỏi hệ thống. | |
| **Trigger:** Khi tác nhân nhấp vào đăng xuất.  **Type:** External | |
| **Các mối quan hệ:**  **+Association (kết hợp):** Researcher  **+Include(bao gồm):** Không  **+Extend(mở rộng):** Không  **+Generalization(tổng quát hóa):** Không | |
| **Luồng xử lý bình thường của sự kiện:**   1. Tác nhân chọn đăng xuất 2. Hệ thống về trang chủ 3. Kết thúc một sự kiện. | |
| **Các luồng sự kiện con:** | |
| **Luồng luân phiên/đặc biệt (Alternate/Exceptional flows)**: | |

#### Xem các loài cần duyệt

|  |  |
| --- | --- |
| **Tên use case:** Xem các loài cần duyệt | **ID:** UC-17 |
| **Actor chính:** Expert | **Mức độ cần thiết:** Phải có |
| **Phân loại:** Cao |
| **Các thành phần tham gia và mối quan tâm*:***  **Xem danh sách chi tiết sản phẩm:** Khi tác nhân muốn xem các loài cần duyệt. | |
| **Mô tả tóm tắt:**  Khi tác nhân muốn xem các loài cần duyệt trong hệ thống. | |
| **Trigger:** Khi tác nhân nhấp vào xem các loài cần duyệt.  **Type:** External | |
| **Các mối quan hệ:**  **+Association (kết hợp):** Expert  **+Include(bao gồm):** Không  **+Extend(mở rộng):** Không  **+Generalization(tổng quát hóa):** Không | |
| **Luồng xử lý bình thường của sự kiện:**   1. Tác nhân chọn xem các loài cần duyệt 2. Hệ thống hiển thị danh sách các loài 3. Kết thúc một sự kiện. | |
| **Các luồng sự kiện con:** | |
| **Luồng luân phiên/đặc biệt (Alternate/Exceptional flows)**: | |

#### Sửa các loài cần duyệt

|  |  |
| --- | --- |
| **Tên use case:** Sửa các loài cần duyệt | **ID:** UC-18 |
| **Actor chính:** Expert | **Mức độ cần thiết:** Phải có |
| **Phân loại:** Cao |
| **Các thành phần tham gia và mối quan tâm*:***  **Xem danh sách chi tiết sản phẩm:** Khi tác nhân muốn sửa các loài cần duyệt. | |
| **Mô tả tóm tắt:**  Khi tác nhân muốn sửa các loài cần duyệt trong hệ thống. | |
| **Trigger:** Khi tác nhân nhấp vào sửa các loài cần duyệt.  **Type:** External | |
| **Các mối quan hệ:**  **+Association (kết hợp):** Expert  **+Include(bao gồm):** Không  **+Extend(mở rộng):** Không  **+Generalization(tổng quát hóa):** Không | |
| **Luồng xử lý bình thường của sự kiện:**   1. Tác nhân chọn sửa các loài cần duyệt 2. Hệ thống hiển thị loài 3. Tác nhân sửa loài 4. Kết thúc một sự kiện. | |
| **Các luồng sự kiện con:** | |
| **Luồng luân phiên/đặc biệt (Alternate/Exceptional flows)**: | |

#### Xem thống kê

|  |  |
| --- | --- |
| **Tên use case:** Xem thống kê | **ID:** UC-19 |
| **Actor chính:** Expert | **Mức độ cần thiết:** Phải có |
| **Phân loại:** Cao |
| **Các thành phần tham gia và mối quan tâm*:***  **Xem danh sách chi tiết sản phẩm:** Khi tác nhân muốn xem thống kê. | |
| **Mô tả tóm tắt:**  Khi tác nhân muốn xem thống kê trong hệ thống. | |
| **Trigger:** Khi tác nhân nhấp vào xem thống kê.  **Type:** External | |
| **Các mối quan hệ:**  **+Association (kết hợp):** Expert  **+Include(bao gồm):** Không  **+Extend(mở rộng):** Không  **+Generalization(tổng quát hóa):** Không | |
| **Luồng xử lý bình thường của sự kiện:**   1. Tác nhân chọn xem thống kê 2. Hệ thống hiển thị thống kê loài 3. Kết thúc một sự kiện. | |
| **Các luồng sự kiện con:** | |
| **Luồng luân phiên/đặc biệt (Alternate/Exceptional flows)**: | |

#### Xem danh sách Researcher

|  |  |
| --- | --- |
| **Tên use case:** Xem danh sách Researcher | **ID:** UC-20 |
| **Actor chính:** Expert | **Mức độ cần thiết:** Phải có |
| **Phân loại:** Cao |
| **Các thành phần tham gia và mối quan tâm*:***  **Xem danh sách chi tiết sản phẩm:** Khi tác nhân muốn xem danh sách Researcher. | |
| **Mô tả tóm tắt:**  Khi tác nhân muốn xem danh sách Researcher trong hệ thống. | |
| **Trigger:** Khi tác nhân nhấp vào xem danh sách Researcher.  **Type:** External | |
| **Các mối quan hệ:**  **+Association (kết hợp):** Expert  **+Include(bao gồm):** Không  **+Extend(mở rộng):** Không  **+Generalization(tổng quát hóa):** Không | |
| **Luồng xử lý bình thường của sự kiện:**   1. Tác nhân chọn xem danh sách Researcher 2. Hệ thống hiển thị danh sách 3. Kết thúc một sự kiện. | |
| **Các luồng sự kiện con:** | |
| **Luồng luân phiên/đặc biệt (Alternate/Exceptional flows)**: | |

#### Xem chi tiết Researcher

|  |  |
| --- | --- |
| **Tên use case:** Xem chi tiết Researcher | **ID:** UC-21 |
| **Actor chính:** Expert | **Mức độ cần thiết:** Phải có |
| **Phân loại:** Cao |
| **Các thành phần tham gia và mối quan tâm*:***  **Xem danh sách chi tiết sản phẩm:** Khi tác nhân muốn xem chi tiết Researcher. | |
| **Mô tả tóm tắt:**  Khi tác nhân muốn xem chi tiết Researcher trong hệ thống. | |
| **Trigger:** Khi tác nhân nhấp vào xem chi tiết Researcher.  **Type:** External | |
| **Các mối quan hệ:**  **+Association (kết hợp):** Expert  **+Include(bao gồm):** Không  **+Extend(mở rộng):** Không  **+Generalization(tổng quát hóa):** Không | |
| **Luồng xử lý bình thường của sự kiện:**   1. Tác nhân chọn xem chi tiết Researcher 2. Hệ thống hiển thị chi tiết 3. Kết thúc một sự kiện. | |
| **Các luồng sự kiện con:** | |
| **Luồng luân phiên/đặc biệt (Alternate/Exceptional flows)**: | |

#### Tạo Researcher mới

|  |  |
| --- | --- |
| **Tên use case:** Tạo Researcher mới | **ID:** UC-22 |
| **Actor chính:** Expert | **Mức độ cần thiết:** Phải có |
| **Phân loại:** Cao |
| **Các thành phần tham gia và mối quan tâm*:***  **Xem danh sách chi tiết sản phẩm:** Khi tác nhân muốn tạo Researcher mới. | |
| **Mô tả tóm tắt:**  Khi tác nhân muốn tạo Researcher mới trong hệ thống. | |
| **Trigger:** Khi tác nhân nhấp vào tạo Researcher mới.  **Type:** External | |
| **Các mối quan hệ:**  **+Association (kết hợp):** Expert  **+Include(bao gồm):** Không  **+Extend(mở rộng):** Không  **+Generalization(tổng quát hóa):** Không | |
| **Luồng xử lý bình thường của sự kiện:**   1. Tác nhân chọn tạo Researcher mới 2. Hệ thống hiển thị form điền thông tin 3. Tác nhân nhập vào thông tin 4. Kết thúc một sự kiện. | |
| **Các luồng sự kiện con:** | |
| **Luồng luân phiên/đặc biệt (Alternate/Exceptional flows)**: | |

#### Đổi tên Researcher

|  |  |
| --- | --- |
| **Tên use case:** Đổi tên Researcher | **ID:** UC-23 |
| **Actor chính:** Expert | **Mức độ cần thiết:** Phải có |
| **Phân loại:** Cao |
| **Các thành phần tham gia và mối quan tâm*:***  **Xem danh sách chi tiết sản phẩm:** Khi tác nhân muốn đổi tên Researcher. | |
| **Mô tả tóm tắt:**  Khi tác nhân muốn đổi tên Researcher trong hệ thống. | |
| **Trigger:** Khi tác nhân nhấp vào đổi tên Researcher.  **Type:** External | |
| **Các mối quan hệ:**  **+Association (kết hợp):** Expert  **+Include(bao gồm):** Không  **+Extend(mở rộng):** Không  **+Generalization(tổng quát hóa):** Không | |
| **Luồng xử lý bình thường của sự kiện:**   1. Tác nhân chọn đổi tên Researcher 2. Hệ thống hiển thị form điền thông tin 3. Tác nhân nhập vào thông tin 4. Kết thúc một sự kiện. | |
| **Các luồng sự kiện con:** | |
| **Luồng luân phiên/đặc biệt (Alternate/Exceptional flows)**: | |

#### Khóa tài khoản

|  |  |
| --- | --- |
| **Tên use case:** Khóa tài khoản | **ID:** UC-24 |
| **Actor chính:** Expert | **Mức độ cần thiết:** Phải có |
| **Phân loại:** Cao |
| **Các thành phần tham gia và mối quan tâm*:***  **Xem danh sách chi tiết sản phẩm:** Khi tác nhân muốn khóa tài khoản. | |
| **Mô tả tóm tắt:**  Khi tác nhân muốn khóa tài khoản trong hệ thống. | |
| **Trigger:** Khi tác nhân nhấp vào khóa tài khoản.  **Type:** External | |
| **Các mối quan hệ:**  **+Association (kết hợp):** Expert  **+Include(bao gồm):** Không  **+Extend(mở rộng):** Không  **+Generalization(tổng quát hóa):** Không | |
| **Luồng xử lý bình thường của sự kiện:**   1. Tác nhân chọn khóa tài khoản 2. Hệ thống thông báo thành công 3. Kết thúc một sự kiện. | |
| **Các luồng sự kiện con:** | |
| **Luồng luân phiên/đặc biệt (Alternate/Exceptional flows)**: | |

#### Reset mật khẩu của researcher

|  |  |
| --- | --- |
| **Tên use case:** Reset mật khẩu của researcher | **ID:** UC-25 |
| **Actor chính:** Expert | **Mức độ cần thiết:** Phải có |
| **Phân loại:** Cao |
| **Các thành phần tham gia và mối quan tâm*:***  **Xem danh sách chi tiết sản phẩm:** Khi tác nhân muốn reset mật khẩu của researcher. | |
| **Mô tả tóm tắt:**  Khi tác nhân muốn reset mật khẩu của researcher trong hệ thống. | |
| **Trigger:** Khi tác nhân nhấp vào reset mật khẩu của researcher.  **Type:** External | |
| **Các mối quan hệ:**  **+Association (kết hợp):** Expert  **+Include(bao gồm):** Không  **+Extend(mở rộng):** Không  **+Generalization(tổng quát hóa):** Không | |
| **Luồng xử lý bình thường của sự kiện:**   1. Tác nhân chọn reset mật khẩu của researcher 2. Hệ thống thông báo thành công 3. Kết thúc một sự kiện. | |
| **Các luồng sự kiện con:** | |
| **Luồng luân phiên/đặc biệt (Alternate/Exceptional flows)**: | |

### Các yêu cầu phi chức năng

#### Yêu cầu thực thi

* Yêu cầu services:
* Thời gian phản hồi ngắn.
* Hoạt động 24/7.
* Đáp ứng được tối thiểu 1000 người dùng cùng lúc.
* Yêu cầu client:
* Có thể vận hàng trên các trình duyệt như: Google Chrome, Mozilla Firefox, Microsoft Edge, Safari, Coccoc, Opera,…
* Có mạng.

#### Yêu cầu an toàn

* Sao lưu dữ liệu định kỳ.
* Thay đổi mật khẩu quản trị định kỳ.
* Ngăn chặn tối đa các cuộc tấn công:
* Phishing.
* SQL Injection.
* Gian lận thanh toán.
* Spam.
* DDoS.
* Brute-force Attack.
* XSS.
* Tuân thủ các quy định của pháp luật, thông tư bộ công thương và nghị định Chính Phủ:
* Luật giao dịch điện tử 2005.
* Nghị định 52/2013/NĐ-CP.
* Thông tư 47/2014/TT-BCT.

#### Yêu cầu bảo mật

* Các chức năng được hiển thị dựa trên phân quyền của từng nhóm người sử dụng.
* Mã hóa JWT an toàn.
* Hạn chế tấn công Cross Site Scripting (XSS)
* Sử dụng giao thức HTTPS.
* Người dùng bình thường không thể truy cập trang quản lý.
* Sử dụng chứng chỉ mã hóa SSL.
* Mã hóa mật khẩu SHA-256.
* Không lưu giữ thông tin thẻ của người dùng.
* Sử dụng tường lửa.

#### Các đặc điểm chất lượng phần mềm

* Tài liệu của dự án được quản lý có hệ thống, sẵn sàng cung cấp cho khách hàng khi có yêu cầu.
* Hệ thống tiêu hao tối thiểu tài nguyên của người dùng.
* Sử dụng các quy trình mã hóa, khả năng xử lý và chịu lỗi, khả năng phục hồi và quản lý tài nguyên, đảm bảo tính toàn vẹn và thống nhất của dữ liệu.
* Phát triển các thành phần theo hướng dễ dàng chuyển đổi, bảo trì và phát triển trong tương lai.
* Dễ dàng kiểm thử.
* Giao diện đơn giản, trực quan, thân thiện với người dùng.
* Có các chức năng cần thiết nhằm thực hiện tối đa các yêu cầu của người dùng một cách chính xác và nhanh nhất.

## Kiến Thức Nền

Chương này sẽ trình bày các khái niệm nền tảng của kiến trúc Microservices: kiến trúc Monolithic, định nghĩa kiến trúc Microservices và các mẫu thiết kế của kiến trúc Microservices. Phần tiếp theo là cơ sở lý thuyết các kiến thức nền tảng được áp dụng vào thiết kế hệ thống.

### Tổng quan về Microservices

Mặc dù kiến trúc Microservices chỉ mới nổi lên gần đây như một phong cách kiến trúc phổ biến, các khái niệm đằng sau kiến trúc này đã có từ nhiều thập kỷ. Ví dụ, những khái niệm này đã được triển khai trong hệ điều hành Unix:

Một chương trình nên làm một việc và làm nó đặc biệt tốt. Chương trình chỉ nên xử lý một trách nhiệm. Khi một công việc mới phát sinh, hãy tạo một chương trình mới để tránh làm phức tạp chương trình cũ với một trách nhiệm bổ sung.

Dự đoán rằng đầu ra của một chương trình sẽ là đầu vào của một chương trình khác, chưa được biết đến. Hãy để đầu ra ở dạng đơn giản nhất và tránh thêm thông tin không liên quan vào nó. Định dạng đầu vào nên được giữ đơn giản.

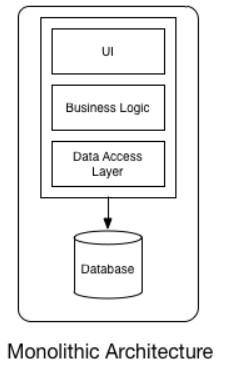
Thiết kế phần mềm với ý định phát triển và phát hành trong thời gian ngắn, tốt nhất là vài tuần, để phần mềm được dùng thử và kiểm tra. Hãy linh hoạt để loại bỏ các phần không hiệu quả hoặc xây dựng lại các chương trình.

—Doug McIlroy, một trong những người sáng lập Unix và là người phát minh ra Unix pipe, 1978

Các ngôn ngữ phát triển server-side nổi tiếng như Java, Python và C được trang bị các khả năng trừu tượng giúp giảm thiểu độ phức tạp của ứng dụng. Tuy nhiên, các khuôn khổ cho các ngôn ngữ này nhằm mục đích xây dựng các monoliths tạo ra các artifacts có thể thực thi duy nhất. Cùng với thời gian, khi các ứng dụng monolithic bắt đầu phát triển, chúng không tuân theo các khái niệm thiết kế được đề cập ở trên và do đó gặp phải những thách thức phát triển lớn liên quan đến độ phức tạp. Trong nhiều thập kỷ, các kỹ sư phần mềm và nhà thiết kế đã tích cực nghiên cứu các cách phá vỡ sự phức tạp của monoliths. Những nỗ lực của họ đã dẫn đến việc phát minh ra các Unix pipe sau đó đến các dynamic-link libraries (DLL), tiếp theo là Lập trình hướng đối tượng (OOP) và cuối cùng là Kiến trúc hướng dịch vụ. Chính vì những nỗ lực nghiên cứu và những tiến bộ trong phương pháp thiết kế và phát triển phần mềm mà kiến trúc Microservices đã xuất hiện và trở nên phổ biến như một sự lựa chọn đáng tin cậy của phong cách kiến trúc.

### Kiến trúc Monolithic

Trong phần mềm, kiến trúc monolithic là một kiểu kiến trúc cho phép xây dựng hệ thống thành một khối duy nhất, trong đó gồm ba thành phần: giao diện người dùng, xử lý nghiệp vụ logic, và phần truy cập dữ liệu.



Hình 7 Các thành phần của kiến trúc Monolithic

Giao diện người dùng: thông qua các thẻ HTML và JavaScript để giao tiếp giữa người dùng và hệ thống.

Nghiệp vụ logic: thực hiện các nghiệp vụ logic để thực hiện các yêu cầu HTTP đồng thời tổng hợp các thông tin từ cơ sở dữ liệu dưới dạng HTML và JavaScript để trả về giao diện người dùng.

Dữ liệu truy cập: thực hiện thao tác chỉnh sửa, cập nhật cơ sở dữ liệu.

Một hệ thống được xây dựng dựa trên kiến trúc Monolithic là một hệ thống hoàn toàn khép kín, độc lập với các hệ thống khác và nó chỉ được xây dựng trên một nền tảng duy nhất, do đó hệ thống được xây dựng dựa trên Monolithic rất dễ xây dựng, dễ quản lý. Cũng chính vì vậy mà các hệ thống xây dựng trên kiến trúc Monolithic có một nhược điểm rất lớn là nếu có bất kì thay đổi nào dù là rất nhỏ trong ba thành phần của hệ thống cũng khiến đội ngũ phát triển cũng phải triển khai một phiên bản mới cho cả hệ thống. Ví dụ người dùng muốn thay đổi màu sắc của một nút ngoài giao diện người dùng thì cả hệ thống sẽ phải được điều chỉnh, tái kiểm thử, tái triển khai trên một phiên bản mới. Điều này khiến cho việc xây dựng và phát triển hệ thống trên Monolithic là rất tốn kém và không linh hoạt. Đặc biệt là các hệ thống lớn việc xây dựng và phát triển hệ thống trên kiến trúc Monolithic sẽ trở nên rất phức tạp. Chi phí, thời gian dành cho việc xây dựng, phát triển, sửa lỗi và kiểm thử mỗi chức năng sẽ tăng lên theo độ lớn của hệ thống.

Mặc dù kiến trúc monolithic còn tồn tại nhiều nhược điểm nhưng nhờ các ưu điểm độc lập, dễ xây dựng và quản lý Monolithic đã được sử dụng để xây dựng và phát triển thành các kiểu kiến trúc phần mềm khác rất phổ biến hiện nay như kiến trúc hướng dịch vụ, Microservices cũng là một kiến trúc được xây dựng và phát triển từ kiến trúc Monolithic, mỗi dịch vụ trong kiến trúc Microservices đều kế thừa các đặc điểm độc lập của kiến trúc Monolithic.

### Định nghĩa về Microservices

Sau đây là một số định nghĩa của kiến trúc Microservices đã được một số bộ điều hợp kiến trúc ban đầu đưa ra.

Định nghĩa 1: Đây là cách để phân tách một ứng dụng về mặt chức năng thành một tập hợp các dịch vụ cộng tác, mỗi dịch vụ có một tập hợp các chức năng hẹp, có liên quan, được phát triển và triển khai độc lập, với cơ sở dữ liệu riêng của nó. (Kharbuja, 2016)

Định nghĩa 2: Microservices là các phần chức năng riêng lẻ được phát triển, triển khai và quản lý độc lập bởi một nhóm nhỏ gồm những người thuộc các lĩnh vực khác nhau. (Goetsch, 2017)

Định nghĩa 3: Phong cách kiến trúc Microservices là một cách tiếp cận để phát triển một ứng dụng đơn lẻ như một tập hợp các dịch vụ nhỏ, mỗi dịch vụ chạy trong quy trình riêng và giao tiếp với các cơ chế nhẹ, thường là API tài nguyên HTTP. Các dịch vụ này được xây dựng dựa trên khả năng kinh doanh và có thể triển khai độc lập bằng máy móc triển khai hoàn toàn tự động. Quản lý tập trung các dịch vụ này ở mức tối thiểu, có thể được viết bằng các ngôn ngữ lập trình khác nhau và sử dụng các công nghệ lưu trữ dữ liệu khác nhau. (Lewis & Fowler, 2014)

### Đặc điểm của Microservices

#### Single purpose

Đặc điểm này dựa trên nguyên tắc rằng một chương trình phải làm một việc và nó phải làm tốt việc đó. Các ứng dụng Monolithic cung cấp nhiều dịch vụ trong một codebase duy nhất có thể chứa hàng triệu dòng code. Ví dụ một ứng dụng thương mại, sẽ xử lý giỏ hàng, đơn đặt hàng, hàng tồn kho, sản phẩm và giá cả trong cùng một ứng dụng. Với kiến trúc Microservices, mỗi dịch vụ xử lý một trách nhiệm duy nhất. Người ta thường kỳ vọng rằng codebase của một dịch vụ sẽ phát triển theo thời gian để đáp ứng nhiều chức năng hơn khi hoạt động kinh doanh phát triển. Tuy nhiên, có thể tránh được sự cồng kềnh và tăng độ phức tạp bằng cách giao trách nhiệm phát triển từng service cho các nhóm nhỏ gồm 2 - 15 lập trình viên. Nếu con số này cần phải được vượt qua, thì đó là dấu hiệu cho thấy Microservices đang làm quá nhiều việc và có lẽ nên được chia thành hai. Một nhóm các lập trình viên nhỏ hơn cũng có nhiều khả năng duy trì sự tập trung và đảm bảo rằng Microservices vẫn nằm trong mục tiêu xử lý một trách nhiệm duy nhất.

#### Encapsulation

Microservices nên sở hữu dữ liệu của chúng và ẩn việc triển khai chúng. Mỗi Microservices phải có kho lưu trữ dữ liệu riêng và phải giữ nó ở chế độ riêng tư. Dữ liệu liên tục của một Microservices chỉ có thể được truy cập thông qua API được xác định rõ ràng của nó. Hạn chế của việc chia sẻ kho và quyền truy cập vào chúng. Việc đóng gói đảm bảo kết hợp lỏng lẻo giữa các dịch vụ và duy trì các phụ thuộc rõ ràng, tránh những phức tạp không chính đáng. Một lợi ích chính khác của tính đóng gói là thực tế là bất kỳ thay đổi nào được thực hiện đối với cơ sở dữ liệu của một dịch vụ cụ thể sẽ không ảnh hưởng đến bất kỳ dịch vụ nào khác.

#### Ownership

Một ứng dụng Monolithic lớn thường có một nhóm lớn các lập trình viên làm việc trên đó. Ví dụ, một ứng dụng bao gồm hàng chục triệu dòng mã có thể có một nhóm khoảng 100 lập trình viên. Một các cá nhân chỉ đóng góp một phần trăm của ứng dụng. Như vậy, ứng dụng thường được xem như một hộp đen mà không ai hiểu hết. Nói cách khác, phần lớn không có cảm giác sở hữu giữa các thành viên trong nhóm. Ứng dụng có nguy cơ trở nên phức tạp hơn do các lập trình viên làm việc vì lợi ích của họ chứ không phải để cải thiện ứng dụng.

Kiến trúc Microservices phát triển mạnh chủ yếu là do quyền sở hữu. Việc tổ chức các lập trình viên xung quanh Microservices dẫn đến các nhóm nhỏ độc lập chỉ gồm 2-15 thành viên chịu trách nhiệm về Microservices từ giai đoạn phát triển cho đến khi triển khai. Nhóm độc lập và đưa ra quyết định của riêng mình, do đó tăng động lực và hiệu quả. Các thành viên trong nhóm cũng phát triển mối quan hệ đồng cảm với những người sử dụng dịch vụ của họ và cam kết cho sự thành công của họ.

#### Autonomy

Quyền sở hữu phụ thuộc vào quyền tự chủ. Như đã đề cập ở trên, một nhóm chịu trách nhiệm về một Microservices là độc lập và phải có thể thiết kế, phát triển, triển khai và duy trì một Microservices hoàn toàn riêng biệt mà không cần phối hợp với một nhóm khác. Nhóm chịu trách nhiệm hoàn toàn về kết quả của Microservices và do đó có quyền tự do đưa ra quyết định đối với tất cả các vấn đề về công cụ, công nghệ và phương pháp luận. Các Microservices khác nhau giải quyết các vấn đề business khác nhau. Mỗi nhóm phải có thể chọn một công nghệ phù hợp nhất để đáp ứng nhu cầu cụ thể của họ. Có thể hiểu, các tổ chức có nhu cầu tiêu chuẩn hóa một số chi tiết bên ngoài như giao thức API, cảnh báo, nhắn tin và ghi nhật ký nhưng không nhất thiết phải là sản phẩm hỗ trợ việc triển khai. Mỗi Microservices được yêu cầu chỉ để lộ API của nó; do đó các kỹ thuật thực hiện nội bộ không có hậu quả lớn.

#### Multiple versions

Nhiều phiên bản Khả năng triển khai nhiều phiên bản dịch vụ vi mô cùng một lúc trong cùng một môi trường là một đặc điểm phân biệt khác của kiến trúc Microservices. Ví dụ: có thể có phiên bản 1.9, 2.0, 2.1 và 2.2 của các sản phẩm Microservices đều chạy trong môi trường sản xuất cùng một lúc. Điều này có thể thực hiện được bằng cách sử dụng URL, ví dụ e (/products/v1.9 hoặc products?version=1.9). Do đó, khách hàng có thể thực hiện các yêu cầu http tới một phiên bản cụ thể của Microservices. Điều này đặc biệt có lợi cho các công ty muốn phát hành nhanh chóng các sản phẩm khả thi tối thiểu (MVP) cho khách hàng của họ. Khả năng của kiến trúc Microservices để hỗ trợ nhiều phiên bản là một điểm mạnh đặc biệt, một điểm mạnh mà các kiến trúc khác như kiến trúc Monolithic không cung cấp.

#### So sánh Monoliths với Microservices

Bảng dưới đây tóm tắt những khác biệt đáng kể nhất giữa hai kiến trúc.

|  |  |
| --- | --- |
| Monolith | Microservices |
| Một ứng dụng duy nhất | Nhiều service với chức năng đơn giản |
| Cần triển khai toàn bộ ứng dụng | Mỗi Microservices có thể triển khai độc lập |
| Một database cho toàn bộ ứng dụng | Mỗi Microservices có một database riêng |
| Liên lạc trong nội bộ ứng dụng | Liên lạc từ xa, dùng REST qua HTTP, etc |
| Dev và ops làm việc riêng lẻ | Kết hợp giữa dev và ops để giữ hệ thống hoạt động bình thường |
| Trạng thái nằm trong ứng dụng bên ngoài lúc chạy | Trạng thái được lưu trữ tập trung |

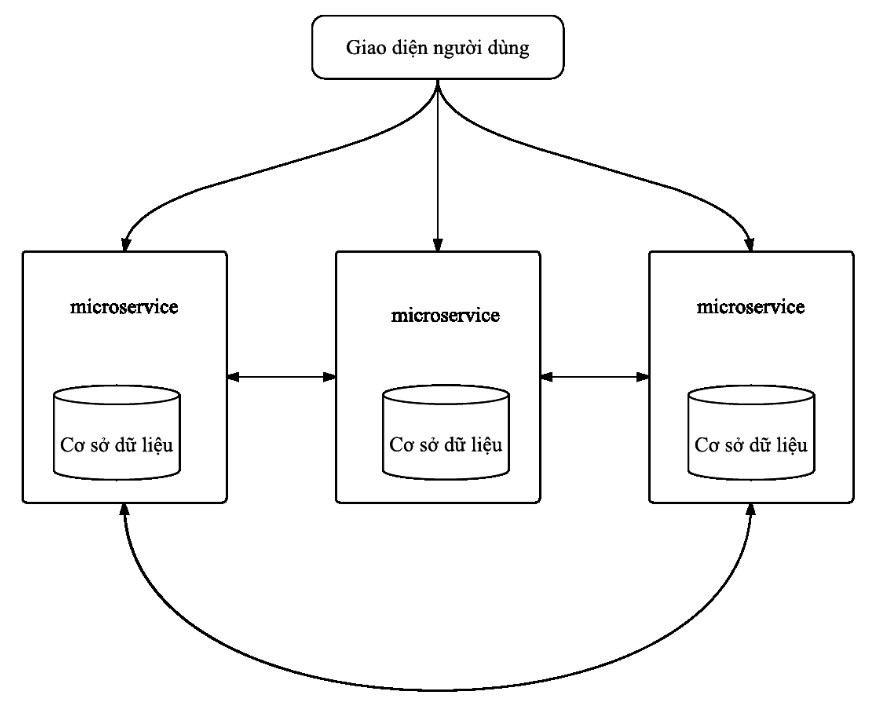
Bảng 2 Bảng so sánh Microservices và monolith

### Mẫu thiết kế của kiến trúc Microservices

#### Kiến trúc vi mô và vĩ mô

Đó là nên phân chia kiến trúc của bạn thành kiến trúc vi mô và vĩ mô. **Kiến trúc vi mô (micro architecture)** liên quan đến quyết định đối với từng Microservices. **Kiến trúc vĩ mô (macro architecture)** liên quan đến tất cả các quyết định ở cấp độ toàn cục áp dụng cho tất cả các Microservices. Có thể mở rộng khái niệm kiến trúc vi mô và vĩ mô đến **quyết định kỹ thuật.** Quyết định kỹ thuật có thể được đưa ra trong khuôn khổ của kiến trúc vĩ mô hoặc vi mô. Ví dụ, hãy xem các quyết định kỹ thuật được đưa ra ở cấp độ vi mô và vĩ mô cho cơ sở dữ liệu:

* **Micro:** Mỗi Microservices có thể có cơ sở dữ liệu riêng. Nếu cơ sở dữ liệu được định nghĩa ở kiến trúc vi mô, sự cố của một cơ sở dữ liệu sẽ chỉ dẫn đến sự cố của một dịch vụ siêu nhỏ. Điều này làm cho toàn bộ ứng dụng mạnh mẽ hơn nhiều.
* **Macro:** Cơ sở dữ liệu cũng có thể được định nghĩa là một phần của kiến trúc macro. Nhiều Microservices sẽ chia sẻ một lược đồ cơ sở dữ liệu.



Hình 8 Ví dụ kiến trúc Microservices

#### Kiến trúc hệ thống khép kín

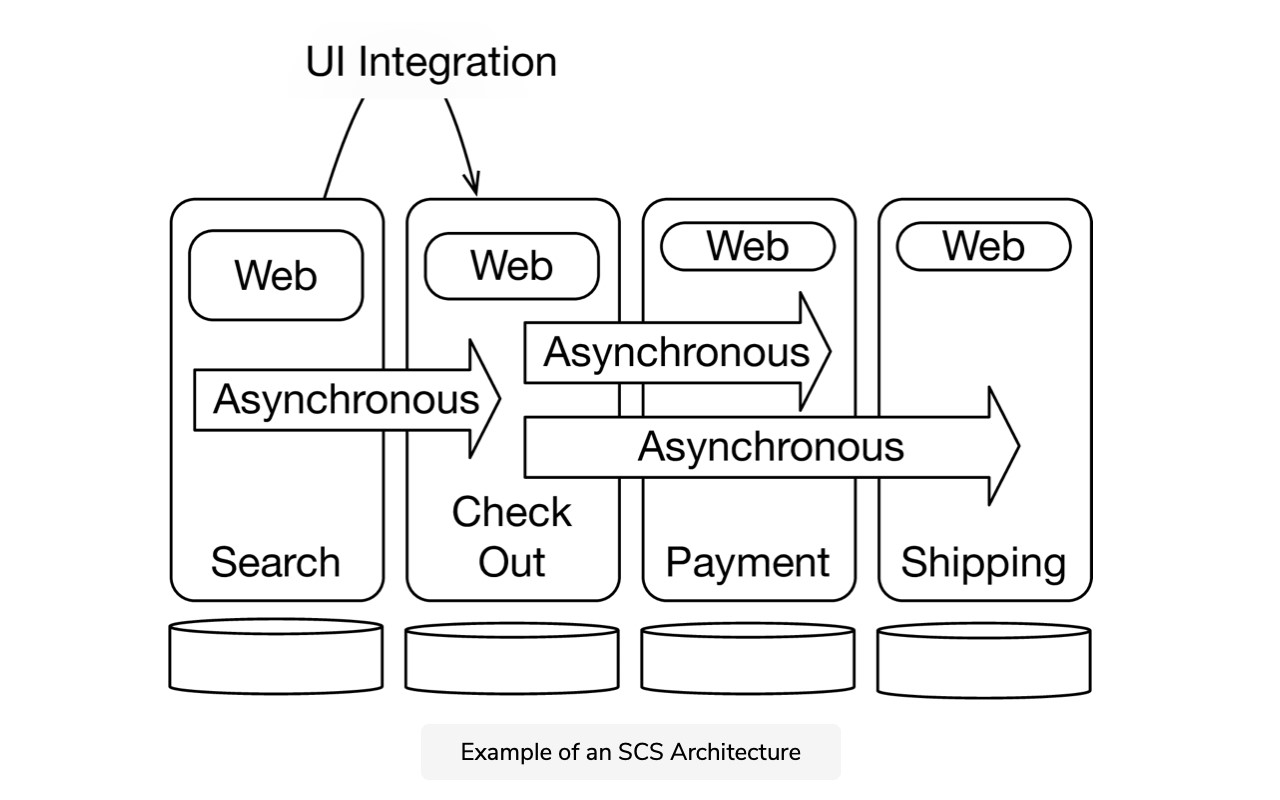
Hệ thống khép kín (SCS) là một loại kiến trúc Microservices để chỉ định các thành phần của kiến trúc macro. Điều này có nghĩa là chúng không đại diện cho toàn bộ hệ thống.

Vì SCS là độc lập, nó cung cấp mọi thứ bạn cần để thực hiện một phần của logic nghiệp vụ, chẳng hạn như dữ liệu nhật ký và giao diện người dùng. SCS cũng có một API tùy chọn.

Ví dụ: SCS cho một Microservices thanh toán sẽ lưu trữ thông tin liên quan đến khoản thanh toán vào cơ sở dữ liệu của nó. Nó cũng sẽ triển khai UI để hiển thị lịch sử thanh toán và dữ liệu về khách hàng sẽ được sao chép từ các SCS khác.

SCS cung cấp các quy tắc chính xác dựa trên các mẫu thiết kế đã thiết lập, cung cấp một điểm tham chiếu về cách xây dựng kiến trúc Microservices.

Tất cả các quy tắc này đảm bảo rằng SCS chỉ triển khai một nghiệp vụ (miền), do đó một tính năng được thêm vào chỉ thay đổi một SCS.



Hình 9 Mẫu kiến trúc SCS

Chúng ta có thể nghĩ về SCS như một kiến trúc Microservices bởi vì nó có thể được triển khai độc lập và phân chia một hệ thống thành các ứng dụng web độc lập.

Trên thực tế, một SCS thậm chí có thể được chia thành nhiều Microservices. Chúng khác với Microservices theo ba khía cạnh chính: chúng lớn hơn Microservices; chúng tập trung vào liên kết lỏng lẻo; chúng phải có giao diện người dùng (UI).

#### Kiến trúc tích hợp Front-End

Microservices cũng có thể được tích hợp với một web Front-End. Việc chia frontend thành các mô-đun khác nhau giúp giải quyết một số vấn đề xuất phát từ việc coi nó là một khối nguyên khối. Kiến trúc Micro Front-End.

Một Front-End được mô đun hóa được tạo thành từ các Microservices có thể triển khai riêng biệt. Điều này có thể mang lại nhiều lợi ích cho Front-End của bạn.

Ví dụ, một Front-End được mô đun hóa có thể có logic nghiệp vụ (miền) độc lập và thay đổi trong miền có thể được thực hiện đơn giản bằng cách sửa đổi chỉ một Microservices. Để kết hợp Front-End riêng biệt, chúng phải được tích hợp, do đó cần có một hệ thống tích hợp.



Hình 10 Chia Frontend thành modules

Điều này có thể được thực hiện thông qua các liên kết, trong đó một Front-End hiển thị một liên kết mà một Front-End khác đọc và xử lý.

Điều này cũng có thể được thực hiện thông qua các chuyển hướng, ví dụ, cách OAuth2 xử lý tích hợp Front-End. Chuyển hướng kết hợp truyền dữ liệu với tích hợp frontend.

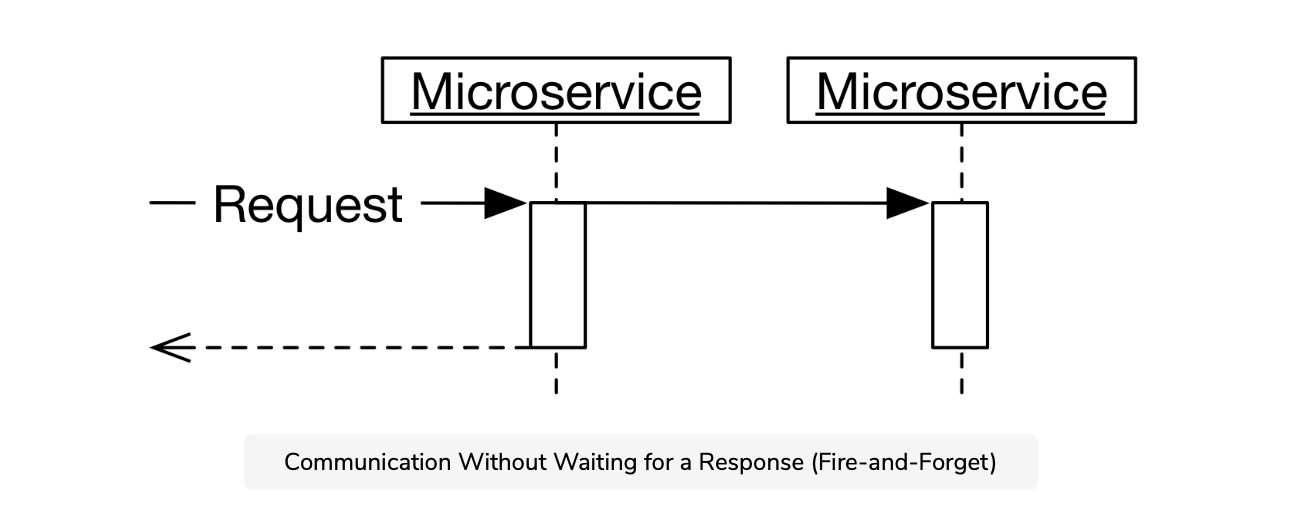
Tuy nhiên, có một vài trường hợp ngoại lệ khi một Front-End nên được triển khai dưới dạng nguyên khối. Ví dụ: các ứng dụng di động gốc phải triển khai nguyên khối hoặc nếu có một nhóm duy nhất chịu trách nhiệm phát triển Front-End.

#### Kiến trúc Microservices không đồng bộ

Các Microservices đồng bộ tạo một yêu cầu cho các Microservices khác trong khi nó xử lý các yêu cầu và chờ kết quả.

Tuy nhiên các giao thức liên lạc không đồng bộ gửi tin nhắn mà người nhận không có phản hồi trực tiếp.

Do đó một Microservices có thể được định nghĩa là không đồng bộ nếu nó không tạo yêu cầu cho các Microservices khác trong khi xử lý hoặc nó thực hiện các yêu cầu nhưng không chờ kết quả.



Hình 11 Liên lạc không đồng bộ

Các Microservices không đồng bộ cung cấp một số lợi thế đáng chú ý cho các Microservices đồng bộ và giải quyết nhiều thách thức của các hệ thống phân tán.

Logic cần thiết để xử lý các yêu cầu Microservices không phụ thuộc vào kết quả, làm cho chúng độc lập hơn nhiều.

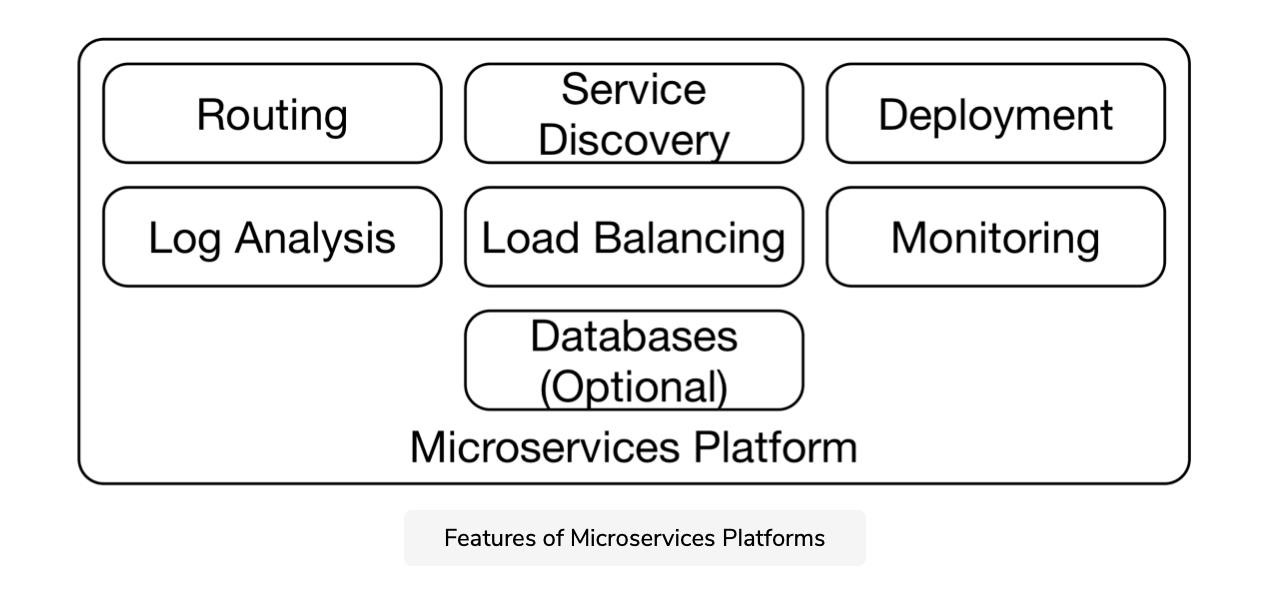
Tương tự, nếu một bên của sự giao tiếp thất bại, nó sẽ không sụp đổ toàn bộ hệ thống, mang lại khả năng phục hồi tổng thể cho hệ thống của bạn.

Một số ví dụ phổ biến về công nghệ cho các Microservices không đồng bộ là Kafka (một MOM thường được sử dụng để nhắn tin), định dạng dữ liệu REST và Atom (để bổ sung cơ sở hạ tầng).

#### Nền tảng Microservices

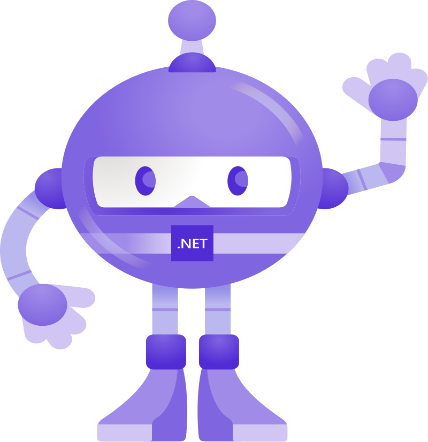
Các nền tảng Microservices, như PaaS và Docker, hỗ trợ hoạt động và liên lạc của các Microservices của bạn. Những công nghệ này cho phép giao tiếp giữa các Microservices để triển khai, phân tích nhật ký và giám sát.

Ví dụ: các nền tảng này hỗ trợ HTTP và REST với tính năng cân bằng tải và khám phá dịch vụ. Hỗ trợ hoạt động hạn chế là cần thiết để triển khai Microservices, vì vậy chúng có thể được triển khai nhanh chóng và hỗ trợ nhiều Microservices.



Hình 12 Chức năng của nền tảng Microservices

### .NET 6 LTS

.NET 6 Long-term support là phiên bản chính thức mới nhất của Microsoft. Tiền thân là .NET Core một nền tảng mã nguồn mở miễn phí trên các hệ điều hành Windows, Linux, và macOS. Đây là phiên bản đa nền tảng thừa kế từ .NET Framework. Dự án này chủ yếu được phát triển bởi các nhân viên Microsoft thuộc tổ chức .NET Foundation và được phát hành theo Giấy phép MIT. Đây là một trong các web framework nhanh nhất theo: <https://www.techempower.com/benchmarks/>

Hình 13 Linh vật .NET

Ngày 12 tháng 11 năm 2014, Microsoft đã công bố .NET Core, trong nỗ lực hỗ trợ đa nền tảng cho .NET trên các hệ điều hành Linux và macOS. Khởi đầu .NET Core áp dụng mô hình phát triển mã nguồn mở thông thường dưới sự quản lý của .NET Foundation, được Miguel de Icaza, nhân viên Microsoft phụ trách phát triển phần mềm của dự án, mô tả .NET Core là "phiên bản .NET được thiết kế lại dựa trên phiên bản đơn giản hóa của các thư viện lớp".

Phiên bản .NET Core 1.0 phát hành vào ngày 27 tháng 6 năm 2016, cho phép phát triển theo bộ công cụ Microsoft Visual Studio 2015 Update 3. Tiếp theo các phiên bản .NET Core 1.0.4 và .NET Core 1.1.1 được phát hành theo .NET Core Tools 1.0 và Visual Studio 2017 vào 7 tháng 3 năm 2017.

Phiên bản .NET Core 2.0 phát hành vào ngày 14 tháng 8 năm 2017, kèm với Visual Studio 2017 15.3, ASP .NET Core 2.0, và Entity Framework Core 2.0. Phiên bản .NET Core 2.1 phát hành ngày 30 tháng 5 năm 2018 với bản 2.1.30 là phiên bản Hỗ trợ dài hạn LTS. Phiên bản .NET Core 2.2 phát hành ngày 4 tháng 12 năm 2018.

Các phiên bản .NET Core 2.1 về sau bao gồm cả .NET 5, hỗ trợ hệ điều hành Alpine Linux một phiên bản gọn nhẹ của của Linux chỉ có 5MB.

Phiên bản .NET 5, hỗ trợ Windows Arm64 gốc. Các phiên bản trước, các ứng dụng .NET trên ARM được biên dịch ở kiến trúc x86, điều này có nghĩa là các ứng dụng này sẽ chỉ chạy trên lớp ARM giả lập.

.NET hỗ trợ đầy đủ cho C# và F# (và C++/CLI từ phiên bản 3.1; chỉ trên windows) và hỗ trợ Visual Basic .NET (từ phiên bản 15.5 trên .NET Core 5.0.100-preview.4, một số phiên bản trước của Visual Basic .NET cũng được hỗ trợ ở các bản .NET Core cũ hơn).

VB.NET biên dịch dịch và chạy .NET, nhưng từ phiên bản .NET Core 3.1, thư viện riêng Visual Basic Runtime không được triển khai nữa. Microsoft thông báo rằng .NET Core 3 sẽ bao gồm cả Visual Basic Runtime, tuy nhiên 2 năm sau tính năng này mới được cập nhật ở phiên bản .NET 5.

### Blazor Web Assembly

Blazor là web framework mã nguồn mở và miễn phí cho phép các nhà phát triển tạo các ứng dụng web bằng C # và HTML. Nó đang được phát triển bởi Microsoft. Có 2 phiên bản chính của Blazor là:

Hình 14 Logo Blazor

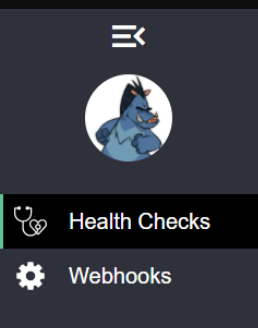
Blazor Server: Các code sau khi biên dịch sẽ được chạy ở server và sau đó gửi đến client thông qua SignalR và giao tiếp với các thư viện DOM để điều khiển.

Blazor WebAssembly: Các code sau khi biên dịch thành các file dll sẽ được đến client vào lúc client đến trang web, việc này làm cho việc lần đầu tải trang sẽ lâu hơn. Các thư viện dll thông qua Web Assembly điều khiển các DOM.

Blazor WebAssembly là một SPA (Single Page Application) tương tự như Angular, React hay là Vue. Sử dụng Blazor để tạo một trang web có một số lợi ích nhất định:

* Compile time checking, vì được viết bằng C# nên các lỗi sẽ được báo lúc biên dịch code thay vì lỗi sẽ phát sinh trong lúc chạy code.
* Không cần học thêm quá nhiều ngôn ngữ khác, có thể dùng C# thay thế cho JavaScript trong trang web.
* Thống nhất ngôn ngữ, như Angular, React, Vue sử dụng thường Nodejs làm server thì JavaScript có thể viết back-end lẫn front-end thì tương tự như vậy sử dụng Blazor có thể dùng C# viết code back-end lẫn front-end.

### HealthChecks

ASP.NET Core cung cấp Health Checks Middleware và thư viện để báo cáo tình trạng của các thành phần đang hoạt trong hệ thống.

Hình 15 Logo watchdogs healthchecks

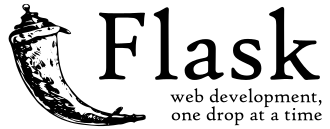
<https://docs.microsoft.com/en-us/dotnet/architecture/microservices/implement-resilient-applications/monitor-app-health>

Kiểm tra sức khỏe được hiển thị bởi một ứng dụng dưới dạng HTTP endpoints. Các Health check endpoints có thể được định cấu hình cho các tình huống giám sát thời gian thực khác nhau:

* Bộ điều phối vùng chứa và bộ cân bằng tải có thể sử dụng các đầu dò sức khỏe để kiểm tra trạng thái của ứng dụng. Ví dụ: bộ điều phối vùng chứa có thể phản hồi khi kiểm tra tình trạng không thành công bằng cách tạm dừng triển khai luân phiên hoặc khởi động lại vùng chứa. Bộ cân bằng tải có thể phản ứng với một ứng dụng không khỏe bằng cách định tuyến lưu lượng truy cập từ service không được khỏe đến một service khác làm nhiệm vụ tượng tự service cũ nhưng khỏe hơn.
* Có thể theo dõi việc sử dụng bộ nhớ, đĩa và các tài nguyên máy chủ vật lý khác để đảm bảo tình trạng hoạt động tốt.
* Kiểm tra sức khỏe có thể kiểm tra các kết nối ngoài của ứng dụng, chẳng hạn như cơ sở dữ liệu và hoặc service bên ngoài, để xác nhận tính khả dụng và hoạt động bình thường.

Kiểm tra sức khỏe thường được sử dụng với dịch vụ giám sát bên ngoài hoặc bộ điều phối vùng chứa để kiểm tra trạng thái của ứng dụng. Trước khi thêm kiểm tra sức khỏe vào ứng dụng, hãy quyết định sử dụng hệ thống giám sát nào. Hệ thống giám sát ra lệnh tạo loại kiểm tra sức khỏe nào và cách định cấu hình các điểm cuối của chúng.

### Flask Framework



Hình 16 Logo Flask

Flask là micro web framework được viết bằng Python. Nó được phân loại là microframework vì nó không yêu cầu các công cụ hoặc thư viện cụ thể. Nó không có lớp trừu tượng cơ sở dữ liệu, xác thực biểu mẫu hoặc bất kỳ thành phần nào khác mà các thư viện bên thứ ba đã có từ trước cung cấp các chức năng phổ biến. Tuy nhiên, Flask hỗ trợ các tiện ích mở rộng có thể thêm các tính năng của ứng dụng như thể chúng được triển khai trong chính Flask. Các tiện ích mở rộng tồn tại cho người lập bản đồ quan hệ đối tượng, xác thực biểu mẫu, xử lý tải lên, các công nghệ xác thực người dùng khác nhau.Các ứng dụng sử dụng Flask bao gồm Pinterest và LinkedIn.

### Seq



Hình 17 Logo Seq

<https://datalust.co/seq>

Seq là một ứng dụng có chức năng là nơi tập trung các log file (nhật ký) của các service mà chúng ta muốn theo dõi. Seq được xây dựng để ghi nhật ký có cấu trúc hiện đại với các mẫu tin nhắn. Thay vì lãng phí thời gian và công sức cố gắng trích xuất dữ liệu từ nhật ký văn bản thuần túy với phân tích cú pháp nhật ký mỏng manh, các thuộc tính liên quan đến mỗi sự kiện nhật ký được ghi lại và gửi tới Seq ở định dạng JSON sạch. Các mẫu thông báo được hỗ trợ bởi ASP.NET Core, Serilog, NLog và nhiều thư viện khác, vì vậy ứng dụng của bạn có thể sử dụng ghi nhật ký chẩn đoán hiện có tốt nhất cho nền tảng của bạn.

Seq chấp nhận nhật ký qua HTTP, GELF, đầu vào tùy chỉnh và máy khách dòng lệnh seqcli, với các trình cắm thêm hoặc tích hợp có sẵn cho .NET Core, Java, Node.js, Python, Ruby, Go, Docker, hàng đợi tin nhắn và nhiều công nghệ khác

### Microsoft SQL Server



Hình 18 Logo SQL Server

Microsoft SQL Server là một hệ quản trị cơ sở dữ liệu quan hệ được phát triển bởi Microsoft. Là một máy chủ cơ sở dữ liệu, nó là một sản phẩm phần mềm có chức năng chính là lưu trữ và truy xuất dữ liệu theo yêu cầu của các ứng dụng phần mềm khác. Có thể chạy trên cùng một máy tính hoặc trên một máy tính khác trên mạng (bao gồm cả Internet).

Microsoft tiếp thị ít nhất một chục phiên bản Microsoft SQL Server khác nhau, nhắm vào các đối tượng khác nhau và cho khối lượng công việc khác nhau, từ các ứng dụng máy đơn nhỏ đến các ứng dụng Internet lớn có nhiều người dùng đồng thời.

### RabbitMQ

#### Tổng quan

RabbitMQ là một chương trình phần mềm trung gian giúp các ứng dụng, hệ thống hay service khác nhau có thể giao tiếp, trao đổi dữ liệu với nhau. Nhiệm vụ của RabbitMQ được hiểu đơn giản là : nơi nhận các message (tin nhắn – thường dưới dạng json) từ các publisher (service gửi tin nhắn) sau đó chuyển đến các queue (hàng đợi) để lưu trữ, rồi các consumer (service xài tin nhắn đó) sẽ lấy các message ra sử dụng.

Hình 19 RabbitMQ

Là một Message Broker mã nguồn mở, dung lượng nhẹ, dễ dàng triển khai trên rất nhiều hệ điều hành lẫn Cloud, vì thế RabbitMQ vô cùng được ưa chuộng và trở nên phổ biến trong thời gian qua.

Không chỉ những doanh nghiệp nhỏ muốn tiết kiệm chi phí sử dụng, ngay cả những doanh nghiệp lớn, họ cũng đang sử dụng RabbitMQ cho công việc.

#### Message Broker là gì?

Message broker là một chương trình trung gian và được phát triển nhằm phục vụ yêu cầu giao tiếp giữa các ứng dụng khác nhau với nhau một cách dễ dàng. Cũng có thể hiểu message broker là một chương trình phần mềm trung gian để gửi tin nhắn.

Message broker có trách nhiệm xác thực, chuyển đổi, vận chuyển các message giữa các ứng dụng với nhau và đảm bảo các biện pháp đề phòng khi tin nhắn không đến đích. Một số message broker nổi tiếng hiện nay:

1. RabbitMq: <https://www.rabbitmq.com/>
2. Kafka: <https://kafka.apache.org/>
3. Redis Bus: <https://redis.io/docs/manual/pubsub/>
4. Của các dịch vụ đám mây: Azure Bus, Aws Bus, ...

#### Các tính năng và lợi ích của RabbitMQ

1. Asynchronous Messaging

RabbitMQ hỗ trợ rất nhiều giao thức message như: sắp xếp hàng đợi, gửi message, khả năng định tuyến hàng đợi linh hoạt và nhiều loại exchange khác

1. Developer Experience

Với mức độ linh hoạt của RabbitMQ, bạn có thể triển khai trên BOSH, Chef, Docket và cả Puppet.

RabbitMQ không chỉ tương thích với ngôn ngữ Erlang “mẹ đẻ” mà còn có khả năng tương thích với hầu hết ngôn ngữ lập trình thông dụng như : C#, JavaScript, Go, Java, Ruby, ..

1. Distributed Deployment

Chúng ta có thể triển khai RabbitMQ dưới dạng các cluster có tính khả dụng cao và thông lượng lớn từ đó chúng ta có thể phát triển liên kết toàn cầu qua nhiều khu vực, lãnh thổ.

1. Enterprise & Cloud Ready

Với những công nghệ xác thực ủy quyền được phát triển dựa trên TLS và LDAP gia tăng độ bảo mật lên rất cao. RabbitMQ có dung lượng nhẹ và khả năng tương thích và phát triển một cách dễ dàng trên cách public cloud cũng như private cloud.

1. Management and Monitoring

Chúng ta hoàn toàn có thể sử dụng HTTP-API, các công cụ dòng lệnh và cả công cụ được phát triển thân thiện với người dùng để quản lý, giám sát RabbitMQ một cách hiệu quả.

1. Tool and Plugin

RabbitMQ cung cấp rất nhiều công cụ và plugin được phát triển liên tục với khả năng như: tích hợp, khả năng đo lường, tương thích với các hệ thống khác của doanh nghiệp.

### MongoDB



Hình 20 MongoDb

#### Một CSDL NoSQL:

NoSQL là 1 dạng CSDL mã nguồn mở và được viết tắt bởi: Non Relational SQL hay có nơi thường gọi là Not-Only SQL.

NoSQL được phát triển trên Javascript Framework với kiểu dữ liệu là JSON và dạng dữ liệu theo kiểu key và value.

NoSQL ra đời như là 1 mảnh vá cho những khuyết điểm và thiếu xót cũng như hạn chế của mô hình dữ liệu quan hệ RDBMS (Relational Database Management System - Hệ quản trị cơ sở dữ liệu quan hệ) về tốc độ, tính năng, khả năng mở rộng, v.v.

NoSQL bỏ qua tính toàn vẹn của dữ liệu và transaction để đổi lấy hiệu suất nhanh và khả năng mở rộng.

NoSQL được sử dụng ở rất nhiều công ty, tập đoàn lớn, ví dụ như Facebook sử dụng Cassandra do Facebook phát triển, Google phát triển và sử dụng BigTable, v.v.

#### Những điều cơ bản về MongoDB:

MongoDB là một hệ quản trị cơ sở dữ liệu mã nguồn mở, là CSDL thuộc NoSql và được hàng triệu người sử dụng. MongoDB là một database hướng tài liệu (document), các dữ liệu được lưu trữ trong document kiểu JSON thay vì dạng bảng như CSDL quan hệ nên truy vấn sẽ rất nhanh.

Với CSDL quan hệ chúng ta có khái niệm bảng, các cơ sở dữ liệu quan hệ (như MySQL, SQL Server, v.v) sử dụng các bảng để lưu dữ liệu thì với MongoDB chúng ta sẽ dùng khái niệm là collection thay vì bảng. So với RDBMS (Relational Database Management System) thì trong MongoDB collection ứng với table, còn document sẽ ứng với row, MongoDB sẽ dùng các document thay cho row trong RDBMS. Các collection trong MongoDB được cấu trúc rất linh hoạt, cho phép các dữ liệu lưu trữ không cần tuân theo một cấu trúc nhất định. Thông tin liên quan được lưu trữ cùng nhau để truy cập truy vấn nhanh thông qua ngôn ngữ truy vấn MongoDB. Ưu nhược điểm của MongoDB:

**Ưu điểm**:

* Do MongoDB sử dụng lưu trữ dữ liệu dưới dạng Document JSON nên mỗi một collection sẽ có các kích cỡ và các document khác nhau, linh hoạt trong việc lưu trữ dữ liệu.
* Dữ liệu trong MongoDB không có sự ràng buộc lẫn nhau, không có join như trong RDBMS nên khi insert, xóa hay update nó không cần phải mất thời gian kiểm tra xem có thỏa mãn các ràng buộc dữ liệu như trong RDBMS.
* MongoDB rất dễ mở rộng. Trường dữ liệu “\_id” luôn được tự động đánh index (chỉ mục) để tốc độ truy vấn thông tin đạt hiệu suất cao nhất.
* Khi có một truy vấn dữ liệu, bản ghi được cached lên bộ nhớ Ram, để phục vụ lượt truy vấn sau diễn ra nhanh hơn mà không cần phải đọc từ ổ cứng.
* Hiệu năng cao: Tốc độ truy vấn (find, update, insert, delete) của MongoDB nhanh hơn hẳn so với các hệ quản trị cơ sở dữ liệu quan hệ (RDBMS).

**Nhược điểm:**

* MongoDB không có các tính chất ràng buộc như trong RDBMS đòi hỏi sự cẩn thận khi sử dụng.
* Tốn bộ nhớ do dữ liệu lưu dưới dạng key-value, các collection chỉ khác về value do đó key sẽ bị lặp lại. MongoDB cũng không hỗ trợ liên kết nên dữ liệu bị dư thừa.
* Khi insert/update/remove bản ghi, MongoDB sẽ chưa cập nhật ngay xuống ổ cứng, mà sau một khoảng thời gian 60 giây MongoDB mới thực hiện ghi toàn bộ dữ liệu thay đổi từ RAM xuống thiết bị lưu trữ, điều này sẽ có nguy cơ bị mất dữ liệu khi xảy ra các tình huống như mất điện, hay service lỗi và tự động xóa.

#### Vì sao sử dụng MongoDB:

Dù MongoDB có nhiều ưu điểm, nhưng tùy vào quy mô hệ thống để lựa chọn. Ví dụ như các hệ thống yêu cầu phản hồi nhanh, Các hệ thống dữ liệu lớn với yêu cầu truy vấn nhanh hay các hệ thống có lượng request lớn thì MongoDB sẽ là sự lựa chọn ưu tiên hơn CSDL quan hệ. Tùy theo dự án và trường hợp cụ thể để sử dụng CSDL quan hệ hay sử dụng MongoDB đem lại hiệu quả cao.

### Docker



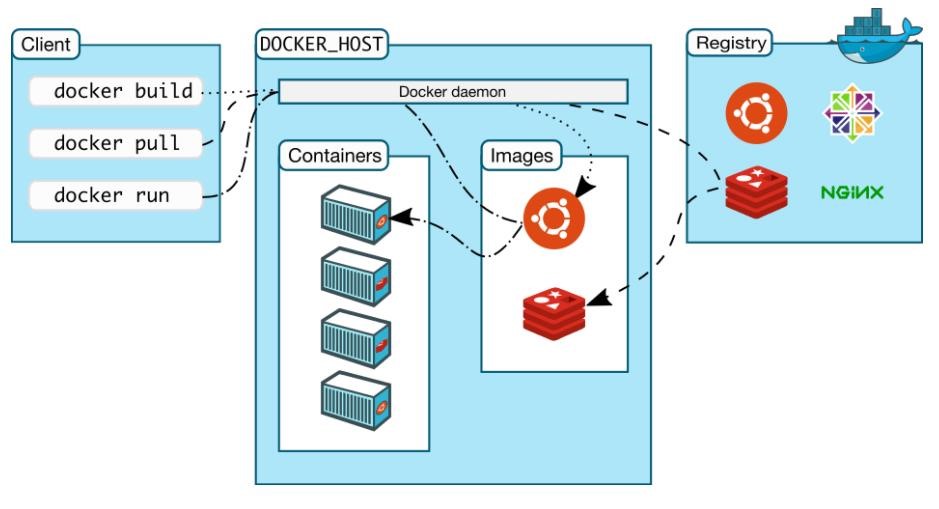
Hình 21 Docker

#### Docker là gì?

Docker là một nền tảng phần mềm cho phép bạn dựng, kiểm thử, triển khai ứng dụng một cách nhanh chóng. Docker đóng gói phần mềm vào các đơn vị tiêu chuẩn hóa được gọi là container có mọi thứ mà phần mềm có thể chạy, trong đó có thư viện, công cụ hệ thống, mã thời gian chạy. Bằng cách sử dụng Docker, chúng ta có thể nhanh chóng triển khai và thay đổi quy mô ứng dụng vào bất kỳ môi trường nào và biết chắc rằng mã của chúng ta sẽ chạy được.

#### Cách thức hoạt động của Docker

Docker hoạt động bằng cách cung cấp phương thức tiêu chuẩn để chạy mã của chúng ta. Docker là hệ điều hành dành cho container. Cũng tương tư như cách máy ảo ảo hóa (loại bỏ nhu cầu quản lý trực tiếp) phần cứng máy chủ, các container sẽ ảo hóa hệ điều hành của máy chủ. Docker được cài đặt trên từng máy chủ và cung cấp các lệnh đơn giản mà bạn có thể sử dụng để dựng và khởi động hoặc dùng container.



Hình 22 Mô hình hoạt động của Docker

#### Lý do nên sử dụng Docker.

Việc sử dụng Docker cho phép bạn vận chuyển mã nhanh hơn, tiêu chuẩn hóa hoạt động của ứng dụng, di chuyển mã một cách trơn tru và tiết kiệm chi phí bằng cách cải thiện khả năng tận dụng tài nguyên. Với Docker, bạn sẽ nhận được một đối tượng duy nhất có khả năng chạy ổn định ở bất kỳ đâu. Cú pháp đơn giản và không phức tạp của Docker sẽ cho bạn quyền kiểm soát hoàn toàn. Việc đưa vào áp dụng rộng rãi nền tảng này đã tạo ra một hệ sinh thái bền vững các công cụ và ứng dụng có thể sử dụng ngay đã sẵn sàng sử dụng với Docker.

## Kiến Trúc Hệ Thống Microservices

### Tổng quan

#### Mô hình hoá các microservices

##### Xác định các Microservices

Các Microservices trong kiến trúc Microservices cần được xác định sao cho các Microservices này phải tách rời nhau nhằm đảm bảo khi thay đổi bất cứ Microservices nào cũng không làm ảnh hưởng đến các Microservices khác nhưng đồng thời cũng cần đảm bảo phải có sự kết nối cao giữa các Microservices trong hệ thống. Để làm được điều đó khi xây dựng các Microservices trong hệ thống ta nên làm một cách riêng rẽ, mỗi Microservices nên có một cơ sở dữ liệu riêng để đảm bảo tính độc lập trong kiến trúc Microservices. Tuy nhiên việc thiết kế cơ sở dữ liệu riêng, độc lập cho mỗi Microservices có thể khiến hệ thống xây dựng trên kiến trúc Microservices bị dư thừa dữ liệu, để tránh việc dư thừa dữ liệu trong kiến trúc Microservices thì cách tốt nhất đó là thiết cơ sở dữ liệu trong mỗi Microservices phải ít ràng buộc, không có khóa ngoài.

Việc xác định các Microservices phải đảm bảo sao cho khi ta muốn thay đổi một hành vi hoặc một chức năng nào đó thì chúng ta chỉ cần thay đổi ở một Microservices. Để làm được điều này, chúng ta nên nhóm các hành vi có liên quan đến nhau lại trong cùng một Microservices nhưng việc nhóm các hành vi này vẫn phải đảm bảo kết nối lỏng lẻo cho các Microservices trong hệ thống. Trong cuốn Domain-Driven Design,

Eric Evans đã đưa ra khái niệm về Bounded Context Để Có thể khoanh vùng được các chức năng trên miền (DDD –Domain-Driven Design). Nó rất hữu ích khi ta áp dụng để xác định các Microservices trong kiến trúc Microservices. Theo Eric Evans thì Bounded Context là một mẫu thiết kế trung tâm của thiết kế hướng miền nhằm khoanh vùng các hành vi có mối quan hệ với nhau và cùng thực hiện một chức năng lại với nhau. Bounded Context giúp các nhà xây dựng phần mềm chia nhỏ các hệ thống lớn và giúp các thành phần sau khi chia nhỏ được độc lập với nhau. Chúng ta có thể sử dụng ưu điểm này của Bounded Context để xác định các Microservices trong kiến trúc Microservices.

##### Thiết kế kiến trúc

Mỗi Microservices này có một API, một cơ sở dữ liệu và các thành phần được cài đặt để thực hiện chức năng của Microservices. Các Microservices giao tiếp với nhau bằng kỹ thuật tương tác IPC (Inter-Process communication) sử dụng giao thức HTTP dựa trên REST. Mỗi Microservices đều có một API để có thể giao tiếp với nhau. Để tạo ra API cho mỗi Microservices, ta phải đảm bảo được ba phần chính sau:

Phải định nghĩa và mô tả được Microservices mà ta muốn viết API. Các thông tin này sẽ được lưu trong controller cho mỗi Microservices.

Nắm được quy mô thay đổi của API bởi vì API của Microservices luôn thay đổi theo thời gian. Khi đã biết rõ quy mô thay đổi của API ta có thể đưa ra các biện pháp xử lý hiệu quả nhất. Đối với các thay đổi nhỏ, tính tương thích giữa API và các máy trạm chưa bị phá vỡ ta có thể thêm thay đổi vào phần yêu cầu và hồi đáp. Đối với các thay đổi lớn, tính tương thích bị phá vỡ ta bắt buộc phải tạo ra thêm phiên bản mới cho API trong Microservices và vẫn duy trì API cũ.

Các API phải có khả năng xử lý được các lỗi cục bộ như Microservices bị ngưng hoạt động tạm thời, bị quá tải. Để có thể xử lý được các lỗi cục bộ ta có thể áp dụng một trong các phương pháp sau: Tạo một khoản thời gian chờ - timeout cho các máy trạm. Tạo fallback cho phép trả lại các thông báo, khuyến cáo để các máy trạm biết Microservices được kết nối đến đang bị lỗi gì.

Các API của các Microservices giao tiếp với nhau bằng REST và sau khi thực hiện xong các HTTP request nó sẽ trả về kết quả dưới dạng JSON. Để dễ dàng xây dựng và phát triển các Microservices, mỗi API trong các Microservices cần phải có một phần tài liệu để mô tả các chức năng cách dùng gọi là tài liệu API. Trong .NET có một thư viện rất hữu ích để viết tài liệu API gọi là Open API Swagger UI. Trong đó summary được dùng để mô tả chung về API, notes cho phép mô tả chi tiết về API, param dùng để khai báo các thuộc tính của mỗi story và response cho phép thể hiện các thông điệp trả về các máy trạm khi có yêu cầu tạo post.

Giao thức HTTP cho phép các Microservices giao tiếp với nhau bằng một cặp yêu cầu và hồi đáp (request/response)trong đó Microservices muốn kết nối trước sẽ đóng vai trò là máy trạm và Microservices nhận kết nối sẽ đóng vai trò là máy chủ. Để bắt đầu kết nối máy trạm khởi tạo HTTP request đến máy chủ và nhận HTTP response trả về từ máy chủ. HTTP request gồm hai thành phần chính là URL và các các phương thức, trong đó các phương thức thường được sử dụng:

1. POST cho phép tạo ra sự thay đổi trong cơ sở dữ liệu, được dùng khi các máy trạm muốn tạo ra đối tượng mới hoặc khi các máy trạm muốn gửi dữ liệu đến các Microservices và ngược lại.
2. GET dùng để truy vấn dữ liệu và tài nguyên với các tham số và giá trị nằm ngay trên URL.
3. PUT/PATCH được dùng khi máy trạm muốn cập nhật dữ liệu.
4. DELETE sẽ xóa đối tượng khỏi cơ sở dữ liệu.

HTTP response cũng gồm hai thành phần là mã trạng thái (Status code)và thông điệp (Message body). Trong đó mã trạng thái sẽ chứa kết quả sau khi xử lý yêu cầu HTTP request ở máy chủ, một số mã trạng thái thường dùng:

* 2xx: thể hiện HTTP request đã được máy chủ thực hiện thành công:
  + 200: OK
  + 201: Created
  + 204: No Content
* 3xx: chuyển hướng
* 4xx: thể hiện các lỗi từ phía máy trạm
  + 400: Bad Request
  + 401: Unauthorized
  + 403: Forbidden
  + 404: Not Found
* 5xx: các lỗi từ phía máy chủ
  + 500: Internal Server Error
  + 503: Service Unavailable

##### Cơ chế kết nối

Khi xây dựng các Microservices ta cần phải xác định được cơ chế giao tiếp(IPC –Inter Process Communication) giữa các Microservices và giữa máy trạm với các Microservices. Hiện nay có hai cơ chế IPC rất phổ biến đó là cơ chế tương tác không đồng bộ dựa trên hệ thống thông điệp và cơ chế tương tác đồng bộ request/response.

Cơ chế tương tác không đồng bộ dựa trên hệ thống thông điệp cho phép các Microservices trao đổi với nhau qua thông điệp (message), Microservices sẽ gửi một thông điệp đến Microservices khác mà nó muốn kết nối để thực hiện kết nối. Có rất nhiều nền tảng sử dụng cơ chế này là mã nguồn mở như: RabbitMQ, Apache Kafka, Apache ActiveMQ,...

Cơ chế tương tác đồng bộ request/response cho phép các Microservices trong hệ thống giao tiếp với nhau bằng cách Microservices muốn kết nối sẽ gửi request đến Microservices mà nó muốn kết nối, Microservices này sẽ xử lý và trả lại Microservices muốn kết nối đến nó một response.

#### Triển khai

##### Đóng gói các Microservices

Trong kiến trúc Microservices để đảm bảo tính độc lập cho mỗi Microservices khi đóng gói các Microservices ta nên đóng gói mỗi Microservices là một khối (Container).

##### Cấu hình cho máy chủ

Có hai phương pháp chính để cấu hình máy chủ cho hệ thống trong kiến trúc Microservices là nhiều Microservices trên một máy chủ và mỗi Microservices một máy chủ. Ưu điểm của phương pháp này là đơn giản, dễ quản lý các Microservices trong hệ thống, tiết kiệm chi phí khi triển khai. Tuy nhiên phương pháp này cũng có khó khăn là việc theo dõi và kiểm tra các microservices, vì các Microservices cùng chạy trên một máy chủ nên rất khó khăn để theo dõi, kiểm tra từng Microservices một. Đồng thời phương pháp này cũng mang lại không ít rủi ro khi máy chủ gặp sự cố thì các Microservices chạy trên máy chủ này cũng ngừng hoạt động theo làm hệ thống bị ảnh hưởng, hoặc khi máy chủ bị các hacker tấn công thì hệ thống cũng sẽ bị dừng hoạt động. Phương pháp thứ hai là triển khai mỗi Microservices một máy chủ. Phương pháp này cho phép mỗi Microservices được triển khai trên một máy chủ. Nó có thể khắc phục được những khó khăn trong phương pháp triển khai nhiều Microservices trên một máy chủ. Nhưng đổi lại chi phí để triển khai theo mô hình này là rất lớn.

#### Kiểm thử trong Microservices

Các Microservices trong kiến trúc Microservices chỉ thực hiện một chức năng duy nhất của hệ thống,điều này đảm bảo các Microservices trong hệ thống luôn ổn định, ít có thay đổi và khi kiểm thử trong kiến trúc Microservices ta cần kiểm thử ở nhiều môi trường khác nhau. Sử dụng kiểm thử tự động sẽ là giải pháp tối ưu cho các hệ thống xây dựng trên kiến trúc Microservices. Kiểm thử tự động là quá trình kiểm tra hệ thống với dữ liệu đầu vào và đầu ra đã xác định trước một cách tự động.Việc áp dụng kiểm thử tự động sẽ mang lại rất nhiều ưu điểm như nâng cao hiệu quả, tăng độ tin cậy, cải thiện chất lượng sản phẩm, tốc độ kiểm thử nhanh và đặc biệt là giảm chi phí cho hệ thống trong khâu kiểm thử.

##### Kiểm thử đơn vị

Kiểm thử đơn vị được thực hiện đồng thời với quá trình xây dựng các Microservices trong kiến trúc Microservices. Tại mỗi Microservices, kiểm thử đơn vị chỉ kiểm thử một hành vi duy nhất trong mỗi Microservices. Vì vậy việc viết mã kiểm thử cho kiểm thử đơn vị sẽ rất nhanh chóng và dễ dàng.

##### Kiểm thử dịch vụ

Kiểm thử dịch vụ thực hiện kiểm thử các dịch vụ mà không quan tâm đến các giao diện người dùng của các dịch vụ đó. Trong Microservices ta có thể thực hiện kiểm thử dịch vụ cho mỗi Microservices một cách độc lập như vậy việc kiểm thử sẽ nhanh chóng xác định và sửa các lỗi trong các Microservices.

##### Kiểm thử đầu cuối

Kiểm thử đầu cuối chính là kiểm thử ở mức giao diện trong mô hình kiểm thử kim tự tháp. Kiểm thử đầu cuối thực hiện kiểm thử hoạt động tổng thể của ứng dụng bằng cách kiểm thử quá trình thực hiện chức năng từ đầu đến cuối quá trình xử lý, đi từ giao diện đến giao tiếp với nguồn dữ liệu.Trong Microservices việc áp dụng kiểm thử đầu cuối cho mỗi Microservices là tiến hành kiểm thử tất cả quá trình thực hiện mỗi Microservices từ API đến các thao tác trên cơ sở dữ liệu của chúng. Như vậy trong phạm vi một Microservices có thể thấy kiểm thử đầu cuối chính là kiểm thử hệ thống cho Microservices đó.

##### Kiểm thử tích hợp

Trong kiến trúc Microservices ta có thể sử dụng phương pháp kiểm thử đầu cuối để kiểm thử tích hợp các Microservices trong hệ thống. Sau khi đã kiểm thử cho từng Microservices trong hệ thống chúng ta sẽ tiến hành kiểm thử tích hợp bằng cách kiểm thử đầu cuối cho toàn hệ thống. Việc kiểm thử đầu cuối cho mỗi Microservices thì rất đơn giản nhưng khi tích hợp chúng lại thì lại rất khó khăn, đặc biệt là với các hệ thống lớn có nhiều Microservices. Ưu điểm của phương pháp này là nhanh chóng phát hiện được lỗi nhưng nó lại cần phải có thêm một đội ngũ nhân viên để thực hiện việc này và mối quan hệ giữa các Microservices trong hệ thống phải được xác định rõ ràng ngay từ đầu.

#### Sử dụng kiến trúc Microservices khi nào là hợp lý

Với những thách thức đối với nhu cầu sử dụng Microservices nên dùng cấu trúc này khi:

Khi phát triển những phiên bản đầu tiên cho một ứng dụng, khi đó bạn thường không phải gặp những vấn đề mà Microservices cần phải giải quyết. Hơn nữa, việc sử dụng một kiến trúc phân tán hoặc phức tạp sẽ làm chậm đi quá trình phát triển của ứng dụng.

Đây là một trong những vấn đề lớn đối với các startup bởi vì họ buộc phải phát triển nhanh mô hình kinh doanh của mình cũng như ứng dụng kèm theo. Chính vì vậy, trừ khi bạn đã có một hệ thống phức tạp để quản lý bằng Monolithic hoặc bạn đã xác định được tương lai của ứng dụng sẽ ra sao; thì có thể dùng Microservices.

##### Hiểu sai về Microservices

Mico là một từ khóa dễ gây hiểu nhầm và bạn nghĩ rằng nên tạo ra services nhỏ hết mức thì đó là cách hiểu hoàn toàn sai.

Services trở thành các cục monolithic với nhiều hàm, chức năng khác được hỗ trợ nhau. Chính vì thế, khi phát triển services kiểu SOA rồi dán nhãn Microservices hoàn toàn bị đánh lạc hướng và không mang lại bất kỳ lợi ích nào.

##### Những điều cần phải tuân thủ

Một Service có phạm vi và chức năng giới hạn thì việc tập trung vào một nhiệm vụ sẽ giúp cho quá trình phát triển cũng như triển khai dịch vụ trở nên nhanh chóng hơn.

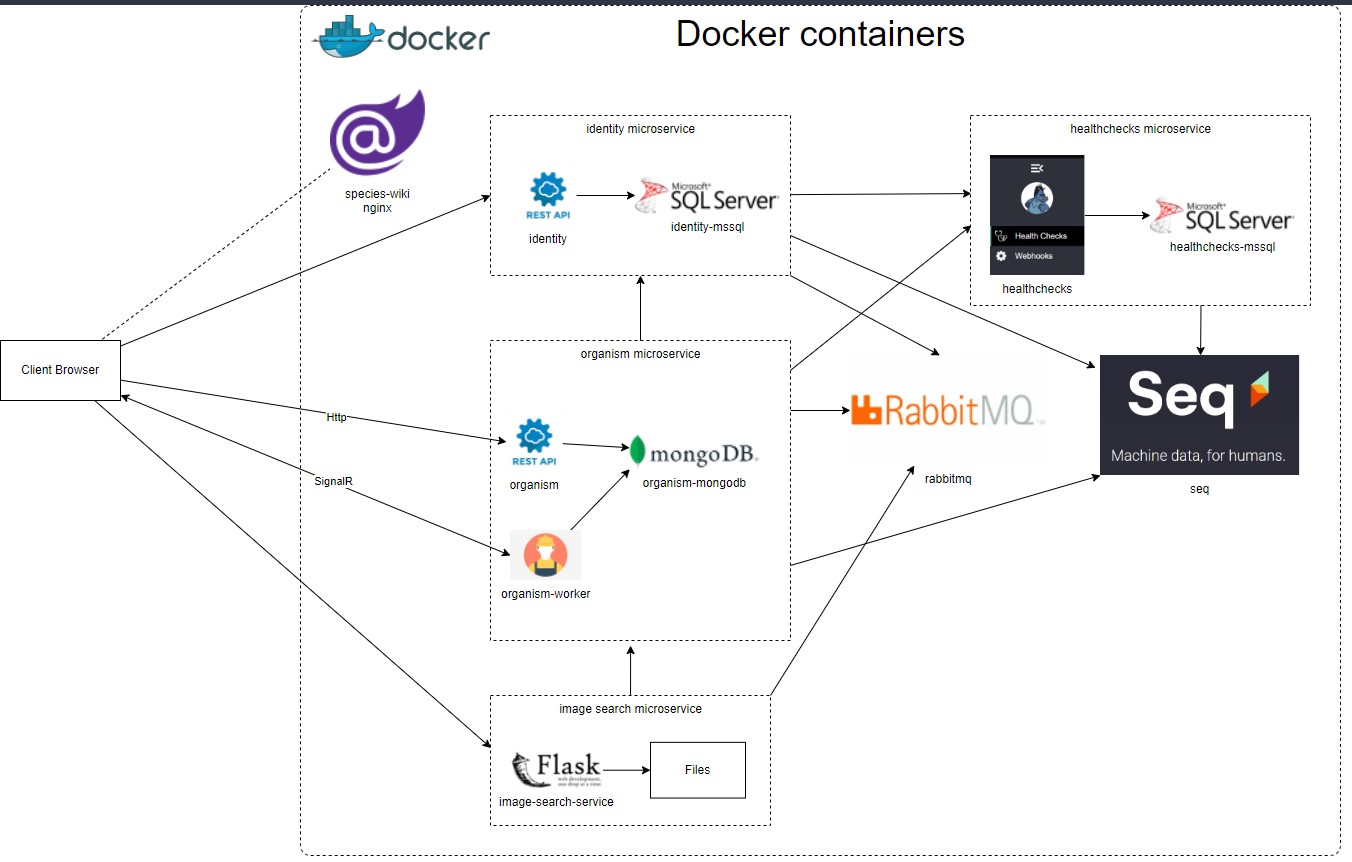
Khi thiết kế, bạn nên xác định và giới hạn cho các service dựa theo chức năng nghiệp vụ thực tế.

Hãy đảm bảo Microservices có thể phát triển cũng như được triển khai độc lập.

Mục tiêu thiết kế là đưa ra phạm vi cho một Microservices phục vụ cho một nghiệp vụ chứ không đơn giản là làm những dịch vụ nhỏ hơn. Khi đó, kích thước hợp lý của một services đó chính là kích thước đủ để đáp ứng cho các yêu cầu của một chức năng bên trong hệ thống.

Khác biệt so với services trong SOA thì một Microservices không nên có quá nhiều hàm hoặc chức năng hỗ trợ xung quanh cũng như định dạng thông báo gửi hoặc gửi tin đơn giản.

### Xây dựng mô hình hệ thống

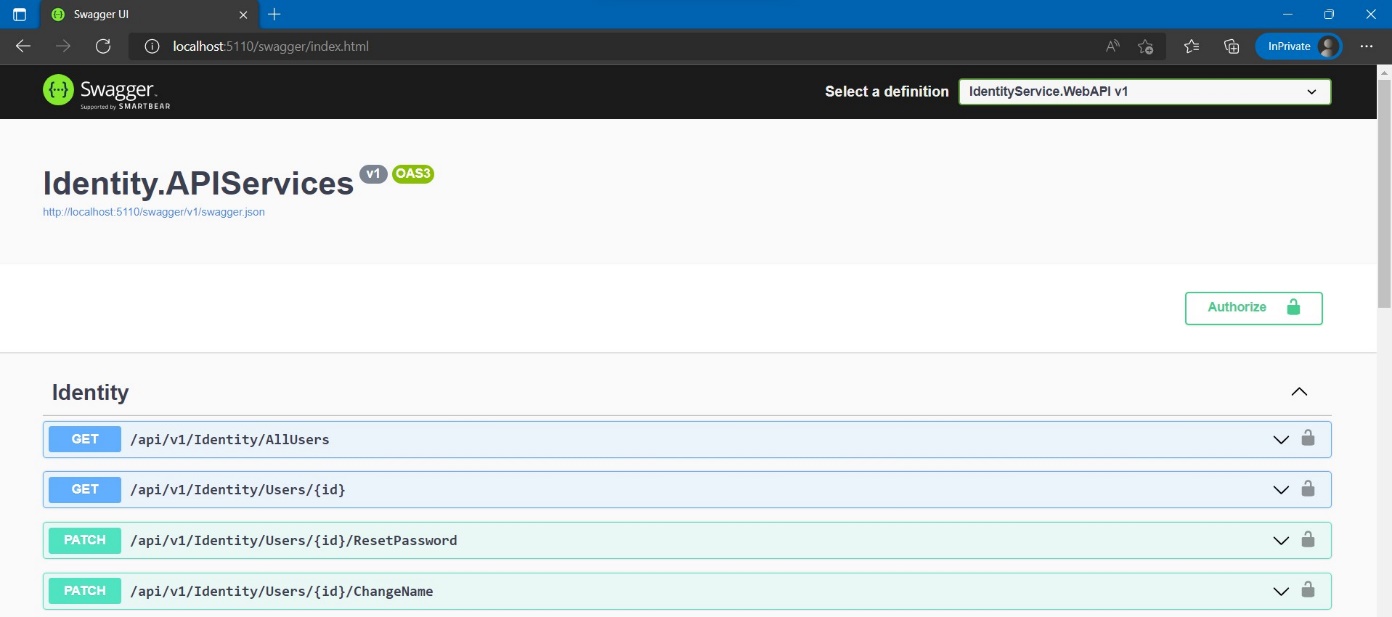


Hình 23 Kiến trúc tổng quan của hệ thống

Tất các services đều chạy dưới dạng một Docker container. Các service bao gồm:

1. Identity service: một web service dùng để xác thực người dùng.
2. Identity-mssql: CSDL SQL Server của identity.
3. Organism: web service dùng để quản lý loài.
4. Organism-mongodb: CSDL Mongo của Organism.
5. Organism-worker: một worker service dùng để xử lý dữ liệu không tuần tự.
6. Image search service: web service dùng để tìm hình ảnh của loài.
7. Rabbitmq: Một message queue là trung tâm trung chuyển các tin nhắn giữa các service với nhau.
8. Healthchecks: giao diện web với chức năng kiểm tra sức khỏe của các service còn lại trong hệ thống.
9. Healthchecks-mssql: CSDL SQL Server của healthchecks.
10. Seq: Trung tâm lưu trữ các log của các service.
11. Species-wiki: Một nginx service nhiệm vụ chính là gửi các static file của trang web species-wiki cho client.

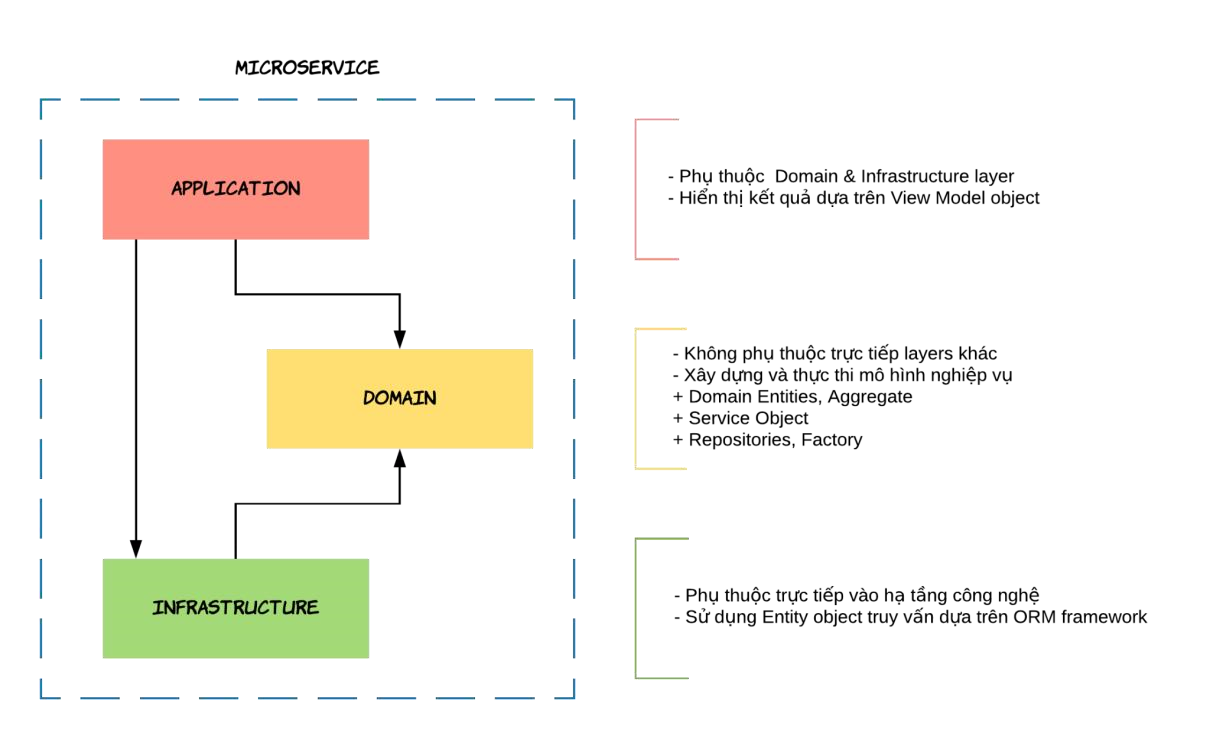
## Identity Service



Hình 24 Identity REST API

### Tổng quan service

Service dùng để xác thực người dùng sử dụng hệ thống sử dụng token. Được xây dựng trên .NET 6 một project WEB API theo mô hình MVC và design pattern DDD (Domain Driven Design), sử dụng Entity Framework để kết nối đến CSDL SQL Server. Kết nối ngoài với các service khác: RabbitMQ, Seq, HealthChecks.



Hình 25 Domain-Driven Design

Hệ thống hoạt động bao gồm các nhóm người dùng sau đây:

1. Researcher (Nhà nghiên cứu) là người dùng đã có tài khoản nhưng quyền hạn bị hạn chế.
2. Expert (Chuyên gia) là nhóm người dùng có toàn quyền trong hệ thống.

Các chức năng mà nhóm người dùng có thể sử dụng:

* Researcher:
  + Đăng nhập
  + Đổi tên
  + Đổi mật khẩu
  + Tạo token mới
  + Lấy thông đăng nhập
* Expert:
  + Lấy tất cả các người dùng
  + Lấy người dùng theo id
  + Đổi tên của 1 người dùng theo id
  + Reset mật khẩu 1 người dùng theo id
  + Tạo tài khoản mới

### Microsoft Authentication

Các project của C# có thêm phần xác thực người dùng nếu chọn trong lúc khởi tạo project nhưng có một hạn chế về cách xác thực tích hợp này. Một số loại xác thực có sẵn là:

Windows authentication: xác thực người dùng sử dụng email của Microsoft. Một số hạn chế là người dùng phải có email Microsoft và phải sử dụng Windows mới có thể sử dụng cách xác thực này. Trong hệ thống Microservice này sử dụng Docker và hệ điều hành là Linux nên sẽ khó để xác thực người dùng theo cách này.

Microsoft Identity Platform: xác thực người dùng bằng nền tảng của Microsoft, cụ thể ở đây là Azure. Sử dụng cách này thì yêu cầu phải có email Azure và có service đang chạy trên đó để xác thực. Điều này sẽ phải tốn tiền duy trì service trong khi phát triển phần mềm cũng như sẽ không hoạt động khi không kết nối được.

Individual Accounts: cái này thường được sử dụng nhiều hơn, xác thực bằng cách này phải có kết nối đến một sở dữ liệu và từ đó Entity Framework sẽ tạo ra các bảng cần thiết để đăng nhập. Ưu điểm là dữ liệu của mình sẽ được lưu nội bộ và còn có sẵn giao diện để đăng nhập cũng như là chỉnh sửa thông tin các nhân.

### Các hạn chế của Microsoft Authentication

Xác thực bằng phương pháp Individual Accounts nghe có vẻ rất tiện lợi như có một số lý do em không chọn cái này cho project:

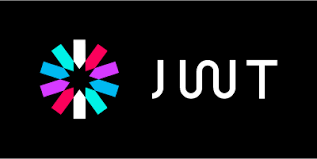
Xác thực người dùng sử dụng cookie: khi đăng nhập vào hệ thống thì người dùng sẽ được cung cấp cookie để duy trì trạng thái đăng nhập mỗi lần gọi server sẽ tự thêm cookie và header. Có một điểm yếu trong cách xác thực này trong hệ thống microservice, các cookie chỉ hoạt động khi có cùng tên miền liên quan đến cookie đó và khi đổi tên miền thì sẽ không thể nào mà xác thực được nữa, đó là một điểm yếu của cookie vì trong microservice các service khác nhau sẽ có các tên miền khác nhau và điều này sẽ gây khó khăn trong việc xác thực. Và khi muốn mở rộng trên nền tảng di động sẽ không hoạt động được vì cookie chỉ có ở trên web.

Khó chỉnh sửa các trang web: từ .Net core 2.1 các trang đăng nhập cũng như là chỉnh sửa thông tin sẽ trở thành Razor class UI và được đóng gói trong nuget package (<https://devblogs.microsoft.com/dotnet/aspnetcore-2-1-identity-ui/>) việc này sẽ khiến các giao diện khỏ để chỉnh sửa hơn làm giảm việc trải nghiệm của người dùng.

Lưu trữ dữ liệu: Individual Accounts sử dụng Entity Framework để kết nối các cơ sở dữ liệu, Entity Framework là chỉ có thể kết nối đến các cơ sở dữ liệu quan hệ như: SQL Server, SQL LocalDb, SQLite, ... . Và khi chạy Entity Framework sẽ tự tạo các bảng trong sở dữ liệu, với nghiệp vụ hiện tại của hệ thống thì việc tạo nhiều bảng khiến hao bộ nhớ để lưu trữ không cần thiết.

Học sử dụng: việc sử dụng hệ thống có sẵn tất nhiên sẽ phải tốn rất nhiều thời gian tìm tòi các tài liệu về cách sử dụng, sẽ phải hiểu các thư viện cũng như là cấu hình để sử dụng. Vì một số hạn chế của việc xác thực tích hợp của Microsoft nên em đã xây dựng một service riêng dành cho việc xác thực người dùng. Là một web service cung cấp một số các chức năng cơ bản để xác thực người dùng và thay vì sử dụng cookie, em sẽ dùng Json Web Token (JWT).

### Xác thực bằng token:



Hình 26 JWT

Json Web Token (<https://jwt.io/>): sử dụng jwt sẽ có một số lợi thế nhất định so với cookie. Jwt sẽ có bảo mật cao hơn cookie vì được mã hóa với khóa điều này khiến cho các hacker khi lấy được token sẽ không thể chỉnh sửa token để dùng CSRF (Cross-site request forgery) tấn công người dùng (<https://en.wikipedia.org/wiki/Cross-site_request_forgery> ). Và nếu hacker có được jwt thì jwt cũng có thời hạn ngắn từ 30p đến 1h nên sẽ hacker sẽ không gây ra quá nhiều rắc rối cho hệ thống. Khi đăng nhập hệ thống sẽ trả về jwt token cho người dùng, và khi người dùng sử dụng các service cần đến việc xác thực, ở đây là các REST api endpoints thì với mỗi request đến endpoint trong phần header đều phải chèn vào jwt token. Các RESTful api thường là stateless có nghĩa là sẽ không có lưu dữ liệu bên api mà là trong CSDL, việc sử dụng jwt các thông tin người dùng đã có sẵn trong jwt nên chỉ cần lấy ra thông tin bên trong jwt.

Refresh token: khác với jwt thì refresh token là một chuỗi mà hóa được tạo ra nhằm mục đích cung cấp cho người dùng một jwt mới, vì theo đề cập bên trên thì jwt có hạn sử dụng ngắn cho nên khi hết hạn người dùng sẽ phải tạo token mới mà việc tạo token mới đòi hỏi người dùng phải đăng nhập vào hệ thống, điều này sẽ làm giảm trải nghiệm của người dùng đi rất nhiều. Vì thế refresh token được tạo ra để cải thiện việc này, thay vì mỗi lúc jwt hết hạn người dùng lại phải đăng nhập lại, các service nào có nhu cầu sử dụng identity service sẽ dùng refresh token để gọi identity service tạo một jwt cho người dùng, từ đó dùng jwt mới để duy trì trạng thái đăng nhập. Refresh token có thời gian sử dụng lâu hơn với jwt token từ vài tuần đến vài tháng.

Lưu trữ các token: jwt token được tạo ra rồi cung cấp cho người dùng và không có lưu lại vào trong CSDL, khi người dùng làm mất thì chỉ cần tạo token mới. Refresh token thì ngược lại vì hạn sử dụng lâu nên sẽ phải lưu lại vào CSDL, refresh token là chuỗi do mình tự đặt ra nên có thể là bất cứ gì ví dụ như là: GUI của C# để tiện trong việc truy vấn, nhưng ở đây em sử dụng luôn jwt token là một refresh token. Khi các service khác gọi identity service để đăng nhập, khi thành công sẽ trả về 2 token là jwt token và refresh token và cả 2 đều là jwt token. Và service khác lưu lại 2 token, species wiki service là một web spa nên các token sẽ được lưu vào local storage và nếu muốn thêm di động thì sẽ lưu vào file như SQLite.

Bảo mật các token: jwt token có hạn sử dụng ngắn để bảo mật, còn đối với refresh token hạn sử dụng dài thì mỗi lúc đăng nhập hệ thống sẽ tạo một refresh token mới và bỏ token cũ nên các hacker sẽ không sử dụng refresh token cũ để tạo jwt token mới.

### Bảo mật

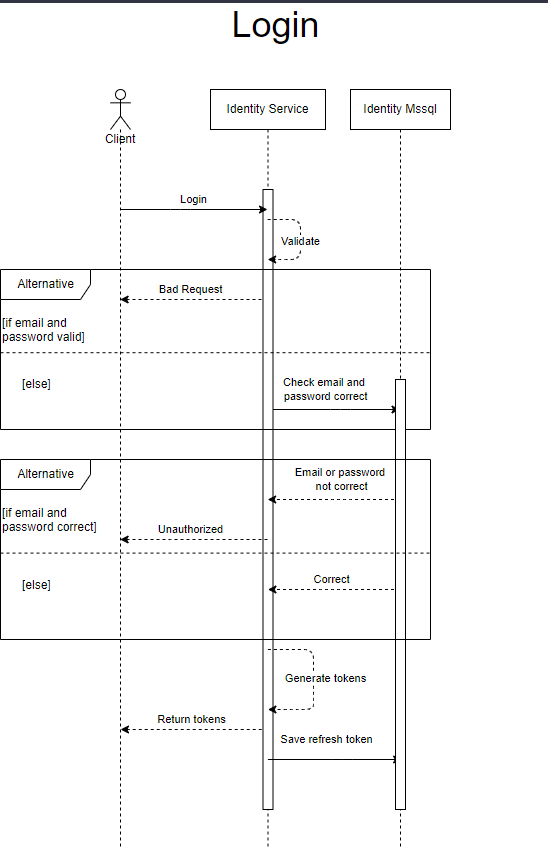
Để tăng tính bảo mật để phòng trường hợp hacker thâm nhập vào được hệ thống và truy xuất CSDL thì việc mã hóa mật khẩu là một điều cần thiết. Mật khẩu của người dùng sẽ được mã hóa 1 chiều sử dụng thuật toán SHA256 và hash cũng như salt sẽ được lưu vào CSDL.

### Các REST API quan trọng

#### Đăng nhập

|  |  |
| --- | --- |
| **Tên yêu cầu** | Đăng nhập |
| **Mục đích** | Chức năng này cho phép các người dùng đã có tài khoản đăng nhập vào hệ thống để tạo token. |
| **URI** | /api/v1/Identity/Login |
| **Param** | Không |
| **Method** | POST |
| **Header** | * accept: \*/\* * Content-Type: application/json |
| **Body** | {  "email": "string",  "password": "string"  } |
| **Response** | 1. 400 - Bad Request   {  "type": "https://tools.ietf.org/html/rfc7231#section-6.5.1",  "title": "One or more validation errors occurred.",  "status": 400,  "traceId": "00-3bba7d14ca8f39edd1dc2d69d18c5f61-77fe7c3a01b5b951-00",  "errors": {  "Email": [  "The field Email is invalid."  ]  }  }   1. 401 – Unauthorized   {  "errors": [  {  "id": "3f6cecc9-3bc7-4fd4-b564-2423bede8c2b",  "message": "Email:alex@gmail.com - Password:123457 incorrect.",  "atTime": "2022-05-07T01:52:48.7634056+00:00"  }  ]  }   1. 200 – Ok   {  "jwtToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1laWQiOiJjNWI2ZjZjNi1mYTRlLTRlNWEtYTJhMi0wYzc4MDIzMDQ2NzEiLCJuYW1lIjoiQWxleCBXYW5nIiwiZW1haWwiOiJhbGV4QGdtYWlsLmNvbSIsIm5iZiI6MTY1MTg4ODU0OCwiZXhwIjoxNjUxODkwMzQ4LCJpYXQiOjE2NTE4ODg1NDh9.zzzbfKV6Wlc-4RieGwpHT5rye9agUy09Z8\_DGOz3q4E",  "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJVc2VySWQiOiJjNWI2ZjZjNi1mYTRlLTRlNWEtYTJhMi0wYzc4MDIzMDQ2NzEiLCJuYmYiOjE2NTE4ODg1NDgsImV4cCI6MTY1NDQ4MDU0OCwiaWF0IjoxNjUxODg4NTQ4fQ.hXQCFRLAUEu7BsakCwzP3tYz2sOR38tRyHAANotZyDM"  } |

Bảng 3 REST API Đăng Nhập

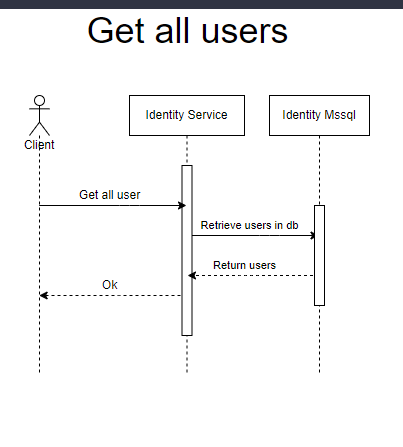


Hình 27 Sequence Diagram Login

#### Lấy tất cả người dùng

|  |  |
| --- | --- |
| **Tên yêu cầu** | Lấy tất cả người dùng |
| **Mục đích** | Chức năng này cho expert lấy các người dùng đã có trong hệ thống. |
| **URI** | /api/v1/Identity/AllUsers |
| **Param** | Không |
| **Method** | GET |
| **Header** | Không |
| **Body** | Không |
| **Response** | 200 – Ok  [  {  "id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",  "name": {  "firstName": "string",  "lastName": "string"  },  "email": "string",  "role": "string",  "dateCreated": "2022-05-07T03:34:23.291Z"  }  ] |

Bảng 4 REST API Lấy tất cả người dùng

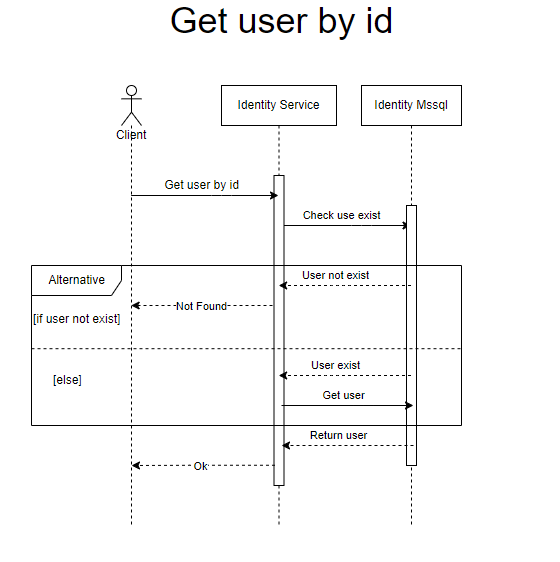


Hình 28 Sequence Diagram Get all users

#### Lấy người dùng theo id

|  |  |
| --- | --- |
| **Tên yêu cầu** | Lấy người dùng theo id |
| **Mục đích** | Chức năng này cho expert lấy các người dùng đã có trong hệ thống theo id. |
| **URI** | /api/v1/Identity/Users/{id} |
| **Param** | Id string($uuid) |
| **Method** | GET |
| **Header** | Không |
| **Body** | Không |
| **Response** | 1. 404 – Not Found   {  "type": "https://tools.ietf.org/html/rfc7231#section-6.5.4",  "title": "Not Found",  "status": 404,  "traceId": "00-5c4c373acc595dff9b168bc87f1bc3fe-b8b291c77ea7f4a9-00"  }   1. 200 – Ok   {  "id": "c5b6f6c6-fa4e-4e5a-a2a2-0c7802304671",  "name": "Alex Wang",  "email": "alex@gmail.com"  } |

Bảng 5 REST API Lấy người dùng theo id

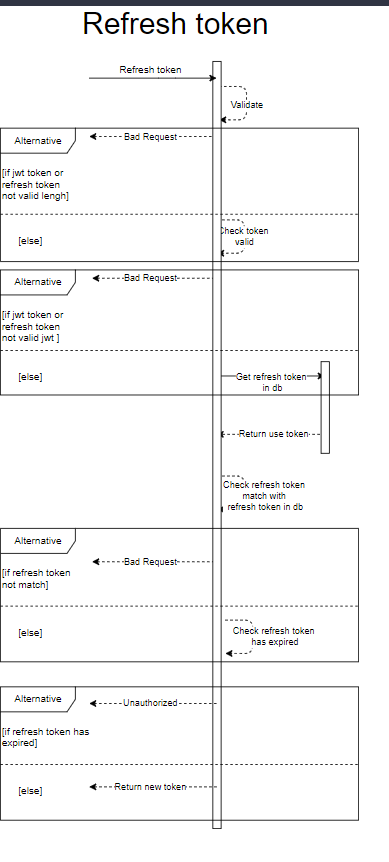


Hình 29 Sequence Diagram Get user by id

#### Refresh token

|  |  |
| --- | --- |
| **Tên yêu cầu** | Tạo một token mới |
| **Mục đích** | Chức năng này tạo một token mới từ refresh token |
| **URI** | /api/v1/Identity/RefreshToken |
| **Param** | Không |
| **Method** | POST |
| **Header** | * accept: \*/\* * Content-Type: application/json |
| **Body** | {  "jwtToken":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1laWQiOiJjNWI2ZjZjNi1mYTRlLTRlNWEtYTJhMi0wYzc4MDIzMDQ2NzEiLCJuYW1lIjoiQWxleCBXYW5nIiwiZW1haWwiOiJhbGV4QGdtYWlsLmNvbSIsIm5iZiI6MTY1MTkwNzEyOSwiZXhwIjoxNjUxOTA4OTI5LCJpYXQiOjE2NTE5MDcxMjl9.vRh3RIsytbHMG3T-58AILwuUn6FnGz-PMrmoI57ukL0",  "refreshToken":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJVc2VySWQiOiJjNWI2ZjZjNi1mYTRlLTRlNWEtYTJhMi0wYzc4MDIzMDQ2NzEiLCJuYmYiOjE2NTE5MDcxMjksImV4cCI6MTY1NDQ5OTEyOSwiaWF0IjoxNjUxOTA3MTI5fQ.okfpS3ZigJH3Qi5VLsaXBtY-8kfln-vQurKsEq-1KFQ"  } |
| **Response** | 1. 400 – Bad Request   [  {  "id": "695fc800-deee-4b4d-9ddc-491f249d1ad0",  "message": "The field JwtToken must be a string with a minimum length of 50 and a maximum length of 2147483647.",  "atTime": "2022-05-07T07:02:33.2011043+00:00"  },  {  "id": "bd9c5647-d640-44e1-9d68-3cfb708f5603",  "message": "The field RefreshToken must be a string with a minimum length of 50 and a maximum length of 2147483647.",  "atTime": "2022-05-07T07:02:33.2012639+00:00"  }  ]   1. 400 – Bad Request (không phải jwt)   [  {  "id": "03ad0082-a262-4bad-9a65-2275ac0ef776",  "message": "JwtToken:eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1laWQiOiJjNWI2ZjZjNi1mYTRlLTRlNWEtYTJhMi0wYzc4MDIzMDQ2NzEiLCJuYW1lIjoiQWxleCBXYW5nIiwiZW1haWwiOiJhbGV4QGdtYWlsLmNvbSIsIm5iZiI6MTY1MTkwNzEyOSwiZXhwIjoxNjUxOTA4OTI5LCJpYXQiOjE2NTE5MDcxMjl9.vRh3RIsytbHMG3T-58AILwuUn6FnGz-PMrmoI57ukLa not valid",  "atTime": "2022-05-07T07:08:52.961783+00:00"  }  ]   1. 400 – Bad Request (Refresh token không tồn tại)   [  {  "id": "2f4bbe43-24ee-4a53-80cf-38b66055148d",  "message": "RefreshToken:eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJVc2VySWQiOiJjNWI2ZjZjNi1mYTRlLTRlNWEtYTJhMi0wYzc4MDIzMDQ2NzEiLCJuYmYiOjE2NDkxMTk1NTIsImV4cCI6MTY1MTE5MzE1MiwiaWF0IjoxNjQ5MTE5NTUyfQ.wbJrqPvpyuboEX9Ft2ia3s6NKeAXjoKoFtqT99cPwC8 not exist.",  "atTime": "2022-05-07T07:35:21.5385318+00:00"  }  ]   1. 401 – Unauthorized (Refresh token hết hạn)   [  {  "id": "3cc70687-f16f-413a-a185-72d02be1bd08",  "message": "RefreshToken:eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJVc2VySWQiOiJjNWI2ZjZjNi1mYTRlLTRlNWEtYTJhMi0wYzc4MDIzMDQ2NzEiLCJuYmYiOjE2NDkxMTk1NTIsImV4cCI6MTY1MTE5MzE1MiwiaWF0IjoxNjQ5MTE5NTUyfQ.wbJrqPvpyuboEX9Ft2ia3s6NKeAXjoKoFtqT99cPwC8 has expired.",  "atTime": "2022-05-07T07:42:44.3532824+00:00"  }  ]   1. 200 – Ok   {  "jwtToken":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1laWQiOiJjNWI2ZjZjNi1mYTRlLTRlNWEtYTJhMi0wYzc4MDIzMDQ2NzEiLCJuYW1lIjoiQWxleCBXYW5nIiwiZW1haWwiOiJhbGV4QGdtYWlsLmNvbSIsIm5iZiI6MTY1MTkwNzEyOSwiZXhwIjoxNjUxOTA4OTI5LCJpYXQiOjE2NTE5MDcxMjl9.vRh3RIsytbHMG3T-58AILwuUn6FnGz-PMrmoI57ukL0"  } |

Bảng 6 REST API Refresh token

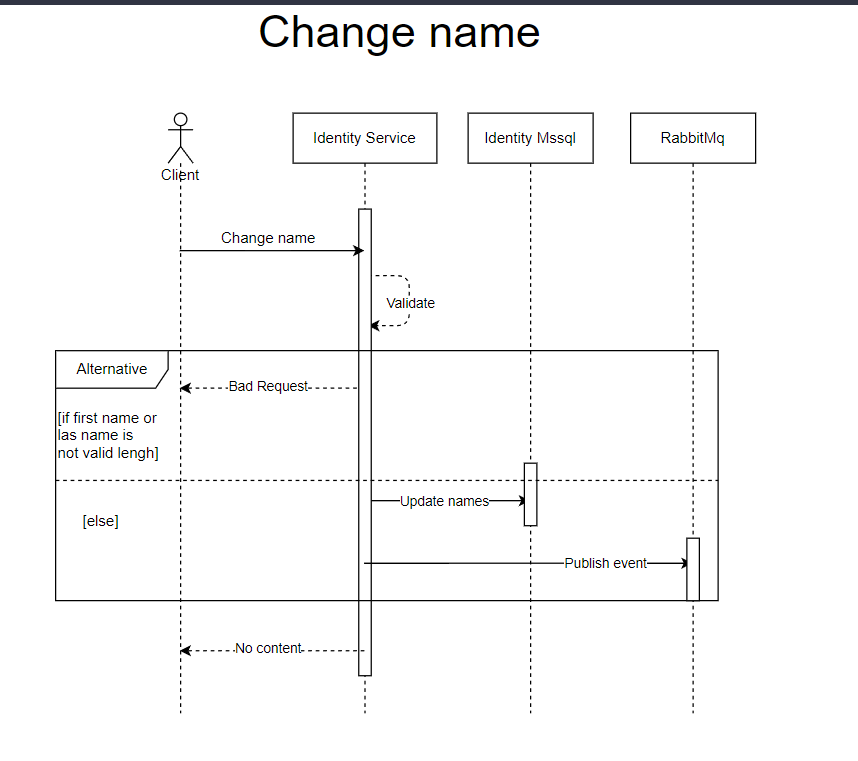


Hình 30 Sequence Diagram Refresh token

#### Change name

|  |  |
| --- | --- |
| **Tên yêu cầu** | Người dùng đổi tên |
| **Mục đích** | Chức năng này cho phép người dùng đổi tên |
| **URI** | /api/v1/Identity/ChangeName |
| **Param** | Không |
| **Method** | PATCH |
| **Header** | * accept: \*/\* * Content-Type: application/json |
| **Body** | {  "firstName": "string",  "lastName": "string"  } |
| **Response** | 1. 400 – Bad Request   {  "type": "https://tools.ietf.org/html/rfc7231#section-6.5.1",  "title": "One or more validation errors occurred.",  "status": 400,  "traceId": "00-6162704397c1deb5a87250b09a85bb4c-4a86f11f300a0ac2-00",  "errors": {  "LastName": [  "The field LastName must be a string with a minimum length of 4 and a maximum length of 50."  ],  "FirstName": [  "The field FirstName must be a string with a minimum length of 4 and a maximum length of 50."  ]  }  }   1. 204 – No Content |

Bảng 7 REST API Change name

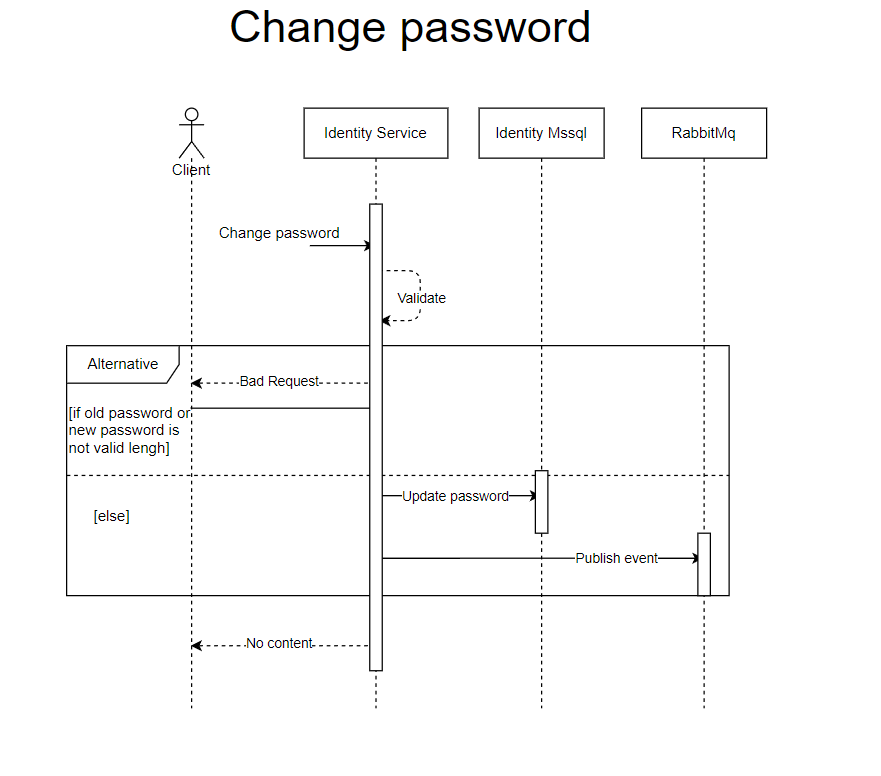


Hình 31 Sequence Diagram Change name

#### Change password

|  |  |
| --- | --- |
| **Tên yêu cầu** | Người dùng đổi mật khẩu |
| **Mục đích** | Chức năng này cho phép người dùng đổi mật khẩu |
| **URI** | /api/v1/Identity/ChangePassword |
| **Param** | Không |
| **Method** | PATCH |
| **Header** | * accept: \*/\* * Content-Type: application/json |
| **Body** | {  "oldPassword": "123456",  "newPassword": "123456",  "confirmNewPassword": "123456"  } |
| **Response** | 1. 400 – Bad Request   {  "type": "https://tools.ietf.org/html/rfc7231#section-6.5.1",  "title": "One or more validation errors occurred.",  "status": 400,  "traceId": "00-bfe8a8f5fd64238b2acf5607218d76c4-61114f53c6d6ee52-00",  "errors": {  "NewPassword": [  "The field NewPassword must be a string with a minimum length of 6 and a maximum length of 100."  ],  "OldPassword": [  "The field OldPassword must be a string with a minimum length of 6 and a maximum length of 100."  ],  "ConfirmNewPassword": [  "The field ConfirmNewPassword must be a string with a minimum length of 6 and a maximum length of 100."  ]  }  }   1. 204 – No Content |

Bảng 8 REST API Change password

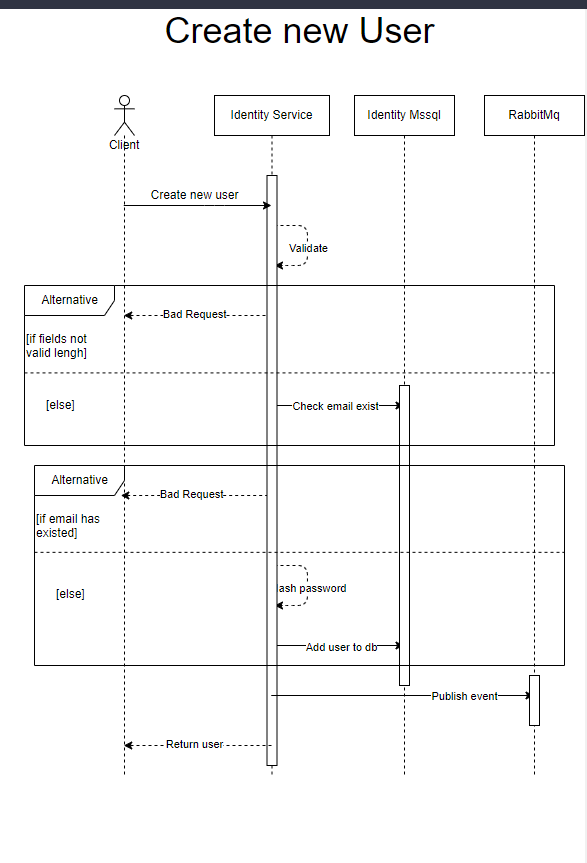


Hình 32 Sequence Diagram Change password

#### Create User

|  |  |
| --- | --- |
| **Tên yêu cầu** | Người dùng tạo tài khoản mới |
| **Mục đích** | Chức năng này cho phép người dùng tạo tài khoản mới |
| **URI** | /api/v1/Identity/Register |
| **Param** | Không |
| **Method** | POST |
| **Header** | * accept: \*/\* * Content-Type: application/json |
| **Body** | {  "firstName": "string",  "lastName": "string",  "email": "user@example.com",  "password": "string"  } |
| **Response** | 1. 400 – Bad Request   {  "type": "https://tools.ietf.org/html/rfc7231#section-6.5.1",  "title": "One or more validation errors occurred.",  "status": 400,  "traceId": "00-efaf09aab63ed91855d90252677c539d-8a81a3ed4e5c6734-00",  "errors": {  "Email": [  "The Email field is not a valid e-mail address."  ],  "LastName": [  "The field LastName must be a string with a minimum length of 4 and a maximum length of 50."  ],  "Password": [  "The field Password must be a string with a minimum length of 6 and a maximum length of 100."  ],  "FirstName": [  "The field FirstName must be a string with a minimum length of 4 and a maximum length of 50."  ]  }  }   1. 400 – Bad Request   [  {  "id": "d81633c6-3fde-4e0b-9a7c-0d90e08b1f73",  "message": "Email:user@gmail.com has been taken",  "atTime": "2022-05-07T08:20:37.6700647+00:00"  }  ]   1. 201 – Created   {  "id": "c5b6f6c6-fa4e-4e5a-a2a2-0c7802304671",  "name": "String String",  "email": "user@gmail.com"  } |

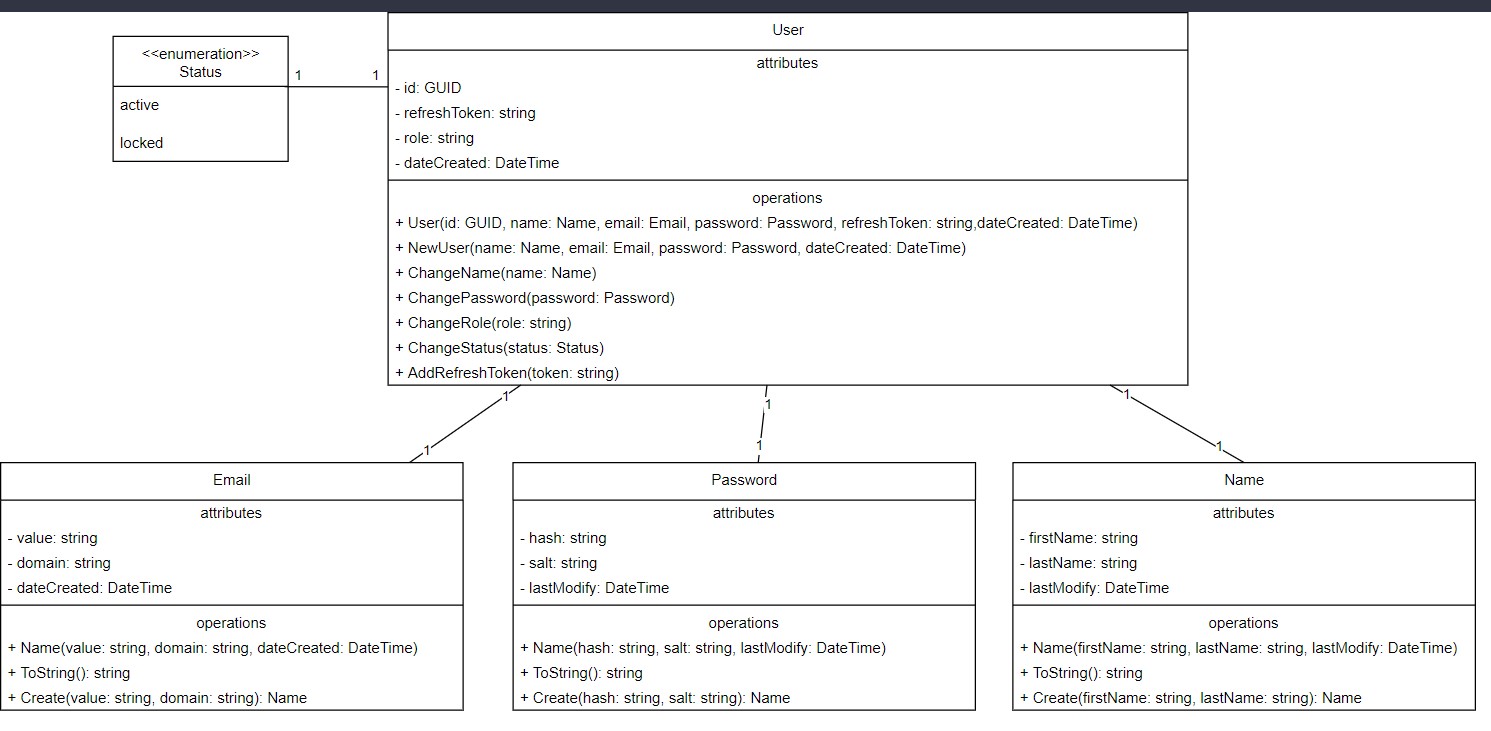
Bảng 9 REST API Create user



Hình 33 Sequence Diagram Create new user

### Thiết kế dữ liệu

#### Sơ đồ lớp

Dữ liệu được thiết kế theo DDD (Domain Driven Design)

Hình 34 Identity Class Diagram

#### Từ điển dữ liệu

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **STT** | **Tên thuộc tính** | **Kiểu** | **Khóa chính** | **Duy nhất** | **Diễn giải** |
| 1 | Id | UUID | X | X | Mã người dùng |
| 2 | Name\_FirstName | NVARCHAR(100) |  |  | Tên người dùng |
| 3 | Name\_LastName | NVARCHAR(100) |  |  | Họ người dùng |
| 4 | Name\_LastModify | DATETIME2(7) |  |  | Thời gian sửa |
| 5 | Email\_Value | VARCHAR(MAX) |  | X | Email của người dùng |
| 6 | Email\_DateCreated | DATETIME2(7) |  |  | Email ngày tạo |
| 7 | Password\_Hash | VARCHAR(500) |  |  | Mật khẩu hash |
| 8 | Password\_Salt | VARCHAR(500) |  |  | Salt cho hash của mật khẩu |
| 9 | Password\_LastModify | DATETIME2(7) |  |  | Thời gian sửa |
| 10 | RefreshToken | VARCHAR(500) |  |  |  |
| 11 | Role | VARCHAR(30) |  |  | Expert, Researcher |
| 12 | Status | VARCHAR(30) |  |  | Locked, Active |
| 13 | DateCreated | DATETIME2(7) |  |  |  |

Bảng 10 Bảng người dùng

### Giao tiếp ngoài

Identity Service phụ thuộc vào Identity Mssql vì là cơ sở dữ liệu của service ngoài ra còn có RabbitMq nơi mà identity sẽ gửi thông báo nếu có dữ liệu thay đổi trong identity. Các log của identity cũng sẽ được gửi đến Seq service để lưu trữ.

#### Identity Mssql

Identity Mssql là một cơ sở dữ liệu SQL Server, đây sẽ là cơ sở dữ liệu của identity service. Identity sẽ sử dụng Entity Framework Core để truy xuất dữ liệu. Identity service phụ thuộc vào service này và sẽ không thể hoạt động nếu identity-mssql không hoạt động.

#### RabbitMq

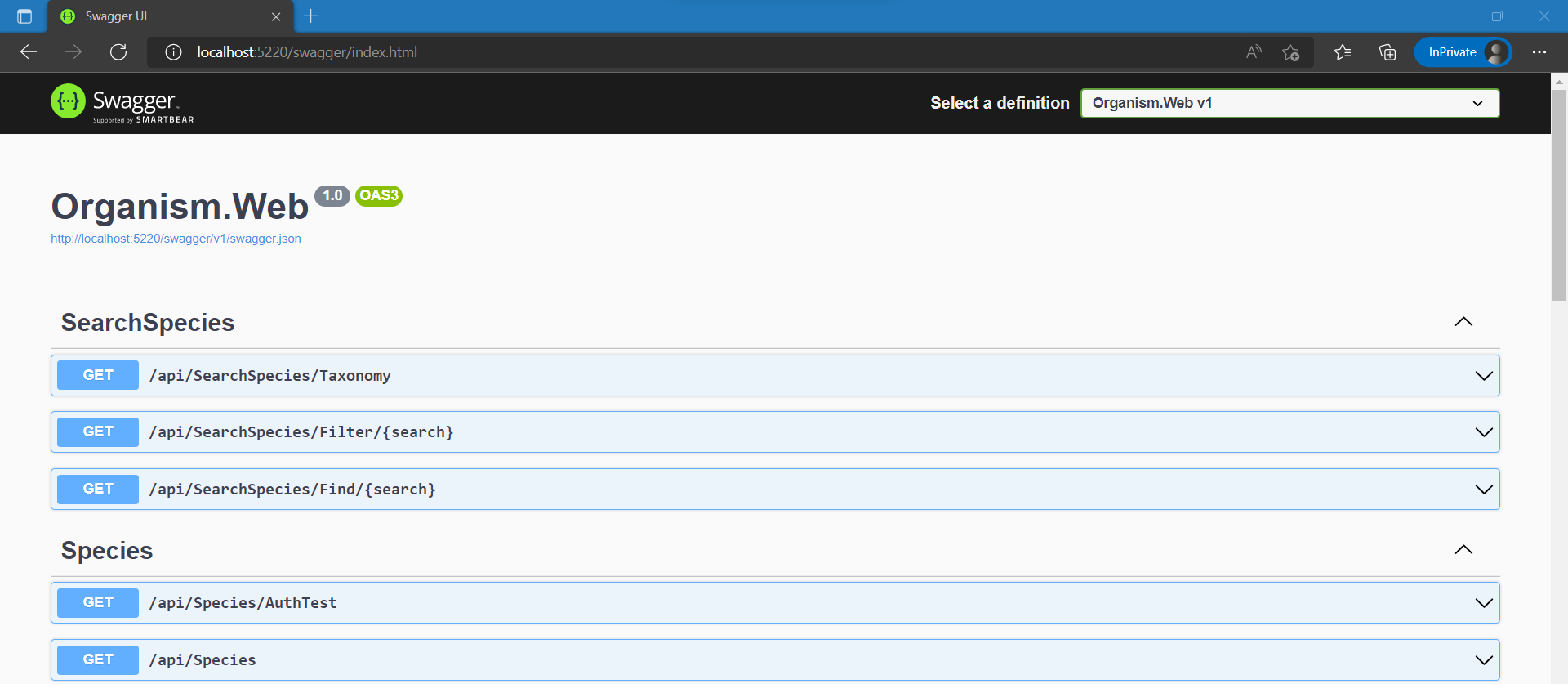
RabbitMq service là nơi mà identity sẽ gửi đi các thông báo nếu các dữ liệu bên trong bị thay đổi. Điều này sẽ giúp bảo vệ tính toàn vẹn của dữ liệu trong các service khác phụ thuộc vào identity.

Vì chỉ dùng để gửi thông báo nên khi RabbitMq không hoạt động sẽ làm ảnh hưởng đến một phần các thao tác chỉnh sửa dữ liệu, các thao tác lấy dữ liệu sẽ không thay đổi.

#### Seq

Seq service là nơi identity sẽ gửi các log của mình đến. Identity vẫn sẽ hoạt động bình thường nếu không có Seq service.

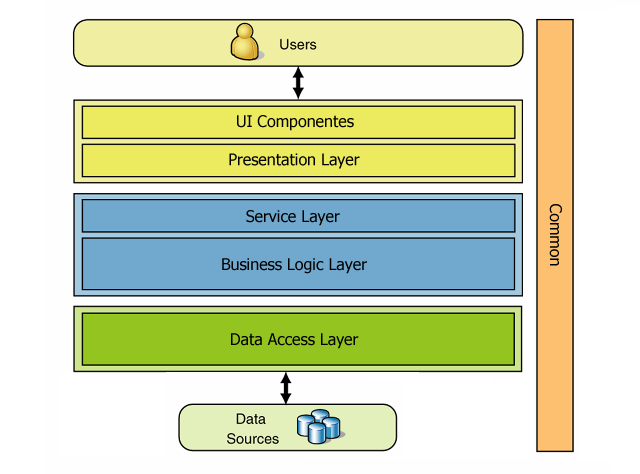
## Organism Service



Hình 35 Organism REST API

### Tổng quan service

Service dùng để quản lý các loài của người dùng sử dụng hệ thống, đăng nhập sử dụng token bên identity. Được xây dựng trên .NET 6 một project WEB API theo mô hình MVC và kiến trúc N-layer, sử dụng CSDL MongoDb. Kết nối ngoài với các service khác: RabbitMQ, Seq, HealthChecks.



Hình 36 Kiến trúc N-Layer

Hệ thống hoạt động bao gồm các nhóm người dùng sau đây:

1. Guest
2. Researcher
3. Expert

Các chức năng mà nhóm người dùng có thể sử dụng:

* Guest:
  + Lấy tất cả loài
  + Lấy loài theo id
  + Lấy loài của tác giả
  + Tìm loài theo tên
* Researcher:
  + Lấy tất cả loài
  + Lấy loài theo id
  + Lấy loài của tác giả
  + Tìm loài theo tên
  + Thêm loài mới
  + Sửa thông tin loài
* Expert:
  + Lấy tất cả loài
  + Lấy loài theo id
  + Lấy loài của tác giả
  + Tìm loài theo tên
  + Thêm loài mới
  + Sửa thông tin loài
  + Duyệt loài
  + Lấy thống kê thông tin loài

### Machine Learning

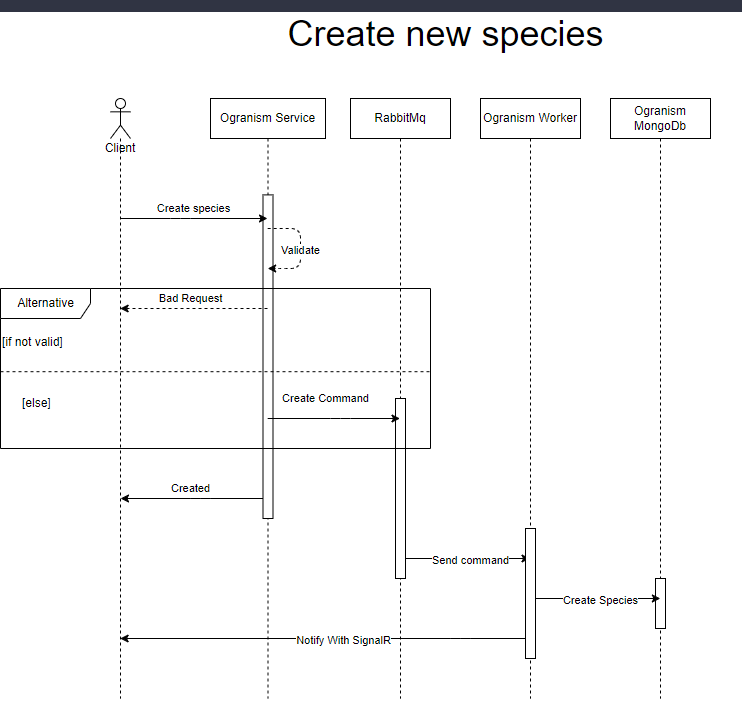
Việc tìm kiếm loài theo tên có áp dụng máy học vào để tăng sự chính xác trong việc tìm kiếm. MongoDb đã có full text searching, em sẽ áp dụng tính năng này của mongo bằng cách tạo index. Khi khởi chạy service, hệ thống sẽ tạo index cho việc tìm kiếm theo tên, tìm kiếm theo tên sẽ bao gồm tên gọi chung, tên địa phương và tên khoa học. Vì mongodb không có hỗ trợ ngôn ngữ tiếng Việt trong full text search nên em sẽ chọn index theo tiếng Pháp vì nó gần tiếng Việt hơn. Vậy khi sử dụng tìm kiếm theo tên đầu vào có thể dùng 3 loại tên đều được.

### Các REST API quan trọng

#### Tạo loài mới

|  |  |
| --- | --- |
| **Tên yêu cầu** | Tạo loài mới |
| **Mục đích** | Chức năng này cho phép người dùng tạo một loài mới. |
| **URI** | /api/Species |
| **Param** | Không |
| **Method** | POST |
| **Header** | * accept: \*/\* * Content-Type: application/json |
| **Body** | {  "name": "string",  "localName": "string",  "images": [  "string"  ],  "taxonomy": {  "kingdom": "string",  "phylum": "string",  "class": "string",  "order": "string",  "family": "string",  "genus": "string",  "species": "string"  },  "description": {  "morphology": "stringstringstringstringstring",  "ecosystem": "stringstringstringstringstring",  "useValue": "string"  },  "distribution": [  [  0  ]  ],  "moreInfo": {  "habitat": "stringstri",  "place": "stringstri",  "conservation": {  "additionalProp1": "string",  "additionalProp2": "string",  "additionalProp3": "string"  }  }  } |
| **Response** | 1. 400 - Bad Request   {  "type": "https://tools.ietf.org/html/rfc7231#section-6.5.1",  "title": "One or more validation errors occurred.",  "status": 400,  "traceId": "00-3bba7d14ca8f39edd1dc2d69d18c5f61-77fe7c3a01b5b951-00",  "errors": {  "Name": [  "The field Name is invalid."  ]  }  }   1. 201 – Created   {  "id": "625579b1d7793c3b2a32496f",  "name": "string",  "localName": "string",  "images": [  "string"  ],  "taxonomy": {  "kingdom": "string",  "phylum": "string",  "class": "string",  "order": "string",  "family": "string",  "genus": "string",  "species": "string"  },  "description": {  "morphology": "stringstringstringstringstring",  "ecosystem": "stringstringstringstringstring",  "useValue": "string"  },  "distribution": [  [  0  ]  ],  "moreInfo": {  "habitat": "stringstri",  "place": "stringstri",  "conservation": {  "additionalProp1": "string",  "additionalProp2": "string",  "additionalProp3": "string"  }  }  } |

Bảng 11 REST API Tạo loài mới

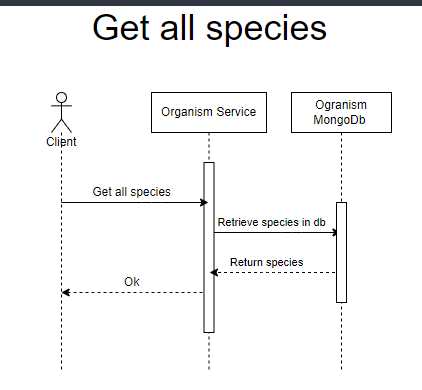


Hình 37 Sequence Diagram Create new species

#### Lấy tất cả loài

|  |  |
| --- | --- |
| **Tên yêu cầu** | Lấy tất cả loài |
| **Mục đích** | Chức năng này cho phép lấy các loài đã có trong hệ thống. |
| **URI** | /api/Species |
| **Param** | Không |
| **Method** | GET |
| **Header** | Không |
| **Body** | Không |
| **Response** | 200 – Ok  [  {  "id": "string",  "name": "string",  "localName": "string",  "taxonomy": {  "kingdom": "string",  "phylum": "string",  "class": "string",  "order": "string",  "family": "string",  "genus": "string",  "species": "string"  },  "description": {  "morphology": "string",  "ecosystem": "string",  "useValue": "string"  },  "distribution": [  [  0  ]  ],  "moreInfo": {  "habitat": "string",  "place": "string",  "conservation": {  "additionalProp1": "string",  "additionalProp2": "string",  "additionalProp3": "string"  }  },  "author": {  "id": "string",  "email": "string"  },  "timestamp": "2022-05-13T11:38:15.589Z",  "status": 0  }  ] |

Bảng 12 REST API Lấy tất cả loài

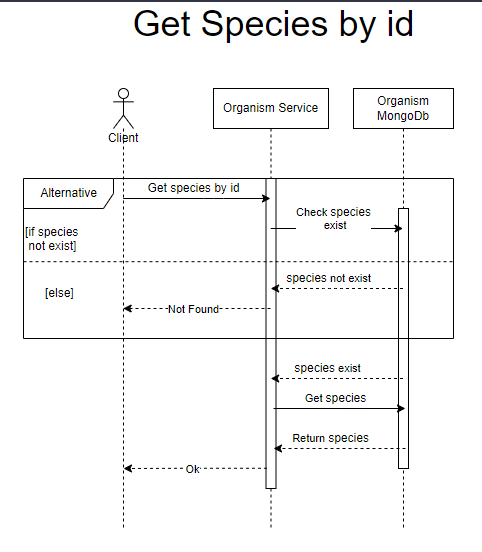


Hình 38 Sequence Diagram Get all species

#### Lấy loài theo id

|  |  |
| --- | --- |
| **Tên yêu cầu** | Lấy loài theo id |
| **Mục đích** | Chức năng này cho phép lấy các loài đã có trong hệ thống theo id. |
| **URI** | /api/ Species/{id} |
| **Param** | Id string |
| **Method** | GET |
| **Header** | Không |
| **Body** | Không |
| **Response** | 1. 404 – Not Found   {  "type": "https://tools.ietf.org/html/rfc7231#section-6.5.4",  "title": "Not Found",  "status": 404,  "traceId": "00-5c4c373acc595dff9b168bc87f1bc3fe-b8b291c77ea7f4a9-00"  }   1. 200 – Ok   {  "id": "string",  "name": "string",  "localName": "string",  "taxonomy": {  "kingdom": "string",  "phylum": "string",  "class": "string",  "order": "string",  "family": "string",  "genus": "string",  "species": "string"  },  "images": [  "string"  ],  "description": {  "morphology": "stringstringstringstringstring",  "ecosystem": "stringstringstringstringstring",  "useValue": "string"  },  "distribution": [  [  0  ]  ],  "moreInfo": {  "habitat": "stringstri",  "place": "stringstri",  "conservation": {  "additionalProp1": "string",  "additionalProp2": "string",  "additionalProp3": "string"  }  },  "author": {  "id": "string",  "email": "string"  },  "timestamp": "2022-05-13T11:41:50.147Z",  "status": "string"  } |

Bảng 13 REST API Lấy loài theo id

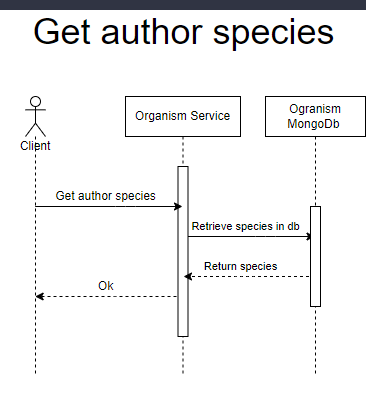


Hình 39 Sequence Diagram Get species by id

#### Lấy loài của tác giả

|  |  |
| --- | --- |
| **Tên yêu cầu** | Lấy loài của tác giả |
| **Mục đích** | Chức năng này cho phép lấy loài của tác giả đã có trong hệ thống. |
| **URI** | /api/Species/Authors/{id} |
| **Param** | Id string |
| **Method** | GET |
| **Header** | Không |
| **Body** | Không |
| **Response** | 200 – Ok  [  {  "id": "string",  "name": "string",  "localName": "string",  "taxonomy": {  "kingdom": "string",  "phylum": "string",  "class": "string",  "order": "string",  "family": "string",  "genus": "string",  "species": "string"  },  "description": {  "morphology": "string",  "ecosystem": "string",  "useValue": "string"  },  "distribution": [  [  0  ]  ],  "moreInfo": {  "habitat": "string",  "place": "string",  "conservation": {  "additionalProp1": "string",  "additionalProp2": "string",  "additionalProp3": "string"  }  },  "author": {  "id": "string",  "email": "string"  },  "timestamp": "2022-05-13T11:38:15.589Z",  "status": 0  }  ] |

Bảng 14 REST API Lấy loài của tác giả

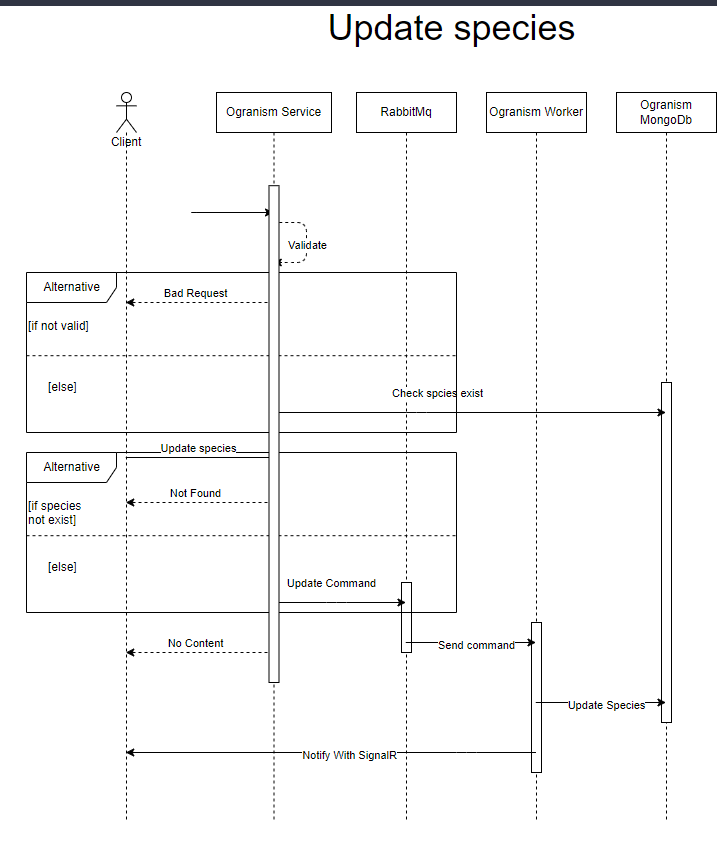


Hình 40 Sequence Diagram Get author species

#### Cập nhật loài

|  |  |
| --- | --- |
| **Tên yêu cầu** | Cập nhật loài |
| **Mục đích** | Chức năng này cho phép người dùng cập nhật loài. |
| **URI** | /api/Species/{id} |
| **Param** | Id string |
| **Method** | PUT |
| **Header** | * accept: \*/\* * Content-Type: application/json |
| **Body** | {  "name": "string",  "localName": "string",  "images": [  "string"  ],  "taxonomy": {  "kingdom": "string",  "phylum": "string",  "class": "string",  "order": "string",  "family": "string",  "genus": "string",  "species": "string"  },  "description": {  "morphology": "stringstringstringstringstring",  "ecosystem": "stringstringstringstringstring",  "useValue": "string"  },  "distribution": [  [  0  ]  ],  "moreInfo": {  "habitat": "stringstri",  "place": "stringstri",  "conservation": {  "additionalProp1": "string",  "additionalProp2": "string",  "additionalProp3": "string"  }  }  } |
| **Response** | 1. 400 - Bad Request   {  "type": "https://tools.ietf.org/html/rfc7231#section-6.5.1",  "title": "One or more validation errors occurred.",  "status": 400,  "traceId": "00-3bba7d14ca8f39edd1dc2d69d18c5f61-77fe7c3a01b5b951-00",  "errors": {  "Name": [  "The field Name is invalid."  ]  }  }   1. 204 – No Content |

Bảng 15 REST API Cập nhật loài

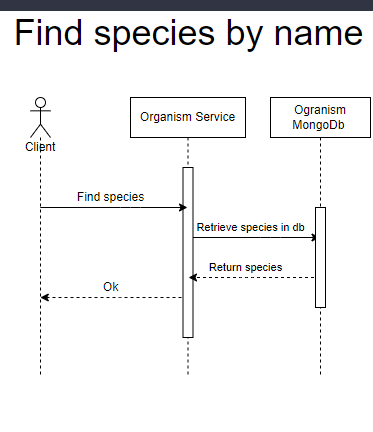


Hình 41 Sequence Diagram Update species

#### Tìm loài theo tên

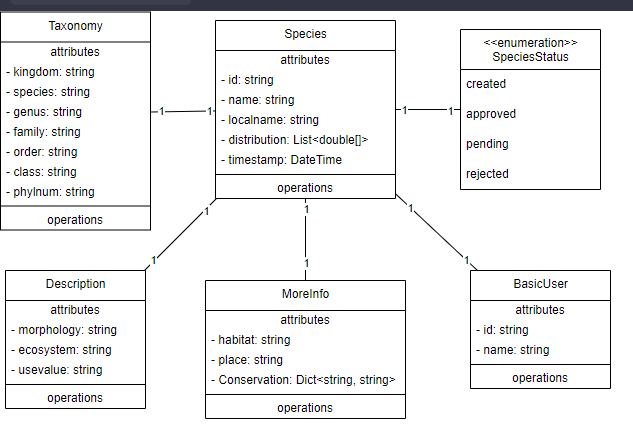
|  |  |
| --- | --- |
| **Tên yêu cầu** | Tìm loài theo tên |
| **Mục đích** | Chức năng này cho phép tìm các loài đã có trong hệ thống theo tên. |
| **URI** | /api/ Species/Find/{search} |
| **Param** | search string |
| **Method** | GET |
| **Header** | Không |
| **Body** | Không |
| **Response** | 200 – Ok  {  "id": "string",  "name": "string",  "localName": "string",  "taxonomy": {  "kingdom": "string",  "phylum": "string",  "class": "string",  "order": "string",  "family": "string",  "genus": "string",  "species": "string"  },  "images": [  "string"  ],  "description": {  "morphology": "stringstringstringstringstring",  "ecosystem": "stringstringstringstringstring",  "useValue": "string"  },  "distribution": [  [  0  ]  ],  "moreInfo": {  "habitat": "stringstri",  "place": "stringstri",  "conservation": {  "additionalProp1": "string",  "additionalProp2": "string",  "additionalProp3": "string"  }  },  "author": {  "id": "string",  "email": "string"  },  "timestamp": "2022-05-13T11:41:50.147Z",  "status": "string"  } |

Bảng 16 REST API Tìm loài theo tên



Hình 42 Sequence Diagram Find species by name

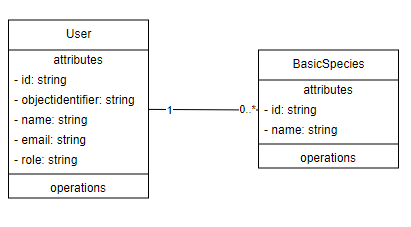
### Thiết kế dữ liệu



Hình 43 Species Class Diagram

Vì sử dụng mongodb nên tất cả bảng sẽ được gom lại thành một document. Vì để tối ưu hóa bộ nhớ khi nên lưu thông tin tác giả sẽ sử dụng BasicUser sẽ lưu 2 trường là id và name.

Hình 44 User Class Diagram



Lưu thông tin người dùng bên identity. Khác với identity, thông tin người dùng bên organism chỉ lưu các thông tin cần thiết bao gồm id, objectIdentifier, name, email. Id sẽ là trường id tự tạo của mongodb và objectIdentifier là sẽ id bên identity, không lưu password vì không có sử dụng đăng nhập.

Lưu các loài của người dùng, một người dùng sẽ có một collection loài. Và collection này sẽ sử dụng BasicSpecies để tối ưu hóa bộ nhớ sẽ chỉ lưu id và name.

### Worker service

Worker service là loại project mới của ASP .NET Core với chức năng chính là thực hiện các tác vụ chạy background. Trong Organism Microservice nhiệm vụ chính của Organism Worker service là lắng nghe các tin nhắn đến từ RabbitMq. Để bảo toàn cho tính toàn vẹn của dữ liệu trong Microservice, worker sẽ lắng nghe mỗi khi dữ liệu bên các service có thay đổi mà làm ảnh hưởng đến Organsim Service.

#### Lắng nghe thông báo từ service khác

##### Trường hợp tốt nhất

Organism service phụ thuộc vào identity service, organism sẽ sử dụng token identity để xác thực người dùng. Khi có một thay đổi dữ liệu bên identity thì identity sẽ gửi tin nhắn đến RabbitMq để thông báo, worker sẽ có nhiệm vụ xử lý thông báo này. Ví dụ về việc tạo người dùng mới bên identity được thể hiện trong sơ đồ tuần tự dưới đây:



Hình 45 Sequence Diagram Create new User event

Sơ đồ tuần tự khác với sơ đồ tuần tự bên identity vì có thêm sự góp mặt của Organism Worker Service và Organism MongoDb. Worker sẽ lấy event từ RabbitMq xuống và thêm người dùng mới vào CSDL MongoDb.

##### Trường hợp tệ nhất

Trường hợp dữ liệu không thống nhất. Theo sơ đồ bên trên Organism sẽ phải phụ thuộc vào RabbitMq để nhận thông báo khi có người dùng mới được thêm vào identity. Có 2 trường hợp cần phải được cân nhắc đến trong việc này:

1. RabbitMq không hoạt động
2. Identity gặp lỗi khi thông báo

Các trường hợp trên sẽ khiến cho dữ liệu chỉ tồn tại bên CSDL của identity mà không tồn tại trong MongoDb của Organism và lỗi sẽ xảy ra khi người dùng sử dụng token để xác thực đăng nhập mà người dùng không tồn tại trong CSDL.

Cách khắc phục tình trạng này, khi người dùng sử dụng token để đăng nhập, nhưng thông tin của người dùng trong token lại không tồn tại trong CSDL Mongo. Thay vì thông báo lỗi vì không tồn tại người dùng trong CSDL, worker service sẽ tạo một lời gọi api đến identity service để kiểm tra xem người dùng có tồn tại bên identity hay không. Khi lấy dữ liệu từ lời gọi api mà có tồn tại người dùng thì worker sẽ tự động thêm người dùng đó vào MongoDb còn không sẽ thông báo lỗi.

#### Xử lý bất đồng bộ

##### Async trong C#

Trong C# ta có thể tránh performance bottlenecks và nâng cao khả năng đáp ứng tổng thể của ứng dụng bằng cách sử dụng lập trình không đồng bộ. Cách viết ứng dụng không đồng bộ cũ thì rất phức tạp và kiến cho việc gỡ lỗi và bảo trì trở nên khó khăn. C # 5 đã giới thiệu một cách tiếp cận đơn giản, lập trình không đồng bộ, thúc đẩy hỗ trợ không đồng bộ trong .NET Framework 4.5 trở lên, .NET Core và Windows Runtime. Trình biên dịch thực hiện công việc khó khăn mà nhà phát triển đã từng làm và ứng dụng của bạn vẫn giữ cấu trúc logic như lập trình tuần tự. Và kết quả là, bạn nhận được tất cả các lợi thế của lập trình không đồng bộ với một phần nhỏ nỗ lực. Async cải thiện khả năng phản hồi Không đồng bộ là điều cần thiết cho các hoạt động có khả năng bị chặn, chẳng hạn như truy cập web. Quyền truy cập vào tài nguyên web đôi khi bị chậm hoặc bị trì hoãn. Nếu một hoạt động như vậy bị chặn trong một quy trình đồng bộ, thì toàn bộ ứng dụng phải đợi. Trong quy trình không đồng bộ, ứng dụng có thể tiếp tục với công việc khác không phụ thuộc vào tài nguyên web cho đến khi tác vụ chặn tiềm năng kết thúc.

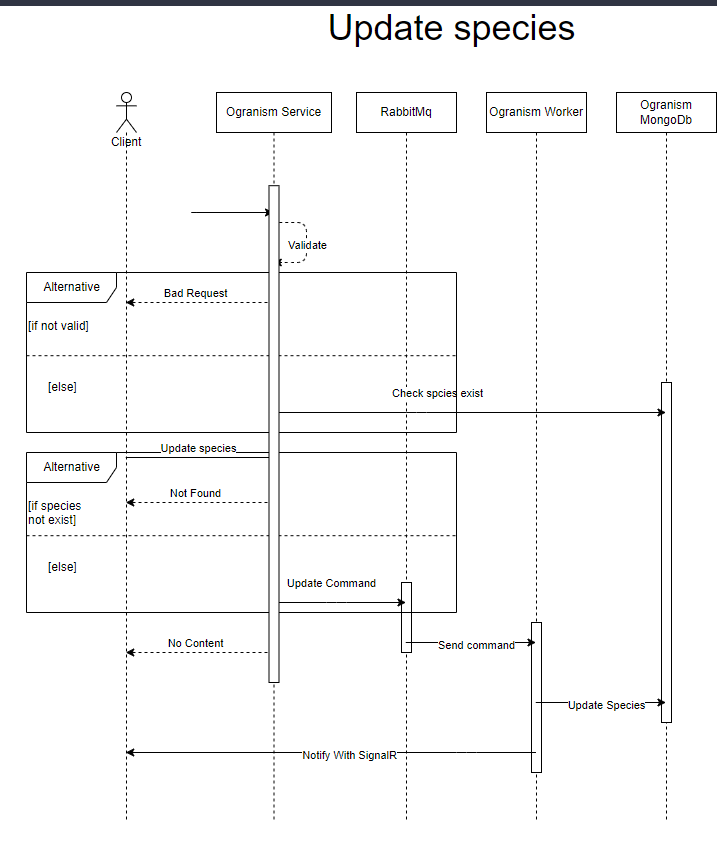
##### Bất cập của async trong C#

Trong C# có lập trình bất đồng bộ. Tuy nhiên bất đồng bộ này chỉ xảy ra trong nội bộ C#. Client giao tiếp với Organism Service sử dụng các api endpoints và dưới góc nhìn toàn cảnh và giao tiếp các api sử dụng chuẩn http. Trong http hoạt động dưới các cơ chế Request và Response và như thế http lại là tuần tự bởi vì dưới góc nhìn của client sử dụng các api này thì khi gửi Request phải luôn chờ cho Response trở về mới thể tiếp tục và thế http trở thành tuần tự.

##### Async trong Organism Service

Vậy Organism đảm nhận việc sử lấy bất tuần tự như thế nào?

Nhờ vào sự góp mặt của message queue mà ở đây chính là RabbitMq. RabbitMq ở đây sẽ hoạt động như một hàng đợi. Khi muốn thực thi các thao tác chỉnh sửa thông tin bên Organism, theo cách thông thường trong lúc nhận Request từ client ta sẽ trực tiếp chỉnh sửa thông tin trong MongoDb và trả về thông báo thành công hay thất bại. Và để làm Request theo cách async, Organism sẽ tạo một tin nhắn sửa thông tin và gửi đến RabbitMq, tiếp theo sẽ trả về 204 http Response code, có ý nghĩa thông báo rằng server đã nhận được và đang xử lý thông tin đó. Ví dụ được miêu tả trong sơ đồ tuần tự dưới đây trong việc cập nhật thông tin loài:



Hình 46 Sequence Diagram Update species

Cập nhật thông tin loài khi người dùng gọi api thì sẽ trả về 204 khi hoàn tất việc gửi tin nhắn đến RabbitMq. Việc xử lý yêu cầu của người dùng sẽ được thực hiện khi mà worker lấy tin nhắn từ RabbitMq, thông báo thành công hay thất bại thông qua SignalR.

### Giao tiếp ngoài

Organism Service phụ thuộc vào Organism MongoDb vì là cơ sở dữ liệu của service ngoài ra còn có RabbitMq nơi mà organism sẽ nhận thông báo nếu có dữ liệu thay đổi trong identity cũng như các công việc chạy async. Các log của identity cũng sẽ được gửi đến Seq service để lưu trữ.

#### Organism MongoDb

Organism MongoDb là một cơ sở dữ liệu phi quan hệ MongoDb, đây sẽ là cơ sở dữ liệu của organism service. Organism sẽ sử dụng Mongo driver để truy xuất dữ liệu. Organism service phụ thuộc vào service này và sẽ không thể hoạt động nếu organism-mongodb không hoạt động.

#### RabbitMq

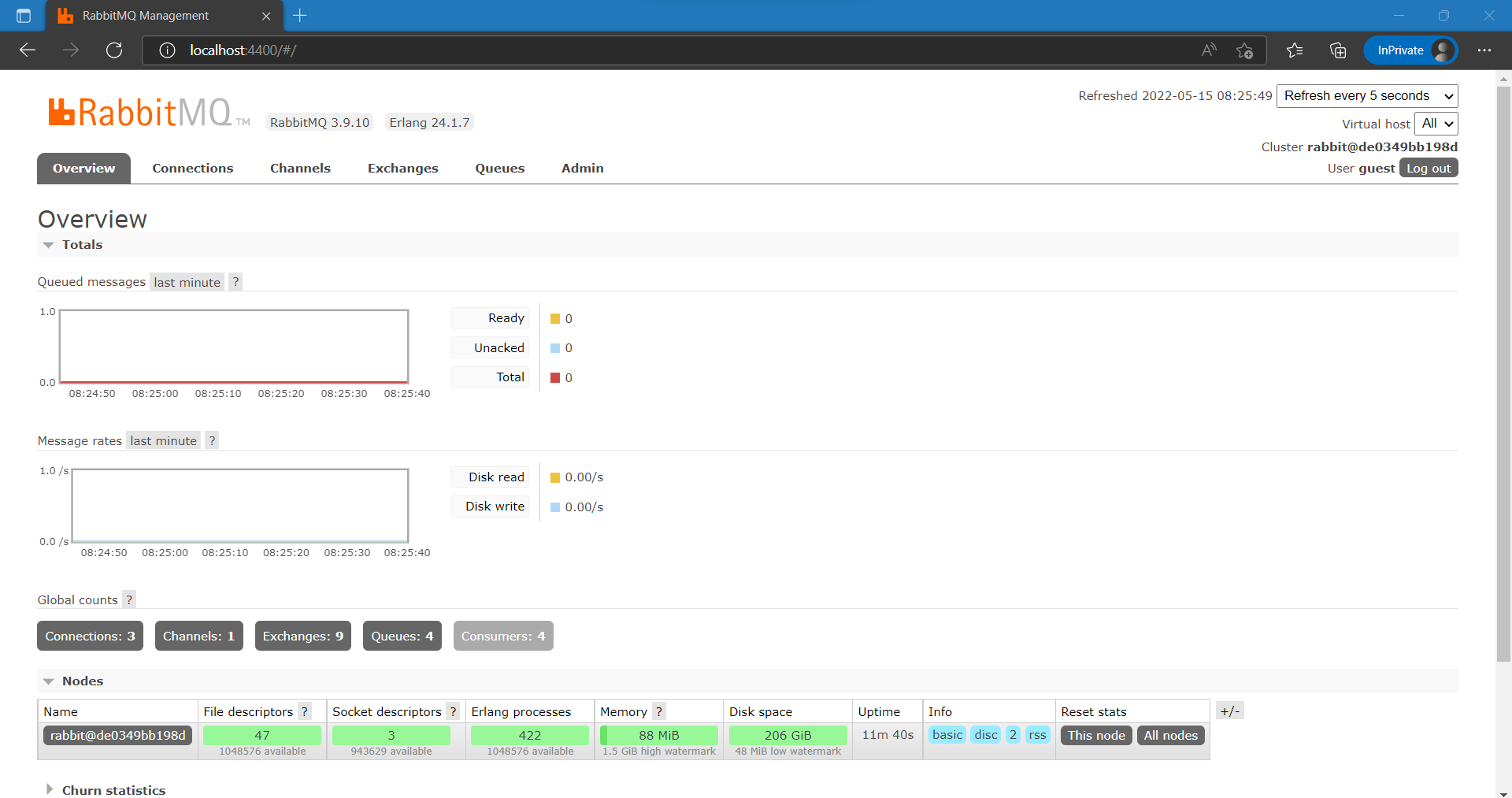
RabbitMq service là nơi mà organism sẽ nhận thông báo nếu các dữ liệu bên identity thay đổi. Điều này sẽ giúp bảo vệ tính toàn vẹn của dữ liệu trong microservice.

Vì chỉ dùng để nhận thông báo nên khi service sẽ không hoạt động nếu RabbitMq không hoạt động.

#### Seq

Seq service là nơi organism sẽ gửi các log của mình đến. Organism vẫn sẽ hoạt động bình thường nếu không có Seq service.

## RabbitMQ



Hình 47 RabbitMQ UI

RabbitMQ service là một message queue. RabbitMq sẽ đóng vai trò là một service trung gian với nhiệm vụ vận chuyển các tin nhắn giữa các service với nhau.

### Giao tiếp giữa các service

Nếu muốn truy cập dữ liệu của một service khác thì chúng ta có nhiều cách khác nhau:

#### Truy cập trực tiếp vào CSDL của service

Ví dụ Organism truy cập trực tiếp vào SQL Server của Identity. Cách làm này là cách tệ nhất và không khuyến khích, việc này không chỉ phá vỡ cấu trúc của Microservice khi định nghĩa rằng mỗi Microservice có CSDL riêng. Việc truy cập CSDL của service khác sẽ làm cho kiến trúc hệ thống từ Microservice mà trở thành Monolithic. Ngoài ra việc này còn làm phá vỡ các ràng buộc bên trong service.

#### Truy cập thông qua web api

Lấy dữ liệu thông qua web api là các làm thông dụng nhất lấy dữ liệu từ các service khác và vẫn bảo toàn tính bao đóng của service khác. Tuy nhiên có một số hạn chế là các lời gọi http là lời gọi tuần tự và việc tạo lời gọi http sẽ gây stress cho service.

#### Truy cập thông qua message queue

Lấy dữ liệu không tuần tự và giảm lời gọi http đến service khác. Sử dụng cách này áp dụng một tính chất giống với Observer Pattern trong desgin pattern GoF được gọi là Hollywood Principle – Don’t Call Us, We’ll Call You!

### Event

Trong các tin nhắn được gửi đến RabbitMq, event miêu tả sự việc đã diễn ra trong service. Ví dụ khi người dùng đổi tên bên identity service thì identity sẽ publish một event là NameChanged để thông báo rằng tên đã thay đổi bên identity. Cách đặt tên event theo thì quá khứ trong tiếng anh theo ví dụ đó chính là chữ Changed. Ví dụ mẫu dữ liệu về NameChanged event:

{

"Id": "891d4dd7-82e7-4818-b78d-62871c0e217d",

"Old Name": "Wang thanh",

"New Name": "Wang Thanh",

"OccurredAt": "2022-05-15T01:28:39.3203084Z",

"Version": 1

}

Thông tin event: người dùng có id là "891d4dd7-82e7-4818-b78d-62871c0e217d" đã đổi tên từ "Wang thanh" thành "Wang Thanh". OccuredAt miêu tả thời gian xảy ra còn Version miêu tả phiên bản của tin nhắn vì tin nhắn sẽ có thể thay đổi trong tương lai.

### Command

Khác với event, command sẽ diễn tả hành động sẽ được thực hiện khi subscriber nhận được command. Ví dụ khi người dùng cập nhật thông tin loài thì organism sẽ publish command là UpdateSpeciesName để tạo một hành động rằng hãy cập nhật loài này. Cách đặt tên command theo thì hiện tại trong tiếng anh theo ví dụ đó chính là chữ Update. Ví dụ mẫu dữ liệu về UpdateSpeciesName command:

{

"Id": "625579c4d7793c3b2a324983",

"Name": "Thằn lằn bóng hoa",

"OccurredAt": "2022-05-15T01:28:39.3203084Z",

"Version": 1

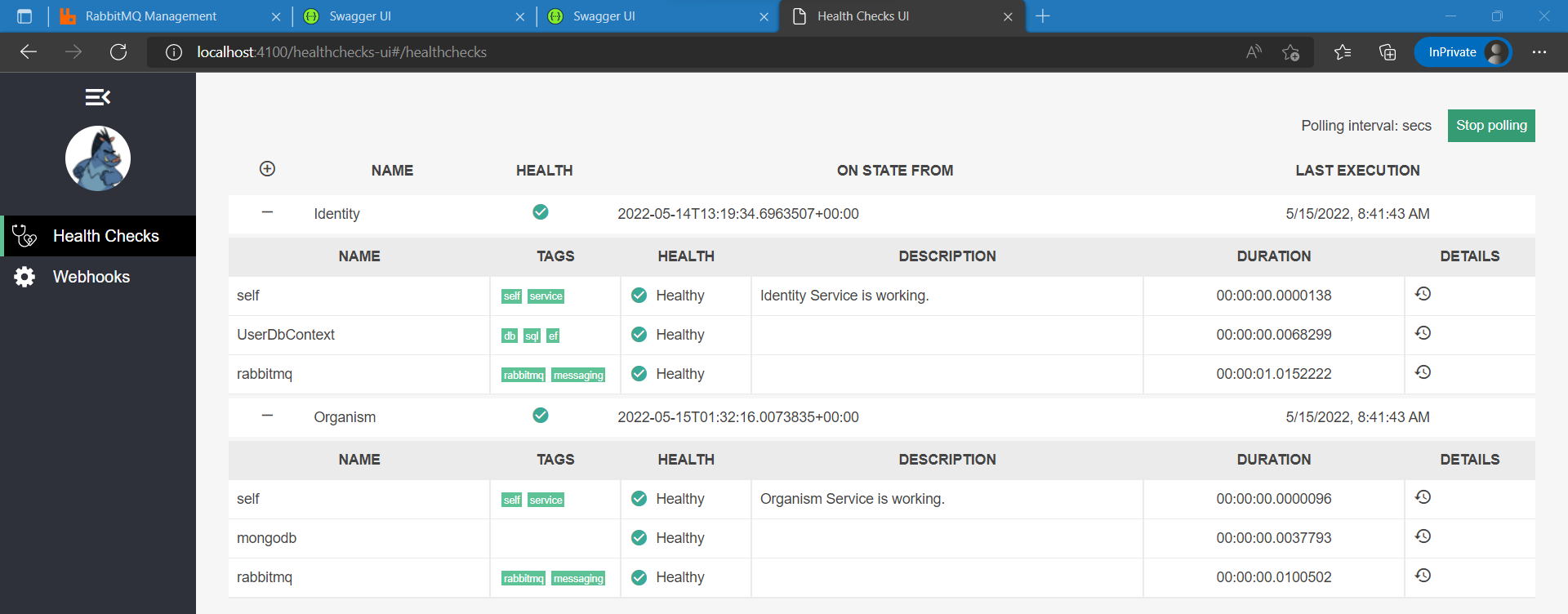
}

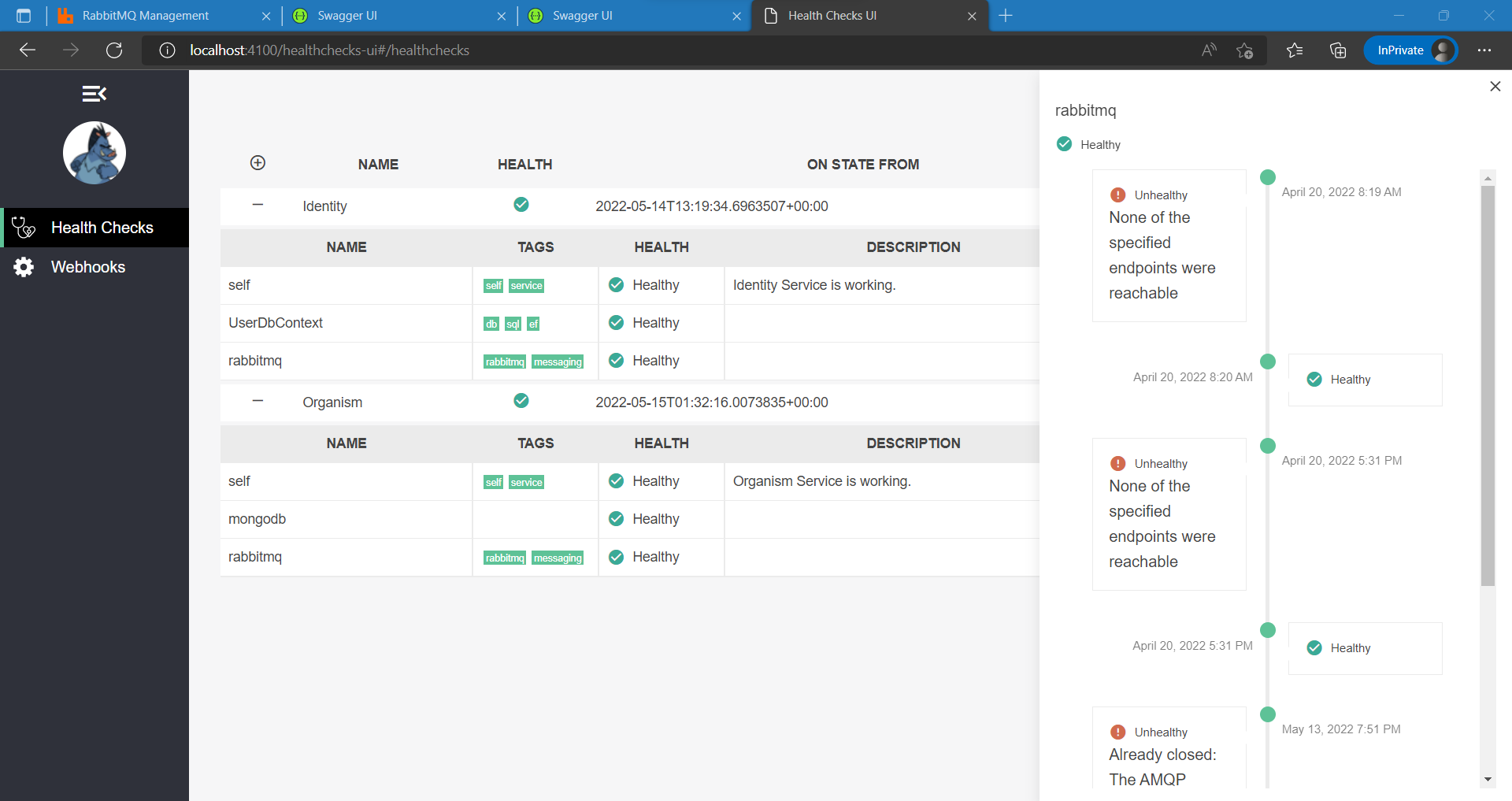
Thông tin command: loài có id là "625579c4d7793c3b2a324983" đổi tên thành "Thằn lằn bóng hoa". OccuredAt miêu tả thời gian xảy ra còn Version miêu tả phiên bản của tin nhắn vì tin nhắn sẽ có thể thay đổi trong tương lai.

### Sự khác giữa event và command

Command và event cả đều là một dạng tin nhắn được gửi đến RabbitMq, tuy nhiên lại có một số điểm không giống nhau để phân biệt. Trong cách đặt tên, event miêu tả sự việc đã diễn ra nên sẽ là thì quá khứ, command mô tả công việc cần thực hiện nên xài thì hiện tại. Subscriber của command và event cũng khác nhau, ta sẽ có nhiều người nhận event vì đó là thông báo nhưng nên chỉ duy nhất 1 người nhận command, nếu nhiều người nhận command và chỉnh sửa CSDL sẽ gây ra đụng độ.

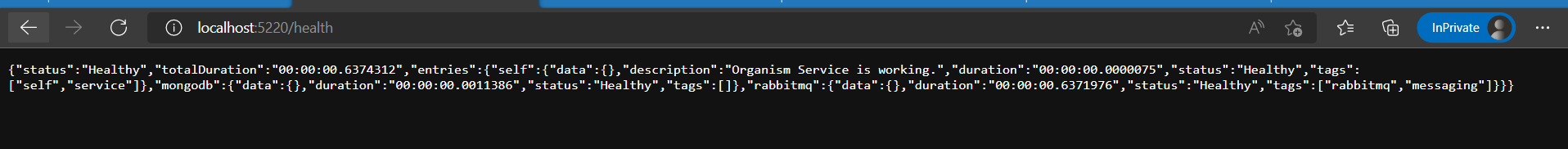
## HealthChecks



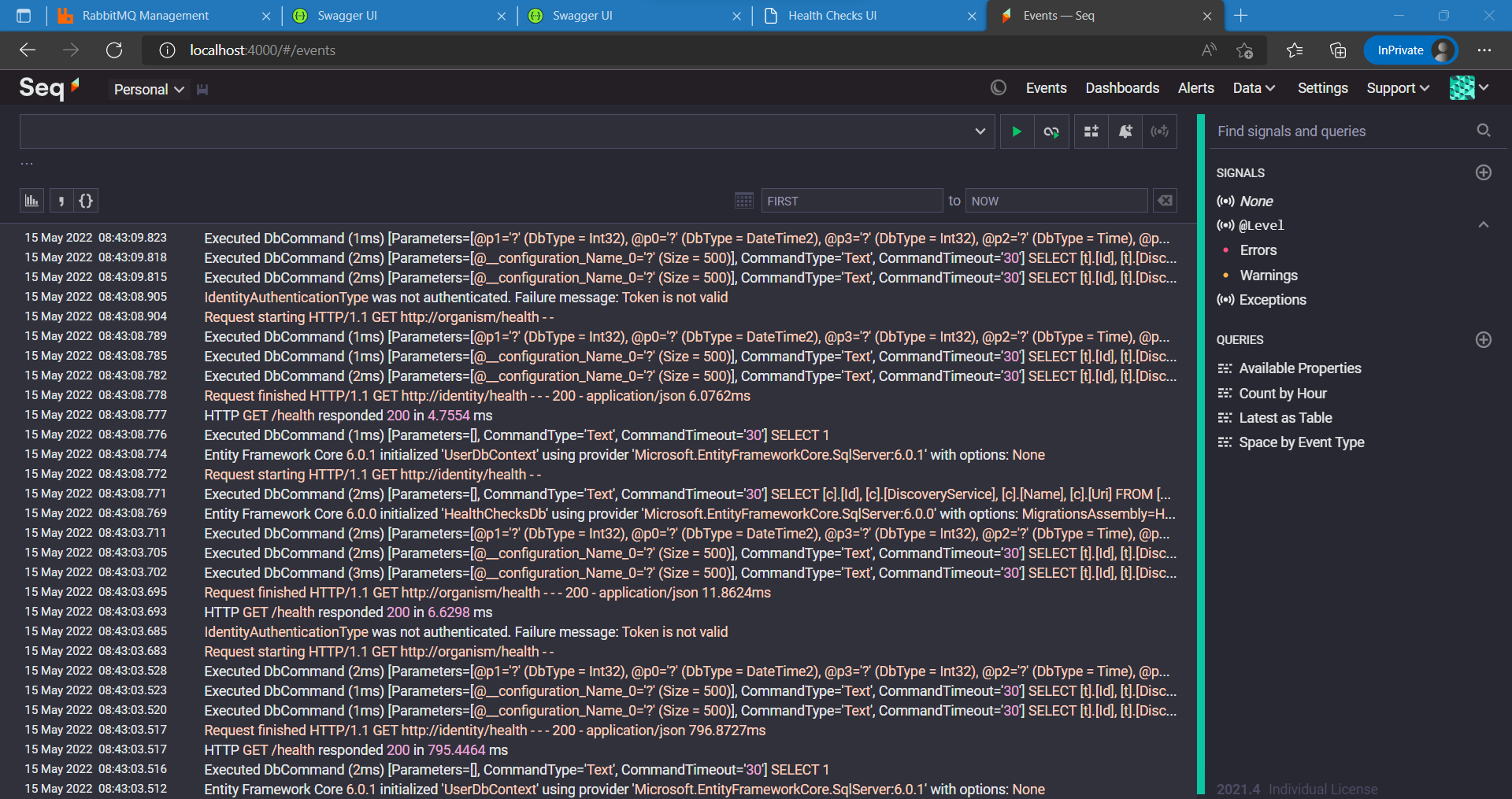
Thông tin chi tiết

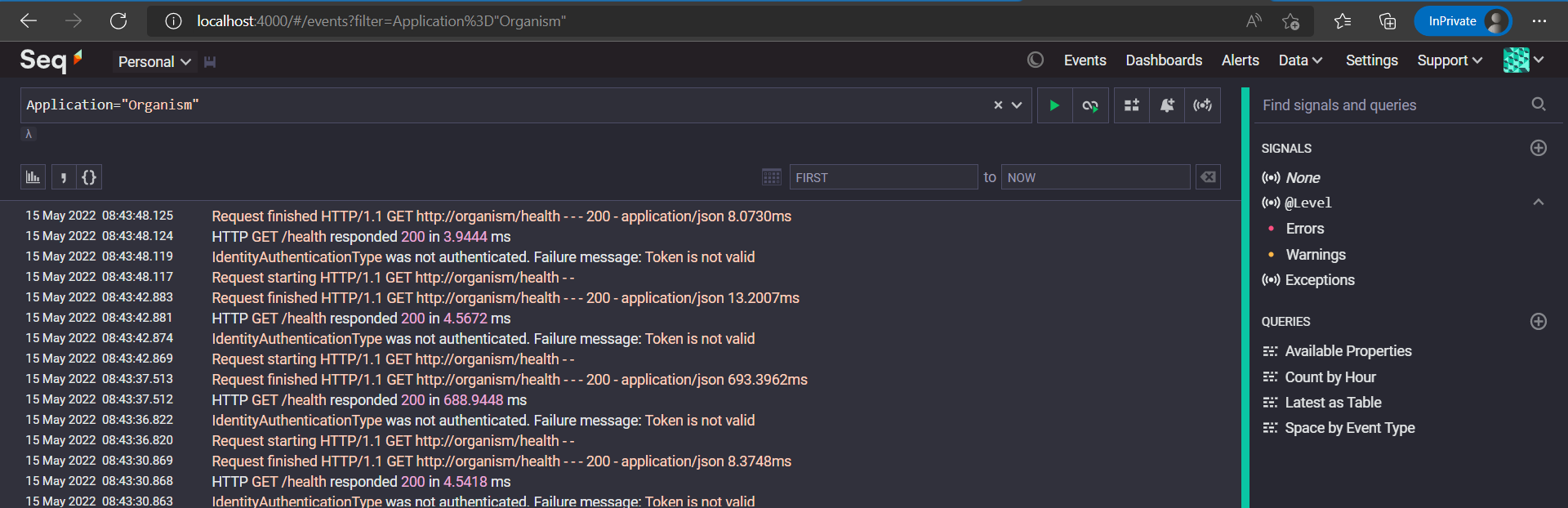
SQL Server

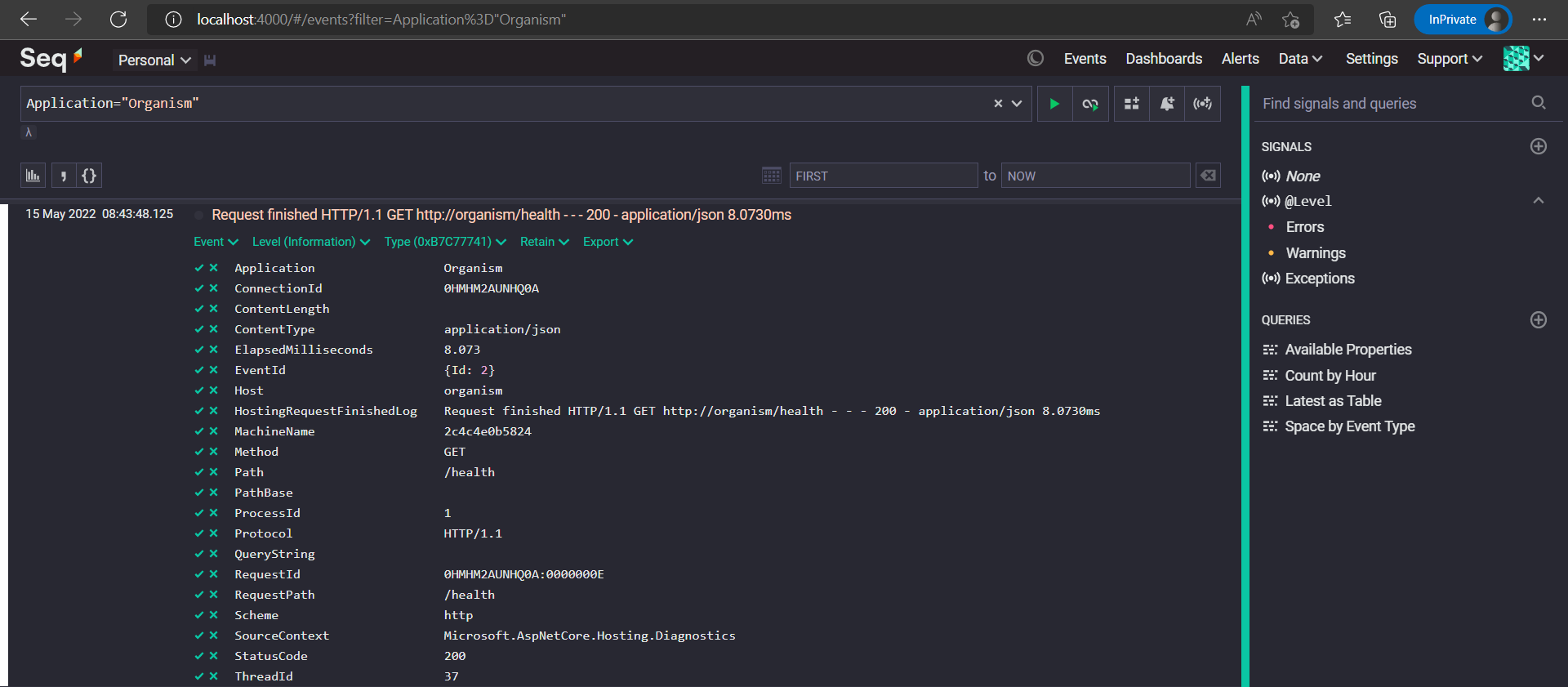
Expose

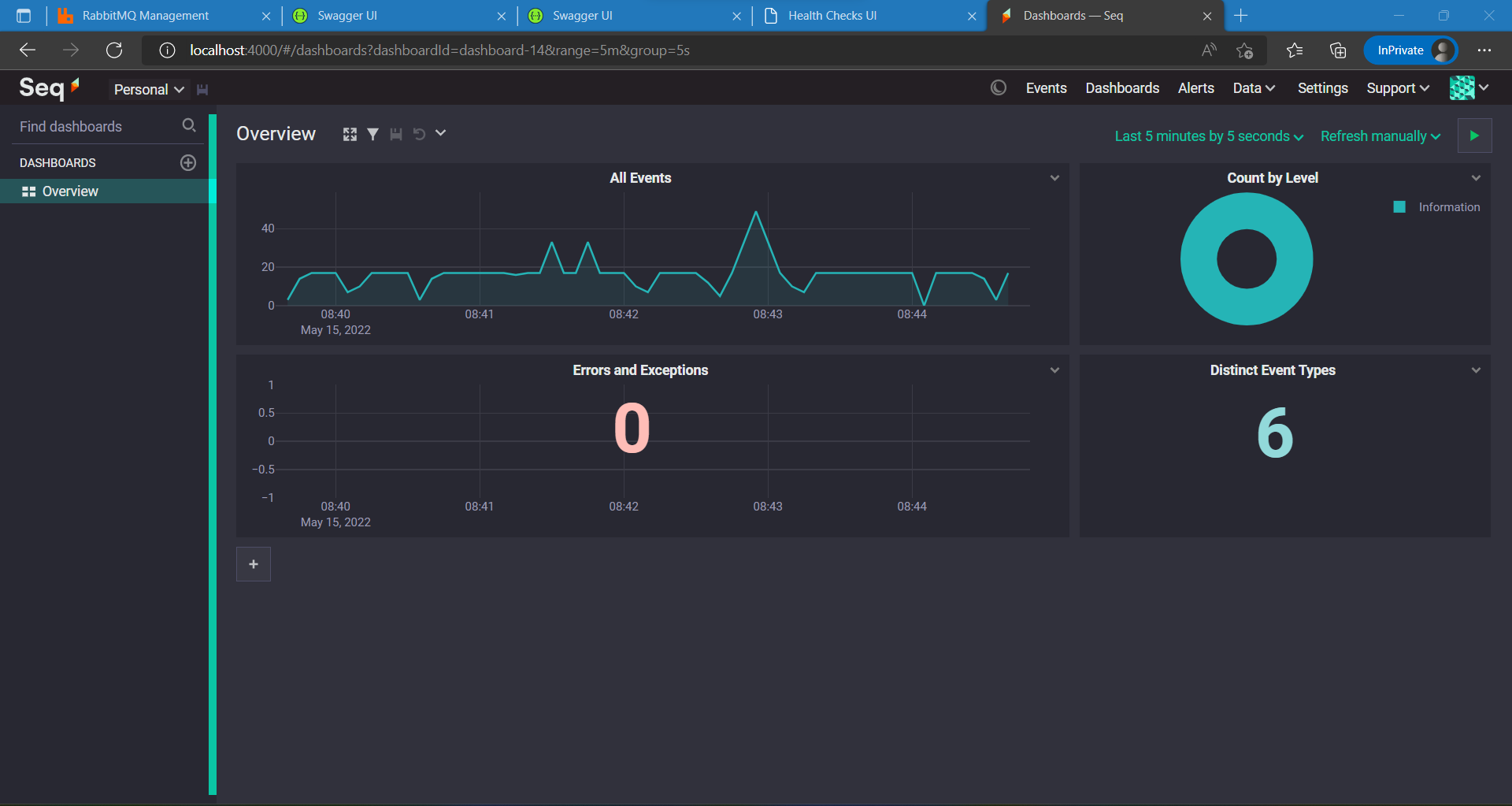


## Seq









việc log có tầm quan trọng nhất định trong vòng đời của phần mềm, chúng ta in ra màn hình khi phần mềm chạy không đúng để kiểm tra lỗi hay việc in ra màn hình các dòng thông tin khi phần mềm đang hoạt động

## Image Search

Miêu tả

CSDl

Gọi RabbitMq

Lưu hình

Build mode

Rest api

## Kiểm thử và đánh giá kết quả

### Giới thiệu

#### Mục tiêu

Tạo các kiểm thử giúp chúng ta theo dõi và đánh giá hành vi của sản phẩm, phần mềm hoàn chỉnh và đã được tích hợp đầy đủ, dựa vào đặc tả và các yêu cầu chức năng đã được xác định trước. Kiểm tra các lỗ hổng hệ thống có thể gây ra trong lúc hoạt động, giảm thiểu các lỗi có thể xảy ra đảm bảo các luồng dữ liệu được xác thực một cách chặt chẽ, nâng cao chất lượng phần mềm.Phát hiện lỗi và kiểm tra hệ thống có hoạt động đúng theo yêu cầu đã nêu ra trong đặc tả hay chưa. Liệt kê kết quả có được sau kiểm thử. Làm tài liệu cho giai đoạn bảo trì.

#### Phạm vi kiểm thử

Quy trình kiểm thử được thực hiện qua những công đoạn như sau:

* Kiểm thử thiết kế: kiểm tra giao diện thiết kế có đúng với đặc tả.
* Kiểm thử chấp nhận: kiểm thử chức năng hệ thống có hoạt động và đáp ứng đặc tả yêu cầu.
* Kiểm thử chức năng: kiểm thử chức năng có xử lý đúng dữ liệu. Kiểm thử cài đặt: tìm và sửa lỗi xảy ra kiểm thử.

### Kế hoạch kiểm thử

#### Các tính năng sẽ được kiểm thử

#### Các tính năng không được kiểm thử

#### Chiến lược kiểm thử

* Chiến lược: thực hiện Black Box Test.
* Sử dụng kiểm thử tích hợp: Functional test.
* Kiểm thử hệ thống (System test): Kiểm tra môi trường sử dụng phần mềm đòi hỏi.

#### Kiểm thử L&P

* Performance profiling là một dạng test hiệu suất trong đó thời gian phản hồi, tỷ lệ giao dịch và các yêu cầu phụ thuộc thời gian khác được đo đạc và đánh giá.
* Mục đích của Performance Profiling là kiểm tra các yêu cầu về hiệu suất có đạt được hay không
* Performance profiling là tiến hành và thực hiện để mô tả sơ lược và điều chỉnh các hành vi hiệu suất của mục tiêu test như một hàm của các điều kiện ví dụ workload hoặc cấu hình phần cứng.

#### Kiểm thử hồi quy

* Kiểm thử hồi quy (Regression Testing) là một hoạt động cần thiết để chỉ ra rằng việc thay đổi code không gây ra những ảnh hưởng bất lợi.
* **Mục đích kiểm thử:**
  + Kiểm thử hồi quy dùng để kiểm tra các phần được sửa chữa trong phần mềm, để đảm bảo rằng những sự thay đổi đó không gây ra lỗi trong những phần khác.
* **Cách thực hiện:**
  + Tái sử dụng các TC từ những phần kiểm thử trước để kiểm thử các module đã được sửa chữa
  + Sử dụng công cụ Rational Robot: Tạo một số script kiểm thử về chức năng. Định nghĩa lịch thực hiện tự động cho chúng
  + 80% các TC được chọn ngẫu nhiên
  + Xây dựng một chương trình phân tích cơ sở hạ tầng. Chúng ta dựng một cơ sở hạ tầng có thể mở rộng được để thực hiện và đánh giá chương trình phân tích. Dựa vào kết quả phân tích chúng ta xác định phạm vi cần test hồi quy.
* **Điều kiện hoàn thành:**
  + Toàn bộ các TC được thực hiện và đạt yêu cầu
  + Toàn bộ các TC được chọn được thực hiện và đạt yêu cầu

#### Cách tiếp cận

Với mỗi tính năng chính hay các nhóm tính năng sẽ được kiểm thử theo thứ tự từ trên xuống dưới và từ trái qua phải để đảm bảo rằng sẽ kiểm thử không bỏ sót chức năng cần kiểm thử.

#### Tiêu chí kiểm thử thành công / thất bại

* Tiêu chí kiểm thử thành công: kết quả cuối cùng của chuỗi các thao tác đúng như mong đợi đặt ra ban đầu.
* Tiêu chí kiểm thử thất bại: kết quả kiểm thử không như mong đợi, xuất hiện lỗi sai lệch so với kỳ vọng ban đầu.

#### Tiêu chí đình chỉ và yêu cầu bắt đầu lại

* Tiêu chí đình chỉ là dừng việc thực hiện công việc khi một chức năng thông báo lỗi.
* Yêu cầu bắt đầu lại khi chức năng đình chỉ đã được sửa lỗi.

### Quản lý kiểm thử

#### Các hoạt động/công việc được lập kế hoạch, sự tiến hành kiểm thử

* Lập kế hoạch kiểm thử
* Tạo các TestCase
* Tiến hành kiểm thử
* Báo cáo kết quả

#### Môi trường

* Phần cứng máy tính: CPU Core i7, RAM 12GB, HDD 500GB.
* Hệ điều hành: Windows 11.
* Phần mềm: Google Chrome, Microsoft Edge.
* Thiết bị có kết nối internet.

#### Trách nhiệm quyền hạn

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Thành viên |  |  | | Vai trò |  |  |
| Quản  lý | Thiết kế | Chuẩn bị | Thực hiện | Chứng kiến | Kiểm tra |
| Vương Cẩm Thanh | x | x | x | x | x | x |

Bảng 17 Trách nhiệm quyền hạn

#### Giao tiếp giữa các nhóm liên quan

Đề tài được thực hiện bởi cá nhân nên không có sự giao tiếp giữa các nhóm liên quan.

#### Tài nguyên và sự cấp phát chúng

* Tài nguyên sử dụng kiểm thử: PC.
* Các công cụ hỗ trợ trong quá trình kiểm thử: Google Chrome, Microsoft Edge.

#### Huấn luyện

Tìm hiểu tài liệu và học hỏi kinh nghiệm của những người từng kiểm thử

#### Kế hoạch dự đoán và chi phí

|  |  |  |
| --- | --- | --- |
| Công việc | Thời gian (ngày) | Công cụ |
| Lập kế hoạch kiểm thử | 1 | Microsoft Word |
| Thiết kế test case | 1 | Microsoft Word |
| Tiến hành kiểm thử | 2 | Google Chrome, Microsoft Edge |
| Đánh giá | 1 | Microsoft Word |

Bảng 18 : Kế hoạch dự đoán và chi phí

#### Các rủi ro

|  |  |
| --- | --- |
|  |  |
| Các rủi ro có thể xảy ra | Kế hoạch khắc phục |
| Kiểm thử không đúng tiến độ | Cần bằng thời gian giữa các lần kiểm thử, tăng cường hiệu suất kiểm thử. |
| Thiếu nguồn nhân sự | Tăng tiến độ làm việc, thay đổi thời gian kiểm thử phù hợp, tập trung kiểm thử các chức năng chính. |
| Kiểm thử không đạt hiệu quả | Tham khảo nhiều nguồn tài liệu kiểm thử. |

Bảng 19 Các rủi ro

#### Kịch bản kiểm thử

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Mã | Tài liệu đặc tả | Mô tả kịch bản kiểm thử | Mức  quan trọng | Số trường hợp kiểm thử |
| TS\_GH\_01 | Mục  1.1.2.3.1 | Kiểm tra người dùng có thể đăng ký tài khoản | P1 | 5 |

Bảng 20 Kịch bản kiểm thử

#### Các trường hợp kiểm thử