

TRƯỜNG ĐẠI HỌC CẦN THƠ
KHOA CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



LUẬN VĂN TỐT NGHIỆP
NGÀNH KỸ THUẬT PHẦN MỀM

Đề tài

PHÁT TRIỂN HỆ THỐNG
LƯU TRỮ VÀ CHIA SẺ DỮ LIỆU GPS DÙNG
KIẾN TRÚC MICROSERVICES

Sinh viên thực hiện:

Nguyễn Tiến Sĩ

Mã số sinh viên: B1704767

Khóa: 43

Cần Thơ, 12/2021

TRƯỜNG ĐẠI HỌC CẦN THƠ
KHOA CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
BỘ MÔN CÔNG NGHỆ PHẦN MỀM



LUẬN VĂN TỐT NGHIỆP
NGÀNH KỸ THUẬT PHẦN MỀM

Đề tài

**PHÁT TRIỂN HỆ THỐNG LƯU TRỮ VÀ CHIA SẺ
DỮ LIỆU GPS DÙNG KIẾN TRÚC
MICROSERVICES**

Cán bộ hướng dẫn

TS. Nguyễn Công Danh

Sinh viên thực hiện

Nguyễn Tiến Sĩ

MSSV: B1704767

Khóa: 43

Cần Thơ, 12/2021

TRƯỜNG ĐẠI HỌC CẦN THƠ CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM

KHOA CNTT&TT

Độc lập – Tự do – Hạnh phúc

XÁC NHẬN CHỈNH SỬA LUẬN VĂN THEO YÊU CẦU CỦA HỘI ĐỒNG

Tên luận văn: Phát Triển Hệ Thống Lưu Trữ Và Chia Sẻ Dữ Liệu GPS Dùng Kiến Trúc Microservices

Họ và tên sinh viên: Nguyễn Tiến Sĩ

MSSV: B1704767

Mã Lớp: DI1796A1

Đã báo cáo tại hội đồng ngành: Kỹ thuật phần mềm

Ngày báo cáo: 21/12/2021

Luận văn đã được chỉnh sửa theo góp ý của Hội đồng

Cần Thơ, ngày 21 tháng 12 năm 2021

Giáo viên hướng dẫn

Nguyễn Công Danh

LỜI CẢM ƠN

Để hoàn thành tốt luận văn tốt nghiệp Đại học, em xin gửi lời cảm ơn chân thành đến Khoa Công Nghệ Thông Tin và Truyền Thông trường Đại học Cần Thơ, đã tạo điều kiện thuận lợi cho em để có thể học tập và làm đề tài luận văn đúng theo yêu cầu nguyện vọng của em trong học kì này. Đồng thời em xin cảm ơn các thầy, cô và nhà trường đã đồng hành, giảng dạy em suốt hơn 4 năm vừa qua.

Em xin gửi lời cảm ơn sâu sắc đến thầy Nguyễn Công Danh – cán bộ trực tiếp hướng dẫn luận văn tốt nghiệp. Thầy đã nhiệt tình hướng dẫn, giành nhiều thời gian trao đổi và cung cấp rất nhiều thông tin bổ ích đến cho em để em có thể trang bị đầy đủ kiến thức làm luận văn. Đó là nguồn động lực to lớn giúp em hoàn thành tốt đề tài này.

Bên cạnh đó là sự hỗ trợ từ phía doanh nghiệp Công ty TNHH Giải pháp phần mềm Tường Minh (TMA Solutions), em xin cảm ơn anh Lê Thanh Phong, anh Nguyễn Đăng Nhật Tân đã tạo điều kiện và hướng dẫn tận tình, đã góp ý kiến với các thông tin quý báu giúp em học hỏi thêm nhiều kỹ năng, kiến thức về microservices. Em chân thành cảm ơn cô Ths. Trương Thị Thanh Tuyền, thầy Cù Vĩnh Lộc đã giành thời gian tham gia Hội đồng bảo vệ và cung cấp các góp ý nhận xét bổ ích

Em cũng xin cảm ơn đến gia đình, tất cả bạn bè cùng anh chị đã luôn ủng hộ và giúp đỡ em có thêm động lực để hoàn thành tốt đồ án tốt nghiệp này. Tuy nhiên do hạn chế về kiến thức, kinh nghiệm thời gian thực hiện và cơ sở vật chất nên chắc hẳn không thể tránh khỏi những sai sót. Em rất mong nhận được sự nhận xét và góp ý của quý thầy cô, bạn bè và quý độc giả. Sự đóng góp của thầy cô, bạn bè và quý độc giả là nguồn kiến thức to lớn đối với em.

Một lần nữa xin gửi đến gia đình, thầy cô, bạn bè và các anh chị đồng nghiệp lời cảm ơn chân thành và lời chúc sức khỏe.

Trân trọng cảm ơn!

Cần Thơ, ngày 21 tháng 12 năm 2021

Nguyễn Tiến Sĩ

CAM KẾT KẾT QUẢ

Em xin cam kết luận văn tốt nghiệp với tên đề tài “Phát triển hệ thống lưu trữ và chia sẻ dữ liệu GPS dùng kiến trúc Microservices” là công trình nghiên cứu của cá nhân em dưới sự hướng dẫn của thầy TS. Nguyễn Công Danh và người đồng hướng dẫn anh Nguyễn Đăng Nhật Tân – cán bộ tại TMA Solutions, trung thực và không sao chép của tác giả khác. Trong toàn bộ nội dung nghiên cứu của luận văn, các vấn đề được trình bày đều là những tìm hiểu và nghiên cứu của chính cá nhân em hoặc là được trích dẫn từ các nguồn tài liệu có ghi tham khảo rõ ràng, hợp pháp. Nếu có vấn đề gì em xin hoàn toàn chịu trách nhiệm

Em xin cam đoan mọi sự giúp đỡ cho luận văn này đã được cảm ơn và các tài liệu tham khảo trong luận văn đã được ghi rõ nguồn gốc.

Cần Thơ, ngày 21 tháng 12 năm 2021

Sinh viên thực hiện

Nguyễn Tiến Sĩ

NHẬN XÉT CỦA CÁN BỘ HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Cần Thơ, ngày ... tháng ... năm
Cán bộ hướng dẫn

NHẬN XÉT CỦA CÁN BỘ PHẢN BIỆN

[illegible]

Cần Thơ, ngày tháng năm 2021

Cán bộ phản biện

MỤC LỤC

LỜI CẢM ƠN.....	ii
CAM KẾT KẾT QUẢ.....	iii
MỤC LỤC	vi
DANH MỤC HÌNH ẢNH.....	viii
DANH SÁCH BẢNG BIỂU	x
KÍ HIỆU VÀ THUẬT NGỮ VIẾT TẮT	xi
TÓM TẮT.....	xii
ABSTRACTS	xiii
PHẦN I. MỞ ĐẦU	1
1. Tổng quan	1
2. Lịch sử giải quyết vấn đề.....	1
3. Mục tiêu đề tài	3
4. Đối tượng và phương pháp nghiên cứu	3
5. Nội dung nghiên cứu	4
6. Những đóng góp chính của đề tài.....	4
7. Bố cục luận văn	4
PHẦN II. NỘI DUNG	5
Chương 1: Mô tả hệ thống.....	5
1.1. Mô tả chi tiết bài toán.....	5
1.2. Đặt vấn đề.....	6
1.3. Các yêu cầu giao tiếp bên ngoài	6
1.4. Các yêu cầu phi chức năng.....	6
1.5. Các yêu cầu khác	7
1.6. Các chức năng chính của hệ thống	7
1.7. Sơ đồ usecase.....	8
Chương 2: Cơ sở lý thuyết.....	9
2.1. Kiến trúc Microservices[16].....	9
2.2. Spring Framework	17
2.3. RabbitMQ	20
2.4. Docker	21
2.5. InfluxDB	22

Chương 3:	Đề xuất giải pháp áp dụng Microservices vào ứng dụng GPS online	23
3.1.	Giới thiệu về GPX File.....	23
3.2.	Sơ đồ kiến trúc của hệ thống	24
3.3.	Sơ đồ tuần tự của hệ thống	25
3.4.	Cài đặt.....	25
3.5.	Kết Luận	43
Chương 4:	Kiểm thử và đánh giá	44
4.1.	Giới thiệu	44
4.2.	Chi tiết kế hoạch kiểm thử.....	44
4.3.	Các chức năng được kiểm thử	44
4.4.	Quản lý kiểm thử	44
4.5.	Kịch bản kiểm thử	45
4.6.	Các trường hợp kiểm thử.....	46
4.7.	Đánh giá.....	46
Chương 5:	Các vấn đề cần quan tâm và hiệu quả của Microservices	47
5. 1.	Các vấn đề cần quan tâm	47
5.2.	Hiệu quả của Microservices	47
5.3.	Đánh giá sự đạt được của hệ thống lưu trữ và chia sẻ dữ liệu GPS	48
5.4.	Kết luận.....	48
PHẦN III.	KẾT LUẬN	49
1.	Kết quả đạt được.....	49
1.1.	Về kiến thức.....	49
1.2.	Về kỹ năng.....	49
1.3.	Về thái độ.....	49
2.	Thuận lợi và khó khăn	49
2.1.	Thuận lợi.....	49
2.2.	Khó khăn.....	50
3.	Hạn chế của phần mềm.....	50
4.	Hướng phát triển.....	50
TÀI LIỆU THAM KHẢO		51

DANH MỤC HÌNH ẢNH

Hình 0.1: Trang web của ứng dụng Netflix.....	2
Hình 0.2: Trang web của Amazon.....	2
Hình 0.3: Website Trackerprofiler.com.....	3
Hình 1.1: Sơ đồ các trường hợp sử dụng.....	8
Hình 2.1: Các thành phần của kiến trúc Monolithic.....	9
Hình 2.2: Ví dụ kiến trúc Microservices	11
Hình 2.3: Mô hình kiến trúc Microservices.....	12
Hình 2.4: Cổng API trong ứng dụng quản lý sản phẩm	14
Hình 2.5: Cơ chế phát hiện phía máy trạm (client-side discovery).....	15
Hình 2.6: Cơ chế phát hiện dịch vụ phía máy khách (server-side discovery).....	15
Hình 2.7: Cơ chế tự đăng ký dịch vụ (self-registration).....	16
Hình 2.8: Cơ chế đăng ký dịch vụ qua bên thứ ba	17
Hình 2.9: Các module của Spring Framework	18
Hình 2.10: Module chính của Spring Framework	19
Hình 2.11: Mô hình hoạt động của Docker	22
Hình 3.1: Ví dụ về File GPX	23
Hình 3.2: Sơ đồ kiến trúc của hệ thống	24
Hình 3.3: Sơ đồ tuần tự của hệ thống	25
Hình 3.4: Class QueueName để cấu hình tên cho RabbitMQ	26
Hình 3.5: Class RabbitMQProvision để tạo queue trên RabbitMQ	27
Hình 3.6: File Docker-compose để triển khai RabbitMQ	27
Hình 3.7: Giao diện đăng nhập của RabbitMQ trên web	28
Hình 3.8: Giao diện chính của RabbitMQ.....	28
Hình 3.9: Giao diện các QueueName đã được cấu hình	28
Hình 3.10: Cấu hình để kết nối SpringBoot với RabbitMQ.....	29
Hình 3.11: Class GPXGateController với chức năng Upload File GPX	29
Hình 3.12: File GPX.....	30
Hình 3.13: Giao diện chính của PostMan với chức năng upload file GPX.....	30
Hình 3.14: Class GpxFileService với chức năng gửi dữ liệu đến RabbitMQ.....	31
Hình 3.15: Class RabbitMQConfiguration cấu hình để nhận dữ liệu từ RabbitMQ.....	32
Hình 3.16: Class TrackerMessageConsumer với chức năng nhận dữ liệu từ RabbitMQ	33
Hình 3.17: Class GpxMessageHandler với chức năng chuyển dữ liệu thành dữ liệu TimeSeries	34
Hình 3.18: Class TimeSeriesArchivingListener với chức năng gửi dữ liệu TimeSeries đến RabbitMQ	35
Hình 3.19: Giao diện Queue sao khi được kết nối với các service	35
Hình 3.20: Cấu hình để SpringBoot kết nối với RabbitMQ.....	35
Hình 3.21: Class RabbitMQConfiguration cấu hình để nhận dữ liệu TimeSeries từ RabbitMQ	36
Hình 3.22: Hình ảnh 2 docker container đang được triển khai	36
Hình 3.23: Giao diện đăng nhập của InfluxDB	37

Hình 3.24: Giao diện chính của InfluxDB.....	38
Hình 3.25: Class InfluxFactory cấu hình để SpringBoot nhận các thông số từ InfluxDB	39
Hình 3.26: Cấu hình để kết nối với InfluxDB	39
Hình 3.27: Cấu hình để kết nối với RabbitMQ	39
Hình 3.28: Class TimeSeriesConsumer với chức năng nhận dữ liệu TimeSeries từ RabbitMQ	40
Hình 3.29: Class TimeSeries với các thuộc tính được gửi vào InfluxDB	40
Hình 3.30: Class TimeSeriesRepository với chức năng insert dữ liệu vào InfluxDB ..	41
Hình 3.31: Cấu hình để nhận các thông số từ InfluxDB	41
Hình 3.32: Cấu hình để kết nối với InfluxDB	42
Hình 3.33: Class TimeSeriesRepository với chức năng lấy dữ liệu từ InfluxDB	42
Hình 3.34: Class TimeSeriesController với chức năng trả dữ liệu đã lấy từ InfluxDB	42
Hình 3.35: Giao diện PostMan với chức năng trả dữ liệu về người dùng.....	43
Hình 4.1: Kịch bản kiểm thử	45
Hình 4.2: Các trường hợp kiểm thử.....	46

DANH SÁCH BẢNG BIỂU

Bảng 1.1: Chức năng upload file gpx	7
Bảng 1.2: Chức năng xem thông tin file gpx đã upload	8
Bảng 4.1: Thời gian kiểm thử	45
Bảng 4.2: Các rủi ro	45
Bảng 5.1: Đánh giá sự đạt được của hệ thống	48

KÍ HIỆU VÀ THUẬT NGỮ VIẾT TẮT

STT	Thuật ngữ/Từ viết tắt	Mô tả
1	HTML	HyperText Markup Language
2	API	Application Program Interface
3	JSON	Javascript Object Notation
4	ORM	Object Relation Mapping
5	SOAP	Simple Object Access Protocol
6	HTTP	HyperText Transfer Protocol
7	URL	Uniform Resource Locator

TÓM TẮT

Hiện nay nước ta đang trong quá trình hội nhập, các công ty, doanh nghiệp cũng bước vào giai đoạn cạnh tranh khốc liệt hơn. Kiến trúc Microservices là một trong những xu hướng kiến trúc phần mềm được thảo luận nhiều nhất tại thời điểm hiện tại và nó đã thay đổi mãi mãi cách thức xây dựng các ứng dụng của doanh nghiệp. Thay vì cách tiếp cận nguyên khối (monolithic) chậm chạp, phức tạp trong quá khứ, các nhà phát triển và công ty ở khắp mọi nơi đang chuyển sang kiến trúc Microservices để đơn giản hóa và mở rộng cấu trúc của họ.

Luận văn “Phát triển hệ thống lưu trữ và chia sẻ dữ liệu GPS sử dụng kiến trúc Microservices” sẽ tìm hiểu và giới thiệu sơ lược về kiến trúc Microservices, những đặc tính vượt trội, ưu việc của nó trong việc xây dựng các ứng dụng đòi hỏi sự module hóa và có khả năng sử dụng lại cao. Sau khi tìm hiểu em sẽ vận dụng kết quả tìm hiểu được vào việc phát triển hệ thống lưu trữ và chia sẻ dữ liệu GPS như một cầu nối giúp giao tiếp giữa các dữ liệu GPS với các thiết bị, phần mềm. Chỉ cần một file GPX, bạn có thể chia sẻ tất cả những thông tin trên file thông qua hệ thống và có thể lưu trữ tất cả những thông tin đó vào cơ sở dữ liệu.

ABSTRACTS

Currently, our country is in the process of importing, companies and enterprises are also entering the stage of fierce competition. Microservices architecture is an in the under the most discussed architectural software trend at the moment and it has changed forever the way enterprise applications are built. Instead of a slow, complex in-process (monolithic) block approach, developers and companies everywhere turn to Microservices architecture to simplify and extend their structure.

Thesis "Development of system storage and sharing of GPS data using Microservices architecture" will learn and briefly introduce Microservices architecture, its outstanding features, priorities in Build applications that use modular and highly reusable questioning. After learning, I will apply the findings to develop storage systems and share GPS data as a bridge to help communicate between GPS data with devices and software. Just one GPX file, you can share all the information on the file through the system and can store all that information into the database.

PHẦN I. MỞ ĐẦU

1. Tổng quan

Dịch vụ (service) ra đời và 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 nên phức tạp, khó phát triển kiểm thử và bảo trì. Chính những điều này làm cho việc xây dựng và phát triển của 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.

Kiến trúc Microservices được coi là lời giải ưu việt cho bài toán xây dựng và phát triển hệ thống dựa trên dịch vụ như hiện nay. Nó đã và đang được nghiên cứu và ứng dụng rộng rãi bởi các công ty lớn như: Netflix, Ebay, Paypal, Twitter, Amazon, ... 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.

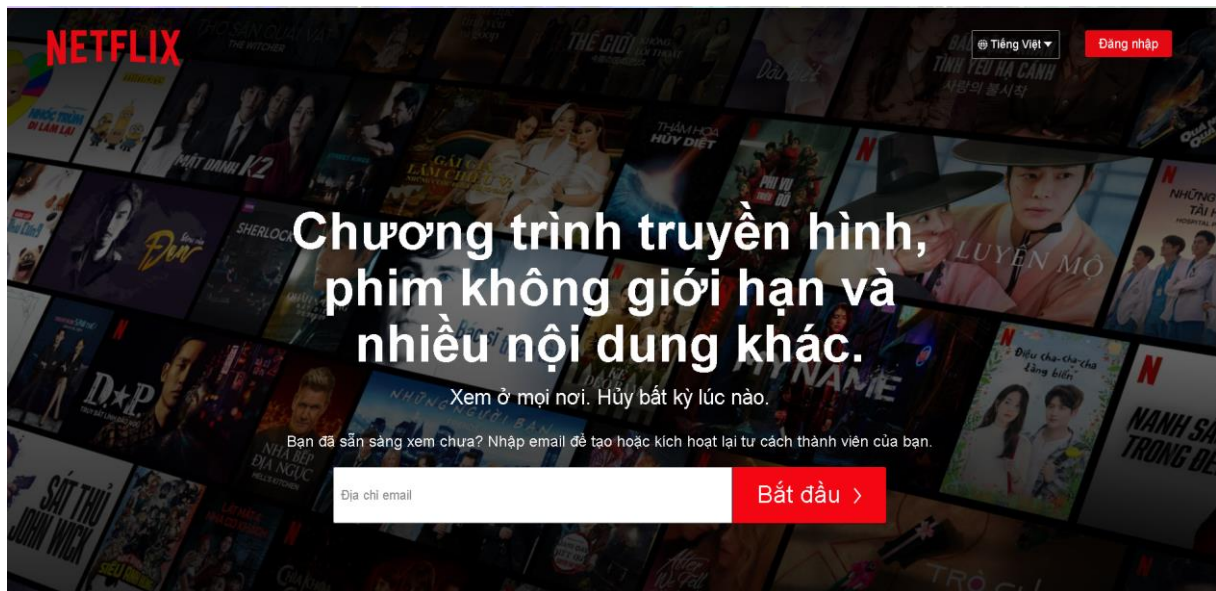
2. Lịch sử giải quyết vấn đề

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 ngày càng tăng đến người dùng. Do đó, sự bất lợi và nhược điểm của Monolithic bắt đầu xuất hiện.

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

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

Netflix là 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 chương trình truyền hình, phim, phim tài liệu đoạt giải thưởng và nhiều nội dung khác trên hàng nghìn thiết bị có kết nối internet.

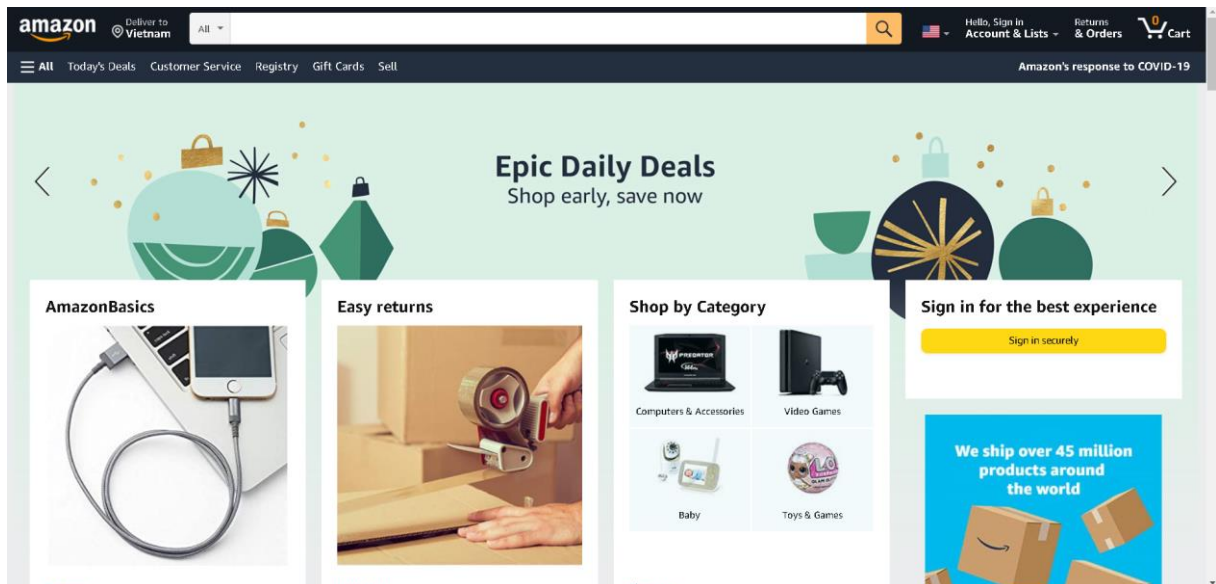


Hình 0.1: Trang web của ứng dụng Netflix

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

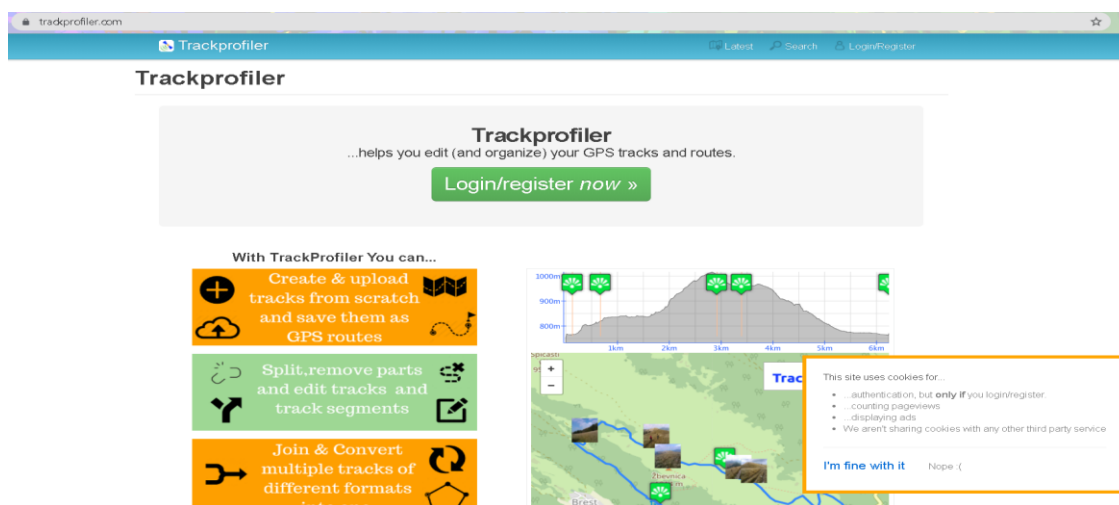
Amazon là công ty hàng đầu thế giới về lĩnh vực bán lẻ trực tuyến. Hơn thế, Amazon đang vươn lên mạnh mẽ với vai trò là nhà cung cấp dịch vụ điện toán đám mây.

Xuất phát điểm của Amazon là trang bán sách. Về sau Amazon mở rộng với hàng loạt hàng hóa tiêu dùng và phương tiện truyền thông kỹ thuật số cũng như các thiết bị điện tử của mình.



Hình 0.2: Trang web của Amazon

<https://trackprofiler.com/>



Hình 0.3: Website Trackerprofiler.com

Với Trackprofile chúng có thể:

- Tạo và upload tracks và lưu dưới dạng GPS routes
- Xóa và chỉnh sửa các tracks và track segments
- Có thể chuyển đổi tracks thành nhiều định dạng khác nhau
- Thêm và chỉnh sửa các điểm tham chiếu, mã hóa các hình ảnh địa lý sang tracks
- Cung cấp những hình ảnh và các biểu đồ trực quan

3. Mục tiêu đề tài

- Tìm hiểu và nghiên cứu về mô hình Microservices với những đặc tính vượt trội và ưu việc trong việc tạo ra hệ thống đòi hỏi sự module hóa cao và có khả năng sử dụng lại và mở rộng trong tương lai.
- 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 các khái niệm và nghiệp vụ liên quan đến việc lưu trữ và chia sẻ dữ liệu GPS.
- Tìm hiểu phân tích thiết kế giải pháp theo hướng đối tượng.
- Phát triển hệ thống lưu trữ và chia sẻ dữ liệu đường đi GPS thỏa mãn các tiêu chí sau:
 - + Người dùng có thể upload một file gpx (GPS exchange format).
 - + Lưu trữ các thông tin bắt buộc như : metadata, waypoint, track.
 - + Cho phép người dùng có thể xem danh sách các đường đi mới nhất.

4. Đối tượng và phương pháp nghiên cứu

Luận văn sẽ tập trung trình bày kết quả nghiên cứu của em về các nội dung sau: Mô hình Microservices và các công nghệ đi kèm như: Spring Framework, Docker, RabbitMQ, InfluxDB. 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

thông lưu trữ và chia sẻ dữ liệu GPS sử dụng kiến trúc Microservices nhằm mục đích minh họa cho phần lý thuyết đã trình bày.

5. 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: Spring framework, Docker, RabbitMQ, InfluxDB, ... 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 lưu trữ và chia sẻ dữ liệu GPS 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.

6. Những đóng góp chính của đề tài

- Kết quả nghiên cứu có thể làm tài liệu tham khảo
- Phần nghiên cứu lý thiết sẽ cung cấp một cách nhìn tổng quát về phát triển hệ thống sử dụng kiến trúc Microservices và các công nghệ đi kèm.
- Hệ thống cũng có thể phát triển để tích hợp vào nhiều thiết bị và phần cứng để theo dõi vị trí, chuyển động.

7. Bố cục luận văn

Tài liệu được chia làm 2 phần chính:

- Phần I. Mở đầu gồm những nội dung chính: Đặt vấn đề, tóm tắt lịch sử giải quyết vấn đề, mục tiêu của đề tài, đối tượng, phạm vi và nội dung nghiên cứu, những đóng góp chính của đề tài, bố cục quyền luận văn.
- Phần II. Nội dung: bao gồm 5 nội dung chính
 - + Chương 1: Giới thiệu về kiến trúc Microservices, nền tảng, định nghĩa, đặc điểm, các lợi ích và mẫu thiết kế của kiến trúc Microservices.
 - + Chương 2: Trình bày các cơ sở lý thiết cũng như các công nghệ có liên quan.
 - + Chương 3: Trình bày các bước thiết kế giải pháp dựa trên kiến trúc Microservices như xác định, triển khai. Từ đó áp dụng vào thiết kế, xây dựng, triển khai giải pháp lưu trữ và chia sẻ dữ liệu đường đi GPS trực tuyến.
 - + Chương 4: kiểm thử Microservices cho hệ thống
 - + Chương 5: Đánh giá kiến trúc Microservices và đưa ra những hiệu quả mà kiến trúc Microservices mang lại.
- Phần III. Kết luận: Trình bày kết quả đạt được cũng như hướng phát triển chung của đề tài.

PHẦN II. NỘI DUNG

Chương 1: Mô tả hệ thống

Mở đầu chương 1 trình bày các yêu cầu cơ bản để phát triển hệ thống chia sẻ và lưu trữ dữ liệu GPS: mô tả chi tiết bài toán, đặt vấn đề, các yêu cầu giao tiếp bên ngoài, các yêu cầu phi chức năng, chức năng chính của hệ thống và cuối cùng là sơ đồ usecase của hệ thống.

1.1. Mô tả chi tiết bài toán

Hiện nay, hầu hết các ứng dụng khi chúng ta sử dụng như điện thoại hoặc các thiết bị trên những chiếc xe ô tô nó sẽ ghi lại lịch sử vị trí các địa điểm mà chúng ta đã đi qua. Sau đó những thiết bị này muốn trao đổi thông tin với các hệ thống với nhau thì sẽ trao đổi bằng cách nào? Để có thể chia sẻ dữ liệu đó trên tất cả các thiết bị GPS hoặc các hệ thống thì chúng ta cần định dạng dữ liệu chuẩn cho các thông tin về lịch sử vị trí.

Định dạng dữ liệu tiêu chuẩn đó được gọi là GPS Exchange hoặc GPX. Tiêu chuẩn này xác định cách các định tuyến, đường đi và điểm tham chiếu được tạo bằng cách sử dụng một định dạng văn bản đặc biệt. tiêu chuẩn GPX cũng cung cấp các phần mở rộng cho định dạng tiêu chuẩn có thể áp dụng cho các tuyến đường, đường đi và điểm tham chiếu. Các phần mở rộng này được xác định bởi nhà cung cấp hoặc ứng dụng cụ thể.

GPX được sử dụng để mô tả các điểm tham chiếu, đường đi và các tuyến đường. Định dạng mở và có thể dựng mà không cần phải trả phí. Dữ liệu vị trí (tùy chọn độ cao, thời gian và các thông tin khác) được lưu trữ trong các thẻ và có thể được hoán đổi cho nhau giữa các thiết bị GPS và phần mềm. Do đó, nó giúp chúng ta có thể theo dõi các thông tin về vị trí các địa điểm mà chúng ta đã đi qua.

“Hệ thống lưu trữ và chia sẻ dữ liệu GPS” cho phép chúng ta upload một file GPX lên hệ thống. Khi đó hệ thống sẽ phân tích tất cả các thông tin của file GPX để có thể lưu trữ thông tin vào cơ sở dữ liệu. Bên cạnh đó, người dùng có thể truy xuất để xem tất cả thông tin về các tuyến đường mà chúng ta đã upload thông qua file GPX, ngoài ra chúng ta cũng có thể xem chi tiết từng thông tin của file GPX về các định tuyến, đường đi hoặc điểm tham chiếu.

1.1.1. Đặc điểm cơ sở dữ liệu

Cơ sở dữ liệu phải được thiết kế hợp lý để khi truy xuất các thông tin về file gpx, các tuyến đường phải nhanh chóng, chính xác không gặp các lỗi về bảng mã, chính tả, kiểu dữ liệu.

1.1.2. Môi trường vận hành

- Ứng dụng chạy trên hệ điều hành MacOS, Window 8 trở lên
- Ngôn ngữ: Java

- Java Framework: Spring MVC
- Cơ sở dữ liệu: InfluxDB

1.2. Đặt vấn đề

Để tiến hành thiết kế các chức năng của ứng dụng trong một dự án phần mềm thì có nhiều cách để thực nghiệm. Trong luận văn này, chúng tôi sẽ tiến hành thiết kế và cài đặt hệ thống lưu trữ và chia sẻ dữ liệu GPS bằng việc áp dụng mô hình Microservices.

Để thực hiện được phương pháp này thì ta lần lượt trả lời các câu hỏi. Sau đó thực nghiệm và cuối cùng là so sánh kết quả.

Câu hỏi 1: Làm thế nào để thiết kế được ứng dụng theo mô hình Microservices?

Câu hỏi 2: Các vấn đề cần quan tâm khi áp dụng mô hình Microservices trong phát triển phần mềm là gì?

Câu hỏi 3: Những lợi ích nào để mô hình Microservices có trong việc thiết kế và cài đặt so với mô hình thông thường?

1.3. Các yêu cầu giao tiếp bên ngoài

Tuân thủ các yêu cầu về vùng miền, quốc gia, các quy định trong thiết kế.

Mã nguồn, cơ sở dữ liệu không dư thừa, nhiều lỗi.

Từ ngữ phải đúng chính tả, rõ ràng, phù hợp phong tục tập quán và văn hóa Việt Nam.

1.4. Các yêu cầu phi chức năng

1.4.1 Yêu cầu thực thi

Phần mềm chạy tốt trên hệ điều hành mà cấu hình tối thiểu được đề ra.

Không bị lỗi phần mềm khi đang chạy chương trình.

Tốc độ phản hồi nhanh, ổn định.

Thời gian đáp ứng các thao tác của người dùng nhanh, dưới 5 giây.

Hiệu suất làm việc hơn 90%.

Hệ thống làm việc tốt, liên tục 24/24, tốn ít tài nguyên, không gây tình trạng lag hay treo máy, thao tác chậm chạp hay xuất hiện lỗi.

1.4.2. Yêu cầu an toàn

Người dùng chỉ được thao tác các chức năng nằm trong phạm vi cho phép được định sẵn.

Các dữ liệu mà ứng dụng cung cấp phải đảm bảo về mặt thông tin.

Ứng dụng hoạt động không làm ảnh hưởng đến hệ điều hành và các ứng dụng khác.

Hệ thống không chứa virus, các phần mềm độc hại hay các tập tin rác.

Cơ sở dữ liệu phải được sao lưu và phục hồi theo định kì.

1.5. Các yêu cầu khác

- Ngôn ngữ sử dụng thuần tiếng việt.
- Mức độ bảo mật cao, đáng tin cậy.
- Tốc độ xử lý nhanh chóng, chính xác.
- Dễ bảo trì, cũng như nâng cấp.

1.6. Các chức năng chính của hệ thống

1.6.1. Chức năng upload file gpx

Tên Usecase: Upload GPX file	ID: UC01
Actor chính: User	Mức độ cần thiết: Bắt buộc
	Phân loại: Trung bình
Các thành phần tham gia và mối quan tâm:	
Mô tả tóm tắt: Cho phép người dùng upload file gpx	
Luồng xử lý bình thường của sự kiện: <ul style="list-style-type: none">- Người dùng truy cập URL : localhost:8080/upload- Chọn file GPX mà người dùng cần upload- Service GPX Gateway có nhiệm vụ upload file gpx của người dùng	

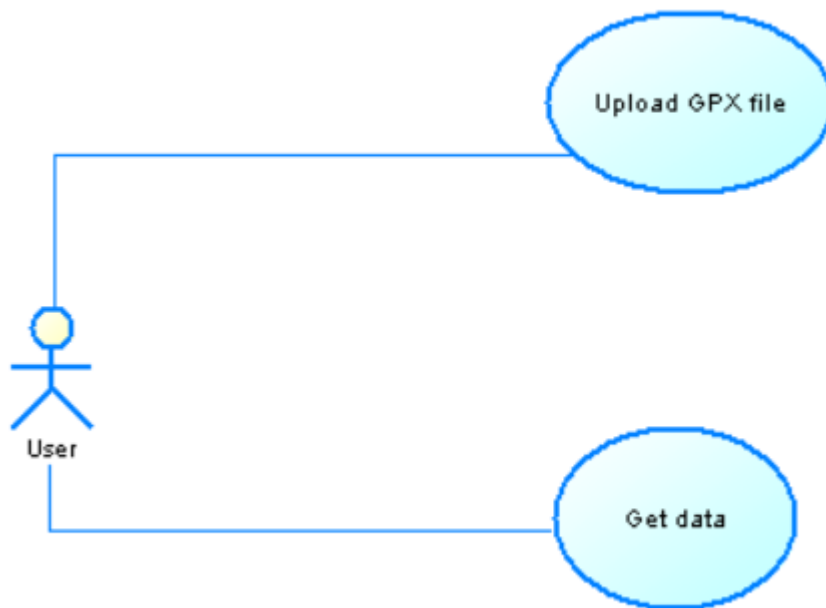
Bảng 1.1: Chức năng upload file gpx

1.6.2. Chức năng xem thông tin file gpx đã upload

Tên Usecase: Get data	ID: UC03
Actor chính: User	Mức độ cần thiết: Bắt buộc
	Phân loại: Trung bình
Các thành phần tham gia và mối quan tâm:	
Mô tả tóm tắt: Cho phép người dùng xem thông tin chi tiết từ file gpx đã upload từ cơ sở dữ liệu	
Luồng xử lý bình thường của sự kiện: <ul style="list-style-type: none">- Người dùng truy cập URL: localhost:8080/rest/timeseries/{trackerID}/latest- Service Timeseries Provider sẽ lấy dữ liệu từ InfluxDb và trả về cho người dùng- Người dùng có thể xem dữ liệu với dạng JSON.	

Bảng 1.2: Chức năng xem thông tin file gpx đã upload

1.7. Sơ đồ usecase



Hình 1.1: Sơ đồ các trường hợp sử dụng

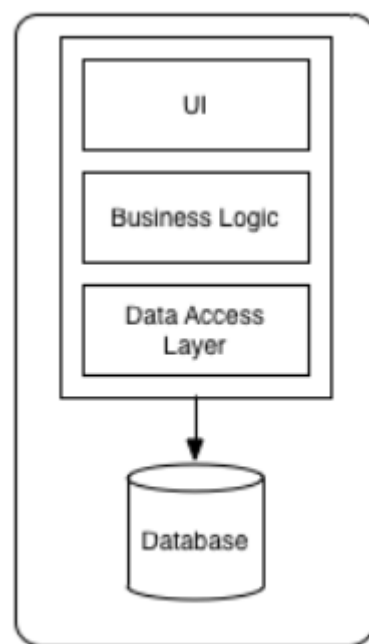
Chương 2: Cơ sở lý thuyết

Chương 2 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.

2.1. Kiến trúc Microservices[16]

2.1.1. 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.



Monolithic Architecture

Hình 2.1: 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.

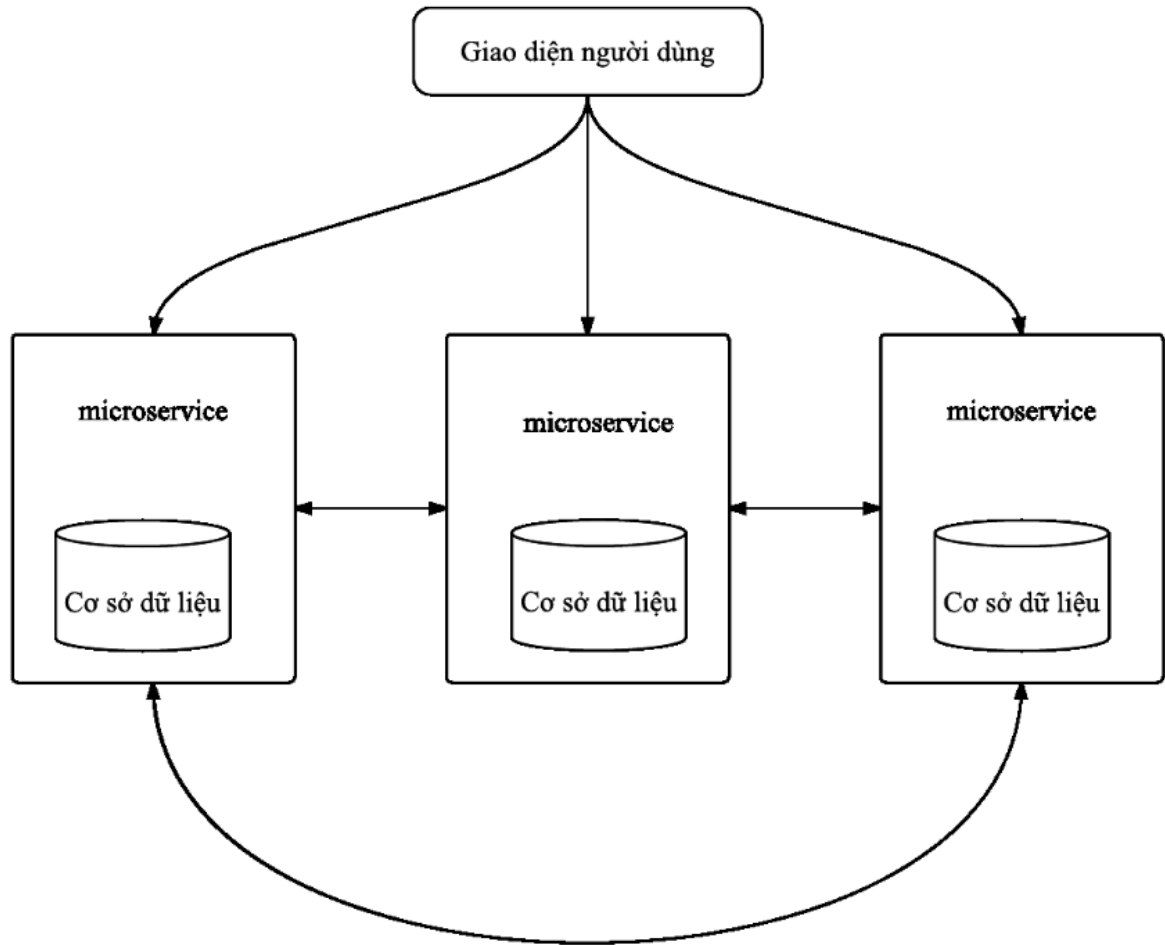
Mặc dù kiến trúc monolithic còn tồn tại nhiều nhược điểm nhưng nhờ các ưu điểm độc lập, dễ xây dựng và quản lý Monolithic đã được sử dụng để xây dựng và phát triển thành các kiểu kiến trúc phần mềm khác rất phổ biến hiện nay như kiến trúc hướng dịch vụ, Microservices cũng là một kiến trúc được xây dựng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

2.1.2. Kiến trúc Microservices

Tư tưởng chia để trị của kiến trúc Microservices thực ra đã được đề xuất từ khá lâu trước đây nhưng phải đến tháng 3/2014 Martin Fowler – tác giả của mô hình phát triển ứng dụng Agile và James Lewis đã có một bài giải thích chi tiết và thú vị trên trang web cá nhân về kiến trúc Microservices. Sau đó tháng 7/2014 Stefan Boijse – Giám đốc, đồng sáng lập công ty bán thiết bị wifi Karma đã viết trên trang blog cá nhân của mình về quá trình xây dựng cửa hàng trực tuyến bán sản phẩm của Karma bằng cách chia nhỏ các ứng dụng thành các phần nhỏ hơn để thực hiện, nhưng ông và các nhân viên của mình đã gặp khó khăn trong khâu kiểm thử, vì việc kiểm thử các dịch vụ nhỏ thì tương đối dễ nhưng khi ghép chúng vào một hệ thống thì lại là một bài toán khó.

Tháng 11/2014 Toby Clemson – một nhà phát triển của ThoughtWorks đã có một bài diễn thuyết cách tiếp cận đa lớp để thử lỗi trong hệ thống xây dựng trên kiến trúc Microservices. Tại đây Toby đã đề nghị thử lỗi cho từng Microservices một cách độc lập để khắc phục khó khăn trong khâu kiểm thử của hệ thống xây dựng trên Microservices mà Stefan và các đồng nghiệp của ông đã gặp phải. Tất nhiên Toby cũng khuyến cáo việc tất cả các Microservices chạy tốt thì không có nghĩa cả hệ thống sẽ chạy tốt. Vì vậy Toby cũng đề xuất một loạt các phương pháp về cách tích hợp các thành phần, tương tác, thử lỗi đầu cuối để các nhà phát triển phần mềm khắc phục được các khó khăn trong quá trình xây dựng phần mềm dựa trên kiến trúc Microservices.

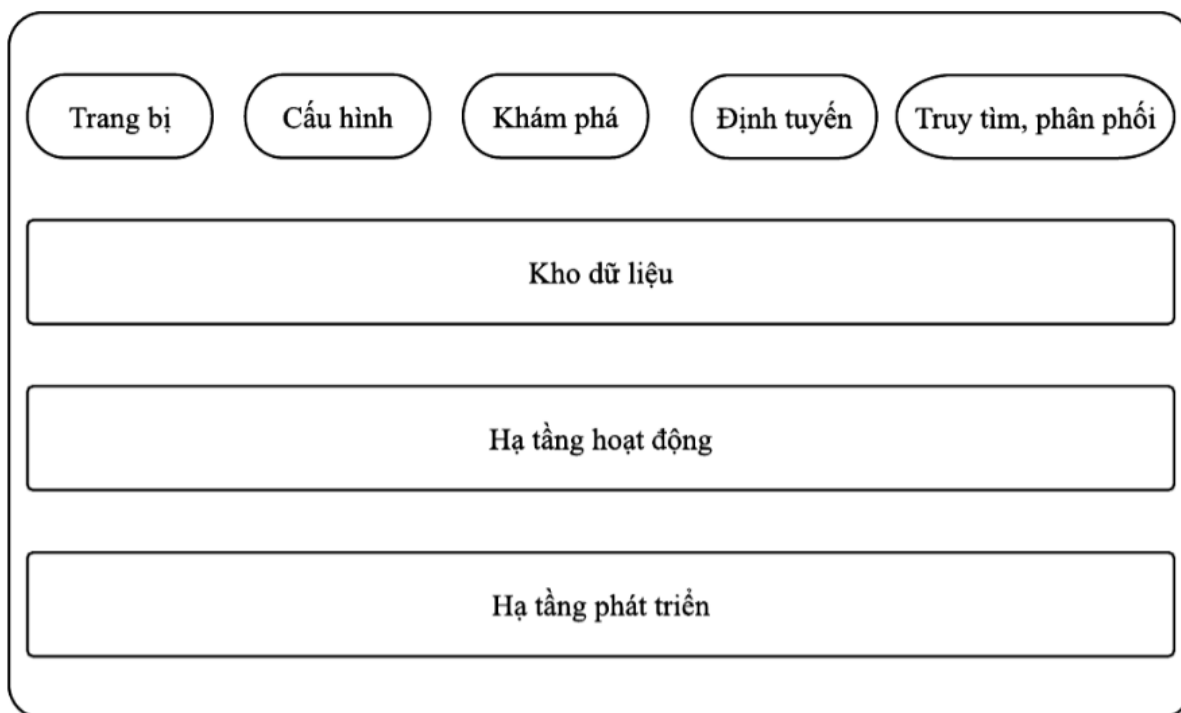
Kiến trúc Microservices là một kiểu kiến trúc phần mềm dành cho các hệ thống được xây dựng dựa trên các dịch vụ, trong đó các dịch vụ này được chia thành các thành phần nhỏ và hoàn toàn độc lập với nhau, mỗi thành phần nhỏ này được gọi là Microservices. Mỗi Microservices chỉ tập trung và giải quyết một chức năng duy nhất trong hệ thống và giao tiếp với nhau thông qua các API. Ví dụ một ứng dụng được xây dựng dựa trên kiến trúc Microservices trong hình.



Hình 2.2: Ví dụ kiến trúc Microservices

Ứng dụng bao gồm ba Microservices độc lập và giao tiếp với nhau để thực hiện chức năng của ứng dụng và mỗi Microservices đều có một cơ sở dữ liệu riêng và các Microservices này giao tiếp với nhau thông qua API.

Mô hình kiến trúc Microservices bao gồm ba thành phần chính: các hạ tầng phát triển (development), các hạ tầng hoạt động (operational) và kho dữ liệu (datastore). Ngoài ra còn có một số thành phần hỗ trợ nhau khác như các công cụ được trang bị (tooling), các công cụ dùng để cấu hình (configuration), định tuyến (routing) và truy tìm phân phối (observability). Các thành phần này có thể có hoặc không.



Hình 2.3: Mô hình kiến trúc Microservices

- Hạ tầng phát triển (development) là các nền tảng được lựa chọn để xây dựng hệ thống như ngôn ngữ lập trình và các ứng dụng đóng gói được sử dụng trong hệ thống.
- Hạ tầng hoạt động (operational) bao gồm các thành phần cơ sở hạ tầng để triển khai hệ thống như máy chủ, ...
- Datastore quản lý các công nghệ mà hệ thống được sử dụng để lưu trữ cơ sở dữ liệu.
- Các thành phần hỗ trợ: sự trang bị (tooling), cấu hình (configuration), khám phá (discovery), định tuyến (routing) và truy tìm, phân phối (observability).

Mỗi Microservices trong kiến trúc Microservices đều có những dịch vụ trong các hệ thống khác. Điều khác biệt lớn nhất là phạm vi của Microservices, nó được giới hạn trong một chỉ tiêu nhất định mà trong định nghĩa được gọi là “nhỏ”. “Nhỏ” ở đây không phải được tính bởi số lượng dòng code được sử dụng trong mỗi Microservices, mà nhỏ ở đây chính là phạm vi để thực hiện mỗi chức năng bất kỳ trong hệ thống – mỗi Microservices trong kiến trúc Microservices chỉ giải quyết một chức năng duy nhất của hệ thống.

Các Microservices giao tiếp với nhau thông qua API. Mỗi API dùng để kết nối các Microservices sẽ có hai kiểu kết nối như sau:

- Đồng bộ: Microservices A sử dụng API kết nối đến Microservices B bằng cách gửi thông báo và chờ Microservices B phản hồi.

- **Bất đồng bộ:** Microservices A gửi thông báo đến Microservices B sau đó tiếp tục chạy và xử lý phản hồi của B sau khi có câu trả lời.

Mỗi Microservices trong kiến trúc Microservices sẽ gọi đến các Microservices khác trên API bằng ba cách gọi sau:

- **REST (Representational State Transfer):** các thông báo được gửi qua HTTP để truy vấn, thao tác dữ liệu. Kiểu dữ liệu sau truy vấn sẽ được trả về dạng XML, JSON, JSONb.
- **RPC (Remote Procedure Call):** cách gọi này giống như mô hình Client-Server trong đó Microservices gửi yêu cầu kết nối sẽ đảm nhiệm vai trò Client và Microservices nhận kết nối sẽ là Server.
- **SOAP (Simple Object Access Protocol):** các Microservices truyền thông điệp cho nhau dưới dạng XML.

Đối với các Microservices dành cho người dùng đầu cuối sẽ không được kết nối trực tiếp vào các Microservices khác mà chúng sẽ giao tiếp với nhau thông qua một cổng API. Cổng API này có nhiệm vụ phân tải, lưu trữ tạm thời và kiểm tra quyền đăng nhập của người dùng đầu cuối.

Mỗi Microservices trong kiến trúc Microservices có thể là một cơ sở dữ liệu riêng và chúng có cách lưu trữ dữ liệu khác nhau. Đây là một điểm mới trong kiến trúc Microservices, khác với cách lưu trữ dữ liệu tập trung của các kiến trúc khác.

2.1.3. Mẫu thiết kế trong Microservices

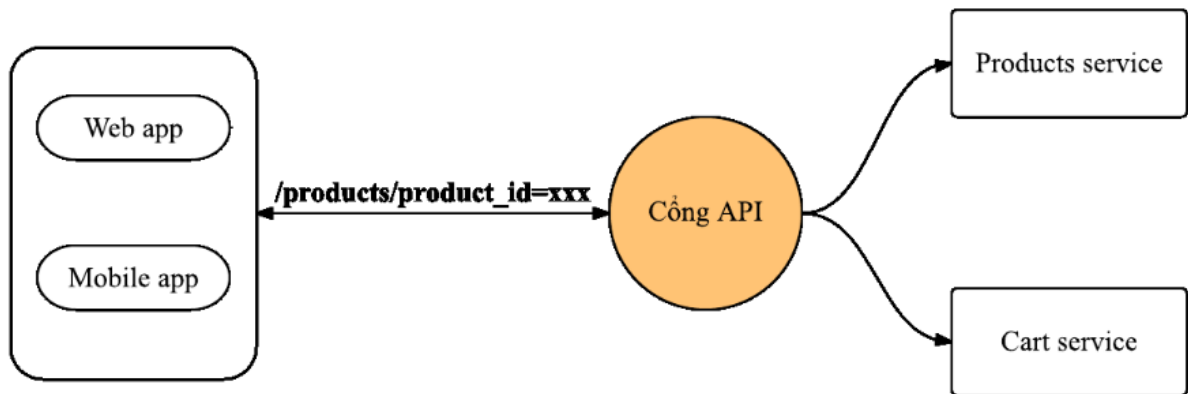
Microservices là kiểu kiến trúc được hình thành từ các dịch vụ nên khi sử dụng kiểu kiến trúc này ta phải lựa chọn phương pháp để các máy trạm có thể kết nối với các Microservices và cơ chế để các Microservices phát hiện và giao tiếp với nhau trên mạng. Ứng với mỗi kiểu kết nối sẽ có một hoặc nhiều mẫu thiết kế riêng cho nó. Các máy trạm kết nối đến Microservices thì kiến trúc Microservices sẽ sử dụng mẫu thiết kế cổng API, để phát hiện dịch vụ thì có thể sử dụng một trong hai mẫu thiết kế phát hiện phía máy trạm (Client-side discovery) và phát hiện phía máy chủ (Server-side discovery). Để đăng ký dịch vụ thì kiến trúc Microservices cho phép các Microservices đăng ký dịch vụ với hai mẫu thiết kế tự đăng ký (self-registration) và đăng ký qua bên thứ ba (third-party registration).

Cổng API

Trên thực tế các máy trạm có thể kết nối trực tiếp đến các Microservices thông qua một số phương thức nhưng sử dụng phương pháp này sẽ gây nguy hiểm cho tường lửa hệ thống. Vì vậy người ta sẽ tránh sử dụng các phương pháp kết nối trực tiếp này mà sử dụng một máy chủ trung gian để thực hiện việc đó, máy chủ trung gian này được gọi là cổng API. Cổng API này có chức năng và vai trò giống với mẫu thiết kế Facade trong kiến trúc hướng đối tượng, nó giúp hệ thống ẩn đi kiến trúc nội bộ của mình với các hệ thống khác và cung cấp các API cho mỗi máy trạm khác nhau. Thông qua cổng API các yêu cầu được gửi đi từ máy trạm sẽ được định tuyến, chuyển đổi giao thức để

được Microservices thích hợp thực hiện yêu cầu và sau đó sẽ tổng hợp các kết quả trả lại về các máy trạm.

Ví dụ trong hình mô tả một ứng dụng quản lý sản phẩm của hai Microservices Product service và Cart service. Cổng API đóng vai trò kết nối máy trạm đến các Microservices của hệ thống.



Hình 2.4: Cổng API trong ứng dụng quản lý sản phẩm

Khi các máy trạm muốn truy xuất thông tin của một sản phẩm, nó sẽ gửi yêu cầu đến cổng API và cổng API sẽ định tuyến các yêu cầu này để chuyển chúng đến Products service xử lý. Sau khi Products service xử lý xong, các câu trả lời sẽ được cổng API tổng hợp lại và trả về các máy trạm qua đường dẫn /products/product_id=xxx.

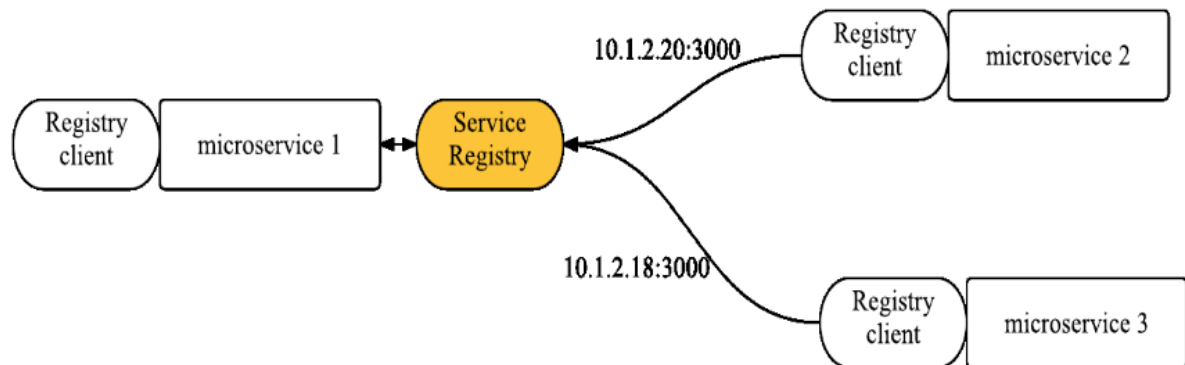
Việc sử dụng cổng API trong kiến trúc Microservices giúp các hệ thống có thể tăng độ bảo mật vì nó giúp hệ thống giấu đi kiến trúc bên trong của mình, các yêu cầu truy xuất từ các máy trạm thay vì phải truy vấn đến các Microservices cụ thể thì nó chỉ cần gửi yêu cầu đến cổng API. Điều này sẽ giúp giảm số lượng trao đổi giữa máy trạm và các Microservices để tối ưu hóa mã nguồn phía máy trạm. Nhưng cũng chính những yêu cầu này của máy trạm phải đi qua cổng API mới truy xuất được đến các Microservices nên có thể sẽ nó trở thành nút thắt cổ chai trong hệ thống.

Phát hiện dịch vụ

Các Microservices trong kiến trúc Microservices có thể trực tiếp giao tiếp với nhau hoặc thông qua cổng API, để thực hiện được việc này thì cổng API và các Microservices cần biết chính xác vị trí của các Microservices trong hệ thống. Tuy nhiên mỗi Microservices được gán một vị trí động và vị trí bản sao của chúng cũng sẽ được thay đổi liên tục để đáp ứng khả năng tự điều chỉnh và nâng cấp, cho nên cổng API hay bất cứ Microservices trong hệ thống khi muốn kết nối đến một Microservices khác cũng cần có một cơ chế để phát hiện dịch vụ. Có hai cơ chế để phát hiện dịch vụ trong kiến trúc Microservices đó là phát hiện phía máy trạm (Client-side discovery) và phát hiện phía máy chủ (Server-side discovery). Cả hai cơ chế này đều thông qua service registry – một cơ sở dữ liệu chứa các thể hiện (instances) và vị trí của các Microservices trong hệ thống để phát hiện dịch vụ. Mỗi service registry sẽ có một API dùng để quản lý việc

đăng kí của các Microservices trong hệ thống và một API truy vấn để thực hiện các truy vấn đến các Microservices đã được đăng kí trong hệ thống.

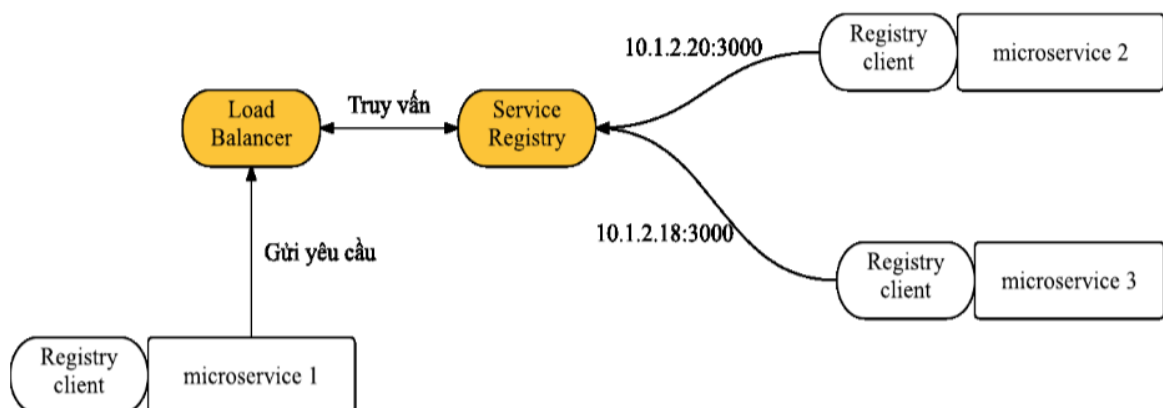
Phát hiện dịch vụ phía máy trạm (Client-side discovery): việc phát hiện dịch vụ được thực hiện từ phía máy trạm, mỗi Microservices đều có một cơ sở dữ liệu để đăng ký dịch vụ (registry client) của mình và các registry client sẽ được kết nối với cơ sở dữ liệu service registry. Máy trạm muốn biết vị trí của Microservices nào thì chỉ việc gửi yêu cầu truy vấn đến service registry và sau đó service registry sẽ trả lại máy trạm địa chỉ của Microservices mà nó muốn kết nối để hai Microservices này kết nối với nhau.



Hình 2.5: Cơ chế phát hiện phía máy trạm (client-side discovery)

Hình 2.5 thể hiện hoạt động cơ chế phát hiện dịch vụ phía máy trạm, mỗi Microservices trong hệ thống đều có một registry client dùng để lưu vị trí của nó và các registry client này đều có kết nối đến service registry. Khi Microservices 1 muốn thực hiện kết nối đến Microservices 2 nó sẽ gửi yêu cầu biết vị trí của Microservices 2 đến service registry. Sau đó service registry sẽ trả về địa chỉ của Microservices 2 cho Microservices 1.

Phát hiện dịch vụ phía máy khách (Server-side discovery) cho phép phát hiện các máy trạm bằng cách máy trạm gửi các yêu cầu kết nối đến các Microservices thông qua cân bằng tải – load balancer. Cân bằng tải này sẽ truy vấn đến service registry và định tuyến các yêu cầu này đến các Microservices thích hợp.



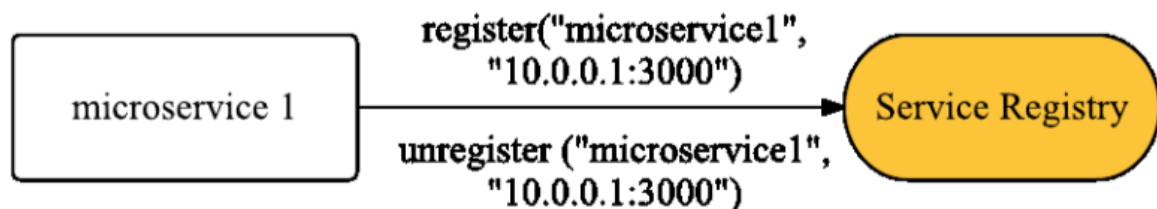
Hình 2.6: Cơ chế phát hiện dịch vụ phía máy khách (server-side discovery)

Hình 2.6 mô tả cơ chế phát hiện dịch vụ phía máy khách, trong đó Microservices 1 muốn kết nối đến Microservices 2 trong hệ thống nó sẽ gửi một yêu cầu kết nối đến load balancer sau đó load balancer sẽ truy vấn đến service registry để lấy vị trí của Microservices 2 và định tuyến yêu cầu này đến Microservices 2 để thực hiện.

Đăng kí dịch vụ

Để các Microservices trong kiến trúc Microservices có thể kết nối với nhau thì mỗi Microservices cần phải được đăng kí dịch vụ. Có hai cách để thực hiện việc đăng kí dịch vụ:

Tự đăng kí (self registry): Mỗi thể hiện (instances) của mỗi Microservices có khả năng tự đăng kí và hủy đăng kí Microservices đó với service registry bằng cách gửi một yêu cầu (registry hoặc unregistry) có chứa thông tin về tên Microservices và địa chỉ của nó.



Hình 2.7: Cơ chế tự đăng kí dịch vụ (self-registration)

Trong Hình 2.7, khi một Microservices 1 muốn đăng kí với service registry nó sẽ gửi đến service registry một thông báo register (“Microservices1”, “10.0.0.1:3000”) và ngược lại khi Microservices 1 gửi đến service registry thông báo unregister (“Microservices”, “10.0.0.1:3000”) thì Microservices 1 sẽ bị hủy đăng kí ra khỏi service registry.

Đăng kí qua bên thứ ba (third-party registration) các thể hiện của Microservices không cần tự đăng ký với service registry mà sẽ có 1 thành phần khác thực hiện việc này, thành phần đó được gọi là registrar. Registrar theo dõi các thay đổi của Microservices trong hệ thống bằng cách bỏ phiếu trong môi trường phát triển (polling) hoặc đăng kí vào các sự kiện (subscribing). Khi registrar phát hiện có một Microservices mới tham gia vào hệ thống nó sẽ đăng kí Microservices đó với service registry bằng cách gửi thông báo register có chứa tên Microservices và địa chỉ của Microservices đó đến service registry và thông báo unregister trong trường hợp hủy đăng kí.



Hình 2.8: Cơ chế đăng ký dịch vụ qua bên thứ ba

Hình 2.8 mô tả cơ chế hoạt động của Registrar trong kiến trúc Microservices, Registrar liên tục kiểm tra hệ thống thông qua bằng cách bỏ phiếu trong môi trường phát triển hoặc đăng ký vào các sự kiện khi Registrar phát hiện Microservices chưa đăng ký với service registry nó sẽ gửi đến service registry một thông báo register(“Microservices 1”, “10.0.0.1:3000”) và ngược lại khi Registrar gửi đến service registry thông báo unregister(“Microservices 1”, “10.0.0.1:3000”) thì Microservices 1 sẽ bị hủy đăng ký ra khỏi service registry.

2.2. Spring Framework

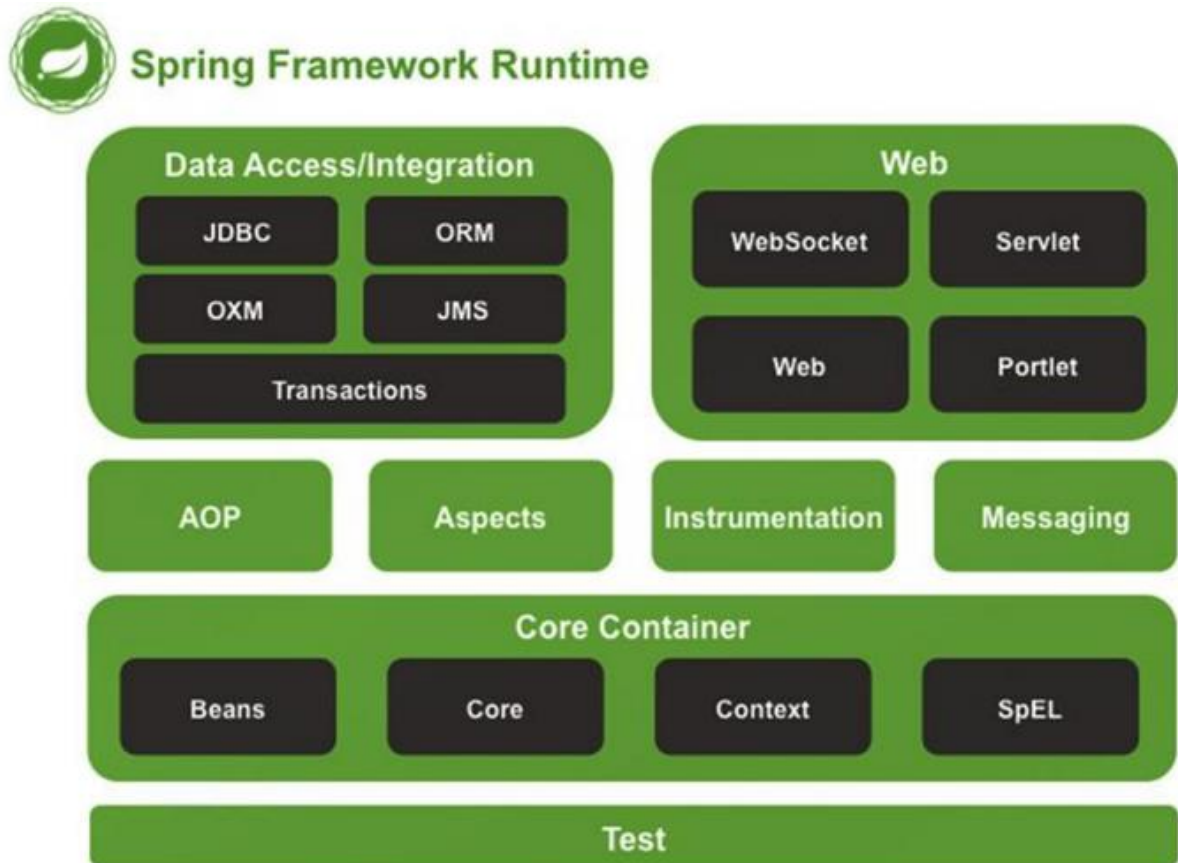
2.2.1. Giới thiệu

Spring framework hay ngắn hơn là Spring là một framework mã nguồn mở được viết bằng ngôn ngữ Java. Nó được xem như là một giải pháp kiến trúc tốt nhất của JavaEE hiện nay.

Spring là một framework mã nguồn mở có phiên bản dùng cho Java Platform và cả .Net Platform. Phiên bản đầu tiên được viết bởi Rod Johnson và đưa ra cùng cuốn sách Expert One-on-One J2EE Design and Development được xuất bản tháng 10 năm 2002.

Spring có thể dùng cho tất cả các ứng dụng viết bằng Java, nhưng nó thành công nhất trên lĩnh vực ứng dụng web trên nền JavaEE

2.2.2. Các module chính trong Spring Framework



Hình 2.9: Các module của Spring Framework

Test: Đây là tầng cung cấp cho người dùng khả năng hỗ trợ kiểm thử với Junit và TestNG.

Spring Core Container: Nó có bao gồm một số module khác như: Spring Core, Bean : có khả năng cung cấp các tính năng như IoC và Dependency Injection.

Spring Context: Hỗ trợ đa dạng ngôn ngữ và các tính năng cho JavaEE cho người dùng như: EJB, JMX.

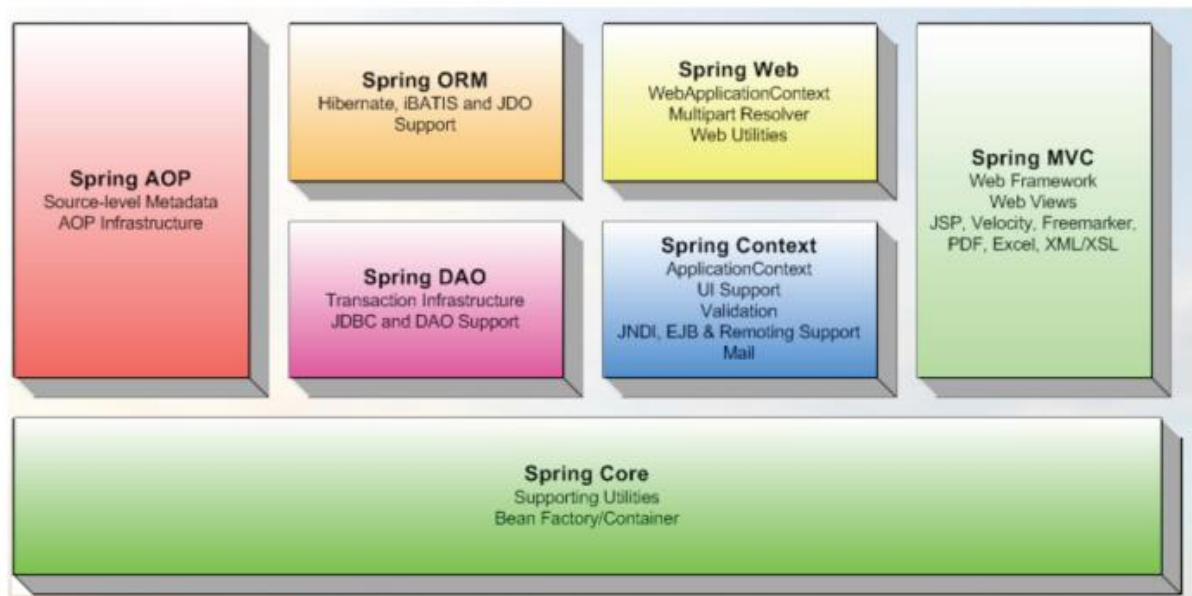
Expression Language: Có khả năng mở rộng từ Expression Language bên trong JSP. Từ đó, cung cấp các hỗ trợ trong quá trình setting hoặc getting các giá trị. Hầu hết các method đều sẽ thực hiện cải tiến cho phép truy cập vào các collections, index, các toán tử logic ...

AOP, Aspects và Instrumentation: các module này sẽ giữ nhiệm vụ hỗ trợ cho các cài đặt lập trình viên thiên hướng khía cạnh và khả năng hỗ trợ tích hợp với AspectJ.

Data Access / Integration: đây là nhóm bao gồm: JDBC, ORM, OXM, JMS, và module Transaction. Chúng có khả năng cung cấp giao tiếp cùng với cơ sở dữ liệu.

Web: nó còn hay được gọi là Spring MVC, đây là một trong những nhóm bao gồm: Web, Web-Servlet, ... nó sẽ hỗ trợ cho việc tạo ra các ứng dụng web.

Spring được xây dựng với 7 module chính:



Hình 2.10: Module chính của Spring Framework

Spring Core: Core package là phần cơ bản nhất của Spring, cung cấp những tính năng như IoC (Inversion of Control) và DI (Dependency Injection). Khái niệm cơ bản là BeanFactory, một cài đặt của Factory Pattern, cho phép “móc nối” sự phụ thuộc giữa các đối tượng trong file cấu hình.

Spring Context: là một file cấu hình để cung cấp thông tin ngữ cảnh của spring. Spring Context cung cấp các service như JNDI access, EIB integration, email, internationalization, validation và scheduling functionality.

Spring AOP (Aspects – Oriented Programming): là module tích hợp tính năng lập trình hướng khía cạnh vào Spring Framework thông qua cấu hình của nó. Spring AOP module cung cấp các dịch vụ quản lý giao dịch cho các đối tượng trong bất kỳ ứng dụng nào sử dụng Spring. Với Spring AOP chúng ta có thể tích hợp declarative transaction management vào trong ứng dụng mà không cần dựa vào EJB component. Spring AOP module cũng đưa lập trình metadata vào trong Spring. Sử dụng cái này chúng ta có thể thêm annotation vào source code để hướng dẫn Spring nơi và làm thế nào để liên hệ với aspects.

Spring DAO (Data Access Object): Tầng JDBC và DAO đưa ra một cây phân cấp exception để quản lý kết nối đến database, điều khiển exception và thông báo lỗi được ném bởi vendor của database. Tầng exception đơn giản điều khiển lỗi và giảm khối lượng code mà chúng ta cần viết như mở và đóng kết nối.

Module này cũng cung cấp các dịch vụ quản lý giao dịch cho các đối tượng trong ứng dụng Spring.

Spring Web: Nằm trên application context module, cung cấp context cho các ứng dụng web. Spring cũng hỗ trợ tích hợp Struts, JSF, và Webwork. Web module cũng

làm giảm bớt công việc điều khiển nhiều request và gắn các tham số request vào các đối tượng domain.

Spring ORM (Object Relational Mapping): Spring có thể tích hợp với một vài ORM Framework để cung cấp các Object Relation tool bao gồm: JDO, Hibernate, OJB và iBatic SQL Maps.

Spring MVC Framework: Spring MVC Framework thì cài đặt đầy đủ đặc tính của MVC pattern để xây dựng các ứng dụng web. Spring MVC framework thì cấu hình thông qua giao diện và chứa được một số kỹ thuật View bao gồm: JSP, Velocity, Tiles và generation of PDF và Excel file.

2.3. RabbitMQ

2.3.1. RabbitMQ là gì?

RabbitMQ là một chương trình phần mềm trung gian giúp các ứng dụng, hệ thống hay server khác nhau có thể giao tiếp, trao đổi dữ liệu với nhau. Nhiệm vụ của RabbitMQ được hiểu đơn giản là : nhận message từ nhiều nguồn => lưu trữ, sắp xếp sao cho hợp lý => đẩy tới đích đến.

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.

2.3.2. Message Broker là gì?

Hiểu một cách đơn giản, 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.

Chương trình message broker sẽ đảm nhận việc xác thực, chuyển đổi, định tuyến cho message giữa các ứng dụng với nhau.

2.3.3. Các tính năng và lợi ích của RabbitMQ

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

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 : Java, PHP, JavaScript, Go, Ruby, ..

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.

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 các public cloud cũng như private cloud.

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.

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.

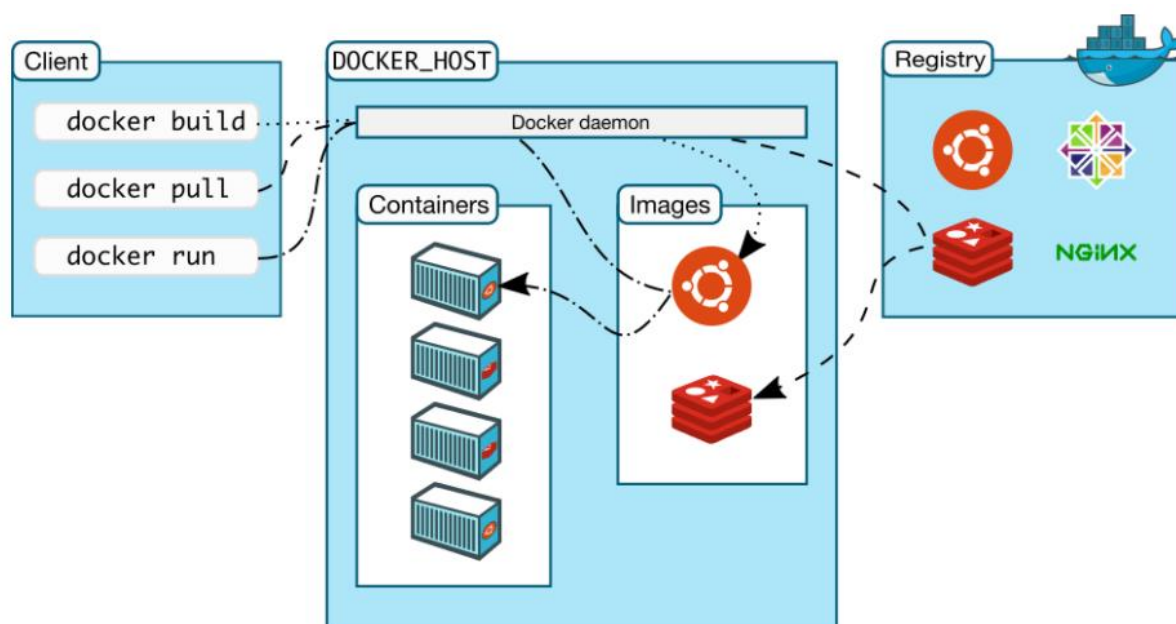
2.4. Docker

2.4.1. 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.

2.4.2. 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 tự 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 dừng container.



Hình 2.11: Mô hình hoạt động của Docker

2.4.3. 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 chu 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.

2.5. InfluxDB

2.5.1. Cơ sở dữ liệu chuỗi thời gian là gì?

Cơ sở dữ liệu chuỗi thời gian có thể được định nghĩa là các điểm dữ liệu được lập chỉ mục theo thứ tự thời gian của chúng, trong đó khoảng cách giữa hai điểm dữ liệu có thể bằng hoặc không. Nếu tần suất tại đó các điểm dữ liệu được lấy là không đổi (Ví dụ: lấy mẫu dữ liệu sau mỗi 10ms) thì chuỗi đó được gọi là chuỗi dữ liệu thời gian rời rạc.

2.5.2. InfluxDB là gì?

Là một cơ sở dữ liệu có mục đích chung là lưu trữ cơ sở dữ liệu chuỗi thời gian (Time-series data). Việc lưu trữ và truy vấn dữ liệu được tối ưu hóa cho các điểm dữ liệu có thành phần thời gian.

Chương 3: Đề xuất giải pháp áp dụng Microservices vào ứng dụng GPS online

Chương 3 sẽ trình bày các bước để thiết kế và xây dựng hệ thống chia sẻ và lưu trữ GPS trực tuyến. Phần đầu chương sẽ giới thiệu về file GPX. Các phần tiếp theo của chương sẽ trình bày chi tiết các bước để xây dựng hệ thống sử dụng kiến trúc Microservices.

3.1. Giới thiệu về GPX File.

Tập GPX là dữ liệu GPS được lưu ở định dạng GPS Exchange, một tiêu chuẩn mở có thể được các chương trình GPS sử dụng tự do. Nó chứa dữ liệu vị trí kinh độ và vĩ độ, bao gồm các điểm mốc, tuyến đường và đường đi. Các tập GPX được lưu trong [XML](#) định dạng cho phép dữ liệu GPS dễ dàng được nhập và đọc hơn bởi nhiều chương trình và dịch vụ web.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>

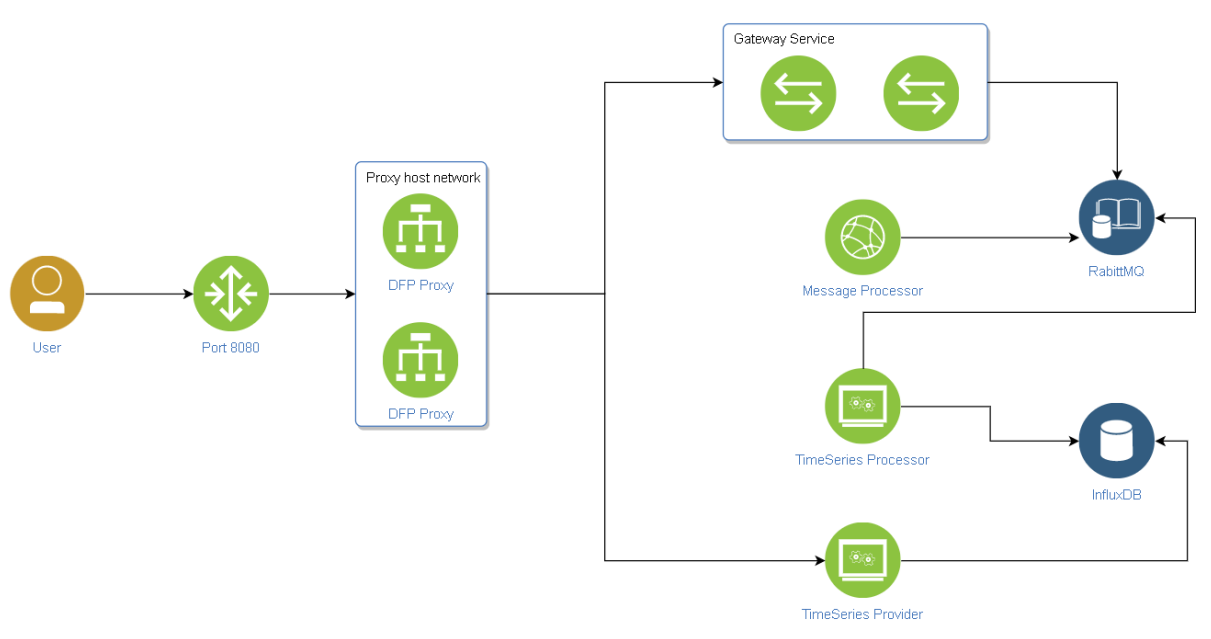
<gpx xmlns="http://www.topografix.com/GPX/1/1" xmlns:gpox="http://-
www.garmin.com/xmlschemas/GpxExtensions/v3" xmlns:gpstpx="http://-
www.garmin.com/xmlschemas/TrackPointExtension/v1" creator="Oregon 400t"
version="1.1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.topografix.com/GPX/1/1 http://-
www.topografix.com/GPX/1/1/gpx.xsd http://www.garmin.com/xmlschemas/-
GpxExtensions/v3 http://www.garmin.com/xmlschemas/GpxExtensionsv3.xsd http://-
www.garmin.com/xmlschemas/TrackPointExtension/v1 http://www.garmin.com/-
xmlschemas/TrackPointExtensionv1.xsd">
  <metadata>
    <link href="http://www.garmin.com">
      <text>Garmin International</text>
    </link>
    <time>2009-10-17T22:58:43Z</time>
  </metadata>
  <trk>
    <name>Example GPX Document</name>
    <trkseg>
      <trkpt lat="47.644548" lon="-122.326897">
        <ele>4.46</ele>
        <time>2009-10-17T18:37:26Z</time>
      </trkpt>
      <trkpt lat="47.644548" lon="-122.326897">
        <ele>4.94</ele>
        <time>2009-10-17T18:37:31Z</time>
      </trkpt>
      <trkpt lat="47.644548" lon="-122.326897">
        <ele>6.87</ele>
        <time>2009-10-17T18:37:34Z</time>
      </trkpt>
    </trkseg>
  </trk>
</gpx>
```

Hình 3.1: Ví dụ về File GPX

Tập GPX lưu trữ ba loại dữ liệu:

- **Waypoint** - Bao gồm GPS tọa độ của một điểm. Nó cũng có thể bao gồm thông tin mô tả khác.
- **Route** - Bao gồm một danh sách các điểm theo dõi, là điểm tham chiếu cho điểm rẽ hoặc điểm giai đoạn, dẫn đến đích.
- **Track** - Bao gồm một danh sách các điểm mô tả một đường dẫn.

3.2. Sơ đồ kiến trúc của hệ thống

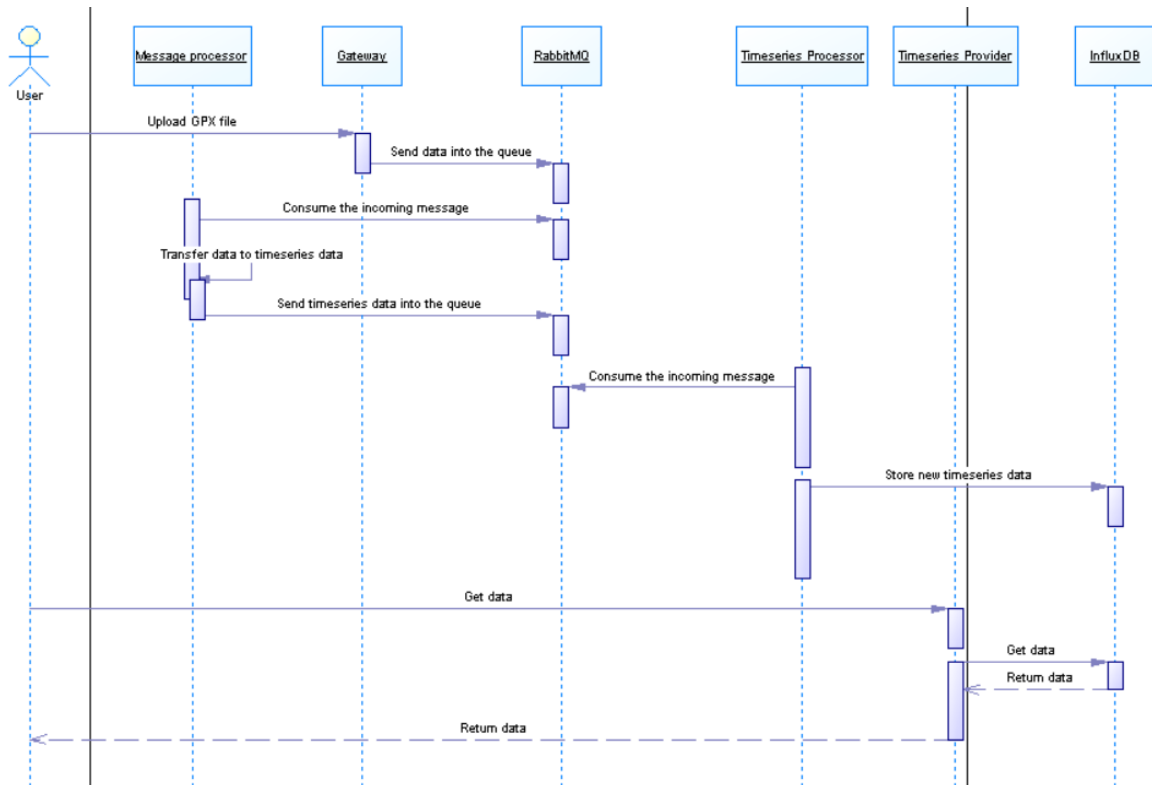


Hình 3.2: Sơ đồ kiến trúc của hệ thống

Trong sơ đồ kiến trúc của hệ thống, bao gồm:

- User: người dùng của hệ thống
- Port: cổng hoạt động của services trong hệ thống
- Các services: Gateway Service, Message Processor, TimeSeries Processor, TimeSeries Provider
- RabbitMQ: dùng để vận chuyển message giúp giao tiếp giữa các service.
- InfluxDB: cơ sở dữ liệu dùng để lưu trữ dữ liệu timeseries.

3.3. Sơ đồ tuần tự của hệ thống



Hình 3.3: Sơ đồ tuần tự của hệ thống

Trong Hình 3.3 là sơ đồ tuần tự của Hệ thống lưu trữ và chia sẻ dữ liệu GPS dùng kiến trúc Microservices mô tả luồng hoạt động của hệ thống một cách chi tiết của các services.

3.4. Cài đặt

3.4.1. Xác định các Microservices cho ứng dụng.

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 kế cơ sở dữ liệu trong mỗi Microservices phải ít ràng buộc, không có khóa ngoại.

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 là được điều này chúng ta nên nhóm các hành vi có liên quan với 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.

Dựa vào các chức năng của hệ thống chia sẻ và lưu trữ GPS trực tuyến, ta có thể xác định được hệ thống có 4 Microservices:

- Gateway service thực hiện chức năng upload file lên RabbitMQ.
- Message Processor service thực hiện chức năng nhận dữ liệu và chuyển đổi dữ liệu thành dữ liệu timeseries sau đó đẩy dữ liệu vào RabbitMQ.
- TimeSeries Processor service thực hiện chức năng nhận dữ liệu timeseries ở RabbitMQ và đưa dữ liệu vào cơ sở dữ liệu InfluxDB.
- TimeSeries Provider service thực hiện chức năng lấy dữ liệu timeseries từ cơ sở dữ liệu và trả về cho người dùng.

3.4.2. Các chức năng chính của ứng dụng

3.4.2.1. Chức năng upload GPX file

Đặt tên cho Queue trên RabbitMQ để chúng ta có thể nhận biết khi service kết nối với RabbitMQ.

```
package model.rabbitmq;  
  
public interface QueueName {  
    String TRACKER_MESSAGE_MAIN = "message.main";  
    String TIMESERIES_MESSAGE_MAIN = "timeseries.main";  
}
```

Hình 3.4: Class QueueName để cấu hình tên cho RabbitMQ

```
package com.nts1.messageprocessor.config;

import model.rabbitmq.QueueName;
import org.springframework.amqp.core.*;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.annotation.Order;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.List;

@Configuration
@Order
public class RabbitMQProvision {

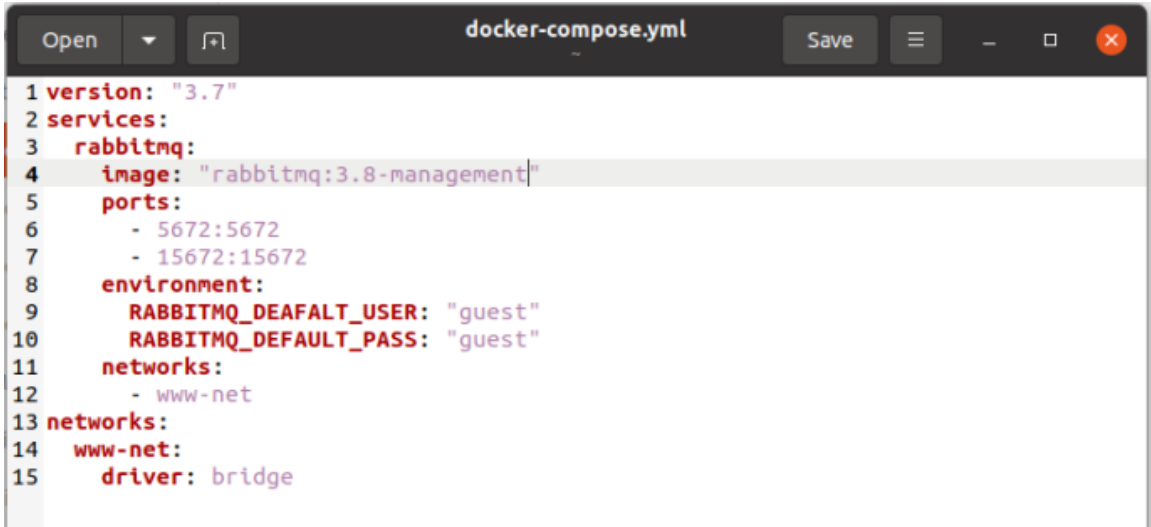
    public static final String DEADLETTER_SUFFIX = ".deadletter";
    public static final String PARKINGLOT_SUFFIX = ".parkinglot";

    @Bean
    public Declarables basicQueuesAndParkingLots() {
        List<Declarable> declarables = new ArrayList<>();
        declarables.addAll(basicQueueAndParkingLot(QueueName.TRACKER_MESSAGE_MAIN));
        declarables.addAll(basicQueueAndParkingLot(QueueName.TIMESERIES_MESSAGE_MAIN));
        return new Declarables(declarables.toArray(new Declarable[0]));
    }

    private Collection<? extends Declarable> basicQueueAndParkingLot(String queueName) {
        FanoutExchange fanout = new FanoutExchange( name: queueName + DEADLETTER_SUFFIX);
        Queue mainQueue = QueueBuilder.durable(queueName)
            .withArgument("x-dead-letter-exchange", queueName + DEADLETTER_SUFFIX).build();
        Queue parkingLot = QueueBuilder.durable(queueName + PARKINGLOT_SUFFIX).build();
        return Arrays.asList(fanout, mainQueue, parkingLot, BindingBuilder.bind(parkingLot).to(fanout));
    }
}
```

Hình 3.5: Class RabbitMQProvision để tạo queue trên RabbitMQ

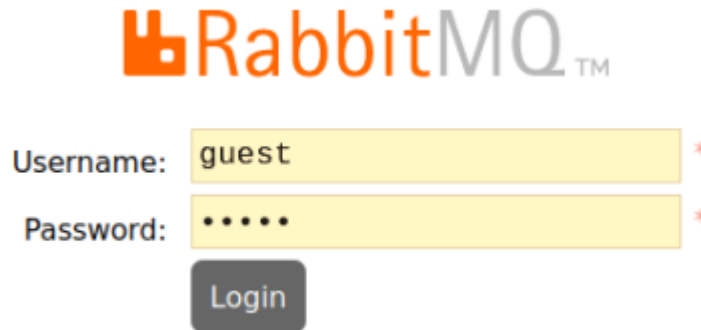
Cấu hình RabbitMQ để tạo Queue



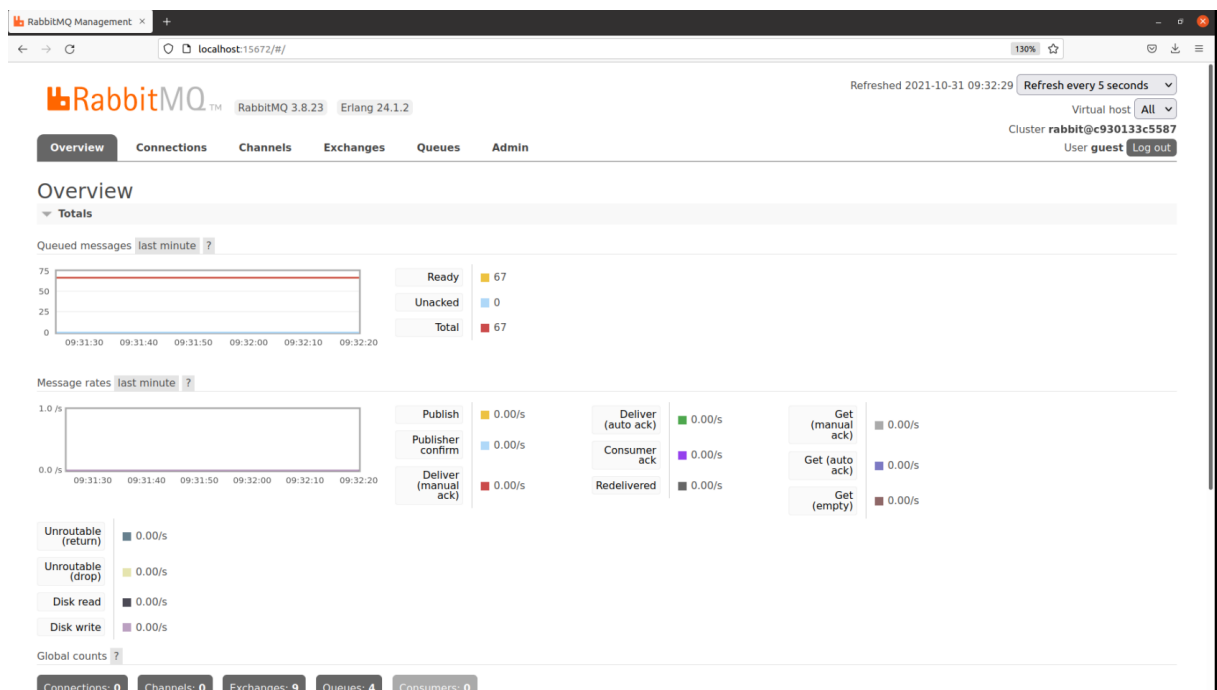
```
1 version: "3.7"
2 services:
3   rabbitmq:
4     image: "rabbitmq:3.8-management"
5     ports:
6       - 5672:5672
7       - 15672:15672
8     environment:
9       RABBITMQ_DEFAULT_USER: "guest"
10      RABBITMQ_DEFAULT_PASS: "guest"
11     networks:
12       - www-net
13 networks:
14   www-net:
15     driver: bridge
```

Hình 3.6: File Docker-compose để triển khai RabbitMQ

File Docker-compose.yml để triển khai giao diện của RabbitMQ lên web để chúng ta có thể dễ dàng theo dõi quá trình làm việc giữa các service và RabbitMQ khi service thực hiện upload file.



Hình 3.7: Giao diện đăng nhập của RabbitMQ trên web
Giao diện đăng nhập của RabbitMQ



Hình 3.8: Giao diện chính của RabbitMQ
Giao diện chính của RabbitMQ khi chúng ta đăng nhập thành công.

Overview				Messages			Message rates			+/-
Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack	
message.main	classic	D DLX	idle	0	0	0	0.00/s	0.00/s	0.00/s	
message.main.parkinglot	classic	D	idle	0	0	0				
timeseries.main	classic	D DLX	idle	0	0	0	0.00/s	0.00/s	0.00/s	
timeseries.main.parkinglot	classic	D	idle	67	0	67				

Hình 3.9: Giao diện các QueueName đã được cấu hình

Khi chúng ta start service thì Queue sẽ được tạo ra đúng với những thông tin mà chúng ta đã cấu hình.

```
server.port=8081

spring.rabbitmq.host=localhost
spring.rabbitmq.port=5672
spring.rabbitmq.username=guest
spring.rabbitmq.password=guest
```

Hình 3.10: Cấu hình để kết nối SpringBoot với RabbitMQ

File application.properties với các thông tin cấu hình của Gateway service, như:

- Port: 8081 là cổng mà service sẽ chạy.
- Các thông số host, port, username, password là các thông số mà service kết nối với RabbitMQ.

```
package com.nts1.gpxgateway.controller;

import com.nts1.gpxgateway.service.GpxFileService;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.multipart.MultipartFile;

@RestController
public class GPXGateController {
    private final GpxFileService gpxFileService;

    public GPXGateController(GpxFileService gpxFileService) { this.gpxFileService = gpxFileService; }

    @PostMapping(value = "/upload", produces = MediaType.APPLICATION_JSON_VALUE, consumes = MediaType.MULTIPART_FORM_DATA_VALUE)
    public ResponseEntity<?> upload(@RequestParam(value = "file") MultipartFile file, @RequestParam(value = "clientId") String clientId, @RequestParam(value = "clientSecret") String clientSecret){
        return gpxFileService.handle(file, clientId, clientSecret);
    }
}
```

Hình 3.11: Class GPXGateController với chức năng Upload File GPX

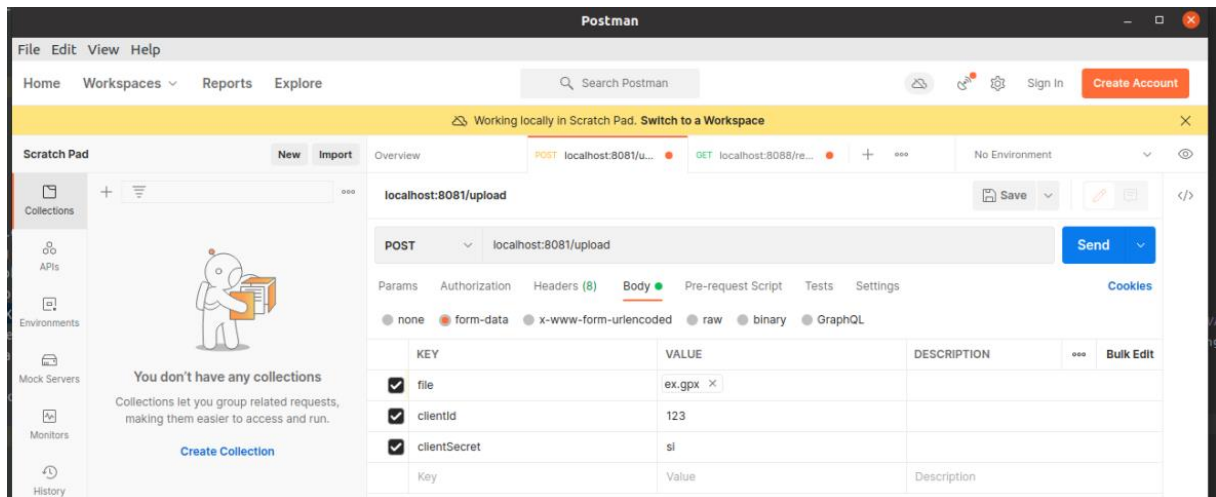
GPXGateController với chức năng upload file GPX.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>

<gpx xmlns="http://www.topografix.com/GPX/1/1" xmlns:gpox="http://-
www.garmin.com/xmlschemas/GpxExtensions/v3" xmlns:gpstpx="http://-
www.garmin.com/xmlschemas/TrackPointExtension/v1" creator="Oregon 400t"
version="1.1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.topografix.com/GPX/1/1 http://-
www.topografix.com/GPX/1/1/gpx.xsd http://www.garmin.com/xmlschemas/-
GpxExtensions/v3 http://www.garmin.com/xmlschemas/GpxExtensionsv3.xsd http://-
www.garmin.com/xmlschemas/TrackPointExtension/v1 http://www.garmin.com/-
xmlschemas/TrackPointExtensionv1.xsd">
  <metadata>
    <link href="http://www.garmin.com">
      <text>Garmin International</text>
    </link>
    <time>2009-10-17T22:58:43Z</time>
  </metadata>
  <trk>
    <name>Example GPX Document</name>
    <trkseg>
      <trkpt lat="47.644548" lon="-122.326897">
        <ele>4.46</ele>
        <time>2009-10-17T18:37:26Z</time>
      </trkpt>
      <trkpt lat="47.644548" lon="-122.326897">
        <ele>4.94</ele>
        <time>2009-10-17T18:37:31Z</time>
      </trkpt>
      <trkpt lat="47.644548" lon="-122.326897">
        <ele>6.87</ele>
        <time>2009-10-17T18:37:34Z</time>
      </trkpt>
    </trkseg>
  </trk>
</gpx>
```

Hình 3.12: File GPX

GPX file.



Hình 3.13: Giao diện chính của PostMan với chức năng upload file GPX

Khi service bắt đầu khởi động và chúng ta có file GPX thì chúng ta có thể truy cập vào Postman để có thể upload file GPX với method POST và localhost:8081/upload với các thông số file, clientId, clientSecret giống với GPXGateController chúng ta đã cấu hình.

```
public class GpxFileService {
    private static Logger log = LogManager.getLogger(GpxFileService.class);

    private AmqpTemplate amqpTemplate;

    public GpxFileService(AmqpTemplate amqpTemplate) { this.amqpTemplate = amqpTemplate; }

    public ResponseEntity<?> handle(MultipartFile multipartFile, String clientId, String clientSecret){
        try (InputStream inputStream = new ByteArrayInputStream(multipartFile.getBytes())) {
            GPX gpx = GPX.read(inputStream);

            model.db.GPX newGpx = new model.db.GPX(gpx);

            GPXRequest gpxRequest = new GPXRequest(newGpx, clientId, clientSecret, multipartFile.getOriginalFilename());

            GpxMessageData gpxMessageData = new GpxMessageData();
            gpxMessageData.setTimestamp(LocalDate.now());
            gpxMessageData.setTrackerDataID("1");
            gpxMessageData.setGpxRequest(gpxRequest);

            TrackerMessageInfo trackerMessageInfo = new TrackerMessageInfo();
            trackerMessageInfo.setImei("123");
            trackerMessageInfo.setTrackerMessageData(gpxMessageData);

            log.info( message: "Receive gpx file request {} from {}", gpxRequest, clientId);
            amqpTemplate.convertAndSend(QueueName.TRACKER_MESSAGE_MAIN, trackerMessageInfo);
            log.info( message: "Queued gpx file request {}", gpxRequest);

            return new ResponseEntity<>(HttpStatus.ACCEPTED);
        } catch (IOException e) {
            log.error( message: "IO exception occurs when handling the gpx file", e);
            return new ResponseEntity<>({ body: "IO exception occurs when handling the gpx file", HttpStatus.INTERNAL_SERVER_ERROR});
        } catch (Exception e) {
            log.error( message: "Unexpected exception occurs when handling the gpx file", e);
            return new ResponseEntity<>({ body: "Unexpected exception occurs when handling the gpx file", HttpStatus.INTERNAL_SERVER_ERROR});
        }
    }
}
```

Hình 3.14: Class *GpxFileService* với chức năng gửi dữ liệu đến RabbitMQ

Khi File được upload lên *GpxFileService* sẽ đọc tất cả các thông tin của File và sẽ chuyển các thông tin này lên RabbitMQ với tên *TRACKER_MESSAGE_MAIN*.

4.2.2. Chức năng trả dữ liệu về cho người dùng

Theo như sơ đồ cấu trúc và sơ đồ tuần tự chúng ta có, thì khi Gateway service upload 1 file GPX và gửi dữ liệu lên RabbitMQ. Khi đó Message Processor service sẽ gọi đến RabbitMQ và nhận những dữ liệu mà Gateway service đã gửi lên.

```
package com.nts1.messageprocessor.config;

import com.nts1.messageprocessor.service.consumer.TrackerMessageConsumer;
import model.rabbitmq.QueueName;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.springframework.amqp.rabbit.connection.ConnectionFactory;
import org.springframework.amqp.rabbit.listener.SimpleMessageListenerContainer;
import org.springframework.amqp.rabbit.listener.adapter.MessageListenerAdapter;
import org.springframework.amqp.support.converter.MessageConverter;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import util.RabbitMQMessageConverter;

@Configuration
public class RabbitMQConfiguration {

    private static Logger log = LogManager.getLogger(RabbitMQConfiguration.class);

    @Bean
    public MessageConverter jsonMessageConverter() { return new RabbitMQMessageConverter(); }

    @Bean
    MessageListenerAdapter trackerMessageListenerAdapter(TrackerMessageConsumer trackerMessageConsumer) {
        MessageListenerAdapter adapter = new MessageListenerAdapter(trackerMessageConsumer, defaultListenerMethod: "consume");
        adapter.setMessageConverter(jsonMessageConverter());
        return adapter;
    }

    @Bean
    SimpleMessageListenerContainer trackerMessageContainer(ConnectionFactory connectionFactory,
        MessageListenerAdapter trackerMessageListenerAdapter) {
        SimpleMessageListenerContainer container = new SimpleMessageListenerContainer();
        container.setConnectionFactory(connectionFactory);
        container.setQueueNames(QueueName.TRACKER_MESSAGE_MAIN);
        container.setMessageListener(trackerMessageListenerAdapter);
        log.info("Message: \"Listening on {}\", QueueName.TRACKER_MESSAGE_MAIN);
        return container;
    }
}
```

Hình 3.15: Class *RabbitMQConfiguration* cấu hình để nhận dữ liệu từ RabbitMQ

File *RabbitMQConfiguration* được cấu hình để Message Processor service có thể lắng nghe RabbitMQ ở Queue *TRACKER_MESSAGE_MAIN*.


```
package com.nts1.messageprocessor.service.consumer;

import ...

@Service
public class TrackerMessageConsumer {
    private static final Logger log = LogManager.getLogger(TrackerMessageConsumer.class);

    private final TrackerService trackerService;

    private final MessageHandlerManager messageHandlerManager;

    public TrackerMessageConsumer(TrackerService trackerService, MessageHandlerManager messageHandlerManager) {
        this.trackerService = trackerService;
        this.messageHandlerManager = messageHandlerManager;
    }

    public void consume(TrackerMessageInfo trackerMessageInfo){
        try {
            log.info( message: "Starting processing a tracker message info {}", trackerMessageInfo);
            Tracker tracker = trackerService.findByImei(trackerMessageInfo.getImei());
            if (tracker == null){
                log.warn( message: "Could not found tracker Imei {}", trackerMessageInfo.getImei());
                return;
            }
            TrackerMessage trackerMessage = new TrackerMessage(tracker, trackerMessageInfo.getTrackerMessageData());
            messageHandlerManager.handle(trackerMessage);
        }
        catch (Exception e){
            log.error( message: "Exception occurs when handling the message: {}.", trackerMessageInfo, e);
            throw new AmqpRejectAndDontRequeueException("Rejecting the message. " + trackerMessageInfo);
        }
    }
}
```

Hình 3.16: Class *TrackerMessageConsumer* với chức năng nhận dữ liệu từ RabbitMQ

File *TrackerMessageConsumer* có chức năng nhận dữ liệu mà Gateway service đã gửi lên RabbitMQ.

Sau khi mà service này nhận được dữ liệu thì service sẽ chuyển dữ liệu về dạng dữ liệu timeseries.

```
@Service("gpxMessageHandler")
public class GpxMessageHandler extends TrackerMessageHandler{
    private AmqpTemplate amqpTemplate;

    public GpxMessageHandler(AmqpTemplate amqpTemplate) { this.amqpTemplate = amqpTemplate; }

    @Override
    public void handle(TrackerMessage trackerMessage) {
        GpxMessageData gpxMessageDataData = (GpxMessageData) trackerMessage.getTrackerMessageData();

        GPXRequest gpxRequest = gpxMessageDataData.getGpxRequest();
        GPX gpxModel = gpxRequest.getGpxModel();
        List<Track> tracks = gpxModel.getTracks();
        if (CollectionUtils.isNotEmpty(tracks)) {
            tracks.forEach(track -> {
                List<TrackSegment> segments = track.getSegments();
                if (CollectionUtils.isNotEmpty(segments)){
                    segments.forEach(trackSegment -> {
                        List<WayPoint> points = trackSegment.getPoints();
                        if (CollectionUtils.isNotEmpty(points)){
                            points.forEach(wayPoint -> {
                                EnumMap<MetricType, String> values = new EnumMap<>(MetricType.class);

                                values.put(MetricType.TRACKER_DATA_GPS_LOCATION_LAT_LNG, String.valueOf(wayPoint.getLatitude()) + "," + wayPoint.getLongitude());
                                values.put(MetricType.TRACKER_DATA_GPS_ELEVATION, String.valueOf(wayPoint.getElevation()));
                                values.put(MetricType.TRACKER_DATA_GPS_SPEED, String.valueOf(wayPoint.getSpeed()));
                                values.put(MetricType.TRACKER_DATA_GPX_TIME, String.valueOf(wayPoint.getTime()));
                                values.put(MetricType.TRACKER_DATA_GPS_NAME, wayPoint.getName());
                                values.put(MetricType.TRACKER_DATA_GPS_COMMENT, wayPoint.getComment());
                                values.put(MetricType.TRACKER_DATA_GPS_DESCRIPTION, wayPoint.getDescription());
                                values.put(MetricType.TRACKER_DATA_GPS_SOURCE, wayPoint.getSource());
                                values.put(MetricType.TRACKER_DATA_GPS_SYMBOL, wayPoint.getSymbol());
                                values.put(MetricType.TRACKER_DATA_GPS_TYPE, wayPoint.getType());

                                insertMultiple(values, gpxMessageDataData.getTimestamp(), trackerMessage);
                            });
                        }
                    });
                }
            });
        }
    }
}
```

Hình 3.17: Class *GpxMessageHandler* với chức năng chuyển dữ liệu thành dữ liệu *TimeSeries*

Sau khi chúng ta có dữ liệu timeseries thì chúng ta sẽ gửi tất cả các dữ liệu timeseries này lên RabbitMQ ở queue `TIMESERIES_MESSAGE_MAIN`.

```
@Service
public class TimeSeriesArchivingListener implements TimeSeriesListener {
    private static Logger log = LogManager.getLogger(RabbitMQConfiguration.class);

    private final TimeSeriesService timeSeriesService;
    private final AmqpTemplate amqpTemplate;

    public TimeSeriesArchivingListener(TimeSeriesService timeSeriesService, AmqpTemplate amqpTemplate) {
        this.timeSeriesService = timeSeriesService;
        this.amqpTemplate = amqpTemplate;
    }

    @PostConstruct
    public void init() { timeSeriesService.registerListener(this); }

    @Override
    public void onInsertMultiple(List<TimeSeriesDataItem> items, TrackerMessage trackerMessage) {
        if(CollectionUtils.isEmpty(items)){
            List<TimeSeriesDataItem> validTimeSeriesDataItems = new ArrayList<>();
            for (TimeSeriesDataItem timeSeriesDataItem : items){
                if(valid(timeSeriesDataItem)){
                    validTimeSeriesDataItems.add(timeSeriesDataItem);
                }else {
                    log.warn( message: "Restricted a time series item because it was invalid {}", timeSeriesDataItem);
                }
            }
            ArchivingTimeSeriesObject archivingTimeSeriesObject = new ArchivingTimeSeriesObject(validTimeSeriesDataItems);
            amqpTemplate.convertAndSend(QueueName.TIMESERIES_MESSAGE_MAIN, archivingTimeSeriesObject);
        }
    }
}
```

Hình 3.18: Class *TimeSeriesArchivingListener* với chức năng gửi dữ liệu *TimeSeries* đến *RabbitMQ*

Overview				Messages			Message rates				+/-
Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack		
message.main	classic	D DLX	idle	0	0	0	0.00/s	0.00/s	0.00/s		
message.main.parkinglot	classic	D	idle	0	0	0		0.00/s	0.00/s		
timeseries.main	classic	D DLX	idle	0	0	0	0.00/s	0.00/s	0.00/s		
timeseries.main.parkinglot	classic	D	idle	67	0	67					

Hình 3.19: Giao diện *Queue* sau khi được kết nối với các service

Giao diện *Queue* trên *RabbitMQ* mà service sẽ kết nối đến và gửi dữ liệu *timeseries*.

```
server.port=8083

spring.rabbitmq.host=localhost
spring.rabbitmq.port=5672
spring.rabbitmq.username=guest
spring.rabbitmq.password=guest
```

Hình 3.20: Cấu hình để *SpringBoot* kết nối với *RabbitMQ*

File `application.properties` sẽ cấu hình mà Message Processor service hoạt động trên cổng 8083 và các thông số cấu hình kết nối với RabbitMQ tương tự như Gateway service.

Sau khi Message Processor hoàn thành nhiệm vụ của mình là nhận dữ liệu từ RabbitMQ và chuyển dữ liệu nhận được sang dữ liệu timeseries và gửi vào RabbitMQ. Khi đó Timeseries Processor service sẽ bắt đầu khởi động và lắng nghe RabbitMQ ở `TIMESERIES_MESSAGE_MAIN`.

```
@Configuration
public class RabbitMQConfiguration {
    private static Logger log = LogManager.getLogger(RabbitMQConfiguration.class);

    @Bean
    public MessageConverter jsonMessageConverter() { return new RabbitMQMessageConverter(); }

    @Bean
    MessageListenerAdapter timeSeriesListenerAdapter(TimeSeriesConsumer timeSeriesConsumer){
        MessageListenerAdapter adapter = new MessageListenerAdapter(timeSeriesConsumer, defaultListenerMethod: "consume");
        adapter.setMessageConverter(jsonMessageConverter());
        return adapter;
    }

    @Bean
    SimpleMessageListenerContainer trackerMessageContainer(ConnectionFactory connectionFactory, MessageListenerAdapter timeSeriesListenerAdapter){
        SimpleMessageListenerContainer container = new SimpleMessageListenerContainer();
        container.setConnectionFactory(connectionFactory);
        container.setQueueNames(QueueName.TIMESERIES_MESSAGE_MAIN);
        container.setMessageListener(timeSeriesListenerAdapter);
        log.info("message: \"Listening on {}\", QueueName.TIMESERIES_MESSAGE_MAIN);
        return container;
    }
}
```

Hình 3.21: Class `RabbitMQConfiguration` cấu hình để nhận dữ liệu TimeSeries từ RabbitMQ

Sau khi kết nối với RabbitMQ và nhận được dữ liệu timeseries thành công, TimeSeries Processor service sẽ gửi tất cả các dữ liệu vào InfluxDB.

Để có thể quản lý được dữ liệu trong InfluxDB chúng ta phải vào được giao diện của InfluxDB để có thể dễ dàng quản lý dữ liệu.

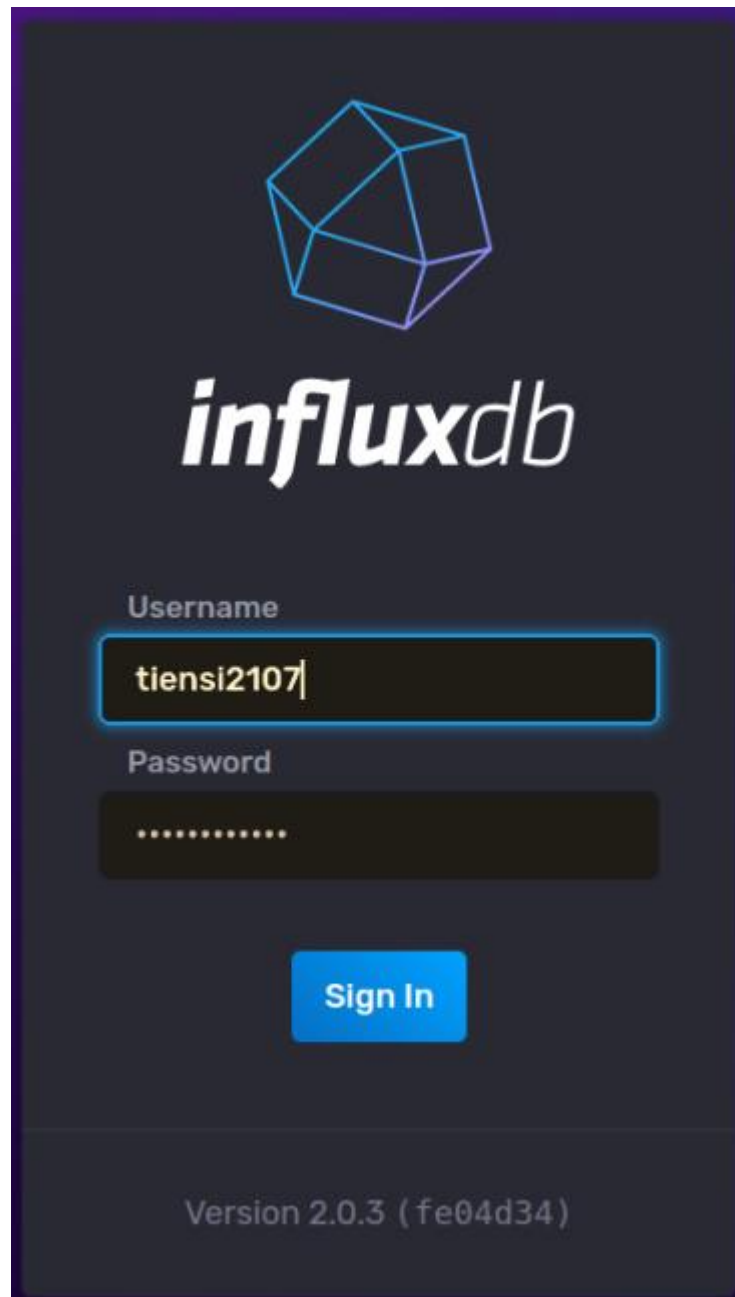
Khi đó, chúng ta thực hiện tải InfluxDB từ Docker Hub và thực hiện chạy InfluxDB để triển khai giao diện của InfluxDB

```
nts1@nts1:~$ docker ps
```

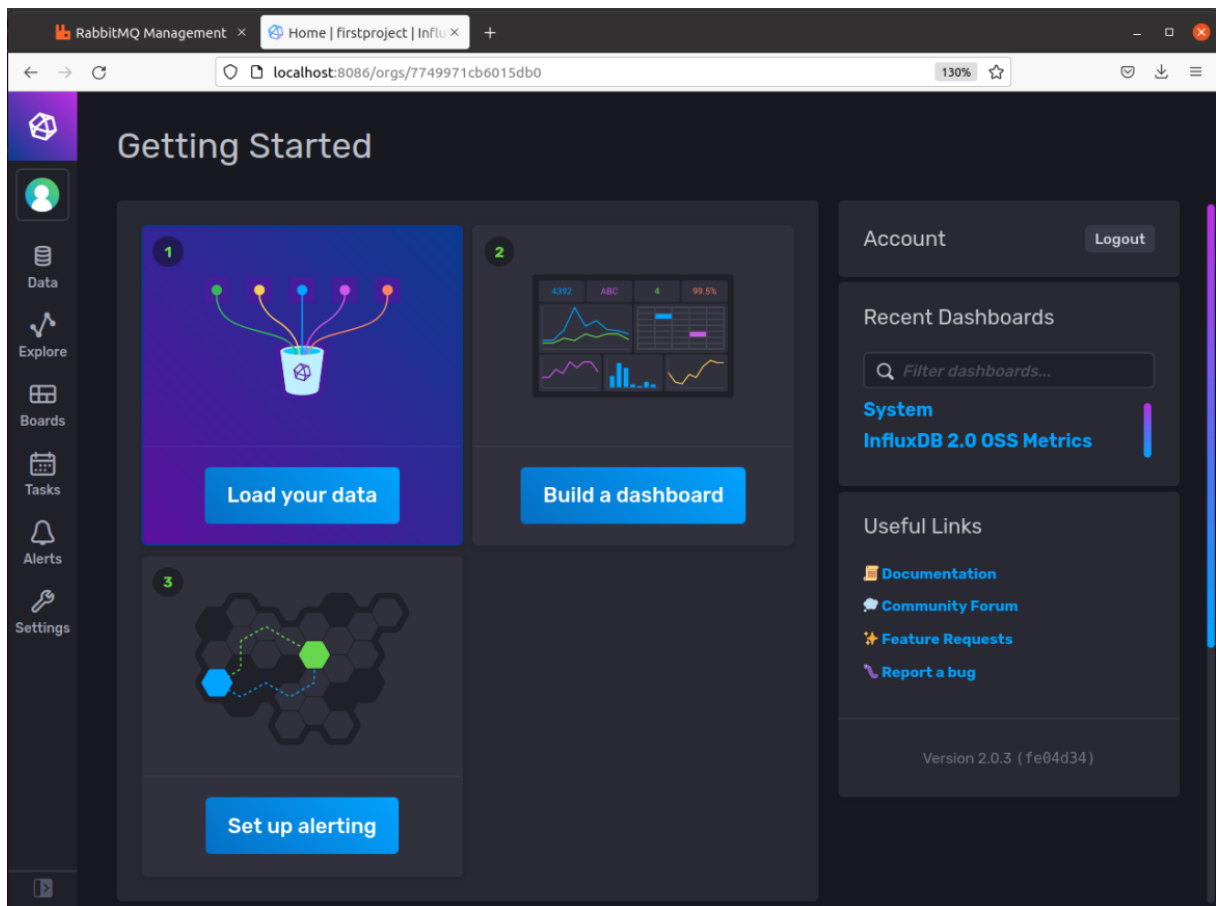
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
3cd83d5b35ec	quay.io/influxdb/influxdb:v2.0.3	"/entrypoint.sh infl..." influxdb	2 weeks ago	Up 2 weeks	0.0.0.0:8086->8086/tcp, :::8086->8086/tcp
c930133c5587	rabbitmq:3.8-management	"docker-entrypoint.s..." nts1_rabbitmq_1	3 weeks ago	Up 2 weeks	4369/tcp, 5671/tcp, 0.0.0.0:5672->5672/tcp, :::5672->5672/tcp, 15671/tcp, 15691-15692/tcp, 25672/tcp, 0.0.0.0:15672->15672/tcp, :::15672->15672/tcp

Hình 3.22: Hình ảnh 2 docker container đang được triển khai

Khi truy cập vào cổng 8086, chúng ta sẽ vào được giao diện đăng nhập của InfluxDB.



Hình 3.23: Giao diện đăng nhập của InfluxDB



Hình 3.24: Giao diện chính của InfluxDB

Sau khi đăng nhập thành công, đây là giao diện chính của InfluxDB.

Để có thể kết nối được với InfluxDB, chúng ta phải cấu hình TimeSeries Processor, như sau:

```
package com.nts.timeseriesprocessor.config.influxdb;

import com.influxdb.client.InfluxDBClient;
import com.influxdb.client.InfluxDBClientFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class influxFactory {
    @Value("${influxdb.host}")
    private String host;

    @Value("${influxdb.org}")
    private String org;

    @Value("${influxdb.bucket}")
    private String bucket;

    @Value("${influxdb.token}")
    private String token;

    @Bean
    public InfluxDBClient influxDBClient(){
        return InfluxDBClientFactory.create(host, token.toCharArray(), org, bucket);
    }
}
```

Hình 3.25: Class *InfluxFactory* cấu hình để *SpringBoot* nhận các thông số từ *InfluxDB*

Với *influxFactory* chúng ta cấu hình các *host*, *org*, *bucket* và *token* sau cho các thông số này phù hợp với trên *InfluxDB* UI là có thể kết nối với nhau.

Do *Timeseries Processor* service vừa kết nối với *RabbitMQ* vừa kết nối với *InfluxDB* nên ta cần cấu hình 2 file được *TimeSeries Processor* service hoạt động được.

```
influxdb:
  host: http://127.0.0.1:8086
  org: firstproject
  bucket: timeseries-processor
  token: fpvUCKlr26gmEP7m1zJkDD6wMT2f_Kzspk69rQtDdL-6MwLlknoiHWq3b1l35a0CKWbX64A7zyPy1UFCdn5rUg==
```

Hình 3.26: Cấu hình để kết nối với *InfluxDB*

File *application.yml* để cấu hình các thông số của *InfluxDB*.

```
server.port=8082

spring.rabbitmq.host=localhost
spring.rabbitmq.port=5672
spring.rabbitmq.username=guest
spring.rabbitmq.password=guest
```

Hình 3.27: Cấu hình để kết nối với *RabbitMQ*

File `application.properties` để cấu hình cổng TimeSeries Processor service hoạt động, cũng như các thông số kết nối với RabbitMQ tương tự như Gateway service và Message Processor service.

```
@Service
public class TimeSeriesConsumer {
    private static final Logger log = LogManager.getLogger(TimeSeriesConsumer.class);

    private final TimeSeriesService timeSeriesService;

    public TimeSeriesConsumer(TimeSeriesService timeSeriesService) { this.timeSeriesService = timeSeriesService; }
    public void consume(ArchivingTimeSeriesObject archivingTimeSeriesObject){
        try{
            log.info( message: "Starting processing a time-series data {}", archivingTimeSeriesObject);
            timeSeriesService.save(archivingTimeSeriesObject);
        }catch (Exception e){
            log.error( message: "Exception occurs when handling the time-series: {}. ", archivingTimeSeriesObject, e);
            throw new AmqpRejectAndDontRequeueException("Reject the time-series." + archivingTimeSeriesObject);
        }
    }
}
```

Hình 3.28: Class `TimeSeriesConsumer` với chức năng nhận dữ liệu `TimeSeries` từ `RabbitMQ`

`TimeSeriesConsumer` nơi mà `TimeSeries Processor service` nhận dữ liệu `timeseries` mà `Message Processor service` đã gửi vào `RabbitMQ` ở queue `TIMESERIES_MESSAGE_MAIN`.

```
@Measurement(name = "timeseries")
public class TimeSeriesData {

    @Column
    private String trackerID;

    @Column
    private MetricType metricType;

    @Column
    private Instant timestamp;

    @Column
    private Instant insertTime = LocalDateTime.now().toInstant(ZoneOffset.UTC);

    @Column
    private String metricValue;

    @Column
    private String trackerDataId;
```

Hình 3.29: Class `TimeSeries` với các thuộc tính được gửi vào `InflucDB`

TimeSeriesData là các dữ liệu mà TimeSeries Processor sẽ gửi vào InfluxDB.

```
@Repository
public class TimeSeriesDataRepository {

    private final InfluxDBClient influxDBClient;

    public TimeSeriesDataRepository(InfluxDBClient influxDBClient) { this.influxDBClient = influxDBClient; }

    public void save(List<TimeSeriesData> timeSeriesDataList){
        try (WriteApi writeApi = influxDBClient.getWriteApi()){
            writeApi.writeMeasurements(WritePrecision.NS, timeSeriesDataList);
        }
    }
}
```

Hình 3.30: Class TimeSeriesRepository với chức năng insert dữ liệu vào InfluxDB

Với TimeSeriesDataRepository các dữ liệu TimeSeriesData sẽ được insert vào cơ sở dữ liệu InfluxDB.

Sau khi dữ liệu đã được gửi vào cơ sở dữ liệu, khi người dùng cần lấy dữ liệu thì TimeSeries Provider sẽ có chức năng truy cập vào cơ sở dữ liệu và trả thông tin về cho người dùng.

Để có thể kết nối với InfluxDB thì TimeSeries Provider cũng phải thiết lập cấu hình tương tự như TimeSeries Processor service với các thông số như: host, org, bucket và token.

```
@Configuration
public class InfluxFactory {

    @Value("${influxdb.host}")
    private String host;

    @Value("${influxdb.org}")
    private String org;

    @Value("${influxdb.bucket}")
    private String bucket;

    @Value("${influxdb.token}")
    private String token;

    @Bean
    public InfluxDBClient influxDBClient(){
        return InfluxDBClientFactory.create(host, token.toCharArray(), org, bucket);
    }

    public String getHost() { return host; }

    public String getOrg() { return org; }

    public String getBucket() { return bucket; }

    public String getToken() { return token; }
}
```

Hình 3.31: Cấu hình để nhận các thông số từ InfluxDB


```
influxdb:
  host: http://127.0.0.1:8086
  org: firstproject
  bucket: timeseries-processor
  token: UyexKMB_En3x0HovdAF9nzKvj1nVa9ihvxecPkuWJptt5S1qEk-PRXHBjW11ZRw_ENB_urYo0Gib8PSjrfsy1Q==
```

Hình 3.32: Cấu hình để kết nối với InfluxDB

File application.yml thiết lập các thông số cần thiết để có thể kết nối.

```
@Repository
public class TimeSeriesDataRepository {

    private final InfluxDBClient influxDBClient;

    private final InfluxFactory influxFactory;

    public TimeSeriesDataRepository(InfluxDBClient influxDBClient, InfluxFactory influxFactory) {
        this.influxDBClient = influxDBClient;
        this.influxFactory = influxFactory;
    }

    public List<FluxTable> getLatestTrackerData(Long trackerID){
        String query = Flux.from(influxFactory.getBucket()) Flux
            .range(LocalDate.now().minusDays(50).toInstant(ZoneOffset.UTC), LocalDateTime.now().plusDays(5).toInstant(ZoneOffset.UTC)) RangeFlux
            .filter(Restrictions.and(
                Restrictions.measurement().equal("timeseries"),
                Restrictions.tag( tagName: "trackerID").equal(String.valueOf(trackerID)))) FilterFlux
            .toString();
        return influxDBClient.getQueryApi().query(query);
    }
}
```

Hình 3.33: Class TimeSeriesRepository với chức năng lấy dữ liệu từ InfluxDB

Với TimeSeriesDataRepository, TimeSeries Provider có thể thực hiện câu truy vấn và lấy dữ liệu để trả về cho người dùng.

```
@RestController
@RequestMapping("/rest/timeseries")
public class TimeSeriesController {

    private final TimeSeriesService timeSeriesService;

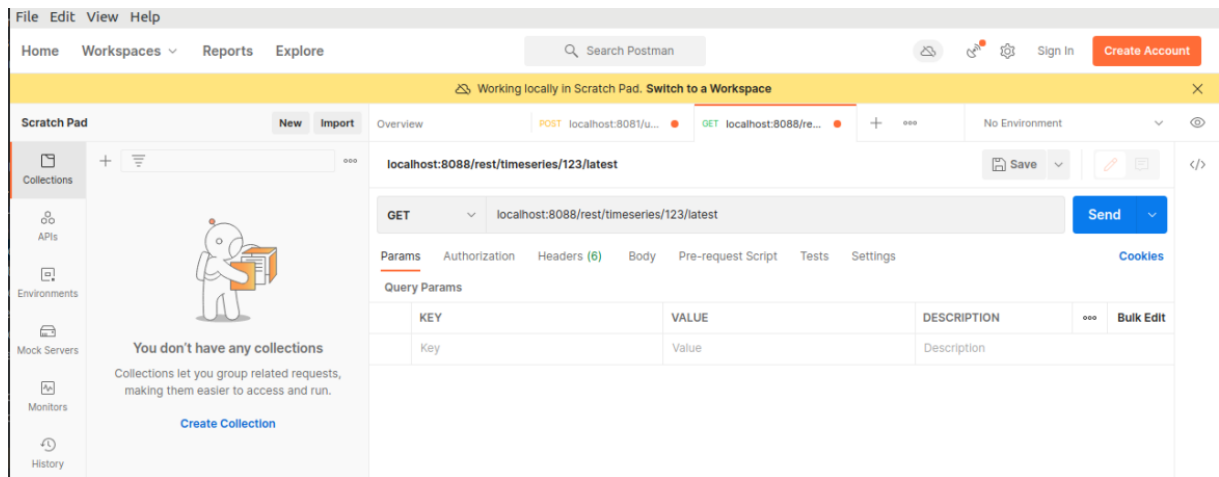
    public TimeSeriesController(TimeSeriesService timeSeriesService) { this.timeSeriesService = timeSeriesService; }

    private Gson getGson(){
        GsonBuilder builder = TrackerGson.getBuilder(TrackerGson.GsonAdapter.ISO_8601_NO_MILLI);
        return builder.create();
    }

    @GetMapping(path = "/{trackerID}/latest", produces = MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<?> getLatestTrackerData(@PathVariable Long trackerID){
        return new ResponseEntity<>(getGson().toJson(timeSeriesService.getLatestTrackerData(trackerID)), HttpStatus.OK);
    }
}
```

Hình 3.34: Class TimeSeriesController với chức năng trả dữ liệu đã lấy từ InfluxDB

Khi đó TimeSeriesController sẽ có chức năng hiển thị các thông tin dữ liệu trên Postman khi người dùng thực hiện phương thức GET.



Hình 3.35: Giao diện PostMan với chức năng trả dữ liệu về người dùng

Truy cập localhost:8088/rest/timeseries/{trackerID}/latest với method GET người dùng có thể nhận được các dữ liệu timeseries được cập nhật trong InfluxDB.

3.5. Kết Luận

Sau khi thiết kế và cài đặt thành công “Hệ thống lưu trữ và chia sẻ dữ liệu GPS dùng kiến trúc Microservices” thì chúng ta có thể trả lời cho câu hỏi số 1 ở phần đặt vấn đề ở Chương 1 mục 1.2

Chương 4: Kiểm thử và đánh giá

Chương 4 sẽ trình bày một cách chi tiết về quá trình kiểm thử và sau đó sẽ đưa ra những đánh giá của hệ thống sau khi kiểm thử.

4.1. Giới thiệu

Phát hiện lỗi và kiểm tra hệ thống có hoạt động theo đúng yêu cầu đã nêu ra trong đặc tả hay chưa

- Liệt kê kết quả có được sau khi kiểm thử
- Làm tài liệu cho giai đoạn bảo trì

4.2. Chi tiết kế hoạch kiểm thử

Quy trình kiểm thử được thực hiện qua các công đoạn:

- Kiểm thử thiết kế: Kiểm tra service có thiết kế 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 các lỗi xảy ra khi kiểm thử

4.3. Các chức năng được kiểm thử

- Upload file
- Lấy dữ liệu từ cơ sở dữ liệu

4.4. Quản lý kiểm thử

4.4.1. 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

4.4.2. Môi trường

Nền tảng phần cứng:

- Bộ vi xử lý: Intel Core i5
- Ram 16GB.
- HDD 1TB.

Phần mềm:

- Hệ điều hành Ubuntu 20.04.3 LTS
- Hệ cơ sở dữ liệu InfluxDB
- Ứng dụng Postman

4.4.3. Giao tiếp giữa các nhóm liên quan

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

4.4.4. Tài nguyên và sự cấp phát nhúng

Con người:

- Người kiểm thử phần mềm: Nguyễn Tiến Sĩ – B1704767

4.4.5. Thời gian kiểm thử

Phiên bản	Ngày bắt đầu	Ngày kết thúc
1.0	04/12/2021	04/12/2021

Bảng 4.1: Thời gian kiểm thử

4.4.6. Các rủi ro

Tên rủi ro	Mức độ	Kế hoạch
Thiếu nhận sự kiểm thử	Cao	Tăng số lượng kiểm thử
Kiểm thử không đúng tiến độ	Thấp	Tăng tiến độ kiểm thử
Kiểm thử không hiệu quả	Trung Bình	Tham khảo các nguồn tài liệu kiểm thử

Bảng 4.2: Các rủi ro


4.5. Kịch bản kiểm thử

Project Name	Ứng dụng Chia sẻ và lưu trữ GPS online
Reference Document	Tài liệu đặc tả ứng dụng Chia sẻ và lưu trữ GPS online
Created by	Nguyễn Tiến Sĩ
Date of creation	31/10/2021
Date of review	

Test scenario ID	Requirement - Reference document index	Test Scenario Description	Importance	No. of Test Cases
TS_01	Phần 5.1	Kiểm tra xem người dùng có thể upload GPX file được hay không	P1	1
TS_02	Phần 5.2	Kiểm tra xem người dùng có lấy được dữ liệu hay không	P1	1

Hình 4.1: Kịch bản kiểm thử

4.6. Các trường hợp kiểm thử

T.O.D	Description	Endpoint	HTTP method type	URL	BODY	SUCCESS RESPONSE	STATUS CODE	Result	Priority	Assignee
1	Upload GPX file	localhost:8081/upload	POST	localhost:8081/upload	{ "file": "ex.gpx", "clientId": "123", "clientSecret": "si" }	NA	202 Accepted	PASS	P1	Nguyen Tien Si
2	Get data	localhost:8088/rest/timeseries/{trackerID}/late	GET	localhost:8088/rest/timeseries/123/latest	NA		200 OK	PASS	P1	Nguyen Tien Si

Hình 4.2: Các trường hợp kiểm thử.

4.7. Đánh giá

- Chức năng upload GPX file hoạt động đúng như mong đợi
- Chức năng lấy dữ liệu từ cơ sở dữ liệu hoạt động đúng như mong đợi.

Chương 5: Các vấn đề cần quan tâm và hiệu quả của Microservices

Trong chương này, chúng tôi liệt kê các vấn đề cần quan trọng tâm khi phát triển phần mềm theo kiến trúc Microservices. Và sau đó chúng tôi đánh giá sự đạt được của hệ thống lưu trữ và chia sẻ GPS trực tuyến của chúng tôi vừa phát triển so với các vấn đề cần quan tâm này.

5.1. Các vấn đề cần quan tâm

Các vấn đề cần quan tâm khi tìm hiểu về kiến trúc Microservices:

- Thiết kế Microservices: Size, Scope, Capabilities.
- Message trong Microservices
 - + Tin nhắn đồng bộ (Synchronous Messaging)
 - + Tin nhắn bất đồng bộ (Asynchronous Messaging)
- Quản lý dữ liệu phân tán.
- Quản trị phân tán.
- Service Registry và Service Discovery
- Deployment
- Bảo mật
- Inter-Service/Process Communication
- Giao dịch
- Thiết kế khi gặp lỗi
- Microservices trong thiết kế doanh nghiệp hiện đại.
- Tích hợp Microservices.

5.2. Hiệu quả của Microservices

5.2.1. Dễ hiểu, dễ quản lý hệ thống

Mỗi Microservices trong kiến trúc chỉ thực hiện một chức năng duy nhất của hệ thống nên nó đòi hỏi ít mã code, dễ hiểu và dễ quản lý và đặc biệt sẽ có ít nguy cơ thay đổi.

5.2.2. Đáp ứng nhiều công nghệ khác nhau

Các Microservices trong hệ thống hoàn toàn độc lập với nhau nên có thể xây dựng mỗi Microservices trên một nền tảng khác nhau mà không ảnh hưởng đến hệ thống, do đó nó có thể lựa chọn công nghệ phù hợp nhất cho mỗi Microservices giúp tăng hiệu quả và chất lượng cho hệ thống. Đặc biệt các Microservices trong kiến trúc Microservices có thể được đẩy lên các máy chủ khác nhau, điều này giúp hệ thống hoạt động linh hoạt hơn rất nhiều đồng thời cũng cho thấy được sự khác biệt rõ ràng của Microservices so với các module.

5.2.3. Dễ dàng mở rộng quy mô

Các Microservices trong hệ thống hoạt động độc lập với nhau khi muốn mở rộng hệ thống ta chỉ việc chỉnh sửa các Microservices cần mở rộng và cập nhật phần cứng (nếu có) cho các Microservices này mà không ảnh hưởng đến hệ thống. Các Microservices khác vẫn chạy trên các nền tảng phần cứng nhỏ hơn. Điều này cho phép tiết kiệm chi phí, tận dụng mọi tài nguyên hiện có khi mở rộng quy mô cho hệ thống.

5.2.4. Tiết kiệm chi phí khi xây dựng và vận hành

Các Microservices có quy mô rất nhỏ giúp cho các nhà phát triển phần mềm dễ dàng xây dựng với chi phí thấp. Khả năng tái sử dụng và ít thay đổi của mỗi Microservices giúp giảm chi phí cho quá trình xây dựng và vận hành phần mềm.

5.3. Đánh giá sự đạt được của hệ thống lưu trữ và chia sẻ dữ liệu GPS

Sau khi phát triển hệ thống lưu trữ và chia sẻ dữ liệu GPS, hệ thống đã được được một số yêu cầu sau:

Các vấn đề cần quan tâm khi thiết kế Microservices	Hệ thống đạt được
Thiết kế Microservices: Size, Scope, Capabilities	Hệ thống được phân chia thành những service và mỗi service mang một chức năng cụ thể.
Message trong Microservices	Hệ thống sử dụng tính nhắn bất đồng bộ (Asynchronous Messaging) để truyền dữ liệu giữa các service.
Quản trị dữ liệu phân tán	Hệ thống sử dụng hệ quản trị cơ sở dữ liệu không quan hệ NoSQL – InfluxDB để lưu trữ những dữ liệu timeseries.
Deployment	Hệ thống sử dụng Docker để khởi chạy RabbitMQ, InfluxDB để dễ dàng cho việc triển khai.
Inter-Service/Process Communication	Hệ thống sử dụng Message Broker – RabbitMQ để giao tiếp giữa các service.
Có khả năng kiểm thử được	Hệ thống sử dụng PostMan để chạy cũng như kiểm thử các chức năng của hệ thống

Bảng 5.1: Đánh giá sự đạt được của hệ thống

5.4. Kết luận

Sau thiết kế, cài đặt thành công và hoàn tất kiểm thử “Hệ thống lưu trữ và chia sẻ dữ liệu GPS dùng kiến trúc Microservices”, chúng ta có thể rút ra được hiệu quả mà kiến trúc microservice mang lại và đánh giá sự đạt được của hệ thống để trả lời câu hỏi số 2 và số 3 đặt ra ở Chương 1 mục 1.2.

PHẦN III. KẾT LUẬN

1. Kết quả đạt được

1.1. Về kiến thức

Qua quá trình nghiên cứu và thực hiện đề tài giúp chúng em củng cố những kiến thức về các học phần mà các Thầy Cô đã truyền đạt như: Phân tích và thiết kế hệ thống thông tin, Ngôn ngữ mô hình hóa UML, Hệ cơ sở dữ liệu, Hệ quản trị cơ sở dữ liệu, lập trình web,... Đồng thời cũng giúp cho bản thân em nâng cao thêm các kỹ năng cứng và các kỹ năng mềm, và có kinh nghiệm trong việc tìm kiếm tra cứu tài liệu, viết báo cáo.

Nắm được kiến trúc tổng quan của Spring Framework cũng như các nguyên lý cơ bản và cơ chế hoạt động của framework này. Nắm được mô hình web MVC trong Spring framework và các cơ chế để bảo mật một ứng dụng web được hỗ trợ trong module Spring Security.

1.2. Về kỹ năng

Phát triển hệ thống lưu trữ và chia sẻ dữ liệu GPS với các chức năng cơ bản để thể hiện được mô hình Microservices được sử dụng trong hệ thống. Hệ thống có các chức năng căn bản, như:

- Cho phép người dùng upload file GPX.
- Cho phép người dùng có thể lưu trữ dữ liệu vào cơ sở dữ liệu InfluxDB.
- Cho phép người dùng xem chi tiết các dữ liệu từ file GPX đã upload.

Áp dụng các công nghệ mới vào hệ thống như: RabbitMQ, Docker.

Áp dụng framework Spring.

Áp dụng cơ sở dữ liệu InfluxDB.

1.3. Về thái độ

Thể hiện được sự nhiệt tình, trách nhiệm công việc, độ tin cậy, trung thực và tôn trọng.

2. Thuận lợi và khó khăn

2.1. Thuận lợi

Sự quan tâm, hỗ trợ nhiệt tình của các bộ nhiệt tình của cán bộ hướng dẫn và bạn bè.

2.2. Khó khăn

Do chưa có kinh nghiệm trong lập trình hệ thống theo hướng Microservices nên không tránh khỏi một số lỗi trong quá trình làm bài.

Tài liệu hướng dẫn còn tương đối ít.

3. Hạn chế của phần mềm

Hệ thống chia sẻ và lưu trữ dữ liệu GPS đã đáp ứng được những chức năng cơ bản trong việc thể hiện mô hình Microservices, song để phát triển hệ hống hơn nữa, hệ thống cần bổ sung và phát triển một số chức năng, như:

- Phát triển theo chức năng đăng nhập, đăng kí cho người dùng
- Mở rộng thêm số lượng file có thể upload không chỉ riêng GPX.

4. Hướng phát triển

Phát triển thêm giao diện cho hệ thống

Tích hợp hệ thống vào các devices/backend để hỗ trợ cho những giải pháp về IoT.

TÀI LIỆU THAM KHẢO

- [1] Huỳnh Xuân Hiệp, Phan Phương Lan (2011). Giáo trình Nhập môn công nghệ phần mềm, NXB Đại học Cần Thơ.
- [2] Huỳnh Xuân Hiệp, Phan Phương Lan (2014). Giáo trình Bảo trì phần mềm, NXB Đại học Cần Thơ.
- [3] Huỳnh Xuân Hiệp, Phan Phương Lan, Võ Huỳnh Trâm (2015). Giáo trình Kiến trúc và Thiết kế phần mềm, NXB Đại học Cần Thơ.
- [4] Huỳnh Xuân Hiệp, Phan Phương Lan, Võ Huỳnh Trâm (2015). Giáo trình Quản lý dự án phần mềm, NXB Đại học Cần Thơ.
- [5] Nguyễn Văn Linh (2010). Giáo trình Phân tích thiết kế thuật toán, Khoa Công nghệ thông tin và Truyền thông, Trường Đại học Cần Thơ.
- [6] Trần Cao Đệ, Đỗ Thanh Nghị (2012). Giáo trình Kiểm thử phần mềm, NXB Đại học Cần Thơ.
- [7] Trần Cao Đệ, Nguyễn Công Danh (2014). Giáo trình Đảm bảo chất lượng phần mềm, NXB Đại học Cần Thơ.
- [8] Võ Huỳnh Trâm (2009). Bài giảng Phân tích yêu cầu phần mềm, Khoa Công nghệ thông tin và Truyền thông, Trường Đại học Cần Thơ.
- [9] Kong, “What are Microservices”, <https://Microservices.io/>
- [10] Latest track, <https://www.trackprofler.com/track/index>
- [11] <https://www.docker.com/why-docker>
- [12] <https://www.rabbitmq.com/>
- [13] <https://www.javatpoint.com/>
- [14] <https://www.influxdata.com/>
- [15] <https://spring.io/>
- [16] Phát triển phần mềm dựa trên Microservices
<http://dlib.vnu.edu.vn/iii/cpro/DigitalItemPdfViewerPage.external?id=4258012940856088&itemId=1061862&lang=vie&file=%2Fiii%2Fcpro%2Fapp%3Fid%3D4258012940856088%26itemId%3D1061862%26lang%3Dvie%26nopassword%3Dtrue%26service%3Dblob%26suite%3Ddef#page=2&zoom=auto,-109,842&locale=vie>
- [17] <https://sites.google.com/cit.ctu.edu.vn/ncdanh/teaching?authuser=0>