

**TRƯỜNG ĐẠI HỌC CẦN THƠ
KHOA CÔNG NGHỆ THÔNG TIN & TRUYỀN THÔNG**



**LUẬN VĂN TỐT NGHIỆP ĐẠI HỌC
NGÀNH KỸ THUẬT PHẦN MỀM**

Đề tài

**PHÁT TRIỂN HỆ THỐNG CHO CỘNG ĐỒNG
NHỮNG NGƯỜI ĐI BỘ ĐƯỜNG DÀI DÙNG
KIẾN TRÚC MICROSERVICES**

**Sinh viên: Trần Công Minh
Mã số: B1704834 Khóa: K43**

Cần Thơ, 12/2021

TRƯỜNG ĐẠI HỌC CẦN THƠ
KHOA CÔNG NGHỆ THÔNG TIN & TRUYỀN THÔNG
BỘ MÔN CÔNG NGHỆ THÔNG TIN



LUẬN VĂN TỐT NGHIỆP ĐẠI HỌC
NGÀNH KỸ THUẬT PHẦN MỀM

Đề tài

**PHÁT TRIỂN HỆ THỐNG CHO CỘNG ĐỒNG
NHỮNG NGƯỜI ĐI BỘ ĐƯỜNG DÀI DÙNG
KIẾN TRÚC MICROSERVICES**

Người hướng dẫn

TS. Nguyễn Công Danh (CTU)
KS. Cao Quang Bình (TMA
Solution)

Sinh viên thực hiện

Trần Công Minh
Mã số: B1704834
Khóa: K43

Cần Thơ, 12/2021

TRƯỜNG ĐẠI HỌC CẦN THƠ
KHOA CNTT&TT

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
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 cho cộng đồng những người đi bộ đường dài
dùng kiến trúc Microservice

Họ tên sinh viên: Trần Công Minh

MSSV: B1704834

Mã lớp: DI1796A2

Đã báo cáo tại hội đồng ngành:

Ngày báo cáo:

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

Cần Thơ, ngày tháng năm 20

Giáo viên hướng dẫn

(Ký và ghi họ tên)

LỜI CẢM ƠN

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

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

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

Cần Thơ, ngày 17 tháng 12 năm 2021
Người Viết

Trần Công Minh

LỜI CAM ĐOAN

Em xin cam đoan luận văn “Phát triển hệ thống cho cộng đồng những người đi bộ đường dài dùng kiến trúc Microservice” được hoàn thành hoàn toàn dựa trên kết quả nghiên cứu của em dưới sự hướng dẫn của TS. Nguyễn Công Danh, KS. Cao Quang Bình, các nguồn tài liệu tham khảo đã được chỉ rõ trong danh mục tài liệu tham khảo.

Cần Thơ, ngày 17 tháng 12 năm 2021
Người viết

Trần Công Minh

MỤC LỤC

MỤC LỤC	ii
DANH MỤC CÁC KÝ HIỆU VÀ CHỮ VIẾT TẮT	vi
DANH MỤC CÁC BẢNG	vi
DANH MỤC CÁC HÌNH ẢNH	ix
TÓM TẮT	xi
ABSTRACT	xii
PHẦN GIỚI THIỆU	1
1. Đặt vấn đề	1
2. Lịch sử giải quyết vấn đề	1
3. Mục tiêu đề tài	1
4. Đối tượng và phạm vi nghiên cứu	2
5. Nội dung nghiên cứu	2
6. Những đóng góp chính của đề tài	2
7. Bố cục của quyển luận văn	2
PHẦN NỘI DUNG	4
CHƯƠNG 1: MÔ TẢ BÀI TOÁN	4
1.1. Yêu cầu cho ứng dụng Go-Hiking	4
1.1.1. Mô tả chi tiết bài toán	4
1.1.2. Tiếp cận giải quyết vấn đề	4
1.2. Yêu cầu phát triển và yêu cầu nghiên cứu	17
1.2.1. Yêu cầu phát triển	17
1.2.2. Yêu cầu nghiên cứu	17
CHƯƠNG 2: KIẾN THỨC NỀN	18
1.1. Tổng quan về Microservices	18
2.1. Định nghĩa về Microservices	18
2.2. Đặc điểm của Microservices	19
2.3. So sánh Monoliths với Microservices	20
2.4. Mẫu thiết kế của kiến trúc Microservices	21
2.4.1. Kiến trúc vi mô và vĩ mô	21
2.4.2. Kiến trúc hệ thống khép kín	21
2.4.3. Kiến trúc tích hợp Front-End	22
2.4.4. Kiến trúc Microservices không đồng bộ	23
2.4.5. Nền tảng Microservices	24
2.5. Spring	26

2.6. Spring-boot	27
2.7. Angular	27
2.8. MongoDB	27
2.9. Docker	28
2.10. HERE Map	28
2.11. Cloudinary	28
2.12. Nginx	28
CHƯƠNG 3: CÁC VẤN ĐỀ CẦN QUAN TÂM KHI PHÁT TRIỂN PHẦN MỀM THEO KIẾN TRÚC MICROSERVICES	29
3.1. Mô hình hoá các microservices.....	29
3.1.1. Xác định các Microservices.....	29
3.1.2. Thiết kế kiến trúc	29
3.1.3. Cơ chế kết nối	31
3.1.4. Xây dựng cổng API	32
3.2. Triển khai	32
3.2.1. Đóng gói các Microservices	32
3.2.2. Cấu hình cho máy chủ.....	32
3.3. Kiểm thử trong Microservices	32
3.3.1. Kiểm thử đơn vị	33
3.3.2. Kiểm thử dịch vụ	33
3.3.3. Kiểm thử đầu cuối	33
3.3.4. Kiểm thử tích hợp	33
3.4. Sử dụng kiến trúc Microservices khi nào là hợp lý.....	33
3.5. Các vấn đề nên lưu ý khi thiết kế Microservices	34
3.5.1. Hiểu sai về Microservices.....	34
3.5.2. Những điều cần phải tuân thủ	34
CHƯƠNG 4: XÂY DỰNG ỨNG DỤNG GO-HIKING.....	35
4.1. Tổng quan hệ thống.....	35
4.2. Kiến trúc hệ thống	35
4.3. Thiết kế dữ liệu	37
4.3.1. Doanh mục các bảng	37
4.3.2. Các kiểu dữ liệu liên lạc giữa các service.....	39
4.4. Thiết kế theo chức năng.....	41
4.4.1. Auth Service	41
4.4.2. Post Service	56

4.4.3. Geo Service	67
CHƯƠNG 5: KIỂM THỬ VÀ ĐÁNH GIÁ KẾT QUẢ	70
5.1. Giới thiệu	70
5.1.1. Mục tiêu	70
5.1.2. Phạm vi kiểm thử	70
5.2. Kế hoạch kiểm thử.....	70
5.2.1. Các tính năng sẽ được kiểm thử.....	70
5.2.2. Các tính năng không được kiểm thử	70
5.2.3. Cách tiếp cận	70
5.2.4. Tiêu chí kiểm thử thành công / thất bại	70
5.2.5. Tiêu chí đình chỉ và yêu cầu bắt đầu lại.....	71
5.3. Quản lý kiểm thử	71
5.3.1. Các hoạt động/công việc được lập kế hoạch, sự tiến hành kiểm thử	71
5.3.2. Môi trường	71
5.3.3. Trách nhiệm quyền hạn.....	71
5.3.4. Giao tiếp giữa các nhóm liên quan	71
5.3.5. Tài nguyên và sự cấp phát chúng.....	71
5.3.6. Huấn luyện.....	71
5.3.7. Kế hoạch dự đoán và chi phí	71
5.3.8. Các rủi ro	72
5.3.9. Kịch bản kiểm thử	72
5.4. Các trường hợp kiểm thử	73
5.4.1. Đăng ký.....	73
5.4.2. Đăng nhập (lấy access token).....	74
5.4.3. Tạo bài viết	75
5.4.4. Tạo địa điểm.....	76
5.4.5. Tạo bình luận.....	77
5.4.6. Lấy địa điểm.....	79
CHƯƠNG 6: ĐÁNH GIÁ MICROSERVICES	81
6.1. So sánh	81
6.1.1. So sánh chức năng	81
6.1.2. So sánh thời gian phản hồi.....	81
6.1.3. So sánh vận hành	82
6.2. Ưu điểm.....	82
6.3. Nhược điểm	83

PHẦN 3. KẾT LUẬN	84
1. KẾT QUẢ ĐẠT ĐƯỢC	84
1.1. Về lý thuyết và công nghệ	84
1.2. Về website	84
1.3. Hạn chế	84
2. HƯỚNG PHÁT TRIỂN	84
TÀI LIỆU THAM KHẢO	85
PHỤ LỤC	86
1. Khởi chạy hệ thống	86
2. Sử dụng trang web.....	86

DANH MỤC CÁC KÝ HIỆU VÀ CHỮ VIẾT TẮT

Thuật ngữ, khái niệm/từ viết tắt	Định nghĩa
HTTP	Hypertext Transfer Protocol
REST	Representational State Transfer
API	Application Programming Interface
URL	Uniform Resource Locator

DANH MỤC CÁC BẢNG

Bảng 1-1: Bảng phân chia quyền sử dụng	6
Bảng 1-2: Bảng usecase đăng ký	9
Bảng 1-3: Bảng usecase đăng nhập (lấy authorization token)	9
Bảng 1-4: Bảng usecase đăng bài viết	10
Bảng 1-5: Bảng usecase xóa bài viết.....	10
Bảng 1-6: Bảng usecase thêm hoặc sửa địa điểm bài viết.....	11
Bảng 1-7: Bảng usecase xóa địa điểm khỏi bài viết.....	11
Bảng 1-8: Bảng usecase xem bài viết	12
Bảng 1-9: Bảng usecase viết bình luận.....	12
Bảng 1-10: Bảng usecase xem trang cá nhân.....	13
Bảng 1-11: Bảng usecase cập nhật thông tin cá nhân	13
Bảng 1-12: Bảng usecase tìm người dùng.....	14
Bảng 1-13: Bảng usecase gửi kết bạn	14
Bảng 1-14: Bảng usecase trả lời kết bạn.....	15
Bảng 1-15: Bảng usecase xóa bạn.....	15
Bảng 1-16: Bảng usecase tra cứu địa điểm	16
Bảng 2-1: Bảng so sánh Microservices và monolith.....	20
Bảng 4-1: Các bảng của hệ thống.....	37
Bảng 4-2: Bảng user.....	38
Bảng 4-3: Bảng friend.....	38
Bảng 4-4: Bảng request.....	38
Bảng 4-5: Bảng post.....	38
Bảng 4-6: Bảng location	39
Bảng 4-7: Bảng comment	39
Bảng 4-8: Bảng geoData.....	39
Bảng 4-9: Chức năng đăng ký.....	41
Bảng 4-10: Chức năng đăng nhập (lấy access token)	42
Bảng 4-11: Chức năng huỷ token	44
Bảng 4-12: Chức năng cập nhật thông tin cá nhân.....	45
Bảng 4-13: Chức năng lấy thông tin cá nhân	46
Bảng 4-14: Chức năng lấy thông tin người dùng	48
Bảng 4-15: Chức năng tạo yêu cầu kết bạn.....	49
Bảng 4-16: Chức năng lấy yêu cầu kết bạn.....	51
Bảng 4-17: Chức năng trả lời yêu cầu kết bạn.....	52
Bảng 4-18: Chức năng lấy thông tin bạn bè.....	54
Bảng 4-19: Chức năng xoá bạn.....	55
Bảng 4-20: Chức năng upload ảnh	56
Bảng 4-21: Chức năng xoá ảnh.....	57
Bảng 4-22: Chức năng tạo bài viết	58
Bảng 4-23: Chức năng lấy bài viết.....	59
Bảng 4-24: Chức năng xoá bài viết	60
Bảng 4-25: Chức năng tạo hoặc sửa địa điểm trong bài viết	61
Bảng 4-26: Chức năng lấy địa điểm trong bài viết.....	62
Bảng 4-27: Chức năng xoá địa điểm trong bài viết.....	63
Bảng 4-28: Chức năng bình luận trong bài viết.....	64

Bảng 4-29: Chức năng lấy bình luận trong bài viết	65
Bảng 4-30: Chức năng xoá bình luận trong bài viết	66
Bảng 4-31: Chức năng tìm địa điểm dựa trên vị trí địa lý.....	67
Bảng 4-32: Chức năng lấy địa điểm đã lưu trên server	68
Bảng 5-1: Trách nhiệm quyền hạn	71
Bảng 5-2: Kế hoạch dự đoán và chi phí.....	72
Bảng 5-3: Các rủi ro	72
Bảng 5-4: Kịch bản kiểm thử	72
Bảng 5-5: Kiểm thử chức năng đăng ký.....	73
Bảng 5-6: Kiểm thử chức năng đăng nhập (lấy access token).....	74
Bảng 5-7: Kiểm thử chức năng tạo bài viết.....	75
Bảng 5-8: Kiểm thử chức năng tạo địa điểm	76
Bảng 5-9: Kiểm thử chức năng tạo bình luận	78
Bảng 5-10: Tìm địa điểm dựa trên vị trí địa lý	79

DANH MỤC CÁC HÌNH ẢNH

Hình 1-1: Chức năng của hệ thống.....	8
Hình 2-1: Mẫu kiến trúc SCS.....	22
Hình 2-2: Chia Frontend thành modules	23
Hình 2-3: Liên lạc không đồng bộ.....	24
Hình 2-4: Chức năng của nền tảng Microservices	25
Hình 2-5: Mẫu kiến trúc Microservices.....	26
Hình 4-1: Kiến trúc của Go-Hiking.....	36
Hình 4-2: Domain-Driven Design.....	37
Hình 4-3: Sơ đồ xử lý chức năng đăng ký.....	41
Hình 4-4: Chức năng đăng ký	42
Hình 4-5: Sơ đồ xử lý chức năng đăng nhập (lấy access token)	43
Hình 4-6: Chức năng đăng nhập (lấy access token).....	43
Hình 4-7: Sơ đồ xử lý chức năng huỷ token	44
Hình 4-8: Chức năng huỷ token.....	44
Hình 4-9: Sơ đồ xử lý chức năng cập nhật thông tin cá nhân.....	45
Hình 4-10: Chức năng cập nhật thông tin cá nhân	46
Hình 4-11: Sơ đồ xử lý chức năng lấy thông tin cá nhân	47
Hình 4-12: Chức năng lấy thông tin cá nhân	47
Hình 4-13: Sơ đồ xử lý chức năng lấy thông tin người dùng	48
Hình 4-14: Chức năng lấy thông tin người dùng.....	49
Hình 4-15: Sơ đồ xử lý chức năng tạo yêu cầu kết bạn.....	50
Hình 4-16: Chức năng tạo yêu cầu kết bạn	50
Hình 4-17: Sơ đồ xử lý chức năng lấy yêu cầu kết bạn.....	51
Hình 4-18: Chức năng lấy yêu cầu kết bạn	52
Hình 4-19: Sơ đồ xử lý chức năng trả lời yêu cầu kết bạn.....	53
Hình 4-20: Chức năng trả lời yêu cầu kết bạn	53
Hình 4-21: Sơ đồ xử lý chức năng lấy thông tin bạn bè	54
Hình 4-22: Chức năng lấy thông tin bạn bè	54
Hình 4-23: Sơ đồ xử lý chức năng xoá bạn.....	55
Hình 4-24: Chức năng xoá bạn	55
Hình 4-25: Sơ đồ xử lý chức năng upload ảnh	56
Hình 4-26: Sơ đồ xử lý chức năng xoá ảnh.....	57
Hình 4-27: Sơ đồ xử lý chức năng tạo bài viết	58
Hình 4-28: Chức năng tạo bài viết	58
Hình 4-29: Sơ đồ xử lý chức năng lấy bài viết	59
Hình 4-30: Chức năng lấy bài viết	59
Hình 4-31: Sơ đồ xử lý chức năng xoá bài viết	60
Hình 4-32: Chức năng xoá bài viết	61
Hình 4-33: Sơ đồ xử lý chức năng tạo và cập nhật địa điểm	61
Hình 4-34: Chức năng tạo và cập nhật địa điểm	62
Hình 4-35: Sơ đồ xử lý chức năng lấy địa điểm trong bài viết.....	62
Hình 4-36: Chức năng lấy địa điểm trong bài viết	63
Hình 4-37: Sơ đồ xử lý chức năng xoá địa điểm trong bài viết.....	63
Hình 4-38: Chức năng xoá địa điểm trong bài viết	64
Hình 4-39: Sơ đồ xử lý chức năng bình luận trong bài viết.....	64

Hình 4-40: Chức năng bình luận trong bài viết	65
Hình 4-41: Sơ đồ xử lý chức năng lấy bình luận trong bài viết.....	65
Hình 4-42: Chức năng lấy bình luận trong bài viết.....	66
Hình 4-43: Sơ đồ xử lý chức năng xoá bình luận trong bài viết.....	66
Hình 4-44: Chức năng xoá bình luận trong bài viết	67
Hình 4-45: Sơ đồ xử lý chức năng tìm địa điểm dựa trên vị trí địa lý.....	68
Hình 4-46: Sơ đồ xử lý chức năng lấy địa điểm đã lưu trên server	69

TÓM TẮT

Kiến trúc Microservices là một kiểu kiến trúc phần mềm mới và đang rất phát triển hiện nay. Trong đó các dịch vụ được chia nhỏ để thực hiện một chức năng duy nhất của hệ thống. Việc chia nhỏ các dịch vụ trong kiến trúc Microservices giúp cho hệ thống đơn giản hơn, dễ phát triển hơn, giảm chi phí xây dựng, tăng khả năng thích ứng công nghệ. Kiến trúc Microservices được coi là lời giải ưu việt cho bài toán xây dựng và phát triển hệ thống dựa trên dịch vụ hiện nay.

Xuất phát từ những ý nghĩa thực tiễn như vậy, tôi đã thực hiện đề tài luận văn “Phát triển hệ thống cho cộng đồng những người đi bộ đường dài dùng kiến trúc Microservice” để tìm hiểu và áp dụng kiến trúc Microservices trong việc xây dựng và phát triển một ứng dụng cụ thể - ứng dụng web cho cộng đồng Đi bộ đường dài (Go-Hiking) một dạng mạng xã hội cho phép người dùng chia sẻ địa điểm của họ, chia sẻ kinh nghiệm của họ và giới thiệu địa điểm an toàn để đi bộ đường dài.

ABSTRACT

Microservices architecture is a new and growing type of software architecture today, where services are broken down to perform a single function of the system. The breakdown of services in the Microservices architecture makes the system simpler, easier to develop, reduces construction costs, and increases technology adaptability. Microservices architecture is considered as the preeminent solution for the current problem of building and developing service-based systems.

Stemming from such practical meanings, we made the thesis topic "Developing a System for Go-Hiking Community using Microservice" to learn and apply Microservices architecture in building build and develop a specific application - web application for the Go-Hiking community a form of social network that allows users to share their locations, share their experiences and recommend safe place for hiking.

PHẦN GIỚI THIỆU

1. Đặt vấn đề

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 đang làm cho việc xây dựng và phát triển hệ thống phần mềm dựa trên dịch vụ đang vượt khỏi khả năng kiểm soát của các kiểu kiến trúc phần mềm hiện có và cần phải có một kiểu kiến trúc mới để giải quyết vấn đề này.

2. Lịch sử giải quyết vấn đề

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

3. Mục tiêu đề tài

Xuất phát từ những ý nghĩa thực tiễn như vậy, tôi đã thực hiện đề tài luận văn “Phát triển hệ thống cho cộng đồng những người đi bộ đường dài dùng kiến trúc Microservices” để tìm hiểu và áp dụng kiến trúc Microservices trong việc xây dựng và phát triển một ứng dụng cụ thể - ứng dụng web cho cộng đồng Đi bộ đường dài (Go-Hiking) một dạng mạng xã hội cho phép người dùng chia sẻ địa điểm của họ, chia sẻ kinh nghiệm của họ và giới thiệu địa điểm an toàn để đi bộ đường dài. Dựa trên việc áp dụng kiến trúc Microservices trong thực tế từ đó đưa ra các phân tích, đánh giá và rút ra các ưu nhược điểm của kiến trúc Microservices.

Trang web có vài chức năng cụ thể sau:

- Cho phép người dùng chia sẻ những địa điểm an toàn để đi bộ đường dài (lộ trình đi bộ đường dài trên bản đồ và ảnh đã chụp)
- Đề xuất địa điểm gần đó để đi bộ đường dài (danh sách địa điểm với đầy đủ thông tin cần thiết hoặc chế độ xem bản đồ có thể được chọn để hiển thị chi tiết)
- Mẹo cho chuyến đi bộ tiếp theo (danh sách chủ đề và kết nối có sẵn với tình nguyện viên địa phương, những người có thể hỗ trợ bạn trong trường hợp khẩn cấp)

- Một blog đơn giản mà những người đi bộ đường dài có thể chia sẻ kinh nghiệm và những người khác có thể đăng bình luận của họ và trả lời về những bình luận cụ thể.

4. Đối tượng và phạm vi nghiên cứu

- Microservices là gì?
- Ưu nhược điểm của kiến trúc Microservices.
- Quá trình phát triển phần mềm theo kiến trúc Microservices.

5. Nội dung nghiên cứu

- Tìm hiểu kiến trúc Microservices.
- Xác định các bước cần thiết để phát triển hệ thống dựa trên kiến trúc Microservices.
- Áp dụng kiến trúc Microservices trong việc phát triển một ứng dụng cụ thể.
- Đánh giá và rút ra các ưu nhược điểm của kiến trúc Microservices.

6. Những đóng góp chính của đề tài

- Kết quả nghiên cứu có thể làm tài liệu tham khảo.
- Phần nghiên cứu lý thuyết sẽ cung cấp một cách nhìn tổng quát về quá trình phát triển phần mềm theo kiến trúc Microservices, ưu nhược điểm và các điểm liên quan.
- Phát triển thành công một trang web sử dụng kiến trúc Microservices.

7. Bố cục của luận văn

Bố cục luận văn được xây dựng gồm 3 phần và 1 tài liệu tham khảo, 1 phụ lục:

- **Phần giới thiệu:** nêu ra các vấn đề cần giải quyết, lý do tại sao phải thực hiện đề tài này, trong quá khứ đã có các hệ thống, các website nào tương tự được xây dựng để giải quyết các vấn đề đã đặt ra và chưa giải quyết được những vấn đề nào, qua đó xác định được mục tiêu của đề tài, đó cũng chính là những vấn đề trọng tâm mà đề tài đang thực hiện muốn giải quyết cũng như xác định được đối tượng và phạm vi nghiên cứu của đề tài.
- **Phần nội dung:** Phần này được chia ra làm 6 chương như sau:
 - Chương 1: Chương này trình bày các yêu cầu cho ứng dụng cũng như các yêu cầu phát triển và yêu cầu nghiên cứu.
 - Chương 2: Chương này trình bày các kiến thức liên quan đến Microservices, đồng thời cũng giới thiệu sơ qua về các công nghệ sử dụng khi phát triển hệ thống.
 - Chương 3: Chương này trình bày các vấn đề cần quan tâm khi phát triển phần mềm theo kiến trúc microservices.
 - Chương 4: Chương này trình bày về các vấn đề thiết kế và cài đặt giải pháp gồm những nội dung như: thiết kế kiến trúc tổng thể của hệ thống, giải thích chức năng trong hệ thống, các lưu đồ chức năng của hệ thống, thiết kế cơ sở dữ liệu.
 - Chương 5: Kiểm thử và đánh giá, trong chương này sẽ mô tả mục tiêu kiểm thử, kịch bản kiểm thử và kết quả kiểm thử xem có chạy được và đúng như mong đợi hay không, có phát sinh lỗi ngoài dự đoán hay không nhằm kiểm soát và phát hiện ra các lỗi tiềm ẩn trong hệ thống và khắc phục, sửa chữa.

- Chương 6: Từ các kiến thức và thực tế phát triển hệ thống, chương này sẽ đánh giá kiến trúc Microservices, đưa ra các ưu nhược điểm của kiểu kiến trúc này.
- Phần kết luận: trình bày kết quả đã đạt được sau khi hoàn thành trang web, đưa ra kết quả đạt được, những tiêu chí, đánh giá mức độ hoàn thành và chưa hoàn thành cũng như những mặt hạn chế, những điều chưa làm được của hệ thống.
- Tài liệu tham khảo: Ghi chú các tài liệu đã tham khảo.
- Phụ lục: Hướng dẫn sử dụng web.

PHẦN NỘI DUNG

CHƯƠNG 1: MÔ TẢ BÀI TOÁN

Chương này trước tiên mô tả các yêu cầu phần mềm của hệ thống cho cộng đồng những người đi bộ đường dài được gọi là Go-Hiking. Sau đó, các yêu cầu phát triển và yêu cầu nghiên cứu được trình bày.

1.1. Yêu cầu cho ứng dụng Go-Hiking

1.1.1. Mô tả chi tiết bài toán

Đề tài “Phát triển hệ thống cho cộng đồng những người đi bộ đường dài dùng Kiến trúc Microservices” đáp ứng những yêu cầu sau:

- Admin: có quyền cao nhất trong hệ thống, quản lý toàn bộ thông tin tài khoản người dùng và cũng có đầy đủ các chức năng của web.
- Người dùng chưa có tài khoản:
 - Xem bài viết: xem các bài viết ở trạng thái công khai.
 - Xác định địa điểm: tìm địa điểm dựa trên kinh độ và vĩ độ.
 - Đăng ký tài khoản.
- Thành viên: người đã đăng ký tài khoản trên hệ thống, có những chức năng sau
 - Đăng nhập.
 - Tìm kiếm bạn bè: người dùng có thể tìm kiếm bạn bè để kết bạn.
 - Đăng bài viết: người dùng có thể tạo bài viết với 3 chế độ xem: công khai, bạn bè, chỉ mình tôi.
 - Kết bạn: chức năng này cho phép bạn có thể tìm kiếm và kết bạn với mọi người.
 - Xem bài viết của bạn bè, công khai, có thể bình luận vào các bài viết.
- Người dùng bên thứ ba là người dùng các api của hệ thống tùy trường hợp mà người dùng này có quyền hạn khác nhau.

1.1.2. Tiếp cận giải quyết vấn đề

1.1.2.1. Mô tả tổng quan

1.1.2.1.1. Bối cảnh sản phẩm

Hiking là gì? Nó là các hoạt động đi bộ đường dài. Bao gồm những hoạt động trong môi trường tự nhiên, thường là các vùng núi, địa danh, danh lam thắng cảnh hay các địa điểm đẹp. Mọi người thường đi bộ trên những con đường mòn. Đó là một hoạt động phổ biến đến mức có nhiều tổ chức đi bộ đường dài trên toàn thế giới.

Các nghiên cứu đã xác nhận lợi ích sức khỏe của go hiking. Như việc giảm cân, giảm huyết áp và cải thiện sức khỏe tim mạch, nâng cao tinh thần.

Ý tưởng của Go-Hiking là một trang web mà ở đó những người đi bộ đường dài có thể chia sẻ thông tin chuyến đi, kinh nghiệm khi thực hiện những chuyến đi bộ đường dài. Họ cũng có thể chia sẻ các hình ảnh ghi lại được trên các tuyến đường mà họ đi qua.

1.1.2.1.2. Các chức năng của sản phẩm

- Đối với admin
 - Đăng nhập.
 - Xem danh sách tài khoản: admin có thể xem danh sách toàn bộ tài khoản người dùng đã đăng ký.
 - Xem danh sách địa điểm: admin có thể xem danh sách các địa điểm lưu trên hệ thống.
- Đối với người dùng chưa có tài khoản
 - Xem bài viết ở chế độ công khai.
 - Tra cứu địa điểm dựa vào kinh độ và vĩ độ.
 - Đăng ký: cho phép người đăng ký tài khoản.
- Đối với người dùng đã đăng ký tài khoản
 - Đăng nhập: định danh người dùng và đăng nhập vào hệ thống để thực hiện các chức năng của người dùng hệ thống.
 - Tìm kiếm bạn bè: chức năng này cho phép người dùng có thể tìm kiếm bạn bè để kết bạn, chỉ có thể tìm thấy người dùng đặt trạng thái công khai cho trang cá nhân.
 - Xem trang cá nhân: mọi người có thể xem trang cá nhân của người dùng đặt trạng thái công khai cho trang cá nhân hoặc người dùng đã kết bạn.
 - Cập nhật thông tin cá nhân: Người dùng có thể chỉnh sửa thông tin cá nhân.
 - Gửi kết bạn: có thể gửi kết bạn cho người dùng đặt trạng thái công khai cho trang cá nhân.
 - Chấp nhận kết bạn: khi chấp nhận kết bạn sẽ trở thành bạn bè của nhau.
 - Xem bài viết: có thể xem các bài viết của bản thân, công khai hoặc của bạn bè.
 - Bình luận: có thể bình luận trên bài viết mà người dùng có thể xem.
 - Xóa bạn: chức năng này áp dụng cho trường hợp cả hai là bạn bè của nhau và muốn hủy bạn.
 - Đăng bài viết: cho phép người dùng đăng bài viết.
 - Thêm địa điểm vào bài viết.
- Đối với người dùng bên thứ ba là người dùng các api của hệ thống tùy trường hợp mà người dùng này có quyền hạn khác nhau.

1.1.2.1.3. Đặc điểm của người sử dụng

Bảng 1-1: Bảng phân chia quyền sử dụng

Nhóm người sử dụng	Đặc trưng	Các chức năng	Vai trò	Quyền hạn	Mức độ quan trọng
Admin	Là người chịu trách nhiệm quản lý toàn bộ hệ thống	Quản lý tất cả người dùng	Admin	Admin	Rất quan trọng
Người dùng chưa có tài khoản	Là người dùng có thể tạo tài khoản để sử dụng các chức năng của mạng xã hội	Đăng ký Xem bài viết Tra cứu địa điểm	Người dùng bình thường	Người dùng bình thường	Quan trọng
Người dùng đã có tài khoản	Là người dùng đã đăng ký tài khoản, người thực hiện các chức năng của mạng xã hội	Xem bài viết Đăng bài viết Tra cứu địa điểm Tham gia ứng tuyển Kết bạn Bình luận bài viết	Thành viên	Thành viên	Quan trọng
Người dùng bên thứ ba	Là người dùng sử dụng các API của hệ thống	Tất cả	Tùy trường hợp	Tùy trường hợp	Quan trọng

1.1.2.1.4. Môi trường vận hành

➤ Phần cứng:

Máy tính có kết nối internet.

Ram: 16GB

HDD: 500GB

CPU: Intel Core i3

Độ phân giải màn hình: 1366x768.

➤ Hệ điều hành và phần mềm:

Hệ điều hành: Windows 10

Trình duyệt web: Google Chrome, Microsoft Edge

Phần mềm API: Postman

1.1.2.1.5. Các ràng buộc về thực thi và thiết kế

➤ Thực thi:

- Máy chủ phải được đặt trong môi trường, nhiệt độ đủ tốt để hoạt động.
- Cần phải có internet với tốc độ ổn định và có thể hoạt động liên tục trong suốt quá trình làm việc.
- Thiết kế:
 - Sử dụng mô hình: Microservices, Client-Server.
 - Ngôn ngữ lập trình sử dụng: Angular, Spring-boot, MongoDB.
 - Giao diện đơn giản, thân thiện với người sử dụng.

1.1.2.2. Các yêu cầu bên ngoài

1.1.2.2.1. Giao diện người sử dụng

- Giao diện nền web: Angular.
- Giao diện thân thiện với người dùng.
- Bố cục gọn gàng, dễ dùng không làm rối mắt.
- Client: giao tiếp với người dùng qua màn hình và các thiết bị nhập xuất (bàn phím, chuột...).
- Sử dụng ngôn ngữ dễ hiểu phù hợp với người sử dụng.

1.1.2.2.2. Giao tiếp phần mềm

- Máy tính có trình duyệt web và có kết nối internet.
- Cơ sở dữ liệu: MongoDB.
- Hệ điều hành: Windows 10.

1.1.2.2.3. Giao tiếp truyền thông tin

- Trình duyệt web: Google Chrome, Microsoft Edge, FireFox (hỗ trợ tốt nhất cho Google Chrome).
- Chuẩn truyền thông tin: dùng giao thức http để truyền và nhận dữ liệu giữa máy chủ với các máy client.

1.1.2.3. Các yêu cầu chức năng của hệ thống



Hình 1-1: Chức năng của hệ thống

1.1.2.3.1. Đăng ký

Bảng 1-2: Bảng usecase đăng ký

Mã yêu cầu	UC-1
Tên yêu cầu	Đăng ký
Mức độ ưu tiên	Cao
Mô tả	Người dùng muốn đăng ký tài khoản để sử dụng hệ thống
Đối tượng sử dụng	Người dùng chưa có tài khoản
Tiền điều kiện	Không
Các thao tác xử lý	Người dùng gửi yêu cầu tạo tài khoản vào hệ thống cùng với dữ liệu user cần tạo Hệ thống xử lý và trả về kết quả
Kết quả	Người dùng đăng ký tài khoản thành công

1.1.2.3.2. Đăng nhập (Lấy authorization token)

Bảng 1-3: Bảng usecase đăng nhập (lấy authorization token)

Mã yêu cầu	UC-2
Tên yêu cầu	Đăng nhập
Mức độ ưu tiên	Cao
Mô tả	Người dùng muốn đăng nhập vào hệ thống để thực các chức năng của hệ thống
Đối tượng sử dụng	Người dùng đã có tài khoản
Tiền điều kiện	Tài khoản của người dùng đã được tạo.
Các thao tác xử lý	Người dùng gửi yêu cầu lấy token vào hệ thống cùng với dữ liệu chứng thực Hệ thống xử lý và trả về kết quả
Kết quả	Người dùng lấy token thành công

1.1.2.3.3. Tạo hoặc sửa bài viết bài viết

Bảng 1-4: Bảng usecase đăng bài viết

Mã yêu cầu	UC- 3
Tên yêu cầu	Tạo bài viết
Mức độ ưu tiên	Cao
Mô tả	Người dùng muốn tạo bài viết web
Đối tượng sử dụng	Người dùng đã có tài khoản.
Tiền điều kiện	Người dùng đã đăng nhập vào hệ thống.
Các thao tác xử lý	Người dùng gửi yêu cầu tạo bài viết vào hệ thống cùng với dữ liệu bài viết cần tạo cần tạo Hệ thống xử lý và trả về kết quả
Kết quả	Người dùng đã tạo bài viết thành công

1.1.2.3.4. Xóa bài viết

Bảng 1-5: Bảng usecase xóa bài viết

Mã yêu cầu	UC-4
Tên yêu cầu	Xóa bài viết
Mức độ ưu tiên	Cao
Mô tả	Người dùng muốn xóa bài viết
Đối tượng sử dụng	Người dùng đã có tài khoản
Tiền điều kiện	Người dùng đã có token.
Các thao tác xử lý	Người dùng gửi yêu cầu xóa bài viết vào hệ thống cùng với dữ liệu bài viết cần tạo cần tạo Hệ thống xử lý và trả về kết quả
Kết quả	Người dùng đã xóa bài viết thành công

1.1.2.3.5. Thêm hoặc sửa địa điểm vào bài viết

Bảng 1-6: Bảng usecase thêm hoặc sửa địa điểm bài viết

Mã yêu cầu	UC-6
Tên yêu cầu	Like bài viết
Mức độ ưu tiên	Cao
Mô tả	Người dùng muốn like bài viết
Đối tượng sử dụng	Người dùng đã có tài khoản
Tiền điều kiện	Người dùng đã có token.
Các thao tác xử lý	Người dùng gửi yêu cầu tạo địa điểm vào hệ thống cùng với dữ liệu bài viết cần tạo cần tạo Hệ thống xử lý và trả về kết quả
Kết quả	Người dùng thêm hoặc cập nhật địa điểm thành công thành công

1.1.2.3.6. Xóa địa điểm khỏi bài viết

Bảng 1-7: Bảng usecase xóa địa điểm khỏi bài viết

Mã yêu cầu	UC-6
Tên yêu cầu	Like bài viết
Mức độ ưu tiên	Cao
Mô tả	Người dùng muốn like bài viết
Đối tượng sử dụng	Người dùng đã có tài khoản
Tiền điều kiện	Người dùng đã có token
Các thao tác xử lý	Người dùng gửi yêu cầu xóa địa điểm vào hệ thống cùng với dữ liệu bài viết cần tạo cần thiết. Hệ thống xử lý và trả về kết quả
Kết quả	Người dùng xóa địa điểm thành công thành công

1.1.2.3.7. Xem chi tiết bài viết

Bảng 1-8: Bảng usecase xem bài viết

Mã yêu cầu	UC-7
Tên yêu cầu	Xem bài viết
Mức độ ưu tiên	Cao
Mô tả	Người dùng muốn xem chi tiết bài viết
Đối tượng sử dụng	Tất cả
Tiền điều kiện	Không
Các thao tác xử lý	Người dùng gửi yêu cầu xem thông tin bài viết vào hệ thống cùng với dữ liệu bài viết cần tạo cần thiết Hệ thống xử lý và trả về kết quả
Kết quả	Người dùng xem bài viết thành công

1.1.2.3.8. Viết bình luận

Bảng 1-9: Bảng usecase viết bình luận

Mã yêu cầu	UC-8
Tên yêu cầu	Viết bình luận
Mức độ ưu tiên	Cao
Mô tả	Người dùng muốn bình luận bài viết
Đối tượng sử dụng	Người dùng đã có tài khoản
Tiền điều kiện	Người dùng đã có token.
Các thao tác xử lý	Người dùng gửi yêu cầu bình luận bài viết vào hệ thống cùng với dữ liệu bài viết cần tạo cần thiết Hệ thống xử lý và trả về kết quả
Kết quả	Người dùng bình luận bài viết thành công

1.1.2.3.9. Xem trang cá nhân

Bảng 1-10: Bảng usecase xem trang cá nhân

Mã yêu cầu	UC-9
Tên yêu cầu	Xem thông tin cá nhân
Mức độ ưu tiên	Cao
Mô tả	Người dùng muốn xem trang cá nhân của mình
Đối tượng sử dụng	Người dùng đã có tài khoản
Tiền điều kiện	Người dùng đã có token.
Các thao tác xử lý	Người dùng gửi yêu cầu lấy thông tin cá nhân vào hệ thống cùng với dữ liệu bài viết cần tạo cần thiết Hệ thống xử lý và trả về kết quả
Kết quả	Người dùng xem trang cá nhân thành công

1.1.2.3.10. Cập nhật thông tin cá nhân

Bảng 1-11: Bảng usecase cập nhật thông tin cá nhân

Mã yêu cầu	UC-10
Tên yêu cầu	Cập nhật thông tin cá nhân
Mức độ ưu tiên	Cao
Mô tả	Người dùng muốn cập nhật thông tin cá nhân
Đối tượng sử dụng	Người dùng đã có tài khoản
Tiền điều kiện	Người dùng đã có token.
Các thao tác xử lý	Người dùng gửi yêu cầu cập nhật thông tin cá nhân vào hệ thống cùng với dữ liệu bài viết cần tạo cần thiết Hệ thống xử lý và trả về kết quả
Kết quả	Người dùng đã chỉnh sửa thông tin cá nhân thành công

1.1.2.3.11. Tìm người dùng

Bảng 1-12: Bảng usecase tìm người dùng

Mã yêu cầu	UC-11
Tên yêu cầu	Tìm kiếm người dùng
Mức độ ưu tiên	Cao
Mô tả	Người dùng muốn tìm kiếm bạn bè để kết bạn, xem thông tin cá nhân và kết bạn
Đối tượng sử dụng	Người dùng đã có tài khoản
Tiền điều kiện	Người dùng đã có token.
Các thao tác xử lý	Người dùng gửi yêu cầu lấy thông tin vào hệ thống cùng với dữ liệu bài viết cần tạo cần thiết Hệ thống xử lý và trả về kết quả
Kết quả	Người dùng thực hiện được việc tìm kiếm người dùng như mong đợi

1.1.2.3.12. Gửi kết bạn

Bảng 1-13: Bảng usecase gửi kết bạn

Mã yêu cầu	UC-12
Tên yêu cầu	Gửi kết bạn
Mức độ ưu tiên	Cao
Mô tả	Người dùng muốn kết bạn giao lưu
Đối tượng sử dụng	Người dùng đã có tài khoản
Tiền điều kiện	Người dùng đã có token.
Các thao tác xử lý	Người dùng gửi yêu cầu kết bạn vào hệ thống cùng với dữ liệu bài viết cần tạo cần thiết Hệ thống xử lý và trả về kết quả
Kết quả	Người dùng thực hiện được việc gửi yêu cầu kết bạn

1.1.2.3.13. Trả lời yêu cầu kết bạn kết bạn

Bảng 1-14: Bảng usecase trả lời kết bạn

Mã yêu cầu	UC-13
Tên yêu cầu	Trả lời kết bạn
Mức độ ưu tiên	Cao
Mô tả	Người dùng muốn chấp nhận kết bạn
Đối tượng sử dụng	Người dùng đã có tài khoản
Tiền điều kiện	Người dùng đã có token.
Các thao tác xử lý	Người dùng gửi trả lời yêu cầu lấy thông tin vào hệ thống cùng với dữ liệu bài viết cần tạo cần thiết Hệ thống xử lý và trả về kết quả
Kết quả	Người dùng thực hiện được việc trả lời yêu cầu kết bạn

1.1.2.3.14. Xóa bạn

Bảng 1-15: Bảng usecase xóa bạn

Mã yêu cầu	UC-14
Tên yêu cầu	Xóa bạn
Mức độ ưu tiên	Cao
Mô tả	Người dùng muốn xóa bạn
Đối tượng sử dụng	Người dùng đã có tài khoản
Tiền điều kiện	Người dùng đã có token và đã là bạn bè.
Các thao tác xử lý	Người dùng gửi yêu cầu xóa bạn vào hệ thống cùng với dữ liệu bài viết cần tạo cần thiết Hệ thống xử lý và trả về kết quả
Kết quả	Người dùng thực hiện được việc xóa bạn

1.1.2.3.15. Tra cứu địa điểm

Bảng 1-16: Bảng usecase tra cứu địa điểm

Mã yêu cầu	UC-15
Tên yêu cầu	Tra cứu địa điểm
Mức độ ưu tiên	Cao
Mô tả	Người dùng muốn tra cứu địa điểm
Đối tượng sử dụng	Tất cả
Tiền điều kiện	Không
Xử lý luồng chính	Người dùng gửi yêu cầu tra cứu địa điểm vào hệ thống cùng với dữ liệu bài viết cần tạo cần thiết Hệ thống xử lý và trả về kết quả
Kết quả	Người dùng tra cứu địa điểm thành công

1.1.2.4. Các yêu cầu phi chức năng

1.1.2.4.1. Yêu cầu thực thi

- Máy chủ (server) phải hoạt động liên tục trong suốt thời gian vận hành của hệ thống.
- Phải có kết nối internet ổn định.
- Đối với các luồng thêm, sửa, xóa thời gian xử lý mỗi luồng này không được vượt quá 5s.

1.1.2.4.2. Yêu cầu an toàn

Phục hồi dữ liệu bị mất ngay lập tức từ bản sao lưu nếu có sự cố xảy ra.

1.1.2.4.3. Yêu Cầu bảo mật

Mật khẩu người dùng được lưu trữ trong cơ sở dữ liệu phải được mã hóa.

1.1.2.4.4. Các đặc điểm chất lượng phần mềm

- Độ chính xác và độ tin cậy cao.
- Có thể kiểm thử.
- Dễ dàng thể bảo trì và nâng cấp khi hệ thống phát sinh các vấn đề mới.
- Có các thông báo xác nhận khi người dùng thực hiện các chức năng xóa hoặc thay đổi dữ liệu.
- Tài liệu của dự án được quản lý có hệ thống.

1.2. Yêu cầu phát triển và yêu cầu nghiên cứu

1.2.1. Yêu cầu phát triển

- Phần mềm cần được phát triển bằng mô hình Microservices để tăng tính mô-đun hoá, tính sử dụng lại.
- Phần mềm không cần phải triển khai lên web nhưng cần chạy được ở máy cục bộ.
- Có thể sử dụng bất kì Java libraries và frameworks, khuyến khích dùng Spring Boot, Spring Data JPA/Hibernate, JUnit, Rabbitmq and React/Angular.

1.2.2. Yêu cầu nghiên cứu

Để thực hiện theo yêu cầu phát triển trên thì cần trả lời các câu hỏi sau:

- Microservices là gì?
- Các vấn đề cần quan tâm khi áp kiến trúc Microservices vào phát triển phần mềm?
- Hiệu quả của Microservices so với API service thông thường?

CHƯƠNG 2: KIẾN THỨC NỀN

Chương này giới thiệu các kiến thức và công nghệ áp dụng vào phần mềm. Phần nội dung Microservices tham khảo từ [3].

1.1. Tổng quan về Microservices

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

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

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

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

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

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

2.1. Định nghĩa về Microservices

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

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

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

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

2.2. Đặc điểm của Microservices

Single purpose

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

Encapsulation

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

Ownership

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

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

Autonomy

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

Multiple versions

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

2.3. So sánh Monoliths với Microservices

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

Bảng 2-1: Bảng so sánh Microservices và monolith

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

2.4. Mẫu thiết kế của kiến trúc Microservices

2.4.1. Kiến trúc vi mô và vĩ mô

Đó là nên phân chia kiến trúc của bạn thành kiến trúc vi mô và vĩ mô. **Kiến trúc vi mô (micro architecture)** liên quan đến quyết định đối với từng Microservices. **Kiến trúc vĩ mô (macro architecture)** liên quan đến tất cả các quyết định ở cấp độ toàn cục áp dụng cho tất cả các Microservices.

Có thể mở rộng khái niệm kiến trúc vi mô và vĩ mô đến **quyết định kỹ thuật**. Quyết định kỹ thuật có thể được đưa ra trong khuôn khổ của kiến trúc vĩ mô hoặc vi mô. Ví dụ, hãy xem các quyết định kỹ thuật được đưa ra ở cấp độ vi mô và vĩ mô cho cơ sở dữ liệu:

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

2.4.2. Kiến trúc hệ thống khép kín

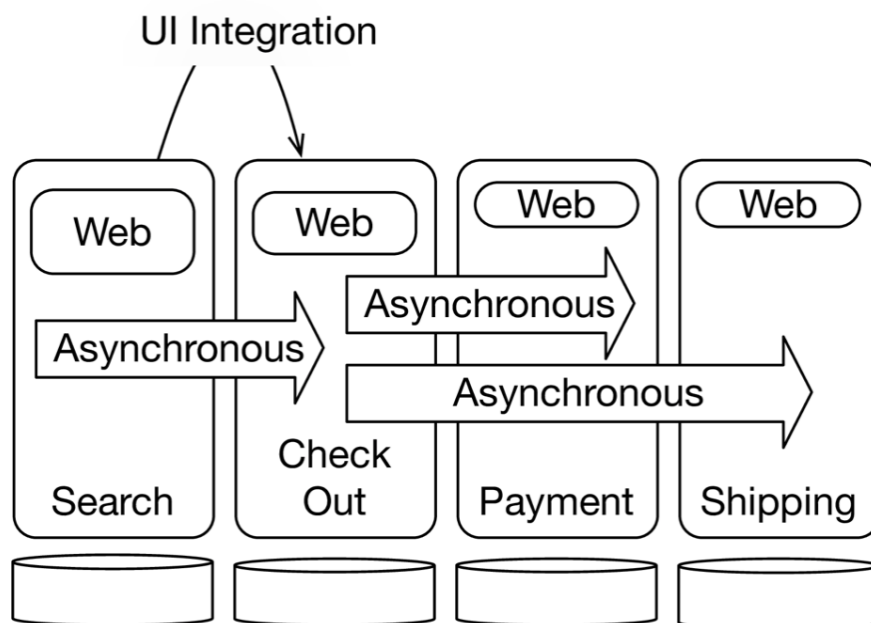
Hệ thống khép kín (SCS) là một loại kiến trúc Microservices để chỉ định các thành phần của kiến trúc macro. Điều này có nghĩa là chúng không đại diện cho toàn bộ hệ thống.

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

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

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

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



Example of an SCS Architecture

Hình 2-1: Mẫu kiến trúc SCS

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

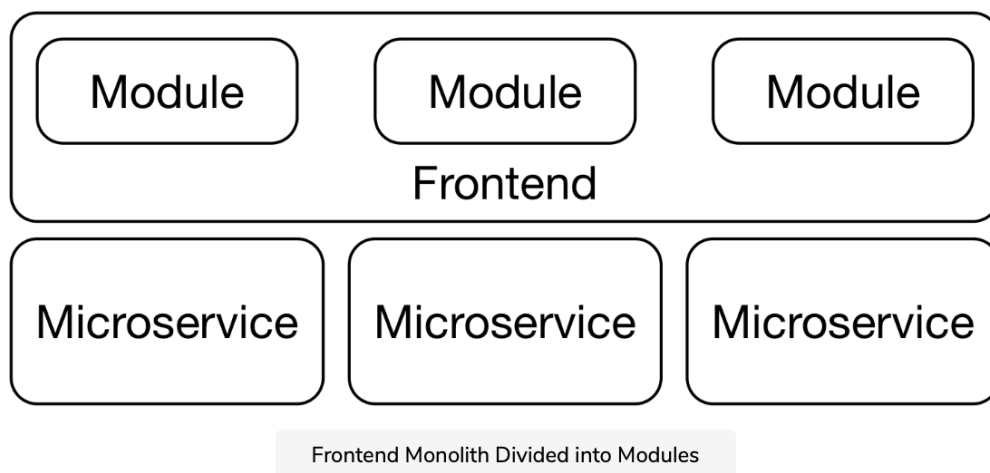
Trên thực tế, một SCS thậm chí có thể được chia thành nhiều Microservices. Chúng khác với Microservices theo ba khía cạnh chính: chúng lớn hơn Microservices; chúng tập trung vào liên kết lỏng lẻo; chúng phải có giao diện người dùng (UI).

2.4.3. Kiến trúc tích hợp Front-End

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

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

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



Hình 2-2: Chia Frontend thành modules

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

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

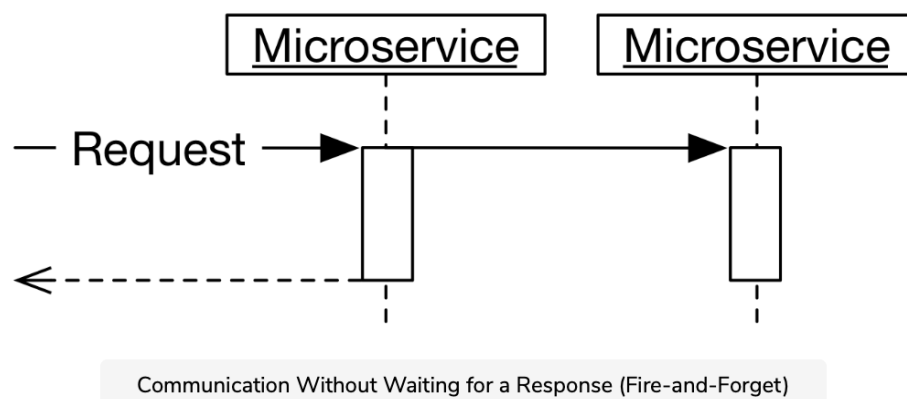
Tuy nhiên, có một vài trường hợp ngoại lệ khi một Front-End nên được triển khai dưới dạng nguyên khối. Ví dụ: các ứng dụng di động gốc phải triển khai nguyên khối hoặc nếu có một nhóm duy nhất chịu trách nhiệm phát triển Front-End.

2.4.4. Kiến trúc Microservices không đồng bộ

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

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

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



Hình 2-3: Liên lạc không đồng bộ

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

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

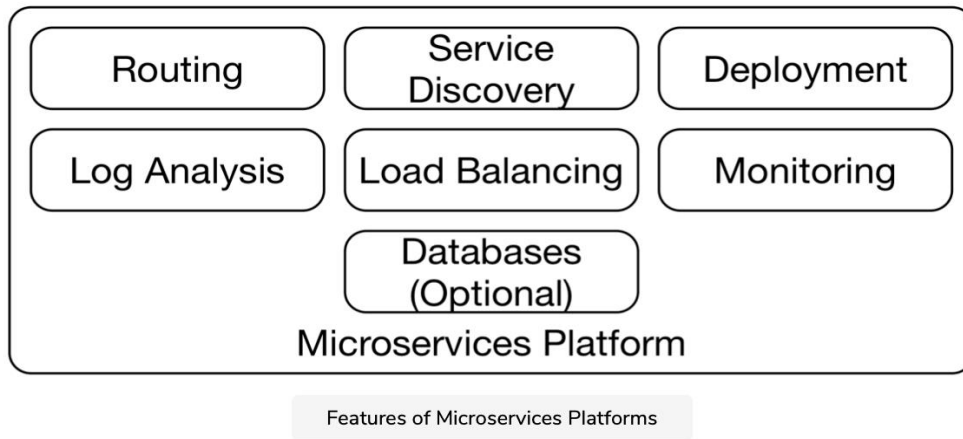
Tương tự, nếu một bên của sự giao tiếp thất bại, nó sẽ không sụp đổ toàn bộ hệ thống, mang lại khả năng phục hồi tổng thể cho hệ thống của bạn.

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

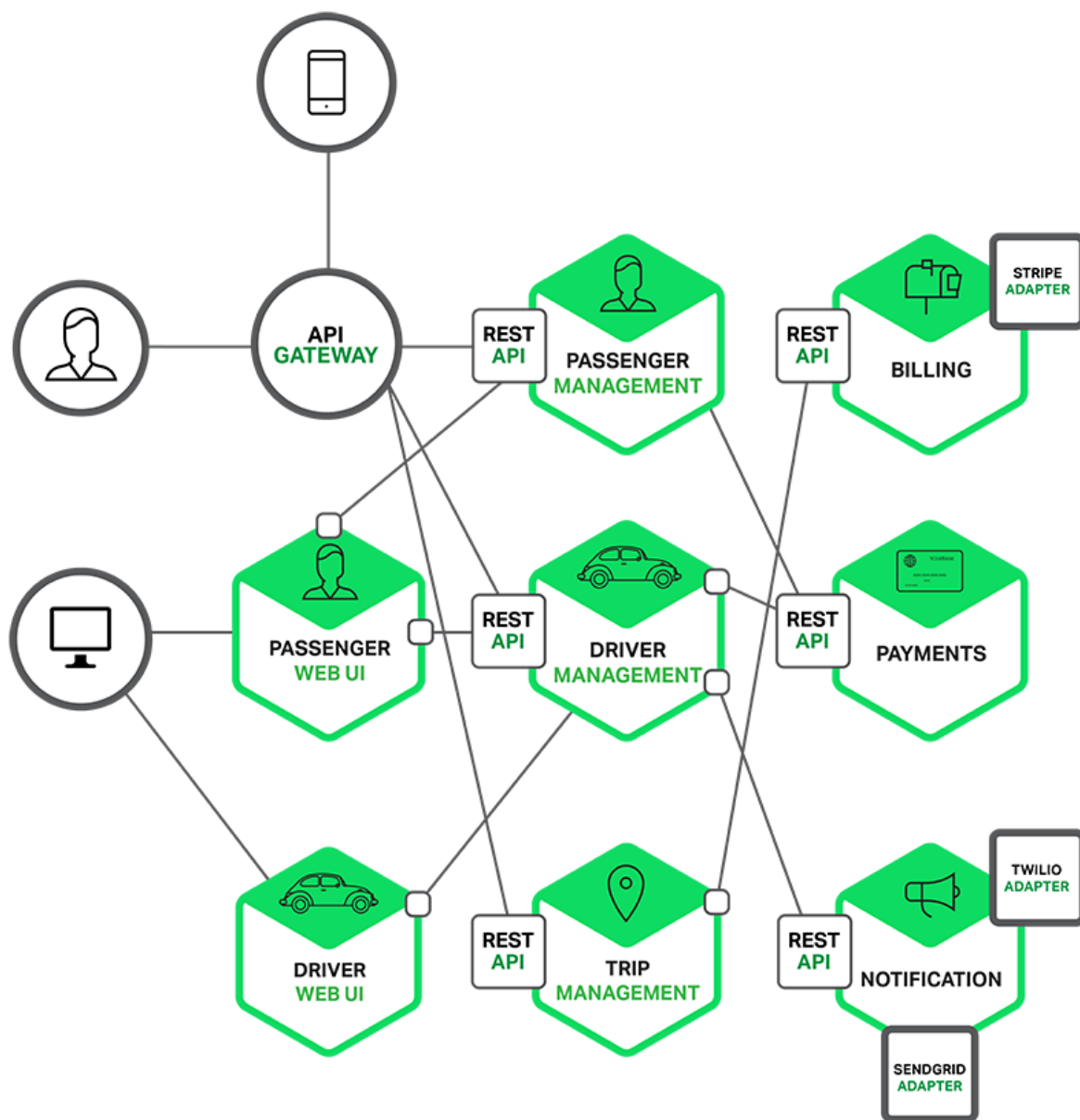
2.4.5. Nền tảng Microservices

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

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



Hình 2-4: Chức năng của nền tảng Microservices



Hình 2-5: Mẫu kiến trúc Microservices

2.5. Spring

Spring là một Framework phát triển các ứng dụng Java được sử dụng bởi hàng triệu lập trình viên. Nó giúp tạo các ứng dụng có hiệu năng cao, dễ kiểm thử, sử dụng lại code...

Spring nhẹ và trong suốt (nhẹ: kích thước nhỏ, version cơ bản chỉ khoảng 2MB; trong suốt: hoạt động một cách trong suốt với lập trình viên)

Spring là một mã nguồn mở, được phát triển, chia sẻ và có cộng đồng người dùng rất lớn.

Spring Framework được xây dựng dựa trên 2 nguyên tắc design chính là: Dependency Injection và Aspect Oriented Programming.

Những tính năng core (cốt lõi) của Spring có thể được sử dụng để phát triển Java Desktop, ứng dụng mobile, Java Web. Mục tiêu chính của Spring là giúp phát triển các ứng dụng J2EE một cách dễ dàng hơn dựa trên mô hình sử dụng POJO (Plain Old Java Object)

2.6. Spring-boot

Spring Boot là một mô-đun của Spring Framework. Nó được sử dụng để tạo các Ứng dụng dựa trên Spring cấp sản xuất, độc lập với những nỗ lực tối thiểu. Nó được phát triển trên nền tảng Spring Framework cốt lõi.

Spring Boot tuân theo một kiến trúc phân lớp, trong đó mỗi lớp giao tiếp với lớp ngay bên dưới hoặc bên trên (cấu trúc phân cấp) nó.

Trước khi hiểu Kiến trúc Spring Boot, chúng ta phải biết các lớp và các lớp khác nhau có trong nó. Có bốn lớp trong Spring Boot như sau:

Lớp trình bày (Application Layer): Lớp trình bày xử lý các yêu cầu HTTP, dịch tham số JSON thành đối tượng và xác thực yêu cầu và chuyển nó đến lớp nghiệp vụ. Nói tóm lại, nó bao gồm các khung nhìn, tức là phần giao diện người dùng.

Lớp nghiệp vụ (Domain Layer): Lớp nghiệp vụ xử lý tất cả các logic nghiệp vụ. Nó bao gồm các lớp dịch vụ và sử dụng các dịch vụ được cung cấp bởi các lớp truy cập dữ liệu. Nó cũng thực hiện ủy quyền và xác nhận.

Lớp Kiến trì (Infrastructure Layer): Lớp bền vững chứa tất cả logic lưu trữ và dịch các đối tượng nghiệp vụ từ và sang các hàng cơ sở dữ liệu.

Lớp cơ sở dữ liệu: Trong lớp cơ sở dữ liệu, các hoạt động CRUD (tạo, truy xuất, cập nhật, xóa) được thực hiện.

2.7. Angular

Angular là một javascript framework do google phát triển để xây dựng các Single Page Application (SPA) bằng JavaScript, HTML và TypeScript. Angular cung cấp các tính năng tích hợp cho animation, http service và có các tính năng như auto-complete, navigation, toolbar, menus,... Code được viết bằng TypeScript, biên dịch thành JavaScript và hiển thị tương tự trong trình duyệt.

2.8. MongoDB

MongoDB là một cơ sở dữ liệu mã nguồn mở và là cơ sở dữ liệu NoSQL hàng đầu, được hàng triệu người sử dụng. MongoDB được viết bằng C++.

Ngoài ra, MongoDB là một cơ sở dữ liệu đa nền tảng, hoạt động trên các khái niệm Collection và Document, nó cung cấp hiệu suất cao, tính khả dụng cao và khả năng mở rộng dễ dàng.

NoSQL là 1 dạng CSDL mã nguồn mở không sử dụng Transact-SQL để truy vấn thông tin. NoSQL viết tắt bởi: None-Relational SQL, hay có nơi thường gọi là Not-Only SQL. CSDL này được phát triển trên Javascript Framework với kiểu dữ liệu JSON. (Cú pháp của JSON là "key:value") NoSQL ra đời như là 1 mảnh vá cho những khuyết điểm và thiếu sót cũng như hạn chế của mô hình dữ liệu quan hệ RDBMS về tốc độ, tính năng, khả năng mở rộng, memory cache,...

2.9. Docker

Docker là nền tảng phần mềm cho phép bạn xây dựng, kiểm thử và 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 cần để chạy, trong đó có thư viện, công cụ hệ thống, mã và thời gian chạy. Bằng cách sử dụng Docker, bạn 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 bạn sẽ chạy được.

2.10. HERE Map

HERE WeGo là một dịch vụ điều hướng và lập bản đồ web, được điều hành bởi Here Technologies. Ban đầu được phát triển bởi Nokia với tên gọi HERE Maps, ứng dụng phần mềm bản đồ lần đầu tiên được phát hành cho Windows Phone và World Wide Web vào năm 2013 dưới dạng phiên bản cải tiến của Nokia Maps. Ứng dụng Here Maps sau đó được phát hành cho nền tảng Android vào ngày 10 tháng 12 năm 2014 và sau đó cho iOS vào ngày 11 tháng 3 năm 2015 và tên được đổi thành Here WeGo vào tháng 7 năm 2016. Đây cũng là nhà cung cấp dịch vụ bản đồ mặc định cho máy tính bảng Amazon Fire và điện thoại thông minh.

Dữ liệu được cung cấp nội bộ bởi HERE, bao gồm chế độ xem vệ tinh, dữ liệu giao thông và các dịch vụ vị trí khác. Bản đồ được cập nhật hai hoặc ba tháng một lần.

2.11. Cloudinary

Cloudinary là một cloud-based service, nó cung cấp một giải pháp quản lý hình ảnh bao gồm upload, lưu trữ, thao tác, tối ưu hóa và delivery.

Với cloudinary bạn có thể dễ dàng upload ảnh lên cloud, tự động thực thi các thao tác với ảnh một cách thông minh mà không cần phải cài đặt bất kỳ một phần mềm phức tạp nào khác. Cloudinary cung cấp các APIs toàn diện và màn hình quản lý giúp chúng ta dễ dàng tích hợp vào các trang web và ứng dụng di động.

2.12. Nginx

Nginx là một máy chủ ủy nhiệm mã nguồn mở rất phổ biến hiện nay, nginx sử dụng các giao thức HTTP, HTTPS, SMTP, POP3 và IMAP. Nginx tập trung vào việc phục vụ số lượng kết nối đồng thời lớn với hiệu suất cao và sử dụng bộ nhớ thấp. Nginx được biết đến bởi sự ổn định cao, nhiều tính năng, cấu hình đơn giản và tiết kiệm tài nguyên.

CHƯƠNG 3: CÁC VẤN ĐỀ CẦN QUAN TÂM KHI PHÁT TRIỂN PHẦN MỀM THEO KIẾN TRÚC MICROSERVICES

Chương này đưa ra các vấn đề cần quan tâm khi phát triển phần mềm trên nền tảng Microservices. Chương này tham khảo từ [4].

3.1. Mô hình hoá các microservices

3.1.1. Xác định các Microservices

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

Việc xác định các Microservices phải đảm bảo sao cho khi ta muốn thay đổi một hành vi hoặc một chức năng nào đó thì chúng ta chỉ cần thay đổi ở một Microservices. Để làm được điều này, chúng ta nên nhóm các hành vi có liên quan đến nhau lại trong cùng một Microservices nhưng việc nhóm các hành vi này vẫn phải đảm bảo kết nối lỏng lẻo cho các Microservices trong hệ thống. Trong cuốn Domain-Driven Design, Eric Evans đã đưa ra khái niệm về Bounded Context Để Có thể khoanh vùng được các chức năng trên miền (DDD –Domain-Driven Design). Nó rất hữu ích khi ta áp dụng để xác định các Microservices trong kiến trúc Microservices. Theo Eric Evans thì Bounded Context là một mẫu thiết kế trung tâm của thiết kế hướng miền nhằm khoanh vùng các hành vi có mối quan hệ với nhau và cùng thực hiện một chức năng lại với nhau. Bounded Context giúp các nhà xây dựng phần mềm chia nhỏ các hệ thống lớn và giúp các thành phần sau khi chia nhỏ được độc lập với nhau. Chúng ta có thể sử dụng ưu điểm này của Bounded Context để xác định các Microservices trong kiến trúc Microservices.

3.1.2. Thiết kế kiến trúc

Kiến trúc hệ thống gồm có một cổng API và ba Microservices: Users service, Post service, Geo service. Mỗi Microservices này có một API, một cơ sở dữ liệu và các thành phần được cài đặt để thực hiện chức năng của Microservices. Các Microservices giao tiếp với nhau bằng kỹ thuật tương tác IPC (Inter-Process communication) sử dụng giao thức HTTP dựa trên REST.

Mỗi Microservices đều có một API để có thể giao tiếp với nhau. Để tạo ra API cho mỗi Microservices, ta phải đảm bảo được ba phần chính sau:

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

Nắm được quy mô thay đổi của API bởi vì API của Microservices luôn thay đổi theo thời gian. Khi đã biết rõ quy mô thay đổi của API ta có thể đưa ra các biện pháp xử lý hiệu quả nhất: Đối với các thay đổi nhỏ, tính tương thích giữa API và các máy trạm chưa bị phá vỡ ta có thể thêm thay đổi vào phần yêu cầu và hồi đáp. Đối với các

thay đổi lớn, tính tương thích bị phá vỡ ta bắt buộc phải tạo ra thêm phiên bản mới cho API trong Microservices và vẫn duy trì API cũ.

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

Các API của các Microservices trong Go-Hiking giao tiếp với nhau bằng REST và sau khi thực hiện xong các HTTP request nó sẽ trả về kết quả dưới dạng JSON.

Để dễ dàng xây dựng và phát triển các Microservices, mỗi API trong các Microservices cần phải có một phần tài liệu để mô tả các chức năng cách dùng gọi là tài liệu API. Trong Spring có một thư viện rất hữu ích để viết tài liệu API gọi là swaggerUI.

Trong đó summary được dùng để mô tả chung về API, notes cho phép mô tả chi tiết về API, param dùng để khai báo các thuộc tính của mỗi story và response cho phép thể hiện các thông điệp trả về các máy trạm khi có yêu cầu tạo post.

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

POST cho phép tạo ra sự thay đổi trong cơ sở dữ liệu, được dùng khi các máy trạm muốn tạo ra đối tượng mới hoặc khi các máy trạm muốn gửi dữ liệu đến các Microservices và ngược lại.

GET dùng để truy vấn dữ liệu và tài nguyên với các tham số và giá trị nằm ngay trên URL.

PUT/PATCH được dùng khi máy trạm muốn cập nhật dữ liệu.

DELETE sẽ xóa đối tượng khỏi cơ sở dữ liệu.

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

2xx: thể hiện HTTP request đã được máy chủ thực hiện thành công

200: OK

201: Created

204: No_content

3xx: chuyển hướng

4xx: thể hiện các lỗi từ phía máy trạm

400 -:bad_request

403 -:forbidden

401 -:unauthorized

404 -:not_found

410 -:gone

422 -:unprocessable_entity

5xx: các lỗi từ phía máy chủ

500 –:internal_server_error

3.1.3. Cơ chế kết nối

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

Cơ chế tương tác không đồng bộ dựa trên hệ thống thông điệp cho phép các Microservices trao đổi với nhau qua thông điệp (message), Microservices sẽ gửi một thông điệp đến Microservices khác mà nó muốn kết nối để thực hiện kết nối. Có rất nhiều nền tảng sử dụng cơ chế này là mã nguồn mở như: RabbitMQ 7, Apache Kafka 8, Apache ActiveMQ 9,...

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

Trong Go-Hiking các máy trạm giao tiếp với Auth service thông qua các phương thức HTTP với giao thức REST. Để tạo tài khoản mới, các máy trạm sẽ gửi một yêu cầu nó được thực hiện bởi phương thức POST của HTTP. Nếu tài khoản được tạo thành công thì thông tin của tài khoản mới này sẽ được lưu vào UserDB và Auth service sẽ trả về các máy trạm một HTTP response có mã trạng thái là 200.

Để lấy thông tin của người dùng,các máy trạm sẽ thông qua một HTTP request có URL chứa id của người dùng cần truy vấn bằng phương thức GET, Authservice sẽ trả về một HTTP responsive có mã trạng thái là 200 và thông điệp chứa thông tin của tài khoản người dùng được truy vấn dưới định dạng JSON.

Ví dụ thông tin của người dùng trong Go-Hiking sau khi các máy trạm thực hiện HTTP request với authorization header vào url: GET/user/info:

```
{
  "id": "616d152914aa9a6791d83b9d",
  "firstName": "Admin",
  "lastName": "I",
  "username": "admin",
  "email": "mail@mail.com",
  "roles": "0",
  "phoneNumber": "0123456789",
  "access": "PRIVATE"
}
```

Các kết nối này được thực hiện tương tự khi các Microservices trong Go-Hiking muốn giao tiếp với nhau. Ví dụ Post service thực hiện các chức năng hiển thị, tạo, sửa, xóa các story, mỗi người dùng muốn thực hiện các chức năng tạo,sửa,xóa post thì bắt buộc phải đăng nhập cho nên các máy trạm khi gửi các yêu cầu để tạo, sửa, xóa các Story sẽ phải thông qua Authservice để xác thực người dùng đã đăng nhập.Để tạo một Story thì các máy trạm sẽ phải thực hiện truy vấn.

Các máy trạm gửi một yêu cầu HTTP bằng phương thức PUT với URL là /post/upsert đến post service để tạo một post, post service sau khi nhận được yêu cầu này nó sẽ gửi một yêu cầu đến Auth service để kiểm tra token của người dùng. Nếu người dùng đã đăng nhập Authservice sẽ gửi HTTP response có mã trạng thái là 200

đến Stories service, khi đó Stories services sẽ thực hiện tạo mới một post vào Post DB và gửi một HTTP response có mã Status code là 201 đến các máy trạm.

3.1.4. Xây dựng cổng API

Cổng API là thành phần rất quan trọng trong các hệ thống được xây dựng trên kiến trúc Microservices. Tất cả các yêu cầu của các máy trạm đến các Microservices trong hệ thống đều phải đi qua cổng API mà chúng ta lại không thể tính chính xác được mỗi ngày sẽ có bao nhiêu lượt yêu cầu đi qua cổng API nên khi xây dựng cổng API điều quan trọng nhất là hiệu năng và tính mở rộng của cổng API. Hiện nay có rất nhiều công nghệ khác nhau để thực hiện việc mở rộng cổng API nhưng nổi bật nhất là xây dựng cổng API dựa trên nền tảng hỗ trợ xử lý bất đồng bộ và cơ chế non-blocking I/O (Input/Output).

Một cổng API không chỉ kết nối các máy trạm đến các Microservices mà cổng API còn định tuyến các yêu cầu được gửi từ các máy trạm đến các Microservices để thực hiện xử lý các yêu cầu đó. Sau đó cổng API lại tổng hợp các kết quả từ các Microservices lại để phản hồi cho các máy trạm. Vì vậy khi thi thiết kế và xây dựng cổng API ta nên xây dựng cổng API có khả năng thực hiện các yêu cầu đồng thời cùng một lúc để giảm thời gian phản hồi tăng hiệu năng hệ thống.

3.2. Triển khai

3.2.1. Đóng gói các Microservices

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

3.2.2. Cấu hình cho máy chủ

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

Phương pháp thứ hai là triển khai mỗi Microservices một máy chủ. Phương pháp này cho phép mỗi Microservices được triển khai trên một máy chủ. Nó có thể khắc phục được những khó khăn trong phương pháp triển khai nhiều Microservices trên một máy chủ. Nhưng đổi lại chi phí để triển khai theo mô hình này là rất lớn.

3.3. Kiểm thử trong Microservices

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

chất lượng sản phẩm, tốc độ kiểm thử nhanh và đặc biệt là giảm chi phí cho hệ thống trong khâu kiểm thử.

3.3.1. Kiểm thử đơn vị

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

3.3.2. Kiểm thử dịch vụ

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

3.3.3. Kiểm thử đầu cuối

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

3.3.4. Kiểm thử tích hợp

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

3.4. Sử dụng kiến trúc Microservices khi nào là hợp lý

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

- Khi phát triển những phiên bản đầu tiên cho một ứng dụng, khi đó bạn thường không phải gặp những vấn đề mà Microservices cần phải giải quyết. Hơn nữa, việc sử dụng một kiến trúc phân tán hoặc phức tạp sẽ làm chậm đi quá trình phát triển của ứng dụng.
- Đây là một trong những vấn đề lớn đối với các startup bởi vì họ buộc phải phát triển nhanh mô hình kinh doanh của mình cũng như ứng dụng kèm theo. Chính vì vậy, trừ khi bạn đã có một hệ thống phức tạp để quản lý bằng Monolithic hoặc bạn đã xác định được tương lai của ứng dụng sẽ ra sao; thì có thể dùng Microservices.

3.5. Các vấn đề nên lưu ý khi thiết kế Microservices

3.5.1. Hiểu sai về Microservices

- Một số dòng code/kích cỡ của một đội lập trình thường là chỉ số tồi.
- Micro là một từ khóa dễ gây hiểu nhầm và bạn nghĩ rằng nên tạo ra services nhỏ hết mức thì đó là cách hiểu hoàn toàn sai.
- Services trở thành các cục monolithic với nhiều hàm, chức năng khác được hỗ trợ nhau. Chính vì thế, khi phát triển services kiểu SOA rồi dần nhận Microservices hoàn toàn bị đánh lạc hướng và không mang lại bất kỳ lợi ích nào.

3.5.2. Những điều cần phải tuân thủ

- Một Service có phạm vi và chức năng giới hạn thì việc tập trung vào một nhiệm vụ sẽ giúp cho quá trình phát triển cũng như triển khai dịch vụ trở nên nhanh chóng hơn.
- Khi thiết kế, bạn nên xác định và giới hạn cho các service dựa theo chức năng nghiệp vụ thực tế.
- Hãy đảm bảo Microservices có thể phát triển cũng như được triển khai độc lập.
- Mục tiêu thiết kế là đưa ra phạm vi cho một Microservices phục vụ cho một nghiệp vụ chứ không đơn giản là làm những dịch vụ nhỏ hơn. Khi đó, kích thước hợp lý của một services đó chính là kích thước đủ để đáp ứng cho các yêu cầu của một chức năng bên trong hệ thống.
- Khác biệt so với services trong SOA thì một Microservices không nên có quá nhiều hàm hoặc chức năng hỗ trợ xung quanh cũng như định dạng thông báo gửi hoặc gửi tin đơn giản.

CHƯƠNG 4: XÂY DỰNG ỨNG DỤNG GO-HIKING

Chương 4 sẽ trình bày các bước để thiết kế và xây dựng ứng dụng web dành cho người đi bộ đường dài Go-Hiking trên kiến trúc Microservices. Phần đầu chương giới thiệu về ứng dụng Go-Hiking. Các phần tiếp theo của chương trình bày chi tiết các bước để xây dựng ứng dụng này dựa trên kiến trúc Microservices.

4.1. Tổng quan hệ thống

Hệ thống hoạt động gồm 4 loại người dùng: Admin (quản trị viên), người dùng bình thường (người dùng chưa đăng ký tài khoản), thành viên (người dùng đã đăng ký tài khoản), người dùng bên thứ 3. Trong đó:

- Admin: có quyền cao nhất trong hệ thống, quản lý toàn bộ thông tin tài khoản người dùng và cũng có đầy đủ các chức năng của web.
- Người dùng chưa có tài khoản:
 - Xem danh sách bài viết
 - Xem chi tiết bài viết.
 - Tìm kiếm bài viết.
 - Xác định địa điểm.
 - Đăng ký tài khoản.
- Thành viên: người đã đăng ký tài khoản trên hệ thống, có những chức năng sau:
 - Tất cả tính năng của người dùng chưa có tài khoản.
 - Thêm/sửa/xóa thông tin bài viết.
 - Thêm/sửa/xóa thông tin bài viết có các địa điểm đi bộ.
 - Bình luận bài viết.
 - Tìm kiếm các người dùng khác có tài khoản ở chế độ công khai.
 - Gửi yêu cầu kết bạn.
 - Huỷ yêu cầu kết bạn.
 - Chấp nhận kết bạn.
 - Xóa bạn.
 - Xem trang cá nhân bạn bè.
 - Xem thông tin cá nhân.
 - Cập nhật thông tin cá nhân.
- Người dùng bên thứ ba: có thể dùng các cổng api của hệ thống để tích hợp vào hệ thống của họ.

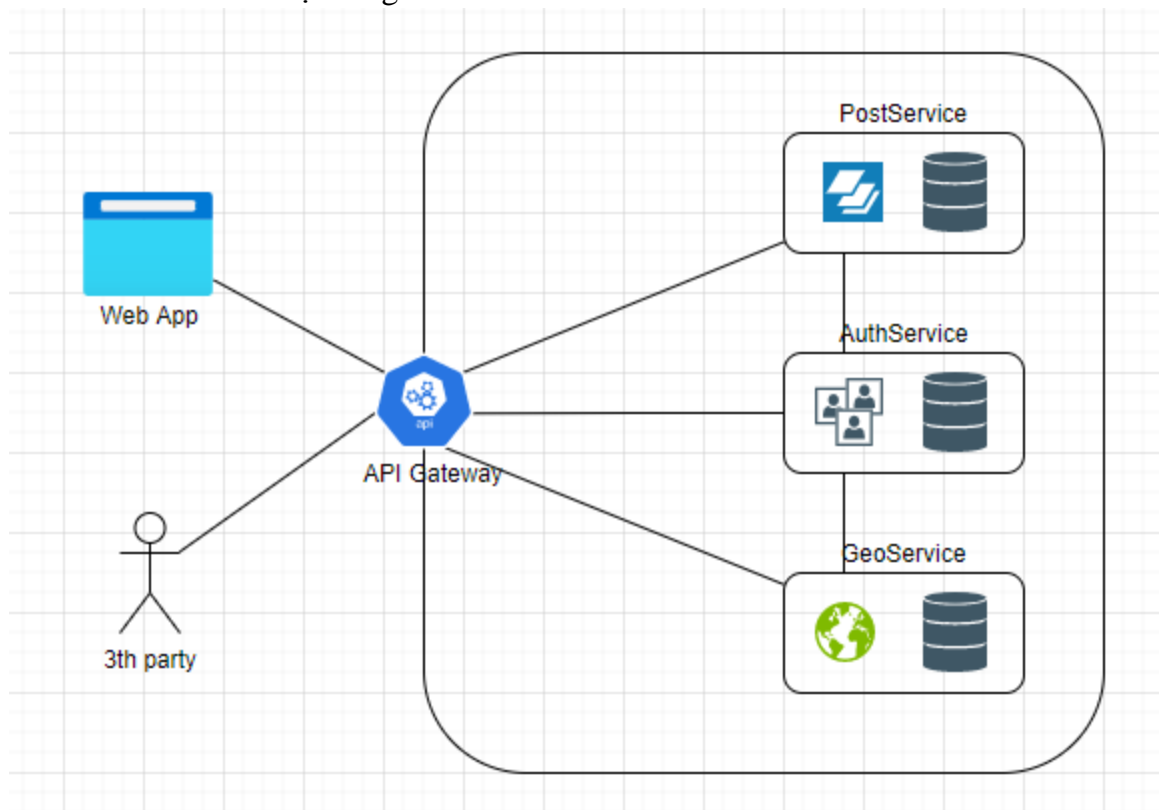
4.2. Kiến trúc hệ thống

Dựa trên các chức năng và các yêu cầu của Go-Hiking mà ta đã xác định ở phần trước, ta có thể xác định sẽ có 4 vùng Bounded Context chính là: vùng chứa các hành vi liên quan đến người dùng (Users Context), vùng chứa các hành vi liên quan đến các bài viết (Post Context), vùng chứa các hành vi liên quan đến vị trí địa lý (Geo Context) và vùng chứa web interface (UI Context)

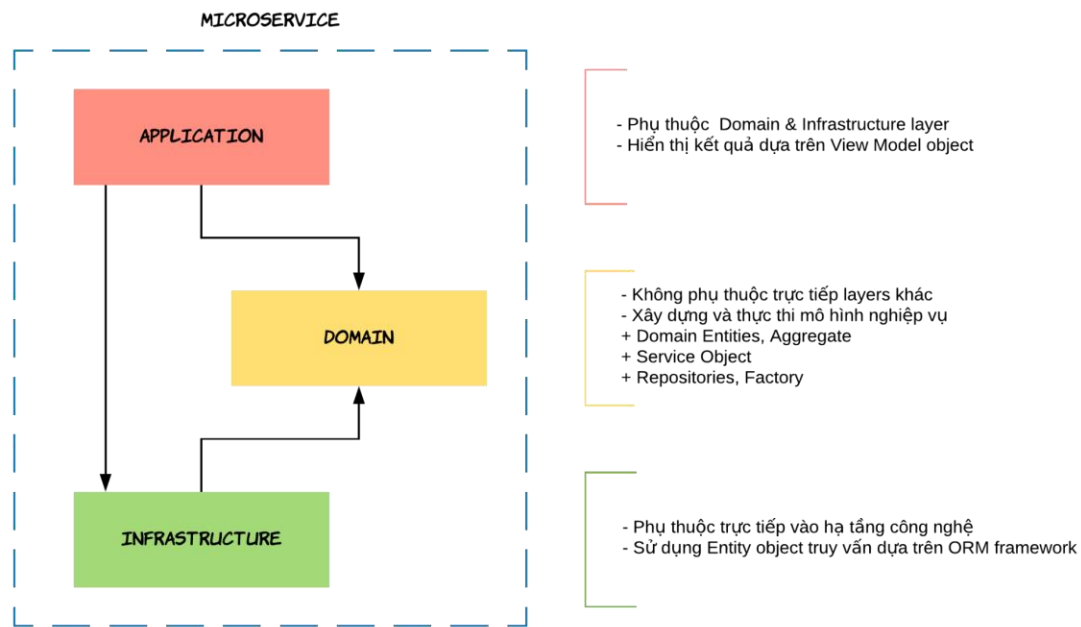
Dựa trên các phân vùng như vậy, ta có thể xác định ứng dụng Go-Hiking có bốn phần:

- Auth service thực hiện chức năng quản lý người dùng.
- Post service thực hiện chức năng quản lý bài viết.
- Geo service thực hiện chức năng liên quan đến vị trí địa lý.

- UI interface cung cấp các chức năng cho người dùng thông thường tương tác với hệ thống.



Hình 4-1: Kiến trúc của Go-Hiking
Các service được thiết kế theo cấu trúc Domain-Driven Design



Hình 4-2: Domain-Driven Design

4.3. Thiết kế dữ liệu

4.3.1. Doanh mục các bảng

Bảng 4-1: Các bảng của hệ thống

STT	Tên bảng	Mô tả	Service
1	user	Người dùng – lưu trữ thông tin của người dùng	auth
2	friend	Bạn bè – lưu trữ thông tin về bạn bè	auth
3	request	Yêu cầu kết bạn – lưu trữ thông tin về các yêu cầu kết bạn	auth
4	post	Bài viết – lưu trữ thông tin bài viết	post
5	location	Địa điểm – lưu trữ thông tin địa điểm người dùng thêm vào	post
6	comment	Bình luận – lưu trữ thông tin bình luận bài viết của người dùng	post
7	geoData	Vị trí địa lý – lưu trữ thông tin về các vị trí địa lý	geo

a) Bảng user (lưu trữ thông tin của người dùng)

Bảng 4-2: Bảng user

STT	Tên thuộc tính	Diễn giải
1	id	Mã người dùng
2	username	Tên tài khoản người dùng
3	password	Mật khẩu
4	email	Tài khoản email
5	roles	Đặc quyền của user
6	firstName	Tên người dùng
7	lastName	Họ người dùng
8	access	Trạng thái của tài khoản người dùng

b) Bảng friend (lưu trữ thông tin về bạn bè)

Bảng 4-3: Bảng friend

STT	Tên thuộc tính	Diễn giải
1	id	Mã của record
2	userId	Mã tài khoản của người dùng
3	friendUsername	username của người bạn

c) Bảng request (lưu trữ thông tin về các yêu cầu kết bạn)

Bảng 4-4: Bảng request

STT	Tên thuộc tính	Diễn giải
1	id	Mã của record
2	requestUsername	Tài khoản người tạo yêu cầu
3	targetUsername	Tài khoản người được yêu cầu
4	msg	Tin nhắn của người yêu cầu

d) Bảng post (lưu trữ thông tin bài viết)

Bảng 4-5: Bảng post

STT	Tên thuộc tính	Diễn giải
1	id	Mã của record
2	type	Loại bài viết
3	title	Tiêu đề bài viết
4	content	Nội dung bài viết
5	username	Tên người dùng tạo bài viết
6	access	Trạng thái của bài viết

e) Bảng location (lưu trữ thông tin địa điểm người dùng thêm vào)

Bảng 4-6: Bảng location

STT	Tên thuộc tính	Diễn giải
1	id	Mã của record
2	parentId	Mã của bài viết
3	lat	Vĩ độ của địa điểm
4	lng	Kinh độ của địa điểm
5	address	Địa chỉ của địa điểm
6	img	Liên kết đến hình ảnh của địa điểm
7	content	Nội dung bài viết
8	username	Tên người dùng tạo địa điểm

f) Bảng comment (lưu trữ thông tin bình luận bài viết của người dùng)

Bảng 4-7: Bảng comment

STT	Tên thuộc tính	Diễn giải
1	id	Mã của record
2	parentId	Mã của bài viết
3	content	Nội dung bài viết
4	username	Tên người dùng tạo bình luận

g) Bảng geoData (lưu trữ thông tin về các vị trí địa lý)

Bảng 4-8: Bảng geoData

STT	Tên thuộc tính	Diễn giải
1	id	Mã của record
2	latitude	Vĩ độ của địa điểm
3	longitude	Kinh độ của địa điểm
4	address	Địa chỉ của địa điểm

4.3.2. Các kiểu dữ liệu liên lạc giữa các service

```
User : {
  id: string (id của người dùng)
  username: string (username của người dùng, phải có)
  email: string (email của người dùng, không bắt buộc)
  roles: string (quyền truy cập của người dùng, mặc định: user)
  firstName: string (Tên người dùng, không bắt buộc)
  lastName: string (Họ người dùng, không bắt buộc)
  access: string (trạng thái trạng cá nhân của người, mặc định private)
}
```

```
Friend: {
  id: string (id của record)
  userId: string (id của người dùng)
  friendUsername: string (username của bạn người dùng)
}

FriendRequest: {
  id: string (id của record)
  requestUsername: string (Tài khoản người tạo yêu cầu, phải có)
  targetUsername: string (Tài khoản người được yêu cầu, phải có)
  msg: string (Tin nhắn của người yêu cầu, không bắt buộc)
}

Post: {
  id: string (Mã của bài viết)
  type: string (Loại bài viết, mặc định post)
  title: string (Tiêu đề bài viết, phải có)
  content: string (Nội dung bài viết, không bắt buộc)
  username: string (Tên người dùng tạo bài viết, phải có)
  access: string (Trạng thái của bài viết, mặc định private)
}

Location: {
  id: string (Mã của bài viết)
  parentId: string (Mã của bài viết, phải có)
  lat: double (Vĩ độ của địa điểm, mặc định: 0)
  lng: double (Kinh độ của địa điểm, mặc định: 0)
  address: string (Địa chỉ của địa điểm, mặc định: Unknown)
  img: string (Liên kết đến hình ảnh của địa điểm, không bắt buộc)
  content: string (Nội dung bài viết, không bắt buộc)
  username: string (Tên người dùng tạo bài viết, phải có)
}

Comment: {
  id: string (Mã của bài viết)
  parentId: string (Mã của bài viết, phải có)
  content: string (Nội dung bài viết, không bắt buộc)
  username: string (Tên người dùng tạo bài viết, phải có)
}

GeoData: {
  id: string (Mã của bài viết)
  latitude: double (Vĩ độ của địa điểm)
  longitude: double (Kinh độ của địa điểm)
  address: string (Địa chỉ của địa điểm)
}
```


4.4. Thiết kế theo chức năng

4.4.1. Auth Service

4.4.1.1. Đăng ký

➤ **Mục đích:** chức năng này cho phép người dùng bình thường (người chưa có tài khoản) có thể đăng ký tài khoản.

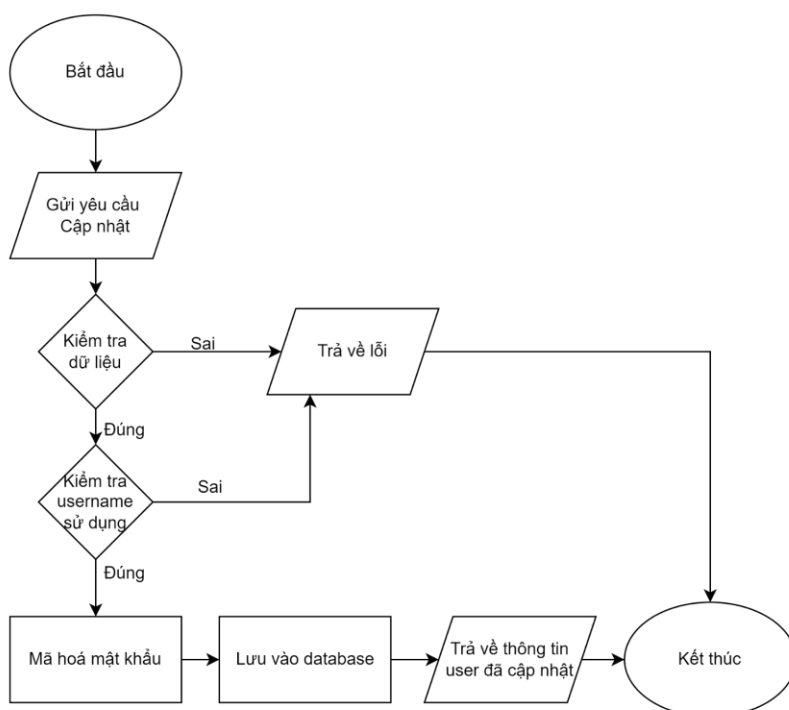
➤ **Đối tượng người sử dụng:** người dùng bình thường.

➤ **Request**

Bảng 4-9: Chức năng đăng ký

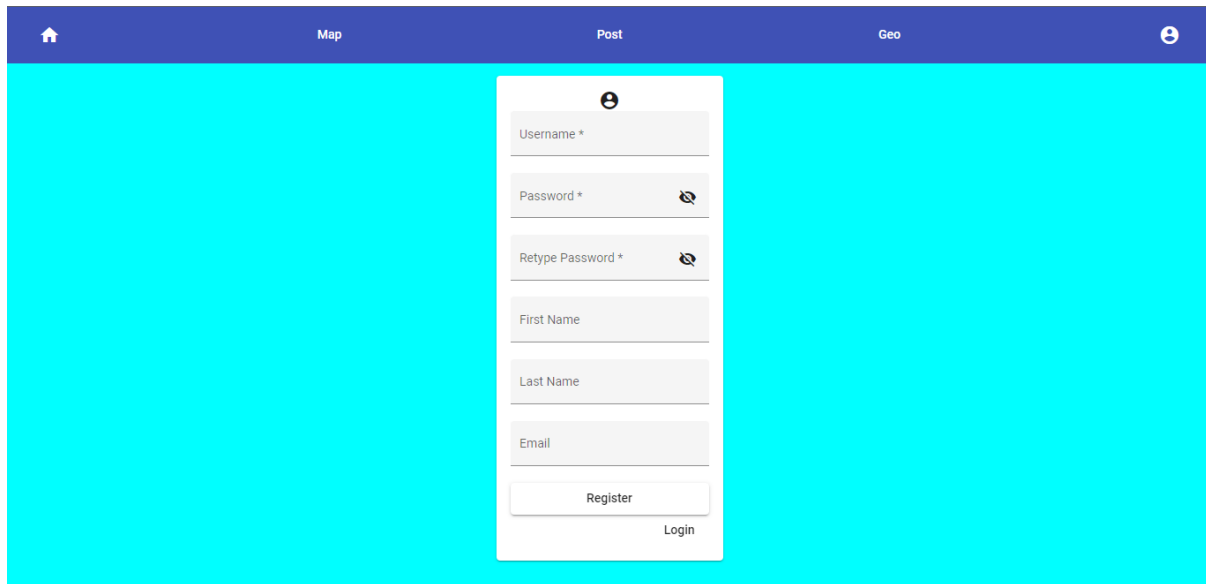
Tên yêu cầu	Đăng ký
Service sử dụng	auth
Request type	PUT
Api	/rest/auth/v1/user/create
Header	Không
Param	Không
Body	User
Kết quả	- User - Error

➤ **Sơ đồ quy trình:**



Hình 4-3: Sơ đồ xử lý chức năng đăng ký

➤ **Giao diện mẫu:**



Hình 4-4: Chức năng đăng ký

4.4.1.2. Đăng nhập (lấy access token)

➤ **Mục đích:** chức năng này cho phép người dùng lấy token để sử dụng các tính năng cần tài khoản.

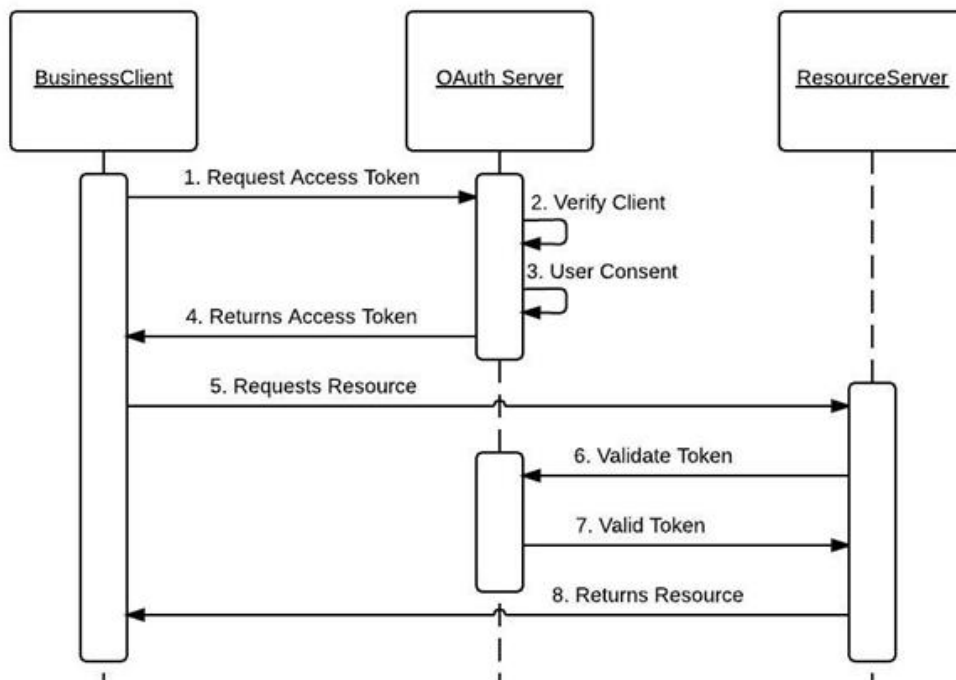
➤ **Đối tượng người sử dụng:** người dùng có tài khoản.

➤ **Request**

Bảng 4-10: Chức năng đăng nhập (lấy access token)

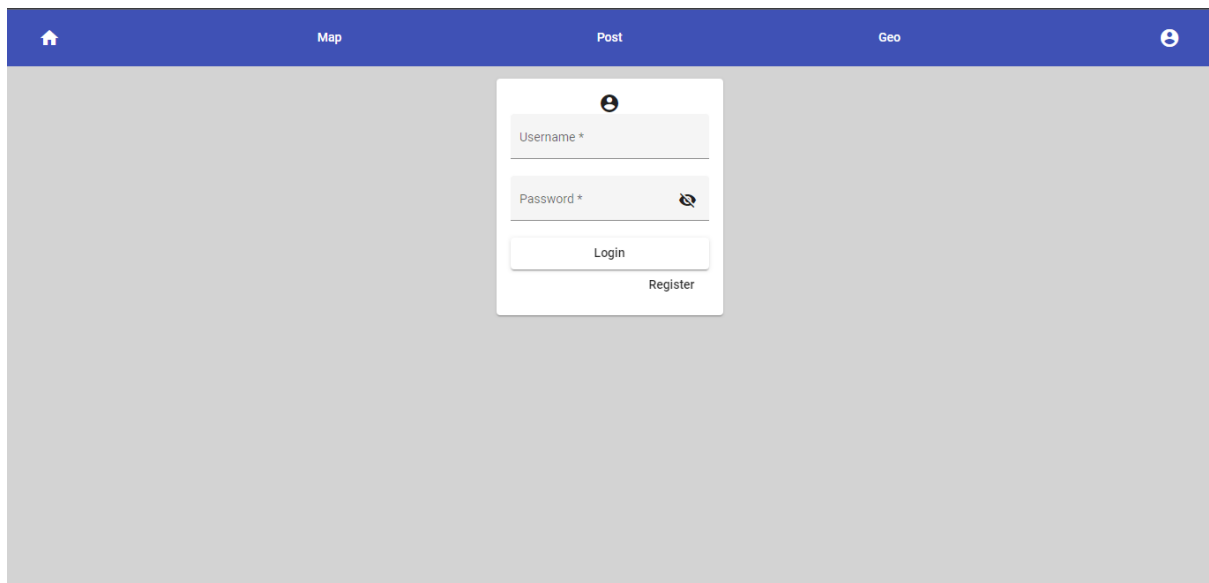
Tên yêu cầu	Lấy access token
Service sử dụng	auth
Request type	POST
Api	/oauth/token
Header	Basic auth: username: tài khoản của service password: mật khẩu của service
Param	username: tài khoản người dùng password: mật khẩu người dùng grant_type: password clientId: định danh client (không bắt buộc)
Body	Không
Kết quả	- Token - Error

➤ Sơ đồ quy trình[13]:



Hình 4-5: Sơ đồ xử lý chức năng đăng nhập (lấy access token)

➤ Giao diện mẫu:



Hình 4-6: Chức năng đăng nhập (lấy access token)

4.4.1.3. Huỷ token đang sử dụng

➤ **Mục đích:** chức năng này cho phép người dùng huỷ token khi không cần dùng đến nữa để tránh các vấn đề bảo mật.

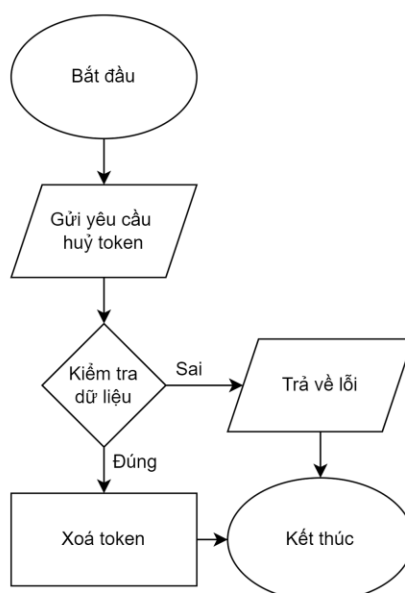
➤ **Đối tượng người sử dụng:** người dùng có tài khoản.

➤ **Request**

Bảng 4-11: Chức năng huỷ token

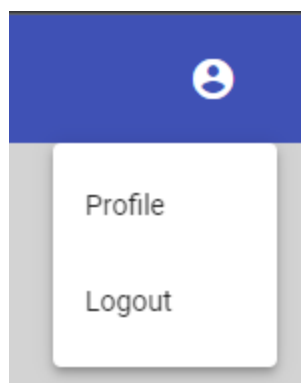
Tên yêu cầu	Huỷ auth token
Service sử dụng	auth
Request type	DELETE
Api	/rest/auth/v1/user/logout
Header	Authorization: token
Param	Không
Body	Không
Kết quả	- Ok - Error

➤ **Sơ đồ quy trình:**



Hình 4-7: Sơ đồ xử lý chức năng huỷ token

➤ **Giao diện mẫu:**



Hình 4-8: Chức năng huỷ token

4.4.1.4. Cập nhật thông tin cá nhân

➤ **Mục đích:** chức năng này cho phép người dùng bình thường (người chưa có tài khoản) có thể đăng ký tài khoản.

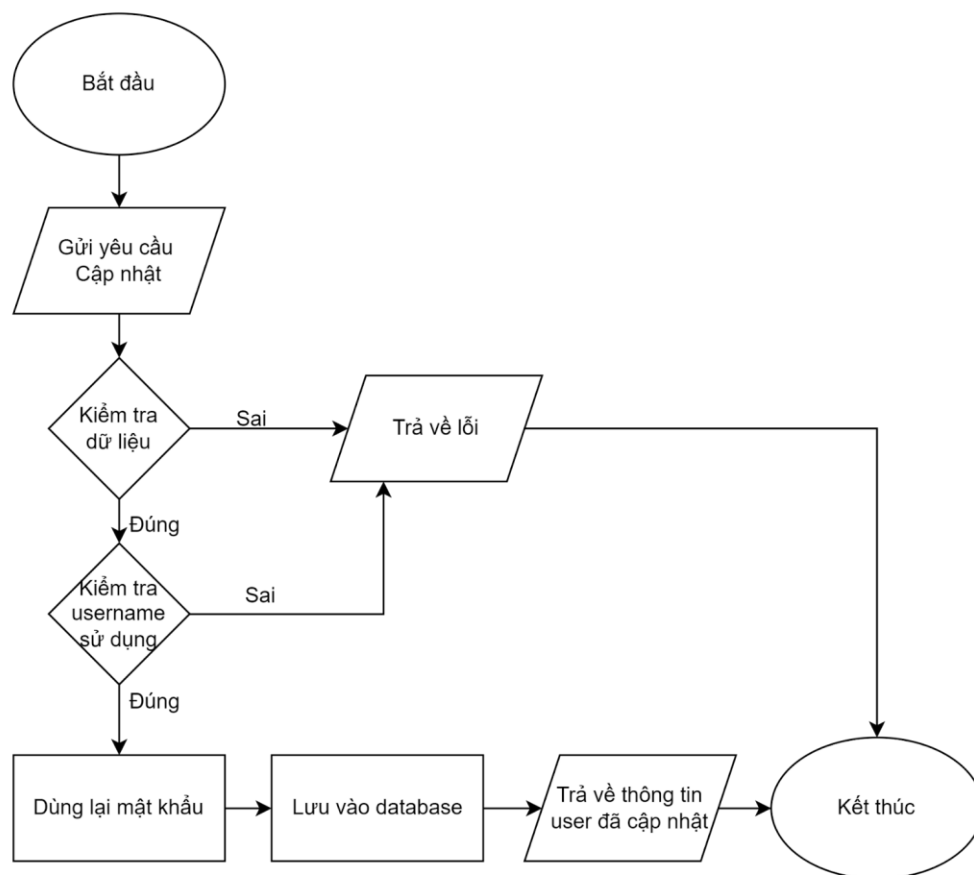
➤ **Đối tượng người sử dụng:** người dùng bình thường.

➤ **Request**

Bảng 4-12: Chức năng cập nhật thông tin cá nhân

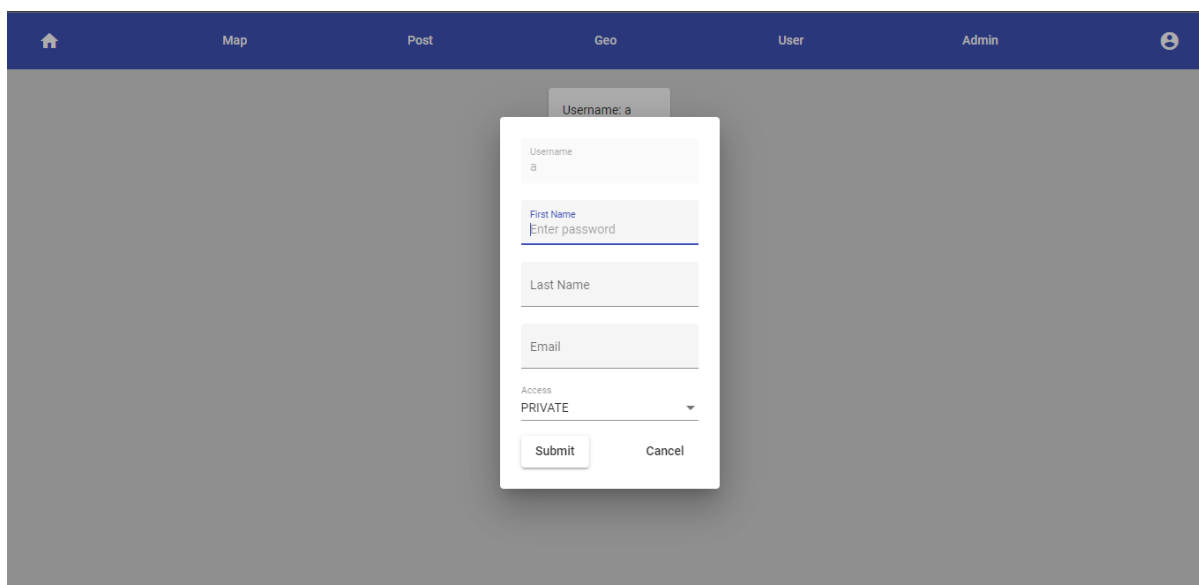
Tên yêu cầu	Cập nhật thông tin cá nhân
Service sử dụng	auth
Request type	PUT
Api	/rest/auth/v1/user/update
Header	Authorization: token
Param	Không
Body	User
Kết quả	- User - Error

➤ **Sơ đồ quy trình:**



Hình 4-9: Sơ đồ xử lý chức năng cập nhật thông tin cá nhân

➤ **Giao diện mẫu:**

The image shows a web application interface with a dark blue header containing navigation links: Home, Map, Post, Geo, User, Admin, and a user profile icon. A modal window is open in the center, titled 'Username: a'. It contains several input fields: 'Username' with the value 'a', 'First Name' with the placeholder 'Enter password', 'Last Name', and 'Email'. Below these is a dropdown menu for 'Access' currently set to 'PRIVATE'. At the bottom of the modal are 'Submit' and 'Cancel' buttons.

Hình 4-10: Chức năng cập nhật thông tin cá nhân

4.4.1.5. Lấy thông tin cá nhân

➤ **Mục đích:** chức năng này cho phép người dùng bình thường (người chưa có tài khoản) có thể đăng ký tài khoản.

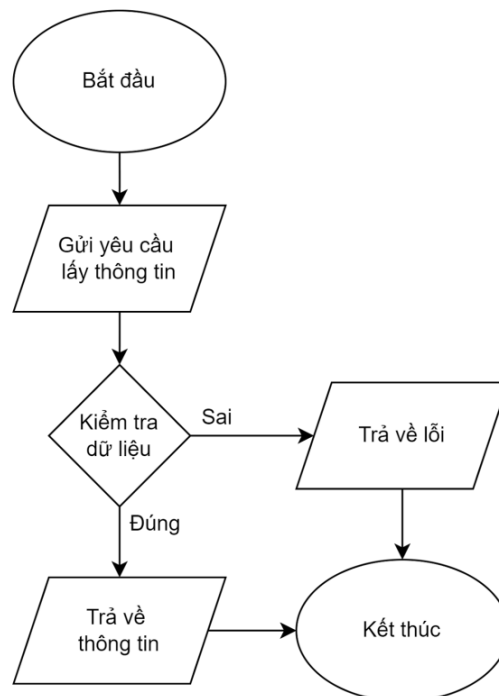
➤ **Đối tượng người sử dụng:** người dùng bình thường.

➤ **Request**

Bảng 4-13: Chức năng lấy thông tin cá nhân

Tên yêu cầu	Lấy thông tin cá nhân
Service sử dụng	auth
Request type	GET
Api	/rest/auth/v1/user/info
Header	Authorization: token
Param	Không
Body	Không
Kết quả	- User - Error

➤ **Sơ đồ quy trình:**



Hình 4-11: Sơ đồ xử lý chức năng lấy thông tin cá nhân

➤ **Giao diện mẫu:**

A sample user profile interface displayed within a light gray border. The profile information is shown in a white box with the following details: Username: a, First Name: , Last Name: , Email: , Access: PRIVATE, and Roles: Admin. At the bottom of the white box is a button labeled 'Edit profile'.

Hình 4-12: Chức năng lấy thông tin cá nhân

4.4.1.6. Lấy thông tin người dùng

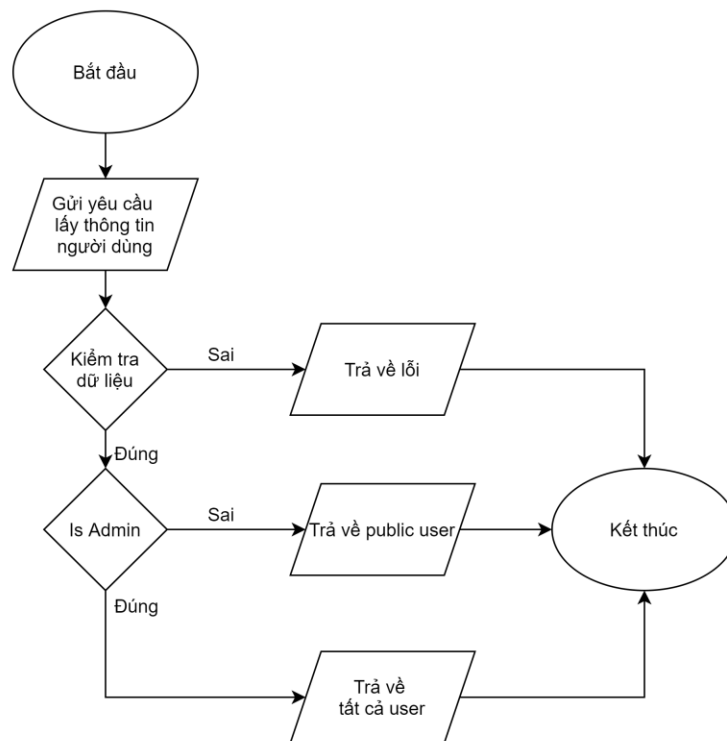
- **Mục đích:** chức năng này cho phép người dùng lấy thông tin người dùng khác.
- **Đối tượng người sử dụng:** người dùng bình thường.

➤ **Request**

Bảng 4-14: Chức năng lấy thông tin người dùng

Tên yêu cầu	Lấy thông tin người dùng
Service sử dụng	auth
Request type	GET
Api	/rest/auth/v1/users
Header	Authorization: token
Param	Không
Body	Không
Kết quả	- List User - Error

➤ **Sơ đồ quy trình:**



Hình 4-13: Sơ đồ xử lý chức năng lấy thông tin người dùng

➤ **Giao diện mẫu:**

Home	Map	Post	Geo	User	Admin	Profile
Username	Email	Role	First Name	Last Name		
admin	admin@mail.com	Admin	admin	admin		
jamsebrown	jamsebrown@mail.com	User	Jamse	Brown		
thienlam	thienlam@mail.com	User	Thien	Lam		
davidmiller	davidmiller@mail.com	User	David	Miller		
tinhlam	tinhlam@mail.com	User	Tinh	Lam		

Hình 4-14: Chức năng lấy thông tin người dùng

4.4.1.7. Tạo yêu cầu kết bạn

➤ **Mục đích:** chức năng này cho phép người dùng tạo yêu cầu kết bạn với người dùng khác.

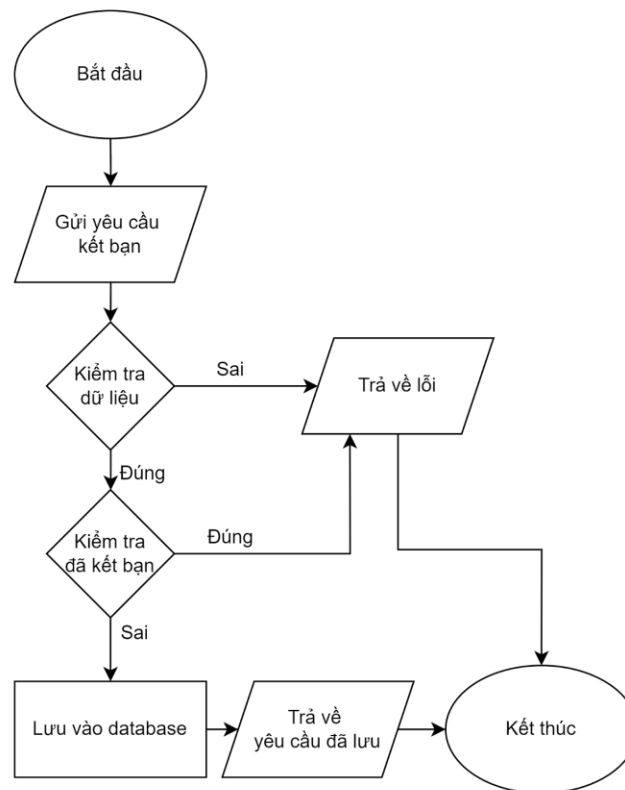
➤ **Đối tượng người sử dụng:** người dùng bình thường.

➤ **Request**

Bảng 4-15: Chức năng tạo yêu cầu kết bạn

Tên yêu cầu	Tạo yêu cầu kết bạn
Service sử dụng	auth
Request type	PUT
Api	/rest/auth/v1/user/friendRequest/upsert
Header	Authorization: token
Param	Không
Body	FriendRequest
Kết quả	- FriendRequest - Error

➤ **Sơ đồ quy trình:**



Hình 4-15: Sơ đồ xử lý chức năng tạo yêu cầu kết bạn

➤ **Giao diện mẫu:**

The image shows a sample user interface for creating a friend request. It displays the following information:

- Username: thienlam
- First Name: Thien
- Last Name: Lam
- Email: thienlam@mail.com
- Access: PUBLIC
- A link: [Add friend message](#)
- A text input field with a blue underline.
- Two buttons: 'Add friend' (blue) and 'Close' (white).

Hình 4-16: Chức năng tạo yêu cầu kết bạn

4.4.1.8. Lấy yêu cầu kết bạn

➤ **Mục đích:** chức năng này cho phép người dùng lấy các yêu cầu kết bạn với người dùng khác.

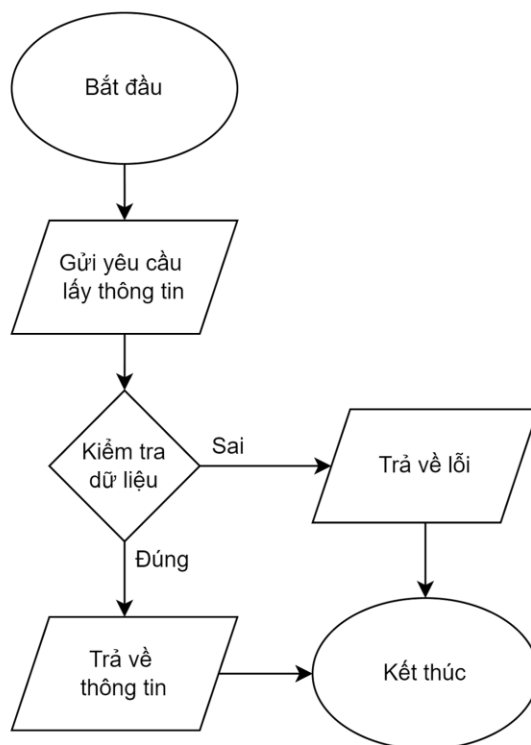
➤ **Đối tượng người sử dụng:** người dùng bình thường.

➤ **Request**

Bảng 4-16: Chức năng lấy yêu cầu kết bạn

Tên yêu cầu	Lấy yêu cầu kết bạn
Service sử dụng	auth
Request type	GET
Api	/rest/auth/v1/user/friendRequests
Header	Authorization: token
Param	Không
Body	Không
Kết quả	- List FriendRequest - Error

➤ **Sơ đồ quy trình:**



Hình 4-17: Sơ đồ xử lý chức năng lấy yêu cầu kết bạn

➤ **Giao diện mẫu:**

Friend requests(1)		
Request User	Target User	Message
admin	thienlam	Add friend

Hình 4-18: Chức năng lấy yêu cầu kết bạn

4.4.1.9. Trả lời yêu cầu kết bạn

➤ **Mục đích:** chức năng này cho phép người dùng được yêu cầu kết bạn với trả lời cho người dùng yêu cầu.

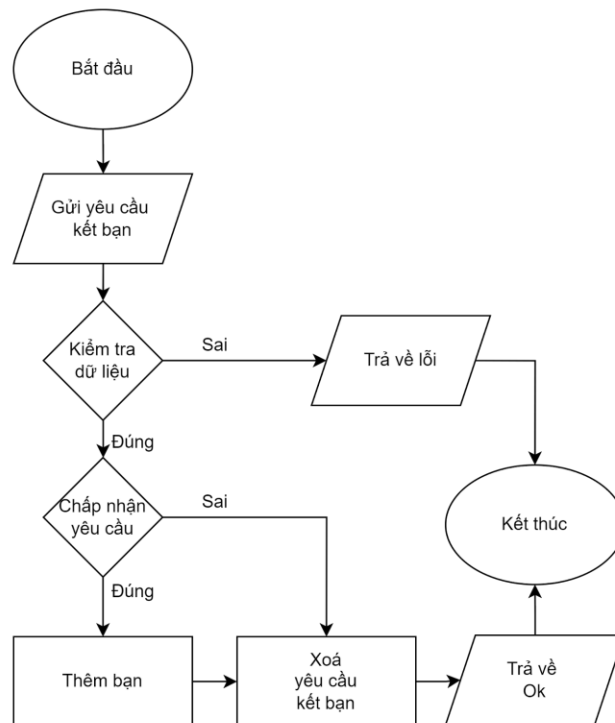
➤ **Đối tượng người sử dụng:** người dùng bình thường.

➤ **Request**

Bảng 4-17: Chức năng trả lời yêu cầu kết bạn

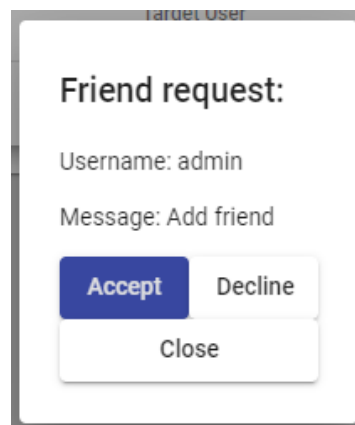
Tên yêu cầu	Trả lời yêu cầu kết bạn
Service sử dụng	auth
Request type	PUT
Api	/rest/auth/v1/user/friendRequest/answer
Header	Authorization: token
Param	requestId: id của yêu cầu kết bạn answer: chấp nhận hoặc từ chối yêu cầu
Body	Không
Kết quả	- Ok - Error

➤ **Sơ đồ quy trình:**



Hình 4-19: Sơ đồ xử lý chức năng trả lời yêu cầu kết bạn

➤ **Giao diện mẫu:**



Hình 4-20: Chức năng trả lời yêu cầu kết bạn

4.4.1.10. Lấy thông tin bạn bè

➤ **Mục đích:** chức năng này cho phép người dùng lấy các người dùng đã kết bạn.

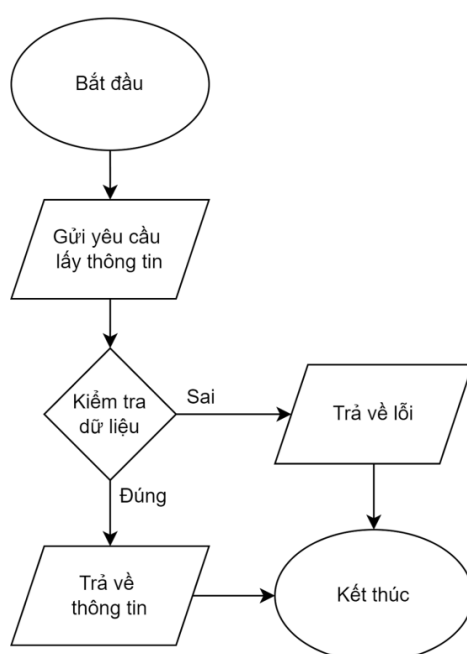
➤ **Đối tượng người sử dụng:** người dùng bình thường.

➤ **Request**

Bảng 4-18: Chức năng lấy thông tin bạn bè

Tên yêu cầu	Lấy yêu cầu kết bạn
Service sử dụng	auth
Request type	GET
Api	/rest/auth/v1/user/friends
Header	Authorization: token
Param	Không
Body	Không
Kết quả	- List Friend - Error

➤ **Sơ đồ quy trình:**



Hình 4-21: Sơ đồ xử lý chức năng lấy thông tin bạn bè

➤ **Giao diện mẫu:**

Friend(1)				
Username	Email	Role	First Name	Last Name
admin	admin@mail.com	Admin	admin	admin

Hình 4-22: Chức năng lấy thông tin bạn bè

4.4.1.11. Xóa bạn bè

➤ **Mục đích:** chức năng này cho phép người dùng xóa kết bạn với người dùng đã kết bạn.

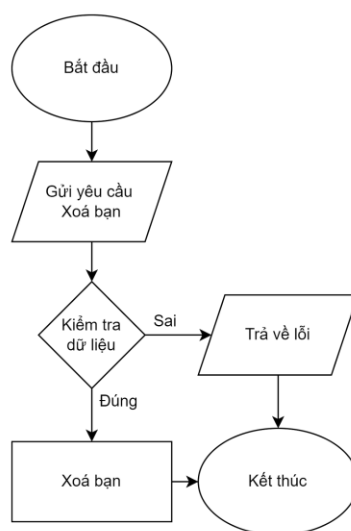
➤ **Đối tượng người sử dụng:** người dùng bình thường.

➤ **Request**

Bảng 4-19: Chức năng xoá bạn

Tên yêu cầu	Xoá bạn
Service sử dụng	auth
Request type	DELETE
Api	/rest/auth/v1/user/friend/delete
Header	Authorization: token
Param	Không
Body	User
Kết quả	- Ok - Error

➤ **Sơ đồ quy trình:**



Hình 4-23: Sơ đồ xử lý chức năng xoá bạn

➤ **Giao diện mẫu:**

Username: admin

First Name: admin

Last Name: admin

Email: admin@mail.com

Access: PRIVATE

Roles: Admin

Delete
Close

Hình 4-24: Chức năng xoá bạn

4.4.2. Post Service

4.4.2.1. Upload hình ảnh lên server

➤ **Mục đích:** chức năng này cho phép người dùng tải hình ảnh lên server.

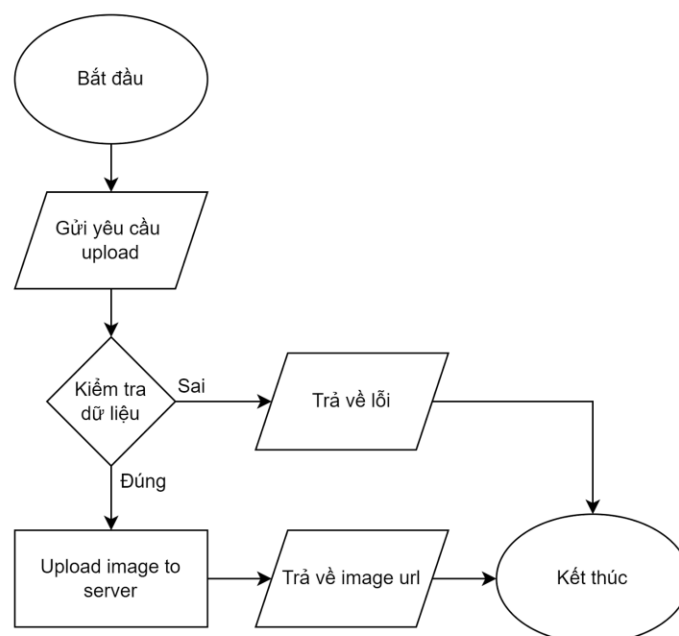
➤ **Đối tượng người sử dụng:** người dùng bình thường.

➤ **Request**

Bảng 4-20: Chức năng upload ảnh

Tên yêu cầu	Upload hình ảnh lên server
Service sử dụng	auth, post
Request type	PUT
Api	/rest/post/v1/image
Header	Authorization: token
Param	Không
Body	file: hình ảnh cần tải lên
Kết quả	- Image Url - Error

➤ **Sơ đồ quy trình:**



Hình 4-25: Sơ đồ xử lý chức năng upload ảnh

4.4.2.2. Xóa hình ảnh khỏi server

➤ **Mục đích:** chức năng này cho phép người dùng xóa hình ảnh khỏi server.

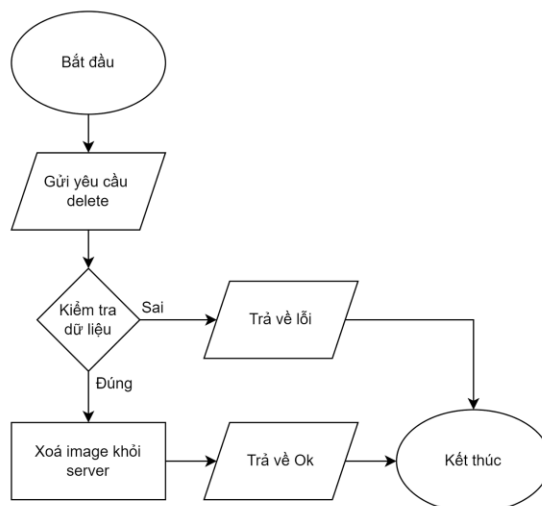
➤ **Đối tượng người sử dụng:** người dùng bình thường.

➤ **Request**

Bảng 4-21: Chức năng xoá ảnh

Tên yêu cầu	Xoá hình ảnh khỏi server
Service sử dụng	auth, post
Request type	PUT
Api	/rest/post/v1/image
Header	Authorization: token
Param	url: url của hình ảnh cần xoá
Body	Không
Kết quả	- Ok - Error

➤ **Sơ đồ quy trình:**



Hình 4-26: Sơ đồ xử lý chức năng xoá ảnh

4.4.2.3. Tạo bài hoặc sửa bài viết

➤ **Mục đích:** chức năng này cho phép người dùng tạo bài viết trên hệ thống.

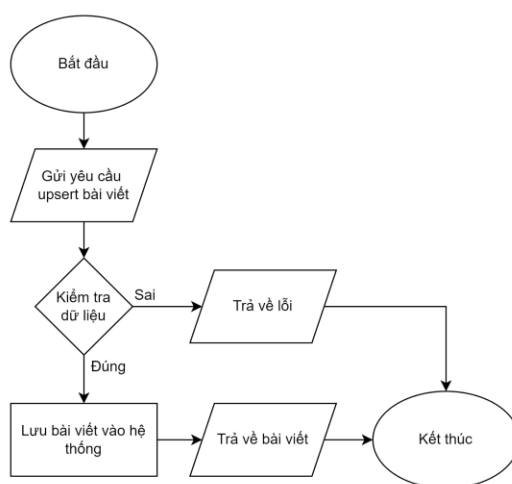
➤ **Đối tượng người sử dụng:** người dùng bình thường.

➤ **Request**

Bảng 4-22: Chức năng tạo bài viết

Tên yêu cầu	Tạo bài viết
Service sử dụng	auth
Request type	PUT
Api	/rest/post/v1/post/upsert
Header	Authorization: token
Param	Không
Body	Post
Kết quả	- Post - Error

➤ **Sơ đồ quy trình:**



Hình 4-27: Sơ đồ xử lý chức năng tạo bài viết

➤ **Giao diện mẫu:**

Hình 4-28: Chức năng tạo bài viết

4.4.2.4. Lấy bài viết

➤ **Mục đích:** chức năng này cho phép người dùng lấy bài viết trên hệ thống.

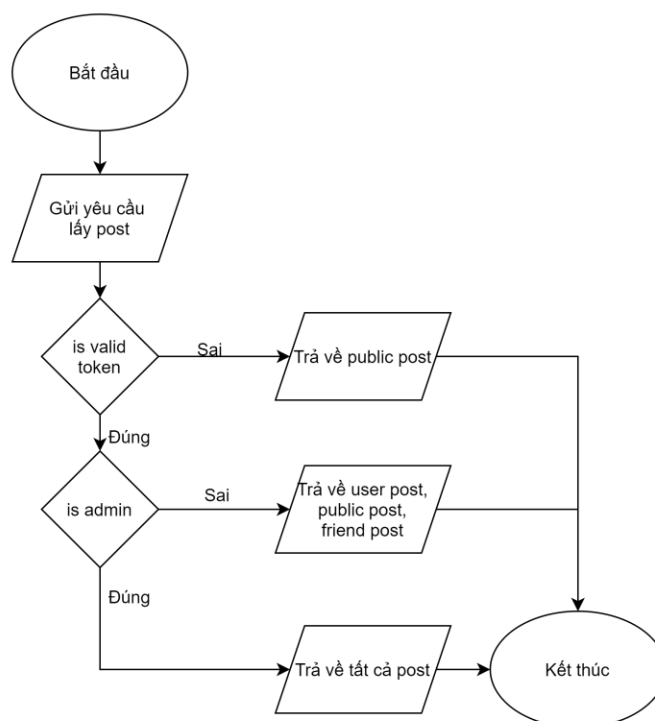
➤ **Đối tượng người sử dụng:** tất cả người dùng.

➤ **Request**

Bảng 4-23: Chức năng lấy bài viết

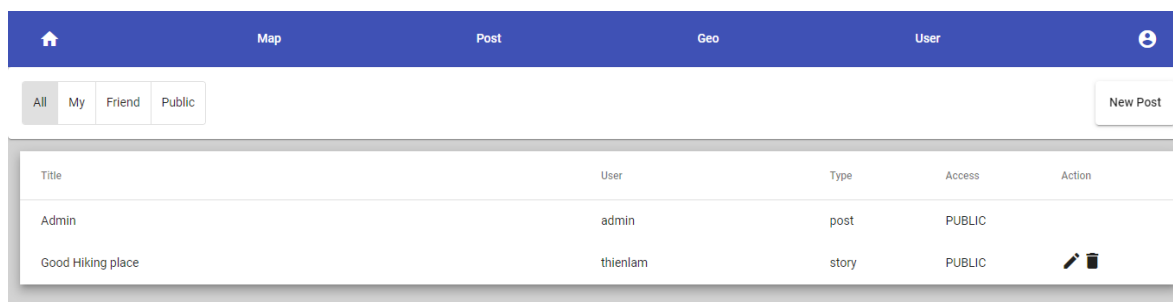
Tên yêu cầu	Lấy bài viết
Service sử dụng	auth, post
Request type	GET
Api	/rest/post/v1/posts
Header	Authorization: token (Không bắt buộc)
Param	Không
Body	Không
Kết quả	- List Post

➤ **Sơ đồ quy trình:**



Hình 4-29: Sơ đồ xử lý chức năng lấy bài viết

➤ **Giao diện mẫu:**



Hình 4-30: Chức năng lấy bài viết

4.4.2.5. Xóa bài viết

➤ **Mục đích:** chức năng này cho phép người dùng xóa bài viết trên hệ thống.

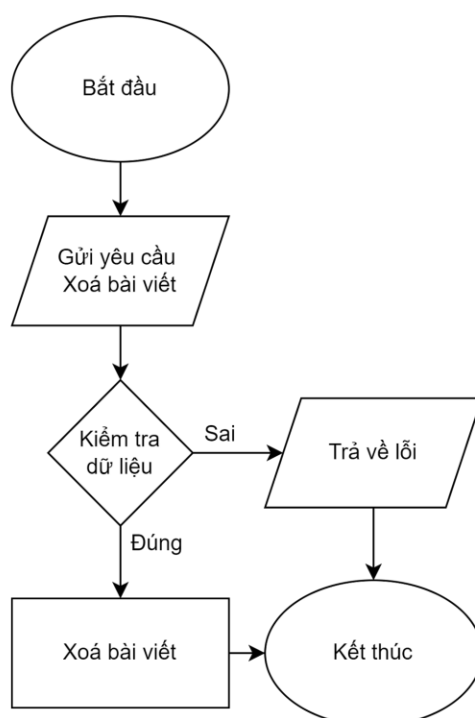
➤ **Đối tượng người sử dụng:** tất cả người dùng.

➤ **Request**

Bảng 4-24: Chức năng xoá bài viết

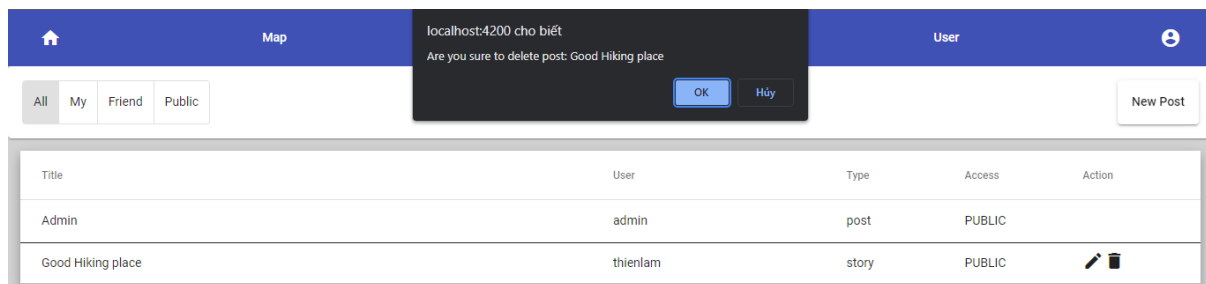
Tên yêu cầu	Xoá bài viết
Service sử dụng	auth, post
Request type	DELETE
Api	/rest/post/v1/post/{id}
Header	Authorization: token
Param	Không
Body	Không
Kết quả	- Ok - Error

➤ **Sơ đồ quy trình:**



Hình 4-31: Sơ đồ xử lý chức năng xoá bài viết

➤ **Giao diện mẫu:**



Hình 4-32: Chức năng xoá bài viết

4.4.2.6. Tạo bài hoặc sửa địa điểm trong bài viết

➤ **Mục đích:** chức năng này cho phép người dùng tạo địa điểm trong bài viết trên hệ thống.

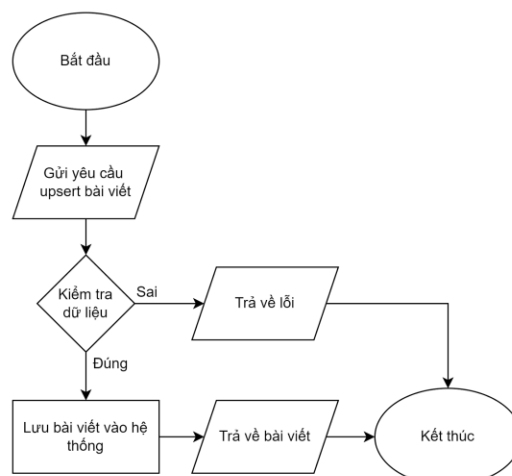
➤ **Đối tượng người sử dụng:** người dùng bình thường.

➤ **Request**

Bảng 4-25: Chức năng tạo hoặc sửa địa điểm trong bài viết

Tên yêu cầu	Tạo bài viết
Service sử dụng	auth
Request type	PUT
Api	/rest/post/v1/location/upsert
Header	Authorization: token
Param	Không
Body	Location
Kết quả	- Location - Error

➤ **Sơ đồ quy trình:**



Hình 4-33: Sơ đồ xử lý chức năng tạo và cập nhật địa điểm

➤ **Giao diện mẫu:**

Hình 4-34: Chức năng tạo và cập nhật địa điểm

4.4.2.7. Lấy địa điểm của bài viết

➤ **Mục đích:** chức năng này cho phép người dùng lấy địa điểm của bài viết trên hệ thống.

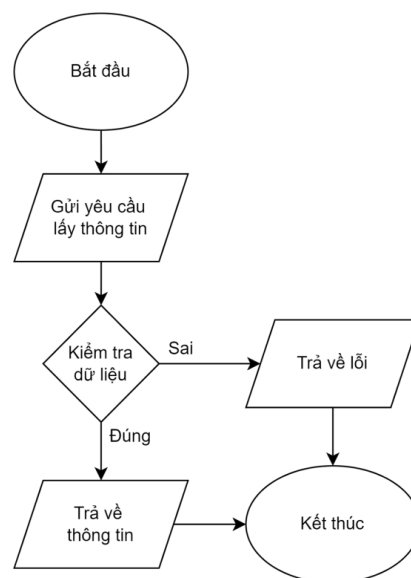
➤ **Đối tượng người sử dụng:** tất cả người dùng.

➤ **Request**

Bảng 4-26: Chức năng lấy địa điểm trong bài viết

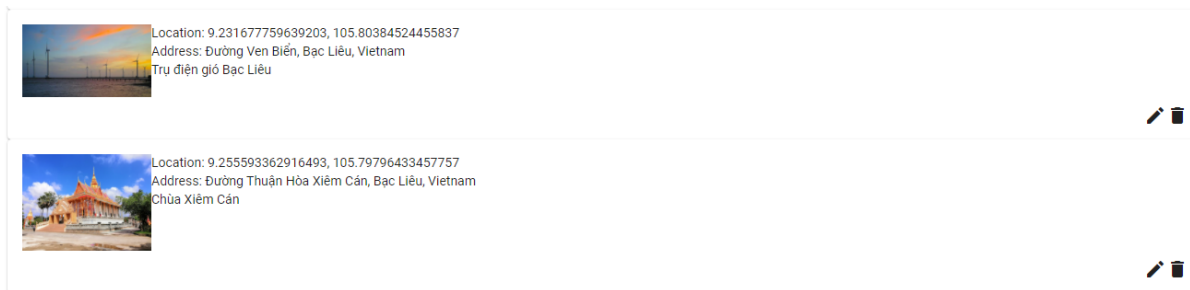
Tên yêu cầu	Lấy bài viết
Service sử dụng	auth, post
Request type	GET
Api	/rest/post/v1/locations/{parentId}
Header	Authorization: token (Không bắt buộc)
Param	Không
Body	Không
Kết quả	- List Location

➤ **Sơ đồ quy trình:**



Hình 4-35: Sơ đồ xử lý chức năng lấy địa điểm trong bài viết

➤ **Giao diện mẫu:**



Hình 4-36: Chức năng lấy địa điểm trong bài viết

4.4.2.8. Xóa địa điểm của bài viết

➤ **Mục đích:** chức năng này cho phép người dùng xóa bài viết trên hệ thống.

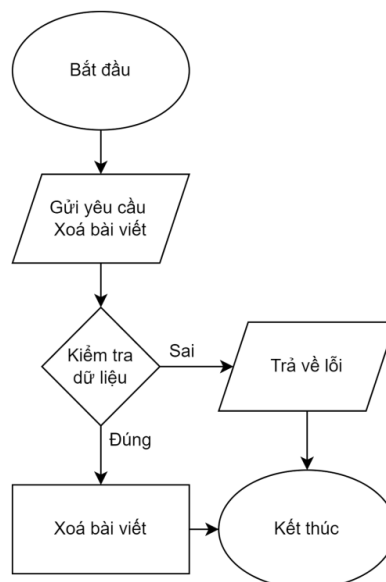
➤ **Đối tượng người sử dụng:** tất cả người dùng.

➤ **Request**

Bảng 4-27: Chức năng xóa địa điểm trong bài viết

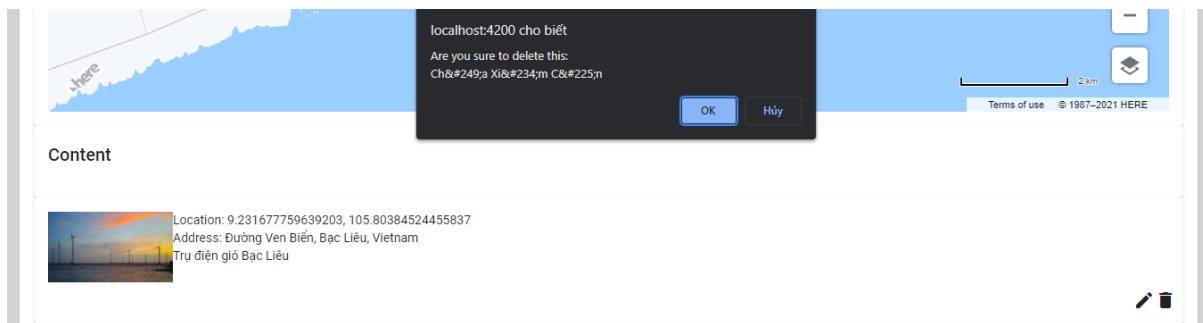
Tên yêu cầu	Xóa bài viết
Service sử dụng	auth, post
Request type	DELETE
Api	/rest/post/v1/location/{id}
Header	Authorization: token
Param	Không
Body	Không
Kết quả	- Ok - Error

➤ **Sơ đồ quy trình:**



Hình 4-37: Sơ đồ xử lý chức năng xóa địa điểm trong bài viết

➤ **Giao diện mẫu:**



Hình 4-38: Chức năng xoá địa điểm trong bài viết

4.4.2.9. Tạo bài hoặc sửa bình luận trong bài viết

➤ **Mục đích:** chức năng này cho phép người dùng tạo bình luận trong bài viết trên hệ thống.

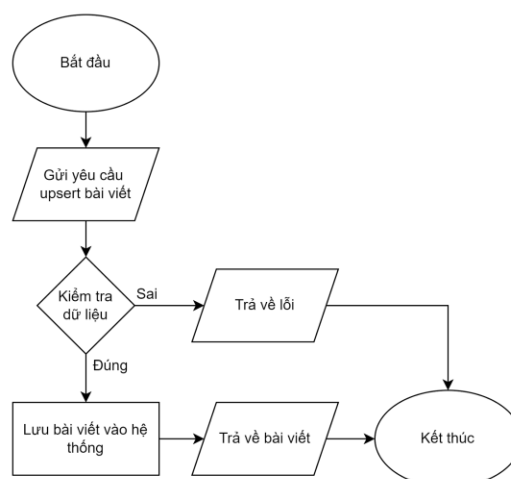
➤ **Đối tượng người sử dụng:** người dùng bình thường.

➤ **Request**

Bảng 4-28: Chức năng bình luận trong bài viết

Tên yêu cầu	Tạo bài viết
Service sử dụng	auth
Request type	PUT
Api	/rest/post/v1/comment/upsert
Header	Authorization: token
Param	Không
Body	Comment
Kết quả	- Comment - Error

➤ **Sơ đồ quy trình:**



Hình 4-39: Sơ đồ xử lý chức năng bình luận trong bài viết

➤ **Giao diện mẫu:**

Hình 4-40: Chức năng bình luận trong bài viết

4.4.2.10. Lấy bình luận của bài viết

➤ **Mục đích:** chức năng này cho phép người dùng lấy bình luận của bài viết trên hệ thống.

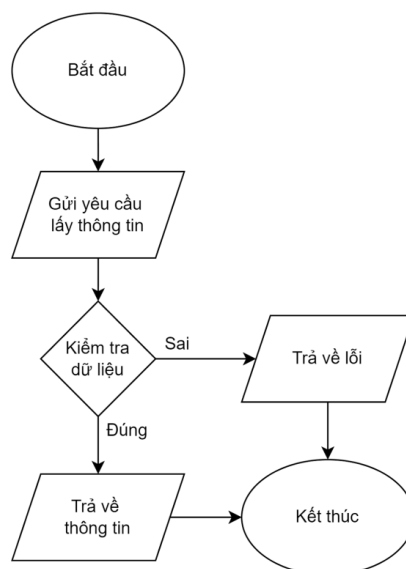
➤ **Đối tượng người sử dụng:** tất cả người dùng.

➤ **Request**

Bảng 4-29: Chức năng lấy bình luận trong bài viết

Tên yêu cầu	Lấy bình luận
Service sử dụng	auth, post
Request type	GET
Api	/rest/post/v1/comments/{parentId}
Header	Authorization: token (Không bắt buộc)
Param	Không
Body	Không
Kết quả	- List Comment

➤ **Sơ đồ quy trình:**



Hình 4-41: Sơ đồ xử lý chức năng lấy bình luận trong bài viết

➤ **Giao diện mẫu:**

Comment

User: admin
Good post

User: thienlam
Thank

Hình 4-42: Chức năng lấy bình luận trong bài viết

4.4.2.11. Xóa bình luận trong bài viết

➤ **Mục đích:** chức năng này cho phép người dùng xóa bình luận trong bài viết trên hệ thống.

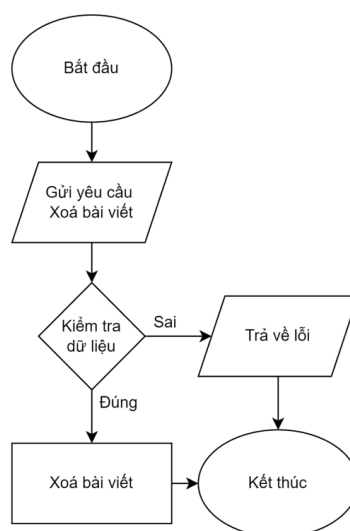
➤ **Đối tượng người sử dụng:** tất cả người dùng.

➤ **Request**

Bảng 4-30: Chức năng xóa bình luận trong bài viết

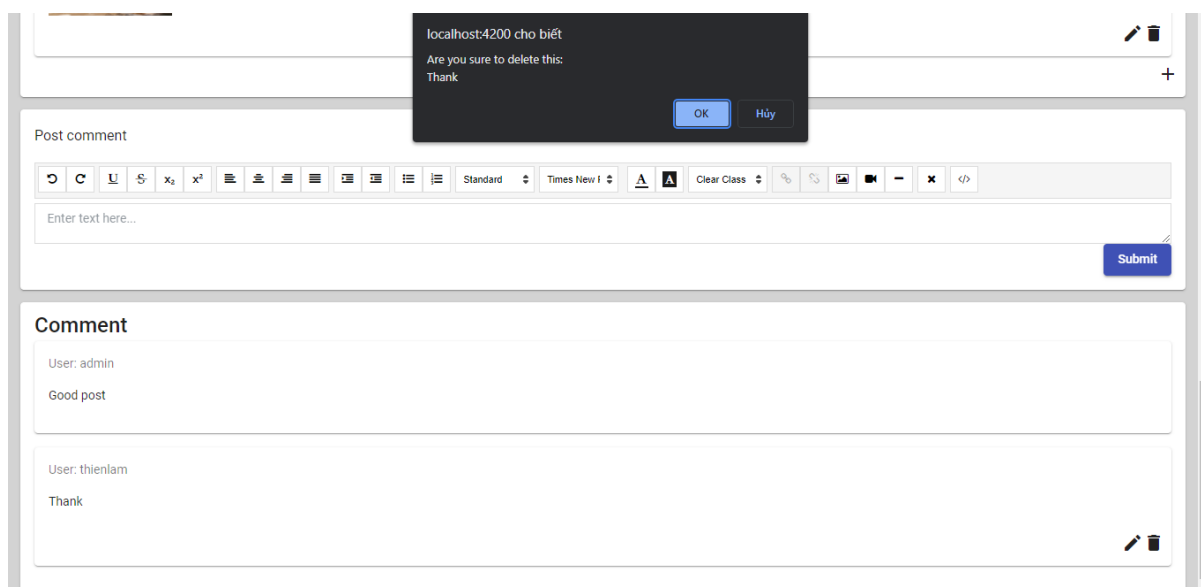
Tên yêu cầu	Xóa bình luận
Service sử dụng	auth, post
Request type	DELETE
Api	/rest/post/v1/comment/{id}
Header	Authorization: token
Param	Không
Body	Không
Kết quả	- Ok - Error

➤ **Sơ đồ quy trình:**



Hình 4-43: Sơ đồ xử lý chức năng xóa bình luận trong bài viết

➤ **Giao diện mẫu:**



Hình 4-44: Chức năng xoá bình luận trong bài viết

4.4.3. Geo Service

4.4.3.1. Tìm địa điểm dựa trên vị trí địa lý

➤ **Mục đích:** chức năng này cho phép người dùng tìm địa điểm dựa trên vị trí địa lý.

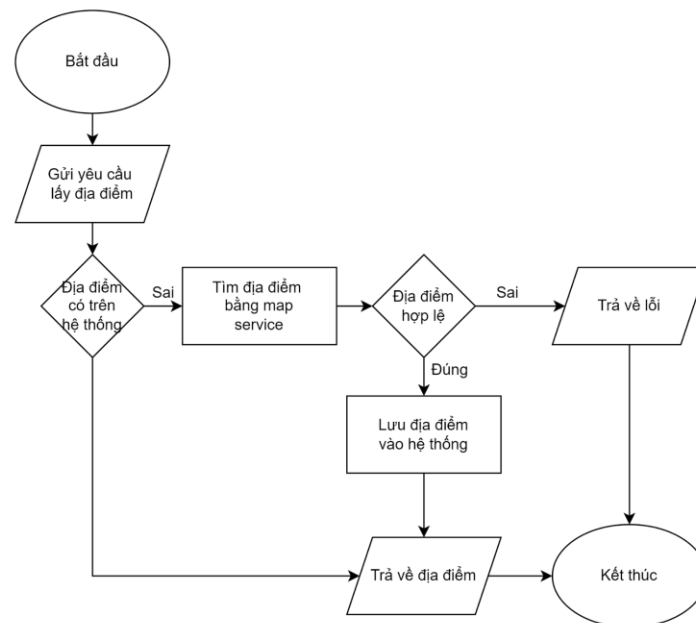
➤ **Đối tượng người sử dụng:** tất cả người dùng.

➤ **Request**

Bảng 4-31: Chức năng tìm địa điểm dựa trên vị trí địa lý

Tên yêu cầu	Tìm địa điểm dựa trên vị trí địa lý.
Service sử dụng	geo
Request type	GET
Api	/rest/geo/v1/geo
Header	Không
Param	lat: kinh độ lng: vĩ độ offset: độ sai lệch
Body	Không
Kết quả	- GeoData - Error

➤ **Sơ đồ quy trình:**



Hình 4-45: Sơ đồ xử lý chức năng tìm địa điểm dựa trên vị trí địa lý

4.4.3.2. Lấy tất cả địa điểm đã lưu trên server

➤ **Mục đích:** chức năng này cho phép người dùng lấy tất cả địa điểm đã lưu trên server.

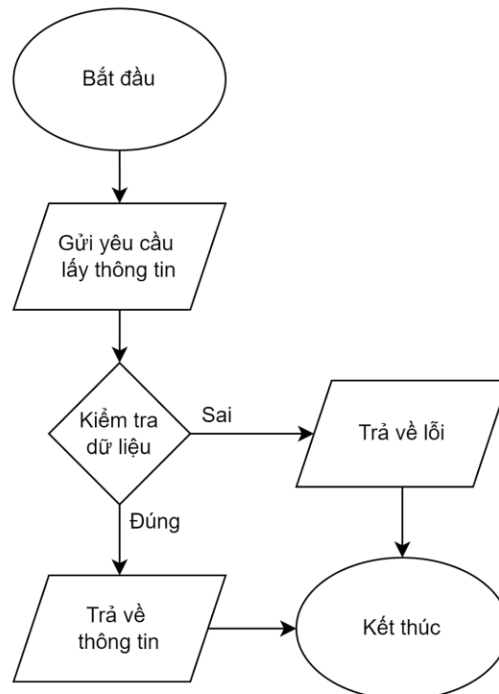
➤ **Đối tượng người sử dụng:** người dùng bình thường.

➤ **Request**

Bảng 4-32: Chức năng lấy địa điểm đã lưu trên server

Tên yêu cầu	Lấy tất cả địa điểm đã lưu trên server
Service sử dụng	geo
Request type	GET
Api	/rest/geo/v1/locations
Header	Không
Param	Không
Body	Không
Kết quả	- List GeoData - Error

➤ **Sơ đồ quy trình:**



Hình 4-46: Sơ đồ xử lý chức năng lấy địa điểm đã lưu trên server

CHƯƠNG 5: KIỂM THỬ VÀ ĐÁNH GIÁ KẾT QUẢ

5.1. Giới thiệu

5.1.1. Mục tiêu

- Phát hiện lỗi và kiểm tra hệ thống có hoạt động đúng theo yêu cầu đã nêu ra trong đặc tả hay chưa.
- Liệt kê kết quả có được sau kiểm thử.
- Làm tài liệu cho giai đoạn bảo trì.

5.1.2. Phạm vi kiểm thử

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

Kiểm thử thiết kế: kiểm tra giao diện thiết kế có đúng với đặc tả.

Kiểm thử chấp nhận: kiểm thử chức năng hệ thống có hoạt động và đáp ứng đặc tả yêu cầu.

Kiểm thử chức năng: kiểm thử chức năng có xử lý đúng dữ liệu.

Kiểm thử cài đặt: tìm và sửa lỗi xảy ra kiểm thử.

5.2. Kế hoạch kiểm thử

5.2.1. Các tính năng sẽ được kiểm thử

- Đăng ký
- Đăng nhập
- Tạo bài viết
- Tạo địa điểm cho bài viết
- Bình luận bài viết
- Lấy địa điểm

5.2.2. Các tính năng không được kiểm thử

Cập nhật thông tin cá nhân, lấy thông tin cá nhân, lấy thông tin người dùng, tạo yêu cầu kết bạn, lấy yêu cầu kết bạn, trả lời yêu cầu kết bạn, lấy thông tin bạn bè, xoá bạn, upload hình ảnh, xoá hình ảnh, lấy bài viết, xoá bài viết, lấy địa điểm bài viết, xoá địa điểm bài viết, lấy bình luận bài viết, xoá bình luận bài viết, lấy các địa điểm đã lưu trên hệ thống.

5.2.3. Cách tiếp cận

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

5.2.4. Tiêu chí kiểm thử thành công / thất bại

➤ Tiêu chí kiểm thử thành công: kết quả cuối cùng của chuỗi các thao tác đúng như mong đợi đặt ra ban đầu.

➤ Tiêu chí kiểm thử thất bại: kết quả kiểm thử không như mong đợi, xuất hiện lỗi sai lệch so với kì vọng ban đầu.

5.2.5. Tiêu chí đình chỉ và yêu cầu bắt đầu lại

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

5.3. Quản lý kiểm thử

5.3.1. 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

5.3.2. Môi trường

- Phần cứng máy tính: CPU Core i3, RAM 16GB, HDD 500GB.
- Hệ điều hành: Windows 10.
- Phần mềm: Google Chrome, Microsoft Edge.
- Thiết bị có kết nối internet.

5.3.3. Trách nhiệm quyền hạn

Bảng 5-1: Trách nhiệm quyền hạn

Thành viên	Vai trò					
	Quản lý	Thiết kế	Chuẩn bị	Thực hiện	Chứng kiến	Kiểm tra
Trần Công Minh	x	x	x	x	x	x

5.3.4. Giao tiếp giữa các nhóm liên quan

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

5.3.5. Tài nguyên và sự cấp phát chúng

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

5.3.6. Huấn luyện

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

5.3.7. Kế hoạch dự đoán và chi phí

Bảng 5-2: Kế hoạch dự đoán và chi phí

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

5.3.8. Các rủi ro*Bảng 5-3: Các rủi ro*

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

5.3.9. Kịch bản kiểm thử*Bảng 5-4: Kịch bản kiểm thử*

Mã	Tài liệu đặc tả	Mô tả kịch bản kiểm thử	Mức quan trọng	Số trường hợp kiểm thử
TS_GH_01	Mục 1.1.2.3.1	Kiểm tra người dùng có thể đăng ký tài khoản	P1	5
TS_GH_02	Mục 1.1.2.3.2	Kiểm tra người dùng có thể đăng nhập vào hệ thống	P1	3
TS_GH_03	Mục 1.1.2.3.3	Kiểm tra người dùng có thể tạo bài viết	P1	3
TS_GH_04	Mục 1.1.2.3.5	Kiểm tra người dùng có thể tạo địa điểm cho bài viết	P1	4
TS_GH_05	Mục 1.1.2.3.8	Kiểm tra người dùng có thể bình luận cho bài viết	P1	5
TS_GH_06	Mục 1.1.2.3.15	Kiểm tra người dùng có thể tra cứu địa điểm dựa vào vị trí địa lý	P1	3

5.4. Các trường hợp kiểm thử

Sau khi thực hiện kiểm thử và sửa lỗi, các trường hợp kiểm thử cuối cùng được trình bày bên dưới.

5.4.1. Đăng ký

- Tiền điều kiện: Phải có kết nối internet và đang truy cập vào trang đăng ký.
- Kịch bản kiểm thử:

Bảng 5-5: Kiểm thử chức năng đăng ký

Mã số test case	Mô tả	Điều kiện	Dữ liệu kiểm thử	Các bước thực hiện	Kết quả mong đợi	Kết quả thực tế	Đánh giá
TC_GH_DK_01	Kiểm thử đăng ký với dữ liệu hợp lệ	Tài khoản chưa tồn tại trước	- username: user1 - password: password1 - Các thông tin còn lại có thể có hoặc không	1. Cài đặt Postman 2. Gửi dữ liệu kiểm thử vào hệ thống: - Type: PUT - API: /rest/auth/v1/user/create	Trả về User có dữ liệu giống dữ liệu kiểm thử	Trả về User có dữ liệu giống dữ liệu kiểm thử	Pass
TC_GH_DK_02	Kiểm thử đăng ký với dữ liệu rỗng	Không cần điều kiện		1. Cài đặt Postman 2. Gửi dữ liệu kiểm thử vào hệ thống: - Type: PUT - API: /rest/auth/v1/user/create	Trả về lỗi	Trả về lỗi	Pass
TC_GH_DK_03	Kiểm thử đăng ký với dữ liệu không hợp lệ (username rỗng)	Không cần điều kiện	- username: - password: password1 - Các thông tin còn lại có thể có hoặc không	1. Cài đặt Postman 2. Gửi dữ liệu kiểm thử vào hệ thống: - Type: PUT - API: /rest/auth/v1/user/create	Trả về lỗi	Trả về lỗi	Pass

TC_GH_DK_04	Kiểm thử đăng ký với dữ liệu không hợp lệ (password rỗng)	Không cần điều kiện	- username: user2 - password: - Các thông tin còn lại có thể có hoặc không	1. Cài đặt Postman 2. Gửi dữ liệu kiểm thử vào hệ thống: - Type: PUT - API: /rest/auth/v1/user/create	Trả về lỗi	Trả về lỗi	Pass
TC_GH_DK_05	Kiểm thử đăng ký với dữ liệu không hợp lệ (tài khoản đã tồn tại)	Tài khoản đã tồn tại trước	- username: user1 - password: password1 - Các thông tin còn lại có thể có hoặc không	1. Cài đặt Postman 2. Gửi dữ liệu kiểm thử vào hệ thống: - Type: PUT - API: /rest/auth/v1/user/create	Trả về lỗi	Trả về lỗi	Pass

5.4.2. Đăng nhập (lấy access token)

- Tiên điều kiện: Phải có kết nối internet và đang truy cập vào trang đăng nhập.
- Kịch bản kiểm thử:

Bảng 5-6: Kiểm thử chức năng đăng nhập (lấy access token)

Mã số test case	Mô tả	Điều kiện	Dữ liệu kiểm thử	Các bước thực hiện	Kết quả mong đợi	Kết quả thực tế	Đánh giá
TC_GH_AT_01	Kiểm thử access token với dữ liệu hợp lệ	Tài khoản đã tồn tại trước	Authorization: - Type: Basic Auth - username: web-ui - password: secret Params - username: user1 - password: password1 - grant_type: password	1. Cài đặt Postman 2. Gửi dữ liệu kiểm thử vào hệ thống: - Type: POST - API: /oauth/token	Trả về Access token	Trả về Access token	Pass

TC_GH_AT_02	Kiểm thử access token với dữ liệu không hợp lệ (username rỗng)	Tài khoản đã tồn tại trước	Authorization: - Type: Basic Auth - username: web-ui - password: secret Params - password: password1 - grant_type: password	1. Cài đặt Postman 2. Gửi dữ liệu kiểm thử vào hệ thống: - Type: POST - API: /oauth/token	Trả về lỗi	Trả về lỗi	Pass
TC_GH_AT_03	Kiểm thử access token với dữ liệu không hợp lệ (password rỗng)	Tài khoản đã tồn tại trước	Authorization: - Type: Basic Auth - username: web-ui - password: secret Params - username: user1 - grant_type: password	1. Cài đặt Postman 2. Gửi dữ liệu kiểm thử vào hệ thống: - Type: POST - API: /oauth/token	Trả về lỗi	Trả về lỗi	Pass

5.4.3. Tạo bài viết

- Tiền điều kiện: Phải có kết nối internet và đang truy cập vào trang tạo bài viết.
- Kịch bản kiểm thử:

Bảng 5-7: Kiểm thử chức năng tạo bài viết

Mã số test case	Mô tả	Điều kiện	Dữ liệu kiểm thử	Các bước thực hiện	Kết quả mong đợi	Kết quả thực tế	Đánh giá
TC_GH_TBV_01	Kiểm thử tạo bài viết với dữ liệu hợp lệ	Có token hợp lệ	Authorization: - Authorization: token Body: - title: Post1 - Các thông tin còn lại có thể có hoặc không	1. Cài đặt Postman 2. Gửi dữ liệu kiểm thử vào hệ thống: - Type: PUT - API: /rest/post/v1/post/upsert	Trả về Post có dữ liệu giống dữ liệu kiểm thử	Trả về Post có dữ liệu giống dữ liệu kiểm thử	Pass

TC_GH_TB V_02	Kiểm thử tạo bài viết với dữ liệu không hợp lệ (Body null)	Có token hợp lệ	Authorization: - Authorization: token Body:	1. Cài đặt Postman 2. Gửi dữ liệu kiểm thử vào hệ thống: - Type: PUT - API: /rest/post/v1/post/upsert	Trả về lỗi	Trả về lỗi	Pass
TC_GH_TB V_03	Kiểm thử tạo bài viết với dữ liệu không hợp lệ (Authorization null)	Không cần điều kiện	Authorization: Body: - title: Post1 - Các thông tin còn lại có thể có hoặc không	1. Cài đặt Postman 2. Gửi dữ liệu kiểm thử vào hệ thống: - Type: PUT - API: /rest/post/v1/post/upsert	Trả về lỗi	Trả về lỗi	Pass

5.4.4. Tạo địa điểm

- Tiền điều kiện: Phải có kết nối internet và đang truy cập vào trang tạo bài viết.
- Kích bản kiểm thử:

Bảng 5-8: Kiểm thử chức năng tạo địa điểm

Mã số test case	Mô tả	Điều kiện	Dữ liệu kiểm thử	Các bước thực hiện	Kết quả mong đợi	Kết quả thực tế	Đánh giá
TC_GH_TDD_01	Kiểm thử tạo địa điểm với dữ liệu hợp lệ	Có token hợp lệ, có bài viết với id: postid	Authorization: - Authorization: token Body: - parentId: postid - Các thông tin còn lại có thể có hoặc không	1. Cài đặt Postman 2. Gửi dữ liệu kiểm thử vào hệ thống: - Type: PUT - API: /rest/post/v1/location/upsert	Trả về Location có dữ liệu giống dữ liệu kiểm thử	Trả về Location có dữ liệu giống dữ liệu kiểm thử	Pass

TC_GH_TDD_02	Kiểm thử tạo bài viết với dữ liệu không hợp lệ (Body null)	Có token hợp lệ	Authorization: - Authorization: token Body:	1. Cài đặt Postman 2. Gửi dữ liệu kiểm thử vào hệ thống: - Type: PUT - API: /rest/post/v1/location/upsert	Trả về lỗi	Trả về lỗi	Pass
TC_GH_TDD_03	Kiểm thử tạo bài viết với dữ liệu không hợp lệ (Authorization null)	Không cần điều kiện	Authorization: - Authorization: token Body: - parentId: postid - Các thông tin còn lại có thể có hoặc không	1. Cài đặt Postman 2. Gửi dữ liệu kiểm thử vào hệ thống: - Type: PUT - API: /rest/post/v1/location/upsert	Trả về lỗi	Trả về lỗi	Pass
TC_GH_TDD_04	Kiểm thử tạo bài viết với dữ liệu không hợp lệ (parent id không tồn tại)	Có token hợp lệ, không có bài viết với id: postid	Authorization: - Authorization: token Body: - parentId: postid - Các thông tin còn lại có thể có hoặc không	1. Cài đặt Postman 2. Gửi dữ liệu kiểm thử vào hệ thống: - Type: PUT - API: /rest/post/v1/location/upsert	Trả về lỗi	Trả về lỗi	Pass

5.4.5. Tạo bình luận

- Tiền điều kiện: Phải có kết nối internet và đang truy cập vào trang tạo bài viết.
- Kịch bản kiểm thử:

Bảng 5-9: Kiểm thử chức năng tạo bình luận

Mã số test case	Mô tả	Điều kiện	Dữ liệu kiểm thử	Các bước thực hiện	Kết quả mong đợi	Kết quả thực tế	Đánh giá
TC_GH_TBL_01	Kiểm thử tạo bình luận với dữ liệu hợp lệ	Có token hợp lệ, có bài viết với id: postid	Authorization: - Authorization: token Body: - parentId: postid - content: good - Các thông tin còn lại có thể có hoặc không	1. Cài đặt Postman 2. Gửi dữ liệu kiểm thử vào hệ thống: - Type: PUT - API: /rest/post/v1/comment/insert	Trả về Comment có dữ liệu giống dữ liệu kiểm thử	Trả về Comment có dữ liệu giống dữ liệu kiểm thử	Pass
TC_GH_TBL_02	Kiểm thử tạo bình luận với dữ liệu không hợp lệ (Body null)	Có token hợp lệ	Authorization: - Authorization: token Body:	1. Cài đặt Postman 2. Gửi dữ liệu kiểm thử vào hệ thống: - Type: PUT - API: /rest/post/v1/comment/insert	Trả về lỗi	Trả về lỗi	Pass
TC_GH_TBL_03	Kiểm thử tạo bình luận với dữ liệu không hợp lệ (Authorization null)	Không cần điều kiện	Authorization: - Authorization: token Body: - parentId: postid - content: good - Các thông tin còn lại có thể có hoặc không	1. Cài đặt Postman 2. Gửi dữ liệu kiểm thử vào hệ thống: - Type: PUT - API: /rest/post/v1/comment/insert	Trả về lỗi	Trả về lỗi	Pass

TC_GH_TBL_04	Kiểm thử tạo bình luận với dữ liệu không hợp lệ (parent id không tồn tại)	Có token hợp lệ, không có bài viết với id: postid	Authorization: - Authorization: token Body: - parentId: postid - content: good - Các thông tin còn lại có thể có hoặc không	1. Cài đặt Postman 2. Gửi dữ liệu kiểm thử vào hệ thống: - Type: PUT - API: /rest/post/v1/comment/insert	Trả về lỗi	Trả về lỗi	Pass
TC_GH_TBL_05	Kiểm thử tạo bình luận với dữ liệu không hợp lệ (Không có nội dung)	Có token hợp lệ, có bài viết với id: postid	Authorization: - Authorization: token Body: - parentId: postid - Các thông tin còn lại có thể có hoặc không	1. Cài đặt Postman 2. Gửi dữ liệu kiểm thử vào hệ thống: - Type: PUT - API: /rest/post/v1/comment/insert	Trả về lỗi	Trả về lỗi	Pass

5.4.6. Lấy địa điểm

- Tiền điều kiện: Phải có kết nối internet và đang truy cập vào trang tạo bài viết.
- Kịch bản kiểm thử:

Bảng 5-10: Tìm địa điểm dựa trên vị trí địa lý

Mã số test case	Mô tả	Điều kiện	Dữ liệu kiểm thử	Các bước thực hiện	Kết quả mong đợi	Kết quả thực tế	Đánh giá
TC_GH_LDD_01	Kiểm thử lấy địa điểm với dữ liệu hợp lệ	Không cần điều kiện	- lat: 48.2181679 - lng: 16.3899064 - offset: 0	1. Cài đặt Postman 2. Gửi dữ liệu kiểm thử vào hệ thống: - Type: GET - API: /rest/geo/v1/geo	Trả về Location với address: Heinestraße 42, 1020 Vienna, Austria	Trả về Location với address: Heinestraße 42, 1020 Vienna, Austria	Pass

TC_GH_LDD_02	Kiểm thử lấy địa điểm với dữ liệu không hợp lệ (Body null)	Không cần điều kiện		1. Cài đặt Postman 2. Gửi dữ liệu kiểm thử vào hệ thống: - Type: GET - API: /rest/geo/v1/geo	Trả về lỗi	Trả về lỗi	Pass
TC_GH_LDD_03	Kiểm thử lấy địa điểm với dữ liệu không hợp lệ (kinh độ vĩ độ không hợp lệ)	Không cần điều kiện	- lat: 148.2181679 - lng: 16.3899064 - offset: 0	1. Cài đặt Postman 2. Gửi dữ liệu kiểm thử vào hệ thống: - Type: GET - API: /rest/geo/v1/geo	Trả về lỗi	Trả về lỗi	Pass

CHƯƠNG 6: ĐÁNH GIÁ MICROSERVICES

6.1. So sánh

6.1.1. So sánh chức năng

Với Microservices thì mỗi service sẽ dùng một database riêng, còn với phần mềm thông thường. Ví dụ chức năng lấy bài viết:

Microservices:

```
check auth code
call auth service to get user
is admin user
    return all post
is normal user
    return access post
return public post
```

Normal Service:

```
check auth code
call auth function to get user
is admin user
    return all post
is normal user
    return access post
return public post
```

Chức năng này dùng 2 service là auth và post khi service auth bị mất kết nối với database thì không ảnh hưởng đến các service khác. Lúc này service post sẽ trả về các public post. Đối với phần mềm thông thường thì khi mất kết nối với database thì toàn bộ các chức năng liên quan đến database sẽ không hoạt động. Lúc này phần mềm thông thường sẽ không thể trả về dữ liệu.

6.1.2. So sánh thời gian phản hồi

Vì Microservices xử lý chức năng riêng biệt nên khi có một yêu cầu phức tạp thì cần phải đi qua nhiều Microservices khác nhau nên thời gian phản hồi sẽ tăng lên. Ví dụ chức năng lấy bài viết:

Microservices:

- Client gửi yêu cầu
- Cổng API gửi dữ liệu đến post service
- Post service gửi yêu cầu lấy user qua auth service
- Auth service lấy user từ database
- Auth service trả lời yêu cầu của post service
- Post service lấy post từ database dựa vào thông tin user
- Post service gửi trả lời yêu cầu đến cổng API
- Cổng API trả lời yêu cầu đến client

Normal service:

- Client gửi yêu cầu
- Hệ thống lấy user từ database
- Hệ thống lấy post từ database dựa vào thông tin user

- Hệ thống trả lời đến yêu cầu đến client

Microservices cần dùng đến bảy bước để xử lý yêu cầu lấy post so với bốn bước của một hệ thống thông thường, như vậy Microservices sẽ cần gấp đôi thời gian để phản hồi nếu nếu tất cả các bước có thời gian xử lý như nhau.

6.1.3. So sánh vận hành

Microservice: Do các service vận hành độc lập nên khi một service ngừng hoạt động thì chỉ những chức năng của service đó bị ảnh hưởng.

Ví dụ 1: Hệ thống đang vận hành tốt thì geo service bị lỗi và ngừng hoạt động. Các chức năng của geo service như tra cứu địa điểm, lấy địa chỉ sẽ ngừng hoạt động.

⇒ Ảnh hưởng nhẹ.

Ví dụ 2: Hệ thống đang vận hành tốt thì auth service ngừng hoạt động. Các chức năng liên quan đến user sẽ ngừng hoạt động, tuy nhiên người dùng vẫn có thể vào để xem bài viết cũng như tra cứu địa điểm.

⇒ Ảnh hưởng trung bình.

Ví dụ 3: Hệ thống đang vận hành tốt thì UI ngừng hoạt động. Khi này hầu hết các người dùng thông thường, thành viên, admin sẽ không dùng được hệ thống do không có UI để tương tác. Tuy nhiên các người dùng bên thứ ba với admin, user có kiến thức về cổng API cũng như thông tin về API của hệ thống vẫn tương tác được qua API.

⇒ Ảnh hưởng nghiêm trọng.

Normal service: Vì các chức năng của hệ thống thông thường được viết trên cùng một service nên nếu có lỗi ngừng hoạt động thì toàn bộ hệ thống sẽ ngừng hoạt động. Vậy khi ta áp dụng hệ thống thông thường vào ba ví dụ của microservices thì sẽ có cùng kết quả là toàn bộ hệ thống sẽ ngừng hoạt động.

⇒ Ảnh hưởng rất nghiêm trọng.

6.2. Ưu điểm

- Microservices cho phép dễ dàng liên tục chuyển giao và triển khai các ứng dụng lớn và phức tạp hơn.
- Có thể cải thiện khả năng bảo trì: bởi vì các service tương đối nhỏ nên dễ hiểu và dễ thay đổi hơn.
- Có khả năng testing dễ dàng: nhờ các services nhỏ.
- Có thể triển khai tốt hơn: các services thường rất dễ cho việc triển khai độc lập.
- Cho phép các services được phát triển nhanh chóng bởi những team khác nhau. Khi đó, mỗi team đều sẽ được phát triển và thử nghiệm để triển khai cũng như mở rộng được quy mô của dịch vụ của mình một cách độc lập nhất với tất cả các team.
- Nếu như có lỗi xảy ra trong một service thì chỉ có service đó bị ảnh hưởng và các service khác sẽ thực hiện xử lý các yêu cầu cần thiết. Trong khi đó, thì mỗi một thành phần nếu như hoạt động sai của kiến trúc một khối thì nó sẽ làm ảnh hưởng đến toàn bộ hệ thống.
- Lập trình viên có thể thay đổi dễ dàng bằng cách sử dụng công nghệ mới khi triển khai các service. Tương tự như khi có thay đổi lớn thì các service đều có thể thực hiện và bạn dễ dàng thay đổi được công nghệ hơn.

6.3. Nhược điểm

- Bởi vì các nhà phát triển thường xuyên phải đối phó với sự phức tạp khi tạo ra một hệ thống phân tán.
- Cần phải cài đặt việc liên lạc giữa các services nội bộ.
- Handle partial failure rất phức tạp bởi vì luồng xử lý cần phải đi qua nhiều service khác nhau.
- Khi thực hiện các requests trải rộng trên nhiều service khó khăn thì điều này cần đòi hỏi sự phối hợp giữa các team.
- Thường rất khó khăn trong việc đảm bảo toàn vẹn cho CSDL nếu như triển khai theo các cấu trúc cơ sở dữ liệu dạng phân vùng.
- Việc triển khai và quản lý Microservices nếu như làm thủ công theo cách làm với ứng dụng thì sẽ rất phức tạp.
- Lập trình viên cần phải xử lý các sự cố kết nối chậm, lỗi nếu như thông điệp không được gửi hoặc nếu như thông điệp được gửi đến nhiều đích đến vào các thời điểm khác nhau.

PHẦN 3. KẾT LUẬN

1. KẾT QUẢ ĐẠT ĐƯỢC

1.1. Về lý thuyết và công nghệ

- Khả năng đọc tài liệu, tự tìm hiểu, tự học và giải quyết vấn đề.
- Hiểu biết được các kiến trúc Microservices
- Hiểu biết được quy trình nghiệp vụ của một trang mạng xã hội.
- Nâng cao kỹ năng lập trình và hiểu rõ hơn về các framework và ngôn ngữ lập trình(Spring Boot, Angular, TypeScript, MongoDB, HTML, CSS).
- Nâng cao khả năng phân tích và thiết kế cơ sở dữ liệu.

1.2. Về website

Xây dựng được một trang web đáp ứng được các yêu cầu về các chức năng cơ bản: đăng ký, đăng nhập, đăng bài viết, bình luận bài viết, tìm kiếm bạn bè, gửi lời mời kết bạn,...

1.3. Hạn chế

- Chưa có nhiều chức năng nâng cao, chỉ dừng ở chức năng cơ bản.
- Giao diện chưa tương thích với màn hình các thiết bị di động.

2. HƯỚNG PHÁT TRIỂN

- Mở rộng hệ thống giúp cho người dùng có thể tạo thực hiện các cuộc gọi video, tính năng chia sẻ bài viết, tính năng cho phép đăng tải video trong bài viết.
- Thiết kế giao diện tương thích với các thiết bị di động.
- Phát triển thêm ứng dụng di động cho dễ sử dụng cho các thiết bị di động.
- Cải thiện UI UX.
- Thêm chức năng gợi ý địa điểm gần với người dùng.
- Thêm chức năng bản đồ thời gian thực để hiển thị các người dùng hoạt động.

TÀI LIỆU THAM KHẢO

- [1] Microservices Architecture, <https://Microservices.io/>
- [2] Tìm hiểu kiến trúc Microservices trong thiết kế phần mềm, <https://stanford.com.vn/kien-thuc-lap-trinh/tin-chi-tiet/cagId/27/id/22548/tim-hieu-kien-truc-Microservices-trong-thiet-ke-phan-mem>
- [3] ADOPTION OF THE MICROSERVICES ARCHITECTURE by Maryanne Ndungu, https://www.doria.fi/bitstream/handle/10024/169535/ndungu_maryanne.pdf
- [4] PHÁT TRIỂN PHẦN MỀM DỰA TRÊN MICROSERVICES, PHẠM THỊ VÂN, <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>
- [5] What are Microservices really all about? - Microservices Basics Tutorial, <https://www.youtube.com/watch?v=j1gU2oGFayY>
- [6] Introduction to Microservices, Docker, and Kubernetes, <https://www.youtube.com/watch?v=1xo-0gCVhTU>
- [7] Principles Of Microservices by Sam Newman, <https://www.youtube.com/watch?v=PFQnNFe27kU>
- [8] Microservices using SpringBoot, <https://www.youtube.com/watch?v=BnknNTN8icw>
- [9] Spring Framework, <https://spring.io/>
- [10] Angular, <https://angular.io/>
- [11] Cloudinary, <https://cloudinary.com/>
- [12] Trackprofiler, <https://www.trackprofiler.com/>
- [13] Oauth 2.0, https://docs.oracle.com/cd/E82085_01/160027/JOS%20Implementation%20Guide/Output/oauth.htm
- [14] Microservices Patterns by Chris Richardson
- [15] Building Microservices: Designing Fine-Grained Systems by Sam Newman

PHỤ LỤC

1. Khởi chạy hệ thống

- Môi trường vận hành
 - Ubuntu
 - Ram: 16GB
 - HDD: 500GB
 - CPU: Intel Core i3
- Yêu cầu
 - Maven
 - Java 11
 - NodeJS
 - Angular CLI
 - Docker
 - Git
- Khởi chạy
 - Tải phần mềm về: git clone <https://github.com/tcminh404/Go-Hiking.git>
 - Build hệ thống:
 - cd Go-Hiking
 - mvn clean install
 - cd web-ui
 - npm i
 - ng build --prod
 - Chạy hệ thống
 - cd Go-Hiking
 - docker-compose up --build
 - Tạo admin account cho mongo
 - docker exec -it mongo sh -c 'exec mongosh'
 - db.createUser({ user: "mongoadmin" , pwd: "mongoadmin",
 - roles: ["userAdminAnyDatabase", "dbAdminAnyDatabase",
 - readWriteAnyDatabase"]})

Note: nếu tạo username và password khác thì cần cập nhật lại dữ liệu trong file docker-compose.yml rồi chạy lại server

2. Sử dụng trang web

Mở trình duyệt và truy cập vào trang web <http://localhost:4200>