

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ



Dương Quang Khải

**XÂY DỰNG NỀN TẢNG IOT MỞ TÍCH HỢP DỊCH VỤ
BÊN THỨ BA CHO NHÀ THÔNG MINH**

KHÓA LUẬN TỐT NGHIỆP ĐẠI HỌC HỆ CHÍNH QUY

Ngành: Khoa học Máy Tính

HÀ NỘI – 2020

**ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ**

Dương Quang Khải

**XÂY DỰNG NỀN TẢNG IOT MỞ TÍCH HỢP DỊCH VỤ
BÊN THỨ BA CHO NHÀ THÔNG MINH**

KHÓA LUẬN TỐT NGHIỆP ĐẠI HỌC HỆ CHÍNH QUY

Ngành: Khoa học Máy Tính

Cán bộ hướng dẫn: PGS.TS. Nguyễn Hoài Sơn

Cán bộ đồng hướng dẫn: ThS. Đào Minh Thư

HÀ NỘI – 2020

**VIETNAM NATIONAL UNIVERSITY, HANOI
UNIVERSITY OF ENGINEERING AND TECHNOLOGY**

Duong Quang Khai

**INTERGRATING THIRD-PARTY SERVICE INTO IOT
PLATFORM FOR SMART HOME**

Major: Computer Science

Supervisor: Assoc. Prof. Nguyen Hoai Son

Co-Supervisor: MSc. Dao Minh Thu

HA NOI – 2020

LỜI CẢM ƠN

-----0o0-----

Được sự dạy dỗ của các thầy cô trong trường Đại học Công Nghệ - Đại học Quốc Gia Hà Nội, sự hướng dẫn tận tình của thầy hướng dẫn PGS. TS. Nguyễn Hoài Sơn và sự giúp đỡ của các bạn học, tôi đã thực hiện thành công khóa luận “Xây dựng nền tảng IoT mở tích hợp dịch vụ bên thứ ba cho nhà thông minh”

Để hoàn thành khóa luận này, tôi xin chân thành cảm ơn các thầy cô trong trường Đại học Công Nghệ - Đại học Quốc Gia Hà Nội đã dạy dỗ tôi kiến thức về các môn đại cương, chuyên ngành, giúp tôi có động lực thúc đẩy khả năng học hỏi trong suốt bốn năm học tập và rèn luyện tại trường.

Tôi xin chân thành cảm ơn thầy hướng dẫn khóa luận của tôi là thầy PGS. TS. Nguyễn Hoài Sơn. Thầy đã tận tình chỉ bảo, động viên tôi trong suốt quá trình thực hiện khóa luận tốt nghiệp này.

Tôi xin gửi lời cảm ơn tới giảng viên đồng hướng dẫn, Ths. Đào Minh Thư đã tận tình góp ý, đưa ra những lời khuyên để tôi có thể hoàn thiện tốt hơn khóa luận tốt nghiệp này.

Tôi cũng xin chân thành cảm ơn gia đình, người thân và bạn bè đã quan tâm, động viên và giúp đỡ tôi trong suốt thời gian học tập và thực hiện khóa luận tốt nghiệp này.

Mặc dù đã cố gắng để hoàn thành tốt nhất khóa luận tốt nghiệp này, tuy nhiên vẫn không thể tránh khỏi thiếu sót. Kính mong quý thầy cô và toàn thể bạn bè góp ý để đề tài dần được hoàn thiện hơn.

Một lần nữa, xin chân thành cảm ơn tới các thầy cô giáo!

Trân trọng.

TÓM TẮT

Tóm tắt: Hiện nay, cùng với sự phát triển không ngừng của ngành công nghệ thông tin, kèm theo đó là sự bùng nổ của cuộc cách mạng công nghiệp lần thứ tư, IoT đang dần trở nên phổ biến hơn bao giờ hết. Gần đây, nhiều nền tảng IoT Platform được xây dựng, với mục đích cung cấp cho người dùng một hệ thống để người dùng có thể dễ dàng điều khiển và quản lý ngôi nhà thông minh của mình. Tuy nhiên, các nền tảng IoT Platform có trên thị trường hiện nay vẫn chưa hỗ trợ tốt khả năng tích hợp các dịch vụ mở, phát triển bởi các bên thứ ba, vào nền tảng của mình. Điều này khiến cho trải nghiệm sử dụng của người dùng đối với các nền tảng đó giảm đi rõ rệt. Do vậy trong đề tài khóa luận tốt nghiệp này, tôi sẽ trình bày về một nền tảng mở IoT hỗ trợ tích hợp dịch vụ của các bên thứ ba và các phương thức xác thực quyền truy cập của những dịch vụ đó vào trong nhà thông minh của người dùng. Khóa luận sẽ tập trung trình bày về phương thức xác thực kết nối từ phía dịch vụ lên hệ thống, cách thức truy xuất dữ liệu và quy trình đăng kí một dịch vụ mới lên hệ thống cho cả phía người dùng và nhà cung cấp dịch vụ.

Từ khóa: *API Server, IoT Platform, JWT, xác thực dịch vụ*

ABSTRACT

Abstract: Nowadays, along with the development of information technology, followed by the burst of the fourth industrial revolution, IoT is now becoming more popular than ever. Recently, many developers/companies has created IoT Platforms to satisfy user's needs. The idea is to provide user the ability to fully control their smart homes, just through a single platform. However, the IoT Platforms that are currently available on the market do not provide the ability to integrate services which are developed by 3rd-party developers/companies (service providers), into user's smart home. This minus point of those services will gradually reduce the user experiences. Because of that, in this graduate thesis, i will present an open IoT Platform that support the intergration of services which are developed by 3rd-party developers/companies and also an method to verify and authenticate incoming request from services to user's smart home system. The thesis will mainly focus on presenting an authentication method between the services and the system, how to access data in user's smart home and the procedure in registering a new service into the system for both service providers and users.

Keywords: *API Server, IoT Platform, JWT, service authentication*

LỜI CAM ĐOAN

-----0o0-----

Tôi xin cam đoan khóa luận tốt nghiệp “Xây dựng nền tảng IoT mở tích hợp dịch vụ bên thứ ba cho nhà thông minh” là công trình nghiên cứu của chính bản thân tôi thực hiện, dưới sự hướng dẫn của PGS. TS. Nguyễn Hoài Sơn.

Tất cả các tài liệu tham khảo đều được nêu rõ nguồn gốc trong danh mục tài liệu tham khảo. Khóa luận không sao chép một cách bất chính từ bất kì công trình nghiên cứu, khóa luận hay đồ án của người khác, mà không được chỉ một cách rõ ràng trong mục tài liệu tham khảo. Các đánh giá, tổng kết trong bài đều thông qua các kết quả thực nghiệm và thống kê số liệu trong thực tế.

Nếu có điều gì sai trái, tôi xin chịu trách nhiệm trước hội đồng về kết quả thực hiện đồ án tốt nghiệp của mình.

Hà nội, ngày 05 tháng 06 năm 2020

Người thực hiện

Dương Quang Khải

MỤC LỤC

MỞ ĐẦU	3
CHƯƠNG 1. TỔNG QUAN	6
1.1. Tổng quan về IoT.....	6
1.1.1. Khái niệm về IoT	6
1.1.2. Khái niệm về IoT Platform	6
1.1.3. Khái niệm về nhà thông minh (Smart Home)	9
1.1.4. Khái niệm về nền tảng mở dành cho Smart Home.....	10
1.1.5. Ví dụ về nền tảng mở openHAB.....	11
1.2. Tổng quan về API.....	12
1.2.1. Khái niệm về API.....	12
1.2.2. Các loại API.....	12
CHƯƠNG 2: THIẾT KẾ HỆ THỐNG	14
2.1. Đặt vấn đề.....	14
2.2. Ý tưởng thiết kế	15
2.3. Tổng quan API Server	18
2.3.1. Thiết kế API dành cho các dịch vụ bên thứ ba.....	18
2.3.2. Xây dựng quy trình đăng kí dịch vụ mới	21
2.3.3. Xây dựng quy trình quản lý dịch vụ bên thứ ba cho người dùng	22
2.4. Thiết kế cơ chế xác thực dịch vụ truy cập vào hệ thống	24
CHƯƠNG 3: TRIỂN KHAI VÀ ĐÁNH GIÁ HỆ THỐNG	28
3.1. Triển khai thực tế.....	28
3.1.1. Cài đặt API Server	28
3.1.2. Cài đặt Database	28
3.1.3. Cài đặt MQTT Broker và bộ thư viện xác thực MQTT Broker	29
3.2. Quy trình đăng kí dịch vụ mới.....	32
3.3. Quy trình quản lý dịch vụ trong hệ thống của người dùng.....	35
3.3.1. Quy trình thêm dịch vụ mới vào hệ thống.....	35
3.3.2. Quy trình khởi chạy dịch vụ.....	37
3.4. Đánh giá kết quả thực hiện	39
3.4.1. Cài đặt các chức năng trên API Server.....	39

3.4.2.	Thời gian chạy xác thực kết nối tới MQTT Broker.....	40
3.4.3.	Thời gian chạy xác thực dịch vụ thực hiện truy vấn tới API Server	43
3.4.4.	Thời gian chạy khi dịch vụ thực hiện truy vấn dữ liệu từ đến API Server	46
3.4.5.	Tài nguyên hệ thống được sử dụng trong các tác vụ.....	48
CHƯƠNG 4: TỔNG KẾT		54
1.	Kết luận.....	54
2.	Hướng phát triển	54
TÀI LIỆU THAM KHẢO		55

MỞ ĐẦU

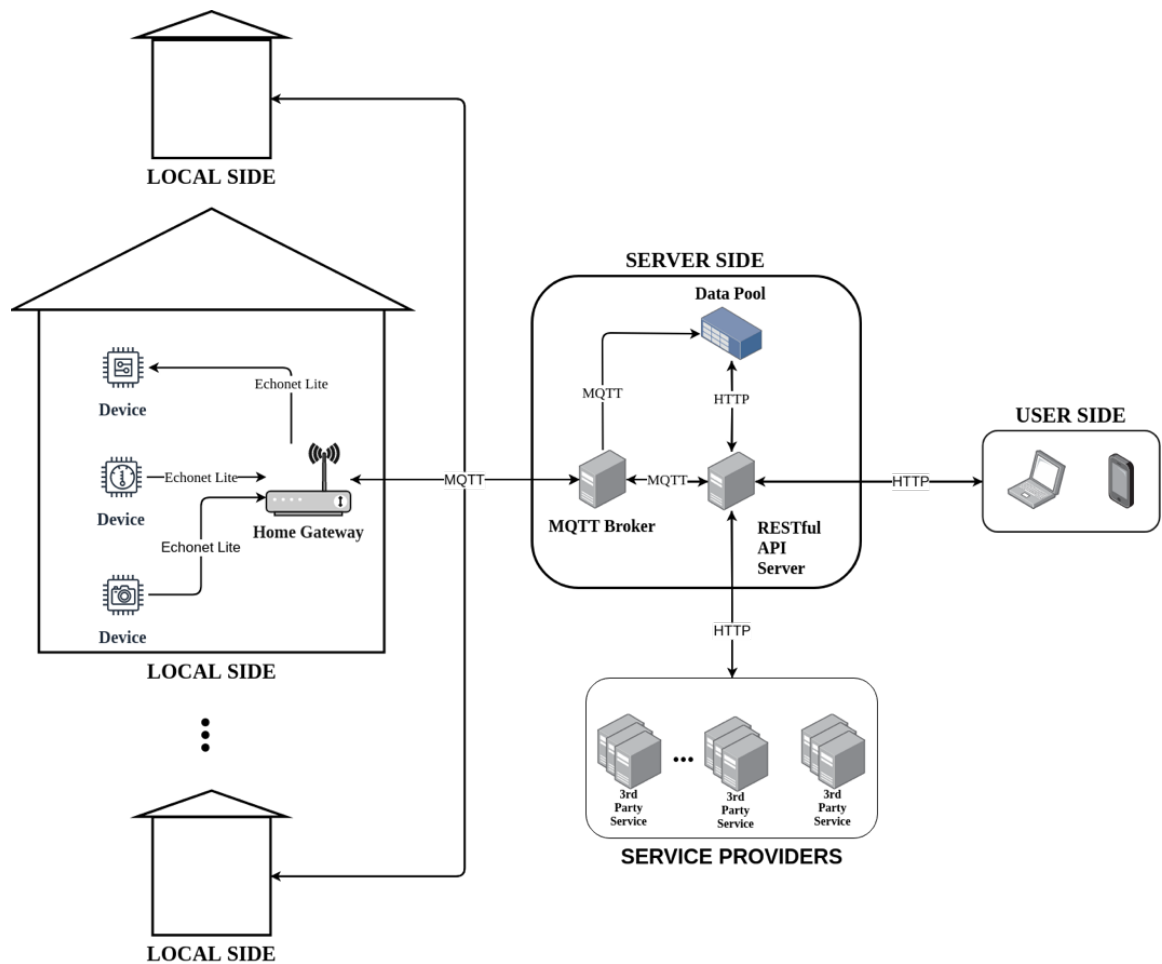
Đặt vấn đề

Internet Of Things (viết tắt là IoT, Internet vạn vật) đang dần trở thành một xu thế tất yếu trong cuộc sống ngày nay. Khái niệm IoT được dùng để miêu tả mạng lưới các thiết bị cảm biến (cảm biến ánh sáng, cảm biến nhiệt độ, độ ẩm,...), thiết bị gia dụng thông minh (máy giặt thông minh, tủ lạnh thông minh,...), kết nối với nhau, trực tiếp trao đổi thông tin, dữ liệu mà không cần đến sự tương tác trực tiếp giữa người với người, hay người với máy tính.

Trên thực tế, các ứng dụng IoT đều chạy trên một nền tảng (Platform) riêng của nhà sản xuất thiết bị đó. Từ đó nảy sinh vấn đề về sự tương thích giữa các thiết bị thông minh đến từ các hãng khác nhau, khiến người dùng bị trói chặt vào hệ sinh thái cố định của hãng đó. Điều này gây ảnh hưởng đến trải nghiệm của người dùng, do phải liên tục chuyển qua lại giữa các nền tảng để điều khiển các thiết bị thông minh trong nhà. Bên cạnh đó, do nền tảng được phát triển riêng bởi mỗi nhà sản xuất nên hoàn toàn là một nền tảng đóng. Người dùng chỉ có thể sử dụng được các tính năng cơ bản được cung cấp bởi chính nhà sản xuất đó, mà không có lựa chọn sử dụng các dịch vụ từ bên thứ ba, VD: Người dùng chỉ có thể thực hiện các thao tác điều khiển từng đèn thủ công trên ứng dụng của nhà sản xuất, mà không thể sử dụng các dịch vụ do các nhà phát triển bên ngoài cung cấp, như là điều khiển đèn tự động, dựa trên lịch bản và hành vi của người dùng. Chính vì là nền tảng đóng, nên các nhà lập trình viên không thể truy cập vào tập API của Platform, để từ đó phát triển các dịch vụ cho người dùng sử dụng, cũng như có quyền được truy cập vào dữ liệu thu thập từ các cảm biến.

Trước đó đã có nhiều hướng nghiên cứu về phát triển một IoT Platform, cụ thể là trong khóa luận tốt nghiệp [1], và trong bài báo [2]. Nhược điểm của nền tảng này, đó là chưa hỗ trợ khả năng tích hợp các dịch vụ của các nhà phát triển vào nền tảng của mình. Việc có thể cung cấp khả năng tích hợp dịch vụ phát triển bởi các nhà cung cấp dịch vụ giúp mở ra nhiều cơ hội mới cho các nhà phát triển trong việc xây dựng các mô hình dự đoán, mô phỏng, xử lý dữ liệu lớn (Big Data), điều khiển thông minh,...

Từ các vấn đề trên, khóa luận tốt nghiệp “Xây dựng nền tảng IoT mở tích hợp dịch vụ bên thứ ba cho nhà thông minh” của tôi hướng đến việc xử lý các vấn đề trên. Hệ thống tôi đề xuất trong khóa luận tốt nghiệp này được cài đặt tại thành phần server trong IoT Platform mà tôi và các cộng sự đã phát triển trước đây, bên cạnh hai thành phần còn lại là thành phần trong nhà thông minh và thành phần phía người dùng.



Hình 1: Tổng quan IoT Platform

API Server này đã được cài đặt trong đề tài “Hệ thống điều khiển đèn thông minh tiết kiệm năng lượng” để tham dự hội nghị sinh viên nghiên cứu khoa học các cấp và đồng thời trong hai bài báo quốc tế [3], [4], tất cả đều cho kết quả rất khả quan. Nhưng nhược điểm của hệ thống API Server này, đó là chưa hỗ trợ khả năng tích hợp dịch vụ bên thứ ba vào trong hệ thống của người dùng.

Vì vậy, API Server xây dựng trong khóa luận tốt nghiệp này sẽ hướng tới việc cung cấp khả năng tích hợp dịch vụ bên thứ ba cho người dùng. Bên cạnh đó, hệ thống còn cung cấp khả năng truy xuất đến một tập các hàm thường dùng, nhằm hỗ trợ các bên thứ ba phát triển các dịch vụ tương thích với Platform. API Server còn có nhiệm vụ xác thực và cung cấp phương thức giao tiếp giữa người dùng (VD: Thông qua Web App hoặc Mobile App) và hệ thống, cũng như phân quyền truy cập các dịch vụ của bên thứ ba tới từng nhà, từng phòng và từng loại thiết bị của người dùng.

API Server cho IoT Platform cần đảm bảo các tính năng:

- Xác thực người dùng và nhà cung cấp dịch vụ (Service Provider), sử dụng JSON Web Token (JWT).
- Cung cấp khả năng phân quyền truy cập tới các dịch vụ của bên thứ ba cho người dùng.
- Cung cấp khả năng truy xuất dữ liệu, cũng như điều khiển các thiết bị trong nhà.
- Hệ thống hoạt động ổn định, có khả năng chịu tải tốt với lượng người dùng đồng thời cao.

Cấu trúc khóa luận tốt nghiệp:

- Chương 1: Tổng quan
Tổng quan các khái niệm về IoT, nhà thông minh (Smart Home), API Server
- Chương 2: Thiết kế hệ thống
- Chương 3: Triển khai và đánh giá hiệu năng, kết quả của hệ thống được mô tả trong chương 2
- Chương 4: Kết luận

CHƯƠNG 1. TỔNG QUAN

1.1. Tổng quan về IoT

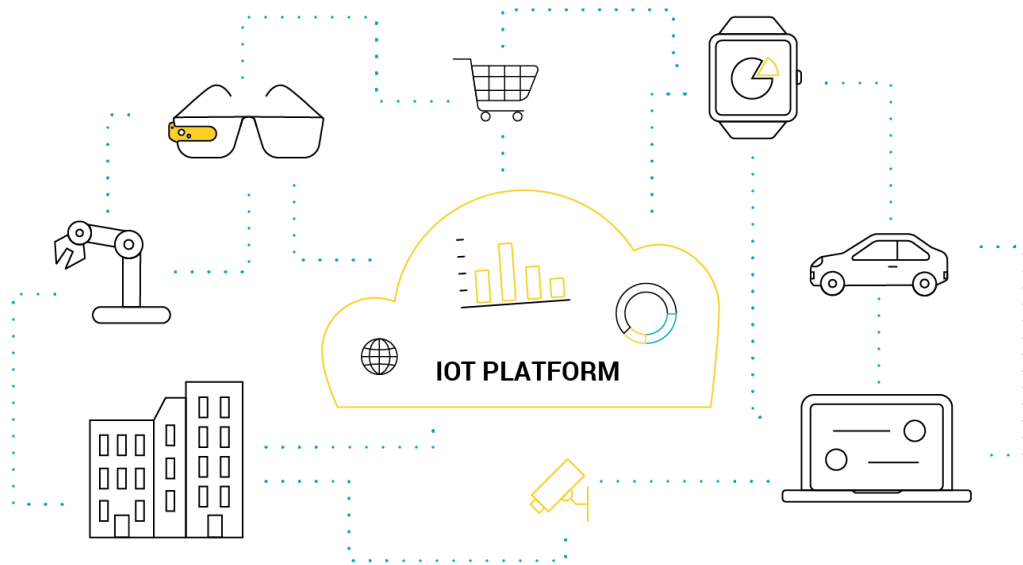
1.1.1. Khái niệm về IoT

IoT, hay còn có tên gọi đầy đủ là Internet of Things, là một khái niệm được nhà khoa học Kevin Ashton giới thiệu vào năm 1999, dùng để mô tả một mạng lưới khổng lồ, nơi mà tất cả mọi “thứ” (Things) kết nối với nhau, bao gồm các thiết bị cảm biến (cảm biến ánh sáng, cảm biến nhiệt độ, độ ẩm,...), thiết bị gia dụng thông minh (máy giặt thông minh, tủ lạnh thông minh,...). Các thiết bị này sẽ trực tiếp trao đổi thông tin, dữ liệu hoặc để thực hiện một công việc nào đó, mà không cần đến sự tương tác trực tiếp giữa người với người, hay người với máy tính.

Ngày nay, IoT đang dần trở thành xu hướng thiết yếu của cuộc sống, bởi những tiện ích mà nó mang lại. Các nhà khoa học đã đưa ra dự báo rằng đến năm 2024, sẽ có khoảng 500 tỷ thiết bị được kết nối vào mạng Internet, tạo thành một mạng lưới Internet Vạn Vật khổng lồ. Những lợi ích IoT mang lại cho con người mà chúng ta có thể kể đến, như là: Cải thiện việc gắn kết giữa doanh nghiệp và khách hàng, tối ưu hóa về mặt công nghệ, giảm sự hao phí và quan trọng nhất, đó là hỗ trợ khả năng thu thập dữ liệu.

Trên thực tế, đã có nhiều ứng dụng về IoT được áp dụng vào trong cuộc sống, ví dụ như điều hòa thông minh, tự điều chỉnh nhiệt độ điều hòa phù hợp với nhu cầu và hành vi của người dùng, cho đến các ứng dụng về xe tự hành, được trang bị hàng loạt các cảm biến phức tạp để có thể đưa ra các quyết định điều khiển xe, tránh các chướng ngại vật. Hay một ứng dụng nữa về IoT mà chúng ta không thể bỏ qua, đó là ứng dụng trong nhà thông minh. Theo thống kê, nhà thông minh hiện đang là ứng dụng liên quan đến IoT được tìm kiếm nhiều nhất trên Google và dần trở thành xu hướng của cuộc sống ngày nay, bởi những tiện ích mà nó đem lại cho con người.

1.1.2. Khái niệm về IoT Platform



Hình 1.1: Tổng quan IoT Platform

IoT Platform được định nghĩa là nền tảng của mọi hệ thống IoT, giúp gắn kết các thiết bị cảm biến, thiết bị thông minh cùng các giao thức kết nối với những giải pháp về phần mềm, với mục đích đem lại một giải pháp hiệu quả trong việc cài đặt và quản lý các thiết bị, cũng như thu thập và xử lý dữ liệu.

Các thành phần cơ bản của một IoT Platform bao gồm:

- **Các thiết bị thông minh:** Gồm các loại máy móc, cảm biến, các thiết bị đeo thông minh như đồng hồ theo dõi nhịp tim với khả năng thu thập thông tin từ môi trường, khả năng giao tiếp lẫn nhau, cũng như khả năng gửi/nhận dữ liệu lên máy chủ đám mây.
- **Các giải pháp kết nối:** Tùy thuộc vào đặc thù của mỗi dự án IoT mà phương thức kết nối giữa các thiết bị với nhau có thể sẽ khác nhau. Nhưng điểm chung của các phương thức kết nối này, đó là phải đảm bảo được tính tin cậy, bảo mật dữ liệu, cũng như đảm bảo được lượng điện năng tiêu thụ, tầm phủ sóng và băng thông dùng để truyền dữ liệu.
- **Các phần mềm/dịch vụ bên thứ ba:** Thường được triển khai trên đám mây hoặc triển khai nội bộ trong hệ thống nhà thông minh của người dùng. Các phần mềm/dịch vụ này sẽ thu thập dữ liệu của các cảm biến và các thiết bị thông minh trong nhà của người dùng, từ đó tiến hành phân tích, xử lý dữ liệu để đưa ra các quyết định thực thi tác vụ tiếp theo trong hệ thống.
- **Giao diện người dùng:** Cung cấp cho người dùng một giao diện trực quan về trạng thái hệ thống IoT trong nhà của mình, mô hình hóa dữ liệu thu thập được từ các cảm biến và đồng thời cung cấp khả năng điều khiển các thiết bị đó trong căn nhà thông minh của họ.

Các tính năng chính của một IoT Platform bao gồm:

- Khả năng mở rộng.
- Thân thiện với người dùng.
- Khả năng tích hợp các dịch vụ bên thứ ba vào hệ thống.
- Cung cấp nhiều tùy chọn về hạ tầng triển khai.
- Bảo mật dữ liệu.

Hiện nay, có hai loại IoT Platform phổ biến, đó là IoT Platform kín (Private IoT Platform) và IoT Platform mở (Public IoT Platform). Các IoT Platform kín thường là những nền tảng được các công ty, tổ chức hoặc các tập đoàn lớn tự phát triển, sau đó được triển khai và sử dụng trong hệ thống nội bộ của mình. Lợi ích của IoT Platform chính là khả năng tương thích cao với hệ thống IoT của chính công ty, tổ chức hoặc các tập đoàn đó do các nền tảng này được phát triển dựa trên chính nhu cầu về tính năng của họ, dẫn đến khả năng tùy biến của nền tảng này là rất cao. Ngoài ra, độ bảo mật của nền tảng là rất cao do các đơn vị đó không công khai mã nguồn của nền tảng đó ra ngoài công chúng, cũng như chỉ triển khai ở trong hệ thống nội bộ của mình, giúp giảm khả năng bị tấn công vào hệ thống IoT. Nhưng nhược điểm của IoT Platform kín, đó là chi phí nghiên cứu và phát triển nền tảng này là rất lớn, không phù hợp với người dùng hay với các công ty tổ chức quy mô nhỏ. Ngoài ra, không phải ai cũng có nhu cầu tự phát triển một IoT Platform của riêng mình, do phần lớn các IoT Platform mở đã đáp ứng được nhu cầu của người sử dụng.

Trái ngược với IoT Platform kín là IoT Platform mở. Các nền tảng này thường được phát triển bởi các tổ chức phi lợi nhuận, hoặc bởi một cộng đồng các nhà phát triển. Với những nền tảng được công khai mã nguồn trên các trang như Github, Gitlab, Bitbucket,... thường là những nền tảng miễn phí, yêu cầu người dùng cần có kiến thức trong việc tự cài đặt, triển khai và vận hành trong hệ thống nội bộ của mình. Những nền tảng này sẽ phù hợp với người dùng thích DIY (Do-it-yourself, tự làm). Còn đối với những nền tảng phát triển theo dạng thuê bao (subscription), sẽ hoạt động bằng cách có một nhà cung ứng dịch vụ xây dựng cơ sở hạ tầng, hệ thống và triển khai IoT Platform lên đó. Người dùng sẽ chỉ cần đăng ký sử dụng với một khoản tiền hàng tháng, mà không cần phải lo lắng về cách vận hành, triển khai và cài đặt IoT Platform.

Bên cạnh đó, các Public IoT Platform sẽ mở ra tiềm năng lớn trong việc tích hợp các dịch vụ bên thứ ba vào trong hệ thống, giúp nâng cao trải nghiệm sử dụng của người dùng với nền tảng. Tuy nhiên hiện nay, nhiều IoT Platform mở chưa hỗ trợ hoặc hỗ trợ rất ít việc tích hợp các dịch vụ bên thứ ba vào trong hệ thống. Lý do là vì một số nền tảng còn đặt nhiều ràng buộc đối với các nhà phát triển, ví dụ như ràng buộc về ngôn ngữ lập trình. Điều này khiến cho nhiều lập trình viên không cảm thấy hứng thú trong việc phát triển các dịch vụ cho nền tảng.

Vì vậy, trong khóa luận tốt nghiệp này, tôi sẽ trình bày về IoT Platform mở cùng với cách thức giải quyết các vấn đề khi tích hợp dịch vụ bên thứ ba vào nền tảng nhà thông minh.

1.1.3. Khái niệm về nhà thông minh (Smart Home)

Nhà thông minh (hay còn gọi là Smart Home, Home Automation), là một ngôi nhà với hàng loạt các loại cảm biến hiện đại, như cảm biến đo nhiệt độ, độ ẩm, phát hiện hành vi người cùng với hệ thống chiếu sáng điều khiển tự động, dựa vào dữ liệu được gửi về từ các cảm biến trong nhà, hệ thống tưới cây tự động,... với mục đích tự động hóa các công việc hàng ngày mà không cần đến sự can thiệp của con người. Trước đây, khái niệm về nhà thông minh thường được dùng để miêu tả về những công nghệ hiện đại chỉ xuất hiện ở trong những bộ phim viễn tưởng. Tuy nhiên, với sự phát triển của các công nghệ, khái niệm về smart home đã dần không còn trở nên xa lạ đối với nhiều người bởi các thiết bị phần cứng thiết kế riêng cho nhà thông minh, cũng như các giải pháp về phần mềm để triển khai smart home đang dần trở nên phổ biến với mọi người.

Ví dụ, vào buổi sáng, khi người dùng thức dậy, căn nhà sẽ tự động mở rèm ra để đón ánh nắng mặt trời, đồng thời ra lệnh cho máy pha cà phê làm việc và phát qua những chiếc loa thông minh bản tin sáng sớm, cùng với thông tin về mật độ giao thông trên tuyến đường đi làm sắp tới của họ. Hay khi người dùng trở về nhà vào chiều tối sau khi kết thúc ca làm ở công ty, căn nhà thông minh sẽ mở sẵn điều hòa trước khi người dùng về từ 15-30 phút, tùy vào mật độ giao thông trên tuyến đường về nhà. Đồng thời, căn nhà cũng tiến hành bật một số lượng đèn cần thiết xung quanh vị trí người dùng đang đứng, hoặc tại căn phòng người dùng đang ở.

Một ví dụ khác về nhà thông minh, đã được tôi và các cộng sự nghiên cứu và phát triển, đó là hệ thống điều khiển đèn thông minh tiết kiệm năng lượng. Hệ thống được lắp đặt bao gồm một bóng đèn nhiều cấp độ sáng, cùng với các cảm biến ánh sáng và camera được lắp đặt trong các phòng. Cảm biến ánh sáng được sử dụng để thu thập thông tin về cường độ ánh sáng tự nhiên. Sau đó các dữ liệu này được đưa qua một thuật toán thông minh để từ đó điều chỉnh mức độ sáng của đèn sao cho phù hợp với mong muốn của người dùng. Còn camera được dùng để phát hiện hành vi người dùng, từ đó lựa chọn kịch bản chiếu sáng hợp lý cho người dùng.

Bên cạnh các kịch bản tự động kể trên, người dùng có thể dễ dàng điều khiển các thiết bị trong nhà của mình, từ những chiếc máy giặt, tủ lạnh đến cả điều hòa, tivi,... hoặc thêm các kịch bản mới vào hệ thống nhà thông minh của mình. Tất cả các thao tác đó, người dùng có thể dễ dàng thao tác thông qua ứng dụng được cài đặt trên điện thoại của mình.

Hiện nay trên thị trường, có rất nhiều nhà sản xuất sản xuất ra các thiết bị phục vụ cho nhà thông minh. Một số tên tuổi có thể kể đến, như là: Lumi, Xiaomi, Tuya, Sonoff,... Tuy nhiên, mỗi hãng sản xuất lại phát triển các thiết bị của họ chỉ có thể hoạt động được với phần mềm do chính họ phát triển. Điều này gây không ít phiền toái đối với người dùng, do người dùng có thể sử dụng nhiều loại thiết bị thông minh cho từng chủng loại trong gia đình, hoặc trong các công ty/tập đoàn lớn của họ, với số lượng lớn. Lúc này, việc quản lý và điều khiển các thiết bị thuộc nhiều hãng khác nhau sẽ trở nên khó khăn, do mỗi hãng lại cung cấp phần mềm, nền tảng và giao thức kết nối riêng của mình, buộc người sử dụng phải chuyển qua chuyển

lại giữa các nền tảng để giao tiếp với thiết bị. Điều này góp phần làm giảm đáng kể trải nghiệm của người dùng đối với nhà thông minh.

Nhận thức được vấn đề đó, nhiều tổ chức phi lợi nhuận được thành lập, với nhiệm vụ phát triển các nền tảng mở cho nhà thông minh như openHAB, Home Assistant (Hass),... Các nền tảng này mở ra cơ hội cho người dùng có thể quản lý các thiết bị thông minh trong nhà của mình thuộc nhiều hãng sản xuất khác nhau, thông qua một nền tảng duy nhất, loại bỏ hoàn toàn việc chuyển qua lại giữa các ứng dụng của các hãng sản xuất để điều khiển từng thiết bị thông minh trong nhà của hãng đó.

1.1.4. Khái niệm về nền tảng mở dành cho Smart Home

Trên thực tế, người dùng có thể sử dụng nhiều loại thiết bị thông minh cho từng chủng loại trong gia đình, hoặc trong các công ty/tập đoàn lớn của họ, với số lượng lớn. Lúc này, việc quản lý và điều khiển các thiết bị thuộc nhiều hãng khác nhau sẽ trở nên khó khăn, do mỗi hãng lại cung cấp phần mềm, nền tảng riêng của mình, buộc người sử dụng phải chuyển qua chuyển lại giữa các nền tảng để giao tiếp. Vì vậy, các nền tảng mở như openHAB hay Hass được ra đời.

Các nền tảng mở được định nghĩa là nơi quản lý tập trung các căn nhà thông minh cùng các thiết bị trong từng căn nhà của người dùng bằng cách hỗ trợ nhiều chuẩn giao tiếp của các thiết bị thông minh trong nhà, ví dụ như Zigbee, ECHONET Lite,... Ngoài ra, nền tảng mở còn hỗ trợ khả năng tích hợp các tiện ích mở rộng từ các nhà phát triển thứ ba vào trong hệ thống, nhằm mở rộng khả năng của hệ thống, cung cấp tiện ích tối đa cho người dùng. Các tiện ích mở rộng có thể kể đến, như là dịch vụ cung cấp khả năng hỗ trợ giao thức kết nối mới với các thiết bị thông minh, dịch vụ điều khiển thiết bị thông minh, sử dụng thuật toán tối ưu và kịch bản được cài đặt sẵn, dịch vụ nhận diện thiết bị dựa trên đặc trưng dòng điện của thiết bị, từ đó kiểm soát được lượng điện năng tiêu thụ của từng loại thiết bị trong nhà,...

Bên cạnh đó, nhu cầu truy xuất dữ liệu từ các thiết bị cảm biến, phục vụ cho mục đích phân tích, thống kê nhu cầu sử dụng của người dùng là một nhu cầu không thể thiếu, nhất là trong thời đại ngày nay, kỉ nguyên của dữ liệu lớn (Big Data), học máy (Machine Learning) và Internet vạn vật (Internet of Things). Vì vậy, các nền tảng mở cho nhà thông minh cần phải hỗ trợ các dịch vụ bên thứ ba truy xuất dữ liệu về các thiết bị trong nhà thông minh của người dùng, tất nhiên sẽ phải được người dùng cấp quyền cho phép truy xuất đến. Từ đó các nhà phát triển có thể tạo ra các dịch vụ điều khiển tự động, sử dụng các thuật toán thông minh, kết hợp với việc dựa trên phân tích dữ liệu truy xuất được từ các cảm biến để từ đó xác định được nhu cầu, hành vi của người dùng.

Hiện nay, các nền tảng thông minh đang được triển khai dưới hai hình thức: Nhà phát triển cài đặt và vận hành nền tảng trên hạ tầng của họ, sau đó cung cấp các gói dịch vụ trả phí cho người dùng và người dùng tự cài đặt và vận hành dịch vụ trên hạ tầng của mình. Đối với phương thức thứ nhất, đăng kí gói dịch vụ trả phí hàng tháng. Ưu điểm của giải pháp này, đó là người dùng sẽ không cần phải tốn kém chi

phí ban đầu cho việc xây dựng hạ tầng thiết bị, cụ thể là máy chủ, để vận hành nền tảng. Ngoài ra, các vấn đề về vận hành dịch vụ, nâng cấp phần mềm, bảo mật hệ thống,... sẽ được quản lý và thực hiện bởi nhà cung cấp dịch vụ, người dùng sẽ chỉ cần chi ra một khoản tiền nhất định hàng tháng, hoặc hàng năm, rồi từ đó có thể dễ dàng bắt đầu sử dụng nền tảng đó. Tuy nhiên, nhược điểm duy nhất của giải pháp này, đó là giá cả của dịch vụ. Tùy vào nhà cung cấp dịch vụ mà giá cả và chất lượng dịch vụ sẽ có sự chênh lệch rõ rệt với nhau.

Với phương thức thứ hai, đó là tự mình cài đặt và vận hành nền tảng IoT cho nhà thông minh. Đây là một giải pháp phù hợp với những người dùng thích tự mình xây dựng một hệ thống nhà thông minh từ con số không và đồng thời có một chút kiến thức về máy tính để vận hành. Ưu điểm của giải pháp này, đó là người dùng sẽ không phải mất chi phí hàng tháng đăng ký sử dụng nền tảng được vận hành sẵn bởi các nhà cung cấp. Bên cạnh đó, người dùng nâng cao có thể toàn quyền can thiệp sâu vào trong hệ thống để tinh chỉnh các thành phần, từ đó có thể nâng cao hiệu năng của nền tảng. Nhưng nhược điểm của giải pháp này, đó là nó chỉ nên được lựa chọn bởi người dùng có một chút kinh nghiệm về vận hành hệ thống, về thao tác với dòng lệnh trên hệ điều hành Linux, bảo mật mạng,... Bởi lẽ nếu các thành phần đó không được thiết lập đúng, hệ thống nhà thông minh có thể sẽ hoạt động không như mong muốn, hoặc tệ hơn, có thể bị tin tặc tấn công chiếm quyền điều khiển căn nhà.

Qua đó ta có thể thấy, sự ra đời các nền tảng mở cùng khả năng tích hợp các dịch vụ từ bên thứ ba vào mở ra tiềm năng mới trong việc phát triển hệ sinh thái nhà thông minh, từ đó nâng cao trải nghiệm cuộc sống của người dùng.

1.1.5. Ví dụ về nền tảng mở openHAB

openHAB có ưu điểm lớn khi so sánh với các nền tảng được cung cấp bởi nhà sản xuất thiết bị thông minh. Đó là khả năng giao tiếp với nhiều loại thiết bị của nhiều hãng khác nhau, đồng thời quản lý các thiết bị đó trên một nền tảng duy nhất. Bên cạnh đó, openHAB còn cung cấp khả năng thêm các tiện ích mở rộng dành cho người dùng (addon) nhằm giúp openHAB có thể giao tiếp được với các thiết bị thông minh. Hiện nay, openHAB được sử dụng rộng rãi trên toàn thế giới và được nhiều người dùng nhà thông minh tin tưởng lựa chọn openHAB làm nền tảng cho căn nhà thông minh của mình.

Bên cạnh những điểm cộng trên, vẫn còn tồn tại một số nhược điểm khi sử dụng giải pháp nền tảng IoT mã nguồn mở, cụ thể là openHAB. Thứ nhất, người dùng phải có kiến thức căn bản về Linux, cấu hình máy chủ, cấu hình mạng thì mới có thể cài đặt và triển khai được openHAB trong căn nhà của mình. Điều này có thể quá phức tạp đối với phần lớn người dùng do không phải ai cũng có những kiến thức cơ bản về vận hành, duy trì, cũng như khả năng để mua một chiếc máy chủ để chạy openHAB 24/7.

Không những vậy, sau khi cài đặt được openHAB, người dùng cũng phải cấu hình, thiết lập các chức năng của openHAB để có thể hoạt động được với thiết bị thông minh của các hãng. Cuối cùng, đối với các nhà phát triển dịch vụ muốn tạo ra

các addons cho openHAB, họ phải sử dụng ngôn ngữ Java và xây dựng thành file .jar để có thể cài đặt vào openHAB. Điều này có thể gây khó khăn đối với nhiều nhà phát triển do có người có thể mạnh về ngôn ngữ Python, có người mạnh về C++,...

1.2. Tổng quan về API

1.2.1. Khái niệm về API

API là cụm viết tắt của **Application Programming Interface**, hay còn được gọi là giao diện lập trình của ứng dụng. Hiểu đơn giản như với ứng dụng web, để người dùng có thể tương tác được với web, cần phải có một giao diện để người dùng sử dụng. Vậy nên giữa các ứng dụng muốn giao tiếp được với nhau cũng cần phải giao tiếp thông qua API. Mục đích chính của API, đó là cung cấp một tập các hàm thường dùng giúp ứng dụng của người dùng cuối có thể truy cập để thực hiện chức năng tương tác hoặc trao đổi dữ liệu.

Trong nhiều trường hợp, một API sẽ là một phần của bộ SDK (Software Development Kit). Một bộ SDK có thể bao gồm API cũng như các công cụ/phần cứng khác, vì thế hai thuật ngữ này không thay thế cho nhau được.

1.2.2. Các loại API

1. Web API

Đây có lẽ là loại API được sử dụng nhiều nhất và thông dụng nhất với nhiều người. API này thường được các website lớn, như Google, Facebook, Accuweather,... cung cấp cho lập trình viên để họ có thể kết nối, lấy dữ liệu hoặc cập nhật thông tin hệ thống.

Một ví dụ đơn giản về web API: khi người lập trình viên phát triển một sản phẩm phần mềm và họ cần hiển thị thông tin về thời tiết lên trên sản phẩm của họ. Để có thể có dữ liệu về thời tiết, cần phải thiết lập hệ thống quan trắc, hệ thống vệ tinh để theo dõi ảnh mây vệ tinh, kết hợp xử lý số liệu thô, rồi từ đó mới có thông tin về thời tiết. Hiển nhiên không phải ai cũng có đủ khả năng để có thể xây dựng một hệ thống phức tạp như vậy để lấy dữ liệu về thời tiết cho vào trong sản phẩm phần mềm của mình. Thay vào đó, người lập trình viên sẽ lựa chọn việc sử dụng API lấy thông tin về thời tiết được cung cấp có thể bởi các trung tâm dự báo thời tiết, khí tượng thủy văn nơi có những máy móc, thiết bị phức tạp phục vụ cho quá trình theo dõi và dự đoán thời tiết.

2. API của hệ điều hành

Các nhà phát triển hệ điều hành, ví dụ như Microsoft, cung cấp cho các nhà phát triển các tài liệu API đặc tả các hàm, phương thức cũng như các giao thức kết nối tới các thành phần của hệ điều hành, giúp cho lập trình viên có thể dễ dàng tạo ra các phần mềm tương tác được tốt với hệ điều hành đó. Ví dụ như API lấy danh sách thư mục, API lấy thông tin một tệp dữ liệu,..

3. API của các thư viện phần mềm hoặc bộ khung phát triển phần mềm (Framework)

Khi phát triển một phần mềm, các lập trình viên đều sử dụng các bộ khung phát triển phần mềm (framework) trong dự án của họ, do tính tiện lợi và dễ phát triển sản phẩm của chúng mang lại. Ví dụ như Node.js thì có Express.js, PHP thì có Laravel, phát triển giao diện web người dùng thì có React, Vue.js, Angular,... Khi phát triển các framework, nhà phát triển sẽ tạo ra các API để mô tả và quy định các hành vi mong muốn mà các thư viện cung cấp, một API có thể có nhiều các triển khai khác nhau và nó cũng giúp cho một chương trình viết bằng ngôn ngữ này có thể sử dụng thư viện được viết bằng ngôn ngữ khác.

Trong phạm vi khóa luận tốt nghiệp này, tôi sẽ chỉ tập trung vào API dành cho nền tảng web, hay còn được gọi là web API.

CHƯƠNG 2: THIẾT KẾ HỆ THỐNG

2.1. Đặt vấn đề

Hiện nay trên thị trường, đã có một số các đơn vị tự phát triển các nền tảng IoT nhằm phục vụ nhu cầu của người dùng về giao tiếp và điều khiển các thiết bị thông minh của mình. Tuy nhiên, phần lớn các nền tảng đó hiện chưa hỗ trợ cho các nhà cung cấp dịch vụ (service provider) phát triển và đem sản phẩm của mình lên các nền tảng đó, hoặc có hỗ trợ nhưng còn tồn tại khá nhiều hạn chế, như là: giới hạn về ngôn ngữ lập trình được phép sử dụng, quy trình tích hợp các thành phần trong dịch vụ của bên thứ ba vào trong nền tảng khá phức tạp, ... Do đó đặt ra vấn đề cần phải xây dựng một nền tảng IoT vừa hỗ trợ tốt nhất cho các nhà phát triển dịch vụ, vừa đảm bảo giới hạn quyền truy cập dữ liệu của người dùng. Việc xây dựng một nền tảng mở sẽ gặp nhiều vấn đề cần giải quyết, bao gồm: xác thực và cấp quyền cho nhà cung cấp dịch vụ và các dịch vụ của họ khi truy cập vào hệ thống; hỗ trợ tối đa đối với các nhà phát triển dịch vụ, bằng cách cho phép sử dụng nhiều ngôn ngữ lập trình khác nhau như Python, C#, C/C++,..., để tạo nên dịch vụ, trong khi vẫn đảm bảo khả năng tích hợp dịch vụ đó với hệ thống; tạo một quy trình để các nhà phát triển có thể tuân theo khi đăng ký một dịch vụ mới lên hệ thống và cách thức để người dùng có thể quản lý được các services trong hệ thống của mình.

Ở vấn đề thứ nhất, việc cần phải có một cơ chế xác thực và cấp quyền cho service khi service thực hiện truy cập vào hệ thống để lấy thông tin, cũng như gửi lệnh điều khiển là nhằm mục đích kiểm soát quyền hạn truy cập của service vào trong hệ thống, tránh việc truy cập trái phép vào, ví dụ như truy cập vào dữ liệu nhà của một người dùng không hề đăng ký sử dụng dịch vụ, hoặc truy cập vào dữ liệu của những căn phòng/ loại thiết bị khi chưa được người dùng cho phép. Bên cạnh đó, việc lựa chọn và cài đặt một cơ chế xác thực các service cần phải đảm bảo được thời gian chạy nằm trong ngưỡng cho phép và đồng thời cũng cần phải đảm bảo tính tin cậy của cơ chế xác thực. Ví dụ như: áp dụng một cơ chế mã hóa không thể làm giả được token, nhằm tránh các service tự tạo ra token với quyền giả để vượt quyền truy cập.

Vấn đề thứ hai, khả năng hỗ trợ các nhà phát triển sử dụng bất kỳ ngôn ngữ lập trình nào để tạo ra các dịch vụ tương thích với nền tảng. Để giải quyết vấn đề này, cần phải cài đặt một cơ chế giao tiếp giữa hệ thống và dịch vụ của các nhà phát triển thứ ba, gọi là API. API, viết tắt của Application Programming Interface, là một phương thức giao tiếp và kết nối giữa các ứng dụng khác nhau, bằng cách cung cấp cho ứng dụng khác khả năng truy xuất đến tập các hàm thường dùng, từ đó có thể dễ dàng trao đổi dữ liệu. Bên cạnh đó, API vẫn phải đảm bảo hoạt động tốt với cơ chế xác thực được nêu ở vấn đề thứ nhất.

Vấn đề thứ ba, nhà phát triển dịch vụ đăng ký sản phẩm của mình để có thể cung cấp cho người dùng tích hợp vào hệ thống. Ở đây, quy trình đăng ký dịch vụ mới dành cho các nhà cung cấp dịch vụ cần phải được xây dựng sao cho đơn giản nhất có thể, nhưng vẫn phải đảm bảo tính nghiêm ngặt trước khi đưa sản phẩm tới

tay người dùng. Bởi nếu như trong sản phẩm của nhà phát triển có chứa mã độc, thì khi triển khai lên cloud cùng với API Server có thể gây lỗi trên toàn hệ thống, làm cho dịch vụ bị gián đoạn. Hoặc nếu trong service chứa đoạn code độc hại với mục đích đánh cắp thông tin người dùng, người kiểm duyệt sẽ biết được kịp thời để thông báo lại cho nhà phát triển, nhằm loại bỏ đoạn code độc hại đó.

Vấn đề thứ tư, cung cấp khả năng cho phép người dùng quản lý các dịch vụ đã được kiểm duyệt và khởi chạy trên cloud trong hệ thống của họ. Để có thể giải quyết được vấn đề này, cần phải xây dựng một quy trình thêm/xóa các dịch vụ trong hệ thống, biểu diễn hóa nó dưới dạng giao diện để người dùng có thể dễ dàng thực hiện các thao tác trên chỉ bằng một vài cú nhấn chuột. Bên cạnh đó, hệ thống cũng cần phải cung cấp giao diện để người dùng cài đặt quyền truy cập khi thêm một dịch vụ mới vào trong hệ thống của mình và cả khi khởi chạy dịch vụ lên.

2.2. Ý tưởng thiết kế

Để giải quyết vấn đề xây dựng một cơ chế xác thực và phân quyền truy cập có tính tin cậy cao cùng với thời gian chạy nằm trong ngưỡng cho phép, dành cho các dịch vụ khi truy cập vào hệ thống, tôi sử dụng token theo chuẩn JSON Web Token (JWT) để xác thực dịch vụ.

Lý do lựa chọn JSON Web Token (JWT) vì JWT giúp tăng đáng kể hiệu năng của hệ thống, cùng với độ tin cậy bảo mật cao do sử dụng các thuật toán mã hóa hiện đại, giúp giảm thời gian cần thiết để xử lý các yêu cầu xác thực. Ngoài ra, việc sử dụng JWT sẽ loại bỏ được việc sử dụng phiên làm việc (session) và cookie để lưu trữ các thông tin xác minh người dùng, do JWT là một dạng token chứa các thông tin liên quan đến xác thực, thời gian hết hạn và các thông tin khác do người dùng tự định nghĩa. Bên cạnh đó, do JWT được hỗ trợ bởi rất nhiều các ngôn ngữ lập trình backend, nên JWT có thể dễ dàng được sử dụng trong việc phát triển các dịch vụ con (micro-service) tích hợp vào trong một hệ thống chính.

Thông tin về quyền hạn truy cập của dịch vụ sẽ được lưu vào trong nội dung của token, sau đó sẽ được “kí” lại, như một hình thức niêm phong để đảm bảo tính hợp lệ của token. Bên cạnh đó, token sẽ chỉ hợp lệ trong một khoảng thời gian nhất định, tính từ thời điểm người dùng thực hiện thành công việc phân quyền truy cập và khởi chạy dịch vụ đó.

Khi một dịch vụ bắt đầu thực hiện việc truy cập vào hệ thống để lấy dữ liệu, dịch vụ đó sẽ phải đính kèm theo token được cung cấp ở trên, theo các cách thức khác nhau được mô tả bởi API Server cho từng cách thức lấy dữ liệu riêng. Như vậy, điều này sẽ đảm bảo được rằng các dịch vụ khi được khởi chạy và truy cập vào API Server để dữ liệu đã được cho phép bởi người dùng, tránh việc vượt quyền, truy cập vào những loại thông tin không mong muốn.

Đối với vấn đề cung cấp khả năng tích hợp các dịch vụ vào hệ thống, tôi sẽ cung cấp một số các hàm truy xuất dữ liệu và điều khiển thiết bị (gọi tắt là API) cho nhà cung cấp dịch vụ, thông qua API Server. Do API Server trong khoá luận tốt nghiệp này được thiết kế theo chuẩn RESTful API, nên sẽ đảm bảo được URL đến

các API là ngắn gọn, dễ hiểu. Lý do tôi thiết kế API Server theo chuẩn RESTful là bởi vì REST có hiệu suất hoạt động tốt, tin cậy, dễ dàng phát triển. Ngoài ra REST còn hỗ trợ nhiều định dạng dữ liệu trả về, như JSON, XML,... Bên cạnh đó, để tạo điều kiện tốt nhất cho các nhà phát triển dịch vụ có thể tạo ra những sản phẩm tốt nhất cho người dùng, API Server sẽ không giới hạn hoặc ràng buộc nhà phát triển phải sử dụng loại ngôn ngữ lập trình nào. Thay vào đó, họ có thể sử dụng bất kì loại ngôn ngữ lập trình nào họ cảm thấy thoải mái nhất, để tạo ra các service cho người dùng.

Hệ thống tôi đề xuất trong khóa luận tốt nghiệp này còn cung cấp cho các dịch vụ khả năng cung cấp dữ liệu theo thời gian thực bằng cách truy cập lấy dữ liệu trực tiếp từ MQTT Broker. Lý do tôi đề xuất tính năng này là vì một số dịch vụ, ví dụ như dịch vụ điều khiển đèn tự động sử dụng thuật toán điều khiển thông minh, sẽ yêu cầu khả năng truy xuất dữ liệu từ các thiết bị cảm biến ánh sáng trong nhà để có thể điều chỉnh mức độ sáng đèn tự động theo thời gian thực. Để giải quyết vấn đề này, bốn giải pháp đã được xem xét, bao gồm:

- **Giải pháp thứ nhất:** tất cả yêu cầu lấy dữ liệu từ service sẽ được gửi trực tiếp tới API Server. Sau khi nhận được yêu cầu, API Server sẽ đăng kí vào topic lấy dữ liệu của thiết bị, theo các tham số về nhà, phòng và thiết bị được yêu cầu từ phía service. Khi nhận được dữ liệu từ topic, API Server sẽ trả lại dữ liệu về cho service thông qua giao thức websocket, cụ thể ở đây là cài đặt thư viện Socket.IO để có thể tạo kết nối socket tới service
 - **Ưu điểm:** Do tất cả yêu cầu và trả gói tin đều được thực hiện thông qua API Server, cho nên có thể kiểm soát được tính chất bảo mật của hệ thống, không gói tin nào
 - **Nhược điểm:** Khi triển khai trên diện rộng, khi chạy API Server trên nhiều phân cụm máy chủ để cân bằng tải cho hệ thống, việc cài đặt giao thức giao tiếp giữa các workers sẽ rất khó khăn. Nếu bỏ qua việc cài đặt này, Socket.IO sẽ hoạt động thiếu ổn định, thậm chí có thể gặp lỗi trong quá trình hoạt động do bản chất Socket.IO không hỗ trợ khả năng hoạt động trên hệ thống phân cụm.
- **Giải pháp thứ hai:** Sử dụng Firebase Realtime Database của Google. Firebase Realtime Database của Google là một hệ cơ sở dữ liệu hỗ trợ cập nhật thay đổi về dữ liệu theo thời gian thực. Tức là khi có sự thay đổi về dữ liệu, hoặc có dữ liệu mới được thêm vào trong cơ sở dữ liệu, người dùng cuối có thể ngay lập tức thấy được những thay đổi về dữ liệu hoặc dữ liệu mới đó.

Khi có yêu cầu truy xuất lấy dữ liệu theo thời gian thực từ service, API Server sẽ bắt đầu đăng kí vào topic lấy dữ liệu của thiết bị, theo các tham số về nhà, phòng và thiết bị được yêu cầu từ phía service, rồi tiến hành lưu những dữ liệu đó vào Firebase Realtime Database. Service chỉ việc kết nối tới Firebase

Realtime Database bằng địa chỉ URL được chỉ định bởi API Server là có thể nhận được dữ liệu về thiết bị.

- **Ưu điểm:** Do sử dụng hạ tầng được cung cấp bởi Google, cho nên tính ổn định của dịch vụ là rất cao. Bên cạnh đó, do sử dụng dịch vụ được phát triển sẵn, nên việc tích hợp vào API Server sẽ khá đơn giản, do giờ đây API Server không cần phải cài đặt phương thức truyền dữ liệu theo thời gian thực tới services.

Bên cạnh đó, Firebase Realtime Database hỗ trợ phân quyền truy cập dữ liệu đối với từng người dùng. Tức là người dùng/service chỉ có thể truy cập vào vùng dữ liệu theo quy định của API Server, mà không thể truy cập chéo/ trái phép sang vùng dữ liệu của người dùng khác

- **Nhược điểm:** Để có thể sử dụng được tính năng phân quyền truy cập cơ sở dữ liệu, tôi cần phải tích hợp cơ chế xác thực Firebase Authentication với cơ chế xác thực người dùng/service hiện đang được cài đặt tại API Server. Bên cạnh đó, do Firebase Realtime Database là một dịch vụ bên ngoài, được cung cấp bởi một hãng thứ ba (cụ thể là Google) nên điều này đồng nghĩa với việc dữ liệu của người dùng sẽ được lưu tại hệ thống bên ngoài, dẫn đến sự bảo mật thông tin người dùng có thể bị xâm phạm.
- **Giải pháp thứ ba:** Cho phép service khả năng truy cập trực tiếp vào MQTT Broker. Với giải pháp này, để xác thực quyền truy cập của service vào MQTT Broker, tôi sử dụng một thư viện mã nguồn mở để cài đặt và triển khai tính năng này. Bộ thư viện này hỗ trợ việc xác thực kết nối đến MQTT Broker thông qua nhiều giao thức, bao gồm: HTTP, MySQL, PostgreSQL, Redis, MongoDB và thậm chí là có cả JSON Web Token – chuẩn token mà tôi đã đề cập trong ý tưởng thiết kế khả năng xác thực các dịch vụ truy cập đến hệ thống. Đồng thời, thư viện này còn hỗ trợ khả năng giới hạn topic mà mỗi service có thể truy cập đến, tùy thuộc vào thiết lập của người dùng.
 - **Ưu điểm:** Việc cho phép các dịch vụ khả năng truy cập trực tiếp vào MQTT Broker sẽ giúp giảm tải cho API Server, do lúc này API Server chỉ phải đảm nhiệm việc xác thực kết nối đến MQTT Broker của dịch vụ, và xác thực topic mà dịch vụ subscribe tới có được cho phép bởi người dùng hay không. Một khi dịch vụ đã subscribe thành công vào topic, dịch vụ sẽ tự động lấy dữ liệu trong topic đó.
 - **Nhược điểm:** Hiện tại tôi chưa thấy bất kì nhược điểm nào với giải pháp này
- **Giải pháp thứ tư:** Service truy xuất dữ liệu về thiết bị thông qua Database. Lợi thế lớn nhất của phương pháp này là tính dễ cài đặt, triển khai do luồng truy xuất và trả dữ liệu khá là đơn giản: Khi có yêu cầu lấy dữ liệu từ service, API Server sẽ trả về bản ghi mới nhất từ cơ sở dữ liệu về cho service. Cách làm này đồng thời còn đảm bảo được tính bảo mật và tính phân quyền truy cập rõ ràng của hệ

thông, do mọi yêu cầu gửi/ nhận dữ liệu đều phải được xác thực thông qua API Server

Tuy nhiên, việc gửi/nhận tất cả dữ liệu thông qua API Server có thể khiến API Server trở nên quá tải. Ngoài ra, dữ liệu lấy thông qua cơ sở dữ liệu không phải là dữ liệu cập nhật theo thời gian thực, do dữ liệu được lưu vào database theo mỗi 5-10 phút/lần, trong khi dữ liệu lấy mẫu từ thiết bị thay đổi theo từng giây.

Sau khi phân tích ưu nhược điểm của cả bốn phương pháp cung cấp dữ liệu theo thời gian thực, tôi sẽ lựa chọn giải pháp thứ ba: cho phép các dịch vụ khả năng gửi nhận dữ liệu trực tiếp từ MQTT Broker để lấy dữ liệu.

Về vấn đề đăng kí dịch vụ mới lên hệ thống, tôi sẽ xây dựng một quy trình hoàn chỉnh để hướng dẫn các nhà cung cấp dịch vụ có thể dễ dàng thực hiện theo. Sau khi đăng kí thành công, mã nguồn của dịch vụ sẽ cần phải được kiểm duyệt bởi người kiểm duyệt, trước khi được khởi chạy trên cloud cùng với API Server. Trong phạm vi khóa luận tốt nghiệp này, tôi sẽ không đi sâu vào việc xây dựng quy trình kiểm duyệt mã nguồn dịch vụ, vì đây là một quy trình phức tạp và khối lượng công việc cần để thực hiện cũng rất lớn. Đây có thể là một đề tài cho một khóa luận tốt nghiệp hoàn chỉnh.

Về vấn đề cung cấp khả năng cho người dùng có thể quản lý các dịch vụ trong hệ thống của mình, bao gồm thêm dịch vụ mới và khởi chạy dịch vụ đó hoạt động, tôi sẽ xây dựng một giao diện ứng dụng web để người dùng có thể lên đó quản lý các dịch vụ của mình. Ngoài ra, tôi cũng xây dựng thêm tính năng cho phép người dùng cấp quyền hoạt động cho dịch vụ tới từng nhà, từng phòng và từng loại thiết bị, sau đó lưu những thông tin đó vào trong nội dung của token được gửi tới dịch vụ để khởi chạy.

Như vậy, tôi đã đưa ra thách thức của hệ thống cũng như giải pháp để giải quyết các vấn đề nêu trên.

2.3. Tổng quan API Server

2.3.1. Thiết kế API dành cho các dịch vụ bên thứ ba

Các dịch vụ do bên thứ ba phát triển, khi tích hợp vào trong platform, sẽ thực hiện một số các tác vụ về truy vấn lấy dữ liệu, cũng như gửi dữ liệu lên, như là:

- Lấy danh sách các thiết bị thuộc nhà, phòng của người dùng.
- Lấy dữ liệu về thiết bị trong một khoảng thời gian cố định từ cơ sở dữ liệu.
- Lấy dữ liệu về thiết bị theo thời gian thực.
- Gửi lệnh điều khiển thiết bị trong nhà.

Để có thể hỗ trợ các nhà phát triển dịch vụ tạo ra các dịch vụ tích hợp được vào trong hệ thống, API Server sẽ cung cấp một tập các hàm thường dùng (gọi tắt là API), bao gồm đủ bốn tác vụ đã liệt kê phía trên, dành cho nhà phát triển. Các tập hàm này có thể truy cập được dưới dạng các URL cùng với các

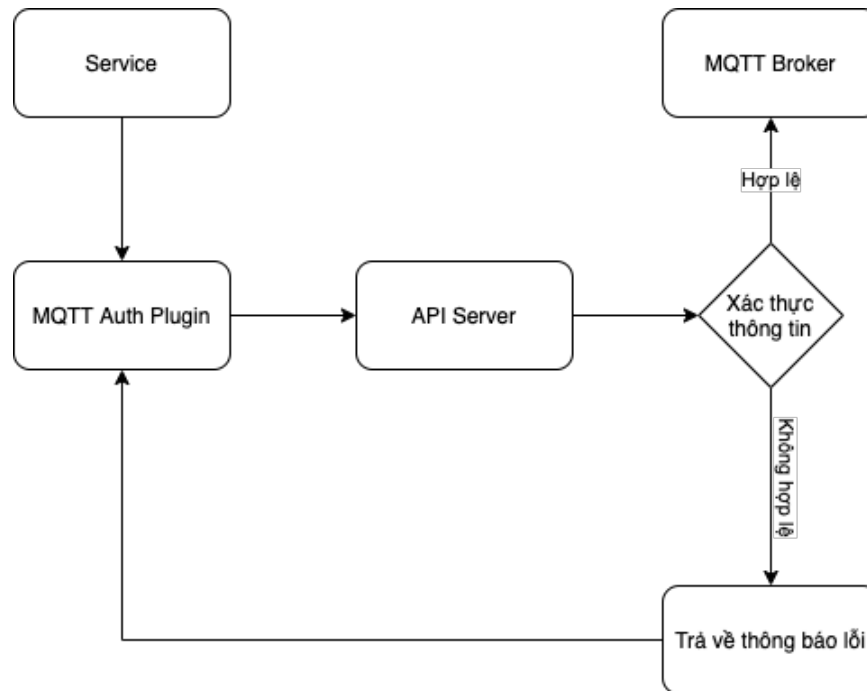
tham số kèm theo. Nhà phát triển sau đó sẽ nhúng các API này vào trong các dịch vụ của họ. Danh sách API dành cho các dịch vụ, bao gồm:

Đường dẫn URL	Phương thức	Mô tả
<code>https://{host}/api/services/devices?home_id={home_id}&&room_id={room_id}&&device_type={device_type}</code>	GET	API dùng để lấy danh sách thiết bị theo từng loại trong phòng
<code>https://{host}/api/services/devices/{device_id}?home_id={home_id}&&room_id={room_id}&&device_type={device_type}&&start_time={start_time}&&end_time={end_time}</code>	GET	API dùng để lấy dữ liệu về thiết bị từ cơ sở dữ liệu trong một khoảng thời gian được chỉ định
<code>https://{host}/api/services/devices/command</code>	POST	API dùng để gửi lệnh điều khiển đến thiết bị

Bảng 2.1: Danh sách các API dành cho các dịch vụ

Ngoài các tham số được mô tả ở trong các API phía trên, các dịch vụ khi gửi yêu cầu truy vấn lấy dữ liệu tới đều phải đính kèm theo token vào trong header của gói tin.

Bên cạnh ba API dành cho nhà phát triển tích hợp vào trong dịch vụ của mình, như đã đề cập trong phần ý tưởng thiết kế, hệ thống tôi đề xuất trong khóa luận này sẽ cho phép các dịch vụ kết nối vào MQTT Broker và subscribe trực tiếp vào topic lấy dữ liệu của thiết bị.



Hình 2.1: Quy trình xác thực kết nối tới MQTT Broker

Khi một dịch vụ muốn truy cập vào MQTT Broker, dịch vụ đó cần phải gửi kèm token của mình vào phần username, phần password còn lại có thể điền kí tự bất kì do khi sử dụng phương thức xác thực là JWT, hệ thống sẽ chỉ quan tâm đến trường dữ liệu username được gửi lên khi kết nối đến MQTT. Token được sử dụng để xác thực kết nối tới MQTT Broker cũng chính là token được sử dụng khi dịch vụ truy cập vào API Server

Sau khi nhận được token, thư viện xác thực MQTT Broker sẽ gửi một POST request tới API Server theo đường dẫn: `https://${host}/api/auth/mqtt-users?type=jwt`. Header gói tin gửi lên sẽ chứa token cần được xác thực. Lúc này, API Server sẽ kiểm tra xem token được gửi lên có chữ kí hợp lệ hay không. Nếu kết quả kiểm tra trả về là đúng, API Server sẽ gửi trả một mã trạng thái HTTP 200 với nội dung:

```

{
  "Ok": true,
  "Error": null
}

```

Nhưng nếu kết quả trả về là token không hợp lệ, API Server sẽ trả một mã trạng thái HTTP 403 với response như sau:

```

{
  "Ok": false,
  "Error": ${error}
}

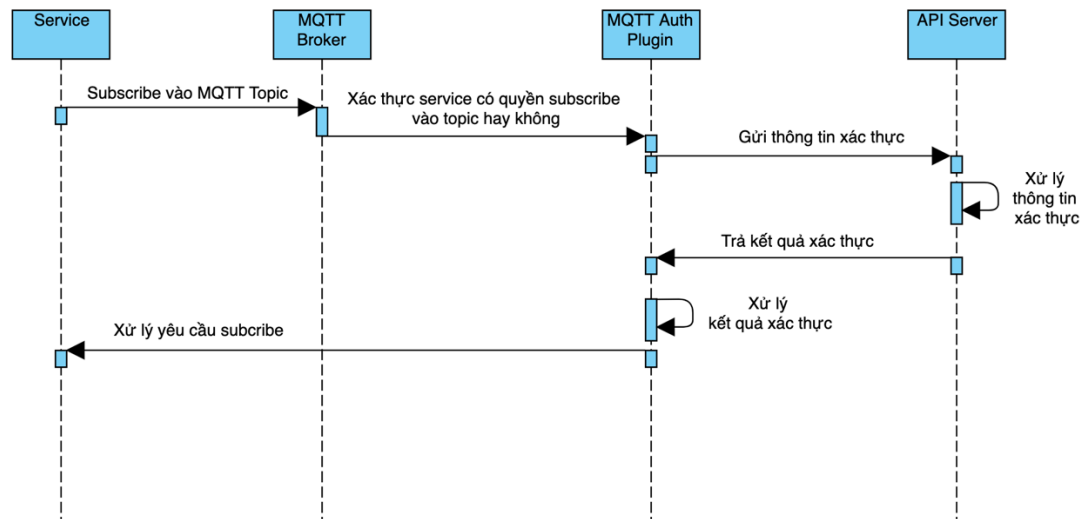
```

Sau khi đã kết nối thành công tới MQTT Broker, lúc này dịch vụ đó có thể tiến hành đăng kí vào topic cần để lấy dữ liệu về thiết bị. Khi nhận được yêu cầu đăng kí vào một topic, bộ thư viện xác thực MQTT Broker sẽ kiểm tra xem topic mà dịch vụ đang đăng kí vào có nằm trong danh sách các topic được phép đăng kí hay không, bằng cách gửi một POST request tới API Server với nội dung:

```
{
  "topic": "/testacc/9/some room/
  IlluminanceSensor/5c:cf:7f:34:35:ce-5/data",
  "clientid": "mock_client",
  "acc": 1
}
```

API Server lúc này sẽ tiến hành kiểm tra topic và acc trong gói tin nhận được, và đem đối chiếu với danh sách topic được phép truy cập đến cùng với quyền hạn đối với topic đó. Danh sách về topic và quyền được lưu ở trong payload của token. Ở đây, quyền hạn đối với topic sẽ được chia làm bốn mức độ: “1” là chỉ đọc, “2” là chỉ viết, “3” là cả đọc và viết, “4” là chỉ được đăng kí vào topic để nhận dữ liệu, theo như mô tả của bộ thư viện xác thực MQTT Broker.

Mã trạng thái HTTP và nội dung của HTTP response trả về giống với quy trình xác thực kết nối tới MQTT Broker ở trên.



Hình 2.2: Quy trình xác thực topic của dịch vụ

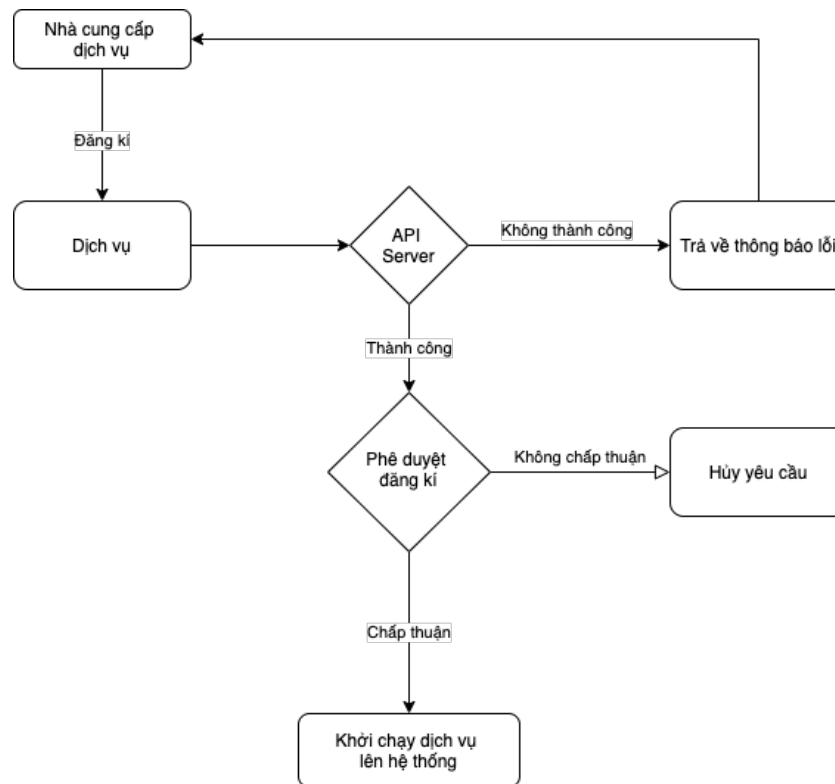
2.3.2. Xây dựng quy trình đăng kí dịch vụ mới

Để các nhà cung cấp dịch vụ có thể đăng tải được dịch vụ của mình lên hệ thống, trước hết họ cần phải đăng kí một tài khoản với người quản trị hệ thống.

Loại tài khoản cung cấp cho nhà phát triển khác biệt hoàn toàn với loại tài khoản cung cấp cho người dùng.

Khi đăng kí một dịch vụ mới, nhà cung cấp dịch vụ cần phải điền đầy đủ các thông tin, bao gồm: Tên dịch vụ, Mô tả dịch vụ và mã nguồn của dịch vụ, kèm theo hướng dẫn chạy (cài đặt thư viện, cấu hình môi trường production, ...). Tất cả các file này cần phải được nén thành một file zip trước khi gửi lên form đăng kí.

Sau khi gửi form đăng kí dịch vụ mới thành công, lúc này dịch vụ mới đang ở trạng thái chờ được phê duyệt bởi người quản trị hệ thống. Quy trình phê duyệt một dịch vụ được mô tả bằng biểu đồ luồng sau:



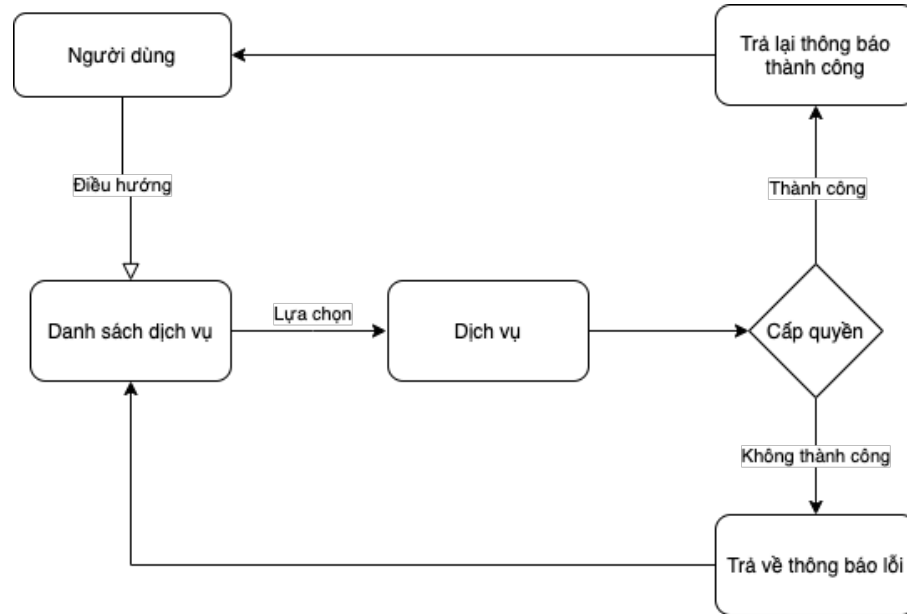
Hình 2.3: Quy trình đăng kí dịch vụ của nhà cung cấp dịch vụ

Khi service được cài đặt và triển khai thành công trên cloud cùng với API Server, người dùng có thể bắt đầu đăng kí sử dụng dịch vụ đó trong nhà của mình.

2.3.3. Xây dựng quy trình quản lý dịch vụ bên thứ ba cho người dùng

Như đã đề cập trong phần đặt vấn đề, hệ thống tôi đề xuất trong khóa luận tốt nghiệp này sẽ cho phép người dùng quản lý và tích hợp các dịch vụ bên thứ ba vào trong hệ thống của mình. Cụ thể hơn, người dùng có thể thêm dịch vụ mới vào hệ thống, cũng như phân quyền khởi chạy dịch vụ đó.

Thứ nhất, đối với tính năng thêm dịch vụ mới vào hệ thống của mình, người dùng sẽ truy cập vào một trang web của API Server và điều hướng sang danh sách các dịch vụ hiện đang khả dụng. Sau khi lựa chọn được dịch vụ mà mình mong muốn, người dùng sẽ tiến hành lựa chọn loại thiết bị trong nhà thông minh mà họ cho phép dịch vụ đó có thể truy cập đến và hoàn tất quá trình thêm dịch vụ vào trong hệ thống của mình.



Hình 2.4: Quy trình người dùng thêm dịch vụ mới vào hệ thống

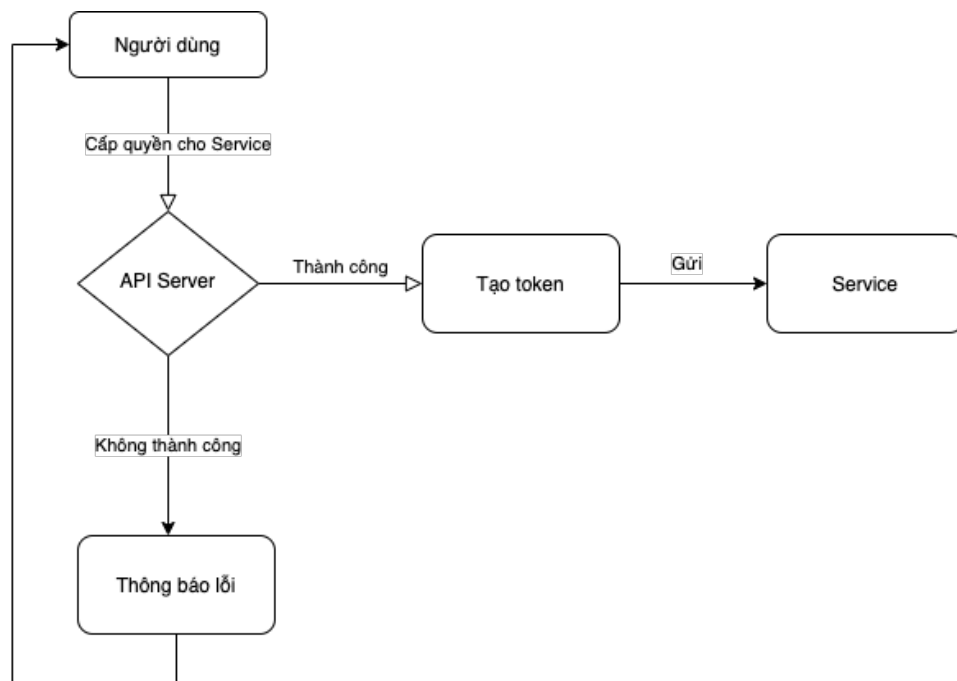
Thứ hai, đó là tính năng quản lý dịch vụ, mà cụ thể là tính năng khởi chạy một dịch vụ để sử dụng. Lúc này, người dùng vẫn sẽ truy cập vào trang web của API Server và bắt đầu lựa chọn dịch vụ mà họ muốn khởi chạy, cũng như các thông số khác về quyền hạn của dịch vụ, như là: ID nhà, danh sách các phòng và danh sách các loại thiết bị được phép truy cập. Danh sách các loại thiết bị trong quy trình này chính là tập con của danh sách các loại thiết bị người dùng đã lựa chọn trong quy trình thêm dịch vụ mới vào trong hệ thống của mình. Có nghĩa là, trong quá trình thêm dịch vụ mới vào trong nhà, người dùng lựa chọn hai loại thiết bị mà dịch vụ có thể truy cập đến, thì tại bước phân quyền và khởi chạy dịch vụ, người dùng chỉ có thể lựa chọn 1 trong 2, hoặc cả 2 loại dịch vụ đó khi khởi chạy dịch vụ.

VD: Khi thêm dịch vụ “Điều khiển đèn thông minh” vào trong nhà của mình, người dùng sẽ lựa chọn dịch vụ đó sẽ có quyền truy cập một số các loại thiết bị, như là: Đèn, cảm biến ánh sáng, cảm biến hồng ngoại, cảm biến phát hiện người.

Sau đó, khi muốn khởi chạy dịch vụ “Điều khiển đèn thông minh”, người dùng sẽ lựa chọn dịch vụ đó trong danh sách các dịch vụ có sẵn, cùng với nhà 1

và phòng ngủ, phòng khách. Tiếp đến, người dùng lúc này chỉ muốn dịch vụ đó truy cập vào hai loại thiết bị là đèn và cảm biến ánh sáng để hoạt động. Vì vậy trong danh sách các loại thiết bị có thể truy cập được thiết lập từ bước thêm dịch vụ vào trong nhà, mặc định hệ thống tích đủ bốn lựa chọn loại thiết bị. Người dùng chỉ cần tích bỏ cảm biến hồng ngoại và cảm biến phát hiện người để khởi chạy dịch vụ.

Khi đã hoàn tất các lựa chọn của mình, người dùng sẽ khởi chạy dịch vụ. Lúc này, API Server sẽ tiến hành lưu các thông số về quyền của dịch vụ do người dùng lựa chọn phía trên vào trong payload của token, kí xác thực token đó và đồng thời gửi token tới dịch vụ bằng một GET request theo URL: `http://{host}:{port}/auth?token={token}`



Hình 2.5: Quy trình khởi chạy dịch vụ phía người dùng

2.4. Thiết kế cơ chế xác thực dịch vụ truy cập vào hệ thống

Để có thể xác thực và định danh dịch vụ đang thực hiện truy vấn tới hệ thống, API Server sẽ nhận diện bằng token được gửi kèm trong các truy vấn. Token này được tạo ra khi người dùng thực hiện khởi chạy dịch vụ thông qua giao diện web của API Server. Tại đây, người dùng sẽ lựa chọn dịch vụ mà mình muốn khởi chạy và lựa chọn nhà, các phòng và các loại thiết bị mà dịch vụ đó có thể truy cập vào. Sau đó, API Server sẽ tiếp nhận các lựa chọn đó của người dùng và tiến hành lưu các lựa chọn đó vào trong payload của token, cùng với thời hạn dịch vụ đó có thể sử dụng token. Cuối cùng, API Server sẽ gửi token đó tới dịch vụ và khởi chạy.

Dịch vụ, sau khi nhận được token từ phía API Server, sẽ phải lưu lại token đó để có thể đính kèm vào trong header của các truy vấn lấy dữ liệu tới API Server. Các nhà phát triển sẽ có nhiều lựa chọn để lưu trữ token cho dịch vụ của mình, như là lưu vào trong bộ nhớ phiên làm việc của trình duyệt (session storage), cơ sở dữ liệu (database),...

Do token được tạo theo chuẩn JSON Web Token nên điều này đồng nghĩa với việc payload của token sẽ được mã hóa bằng BASE64. Vì vậy, dịch vụ có thể dễ dàng giải mã payload của token để lấy được danh sách nhà, các phòng và các loại thiết bị được phép truy cập đến. Ví dụ về payload của token sau khi đã được giải mã:

```
{
  "service_id": "071tt6jp75losdie-ket54pl2-tp64",
  "authenticator": "testacc",
  "home": 9,
  "room": [7, 8, 11],
  "room_name": [
    {
      "room_id": 7,
      "room_name": "some room"
    },
    {
      "room_id": 8,
      "room_name": "some room"
    }
  ],
  "device_type": ["Lighting", "IlluminanceSensor"],
  "topic": [
    {
      "topic": "/testacc/9/some room/Lighting/#",
      "privilege": 4
    },
    {
      "topic": "/testacc/9/some room/IlluminanceSensor/#",
      "privilege": 4
    },
    {
      "topic": "/testacc/9/some room/Lighting/#",
      "privilege": 4
    },
    {
      "topic": "/testacc/9/some room/IlluminanceSensor/#",
      "privilege": 4
    }
  ]
}
```



```

{
  "topic": "/testacc/9/some room/Lighting/#",
  "privilege": 4
},
{
  "topic": "/testacc/9/some room/IlluminanceSensor/#",
  "privilege": 4
}
],
"iat": 1588869698,
"exp": 1588891298
}

```

Ý nghĩa các trường dữ liệu lưu trong payload của token:

- *service_id*: ID định danh của dịch vụ đang được cấp quyền truy cập vào hệ thống.
- *authenticator*: Username của người dùng xác thực cho dịch vụ được truy cập vào hệ thống của mình.
- *home*: ID nhà của *authenticator* (hay còn gọi là username của người dùng) mà dịch vụ được phép truy cập đến.
- *room*: Danh sách ID các phòng thuộc *home* mà dịch vụ có quyền truy cập đến.
- *room_name*: Danh sách các phòng cùng với tên của các phòng đó
- *device_type*: Danh sách các loại thiết bị trong nhà của người dùng mà dịch vụ có quyền truy vấn dữ liệu.
- *topic*: Danh sách các topic MQTT Broker dịch vụ có quyền subscribe vào để lấy dữ liệu theo thời gian thực.

Khi dịch vụ bắt đầu thực hiện gửi truy vấn lấy dữ liệu đến API Server, trong header của gói tin gửi lên API Server cần phải kèm theo một trường chứa token để API Server có thể xác thực truy vấn được gửi lên. Trường dữ liệu trong header của gói tin sẽ có dạng:

x-access-token: \${token}

Tại phía API Server sẽ không lưu dữ liệu về quyền truy cập của dịch vụ vào trong cơ sở dữ liệu. Lý do API Server lưu trữ quyền truy cập của dịch vụ bên thứ ba vào trong payload của JSON Web Token thay vì lưu vào trong cơ sở dữ liệu, là bởi vì:

- Thứ nhất, do mỗi lần khởi chạy dịch vụ là một lần người dùng sẽ cấp quyền mới cho người dùng cùng với một token mới. Đồng thời, mỗi lần khởi chạy có thể là có những quyền hạn nhất định khác nhau. Vì vậy việc lưu các quyền truy cập này vào cơ sở dữ liệu sẽ làm cho bảng cơ sở dữ

liệu bị dư thừa về dữ liệu rất nhiều, bao gồm dữ liệu về những quyền truy cập cũ, đã hết hạn,...

- Thứ hai, nếu mỗi lần service gửi yêu cầu truy vấn dữ liệu lên API Server, đồng nghĩa với việc API Server sẽ phải gửi truy vấn lấy dữ liệu về quyền của dịch vụ từ cơ sở dữ liệu về. Việc làm này sẽ làm giảm đáng kể hiệu năng của API Server và của cơ sở dữ liệu. Chưa kể đến nếu có hàng trăm, hàng nghìn yêu cầu truy vấn dữ liệu từ phía dịch vụ gửi đến API Server, sẽ cần hàng trăm, hàng nghìn yêu cầu tương ứng truy xuất vào cơ sở dữ liệu để lấy thông tin về quyền hạn truy cập của dịch vụ đó. Cách xử lý này sẽ làm tốn kém cả về mặt tài nguyên cũng như thời gian để xử lý yêu cầu của hệ thống.
- Thứ ba, JWT trước khi được truyền tới dịch vụ để khởi chạy, đều được “kí” chứng nhận bằng các thuật toán mã hóa, VD: HS256, RS256,... để đảm bảo tính toàn vẹn của token trong suốt quá trình gửi đi. Như vậy, nếu như một dịch vụ nào đó cố tình tạo ra token giả và sử dụng nó để truy cập vào hệ thống, khả năng cao token đó sẽ là không hợp lệ, do API Server sẽ xác thực token, bao gồm kiểm tra phần chữ kí của token có hợp lệ hay không. Nếu như không hợp lệ, API Server sẽ từ chối truy cập đó của dịch vụ.

API Server khi nhận được yêu cầu lấy dữ liệu từ phía dịch vụ, sẽ bắt đầu tiến hành xác thực tính hợp lệ của token, bao gồm: Kiểm tra xem token có hợp lệ hay không, token còn hạn sử dụng hay không. Tiếp đến, API Server sẽ tiến hành giải mã payload của token, đối chiếu các tham số về nhà, phòng, loại thiết bị mà dịch vụ đang yêu cầu đến, có nằm trong danh sách về nhà, phòng và loại thiết bị được lưu ở trong payload của token hay không. Nếu tất cả phép kiểm tra của API Server đều trả về kết quả là đúng, API Server sẽ tiến hành xử lý yêu cầu từ phía dịch vụ, và trả lại kết quả cho dịch vụ đó. Nhưng ngược lại, nếu một trong các phép kiểm tra trả về kết quả là không hợp lệ, API Server sẽ lập tức từ chối yêu cầu từ phía dịch vụ, và đồng thời trả về mã HTTP 403 cùng lý do vì sao bị từ chối, dưới định dạng JSON.

CHƯƠNG 3: TRIỂN KHAI VÀ ĐÁNH GIÁ HỆ THỐNG

3.1. Triển khai thực tế

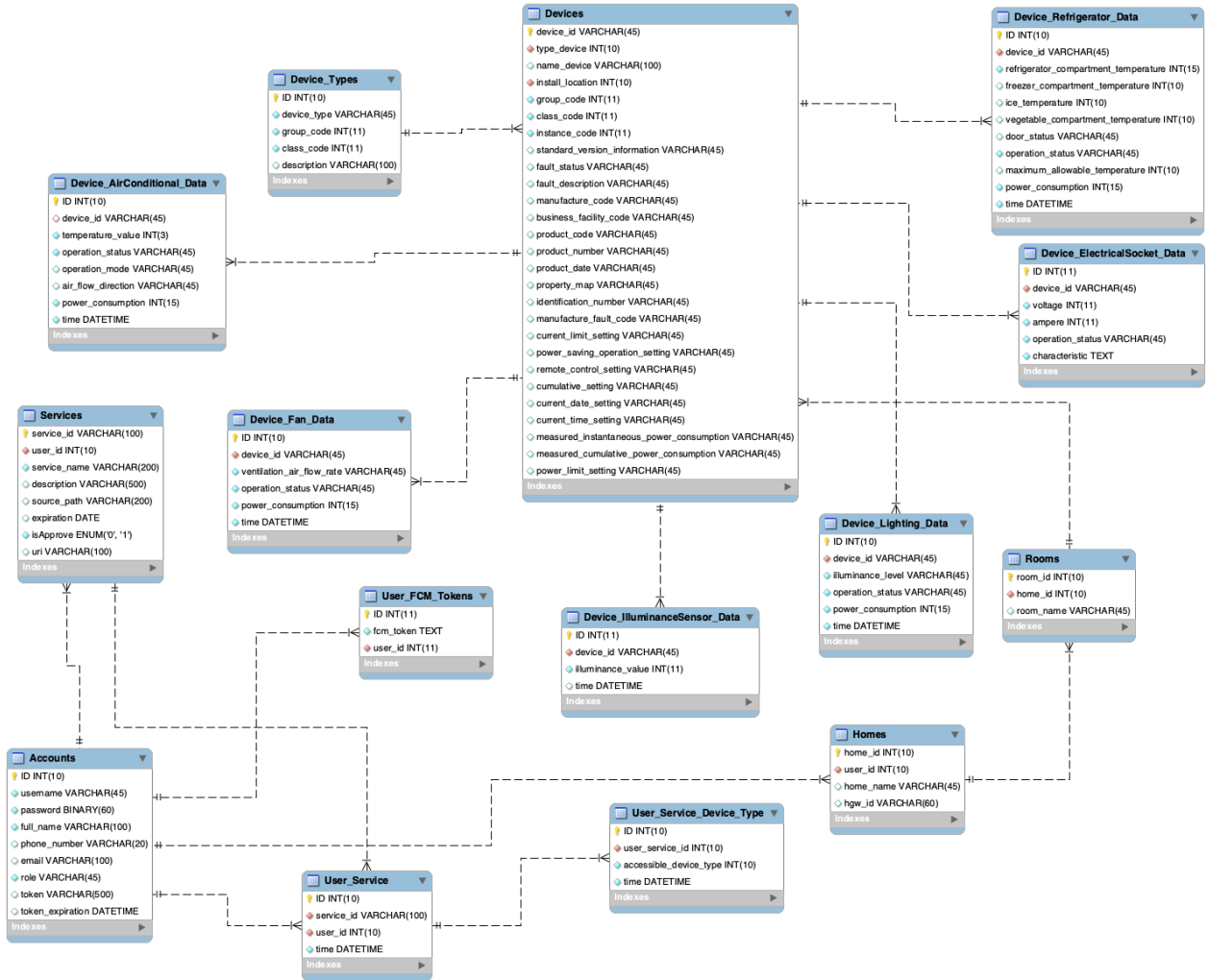
3.1.1. Cài đặt API Server

API Server xây dựng trong khóa luận tốt nghiệp này được phát triển bằng ngôn ngữ Node.js, sử dụng hệ cơ sở dữ liệu MySQL của Oracle, và được triển khai trên một chiếc server ảo chạy hệ điều hành Ubuntu 18.04, với 10 nhân CPU và 10GB RAM.

3.1.2. Cài đặt Database

Trong API Server, tôi cài đặt hệ cơ sở dữ liệu MySQL để lưu các thông tin, bao gồm:

- Thông tin người sử dụng: tên đăng nhập, mật khẩu, và loại tài khoản là người dùng hoặc là nhà cung cấp dịch vụ và một số thông tin phụ khác.
- Thông tin về nhà, phòng và các thiết bị trong nhà thông minh của người dùng.
- Thông tin về các dịch vụ được đăng tải bởi nhà cung cấp dịch vụ, và các dịch vụ được đăng kí sử dụng bởi người dùng
- Dữ liệu về các loại thiết bị trong nhà thông minh của người dùng, lấy trực tiếp từ MQTT Broker. Hiện tại, trong phạm vi khóa luận tốt nghiệp này, API Server mới chỉ hỗ trợ lưu dữ liệu của 6 loại thiết bị, bao gồm:
 - **Dữ liệu về đèn chiếu sáng:** trạng thái hoạt động, cường độ sáng, mức điện năng tiêu thụ của đèn.
 - **Dữ liệu về điều hòa nhiệt độ:** mức nhiệt độ hoạt động, trạng thái hoạt động, chế độ điều hòa, hướng gió và lượng điện năng tiêu thụ.
 - **Dữ liệu về ổ cắm điện:** hiệu điện thế ổ điện, cường độ dòng điện của ổ điện, trạng thái hoạt động và đặc trưng dòng điện của thiết bị đang cắm vào ổ điện.
 - **Dữ liệu về quạt:** tốc độ quạt gió, trạng thái hoạt động và lượng điện năng tiêu thụ.
 - **Dữ liệu về cảm biến ánh sáng:** cường độ sáng đo được tại vị trí lắp đặt cảm biến.
 - **Dữ liệu về tủ lạnh:** nhiệt độ ngăn đựng thực phẩm, nhiệt độ ngăn đá cấp đông, nhiệt độ ngăn mát, nhiệt độ ngăn để rau củ quả, trạng thái cửa (đóng/ mở), trạng thái hoạt động của tủ, nhiệt độ tối đa cho phép, lượng điện năng tiêu thụ.



Hình 3.1: Các bảng cơ sở dữ liệu của API Server

3.1.3. Cài đặt MQTT Broker và bộ thư viện xác thực MQTT Broker

Để xác thực kết nối, cũng như xác thực yêu cầu đăng kí hoặc gửi dữ liệu lên MQTT Broker, tôi sử dụng một thư viện mã nguồn mở trên Github [5]. Do thư viện này được phát triển dựa trên ngôn ngữ Golang của Google, nên trước hết, chúng ta cần phải cài đặt trình biên dịch ngôn ngữ Golang trên server. Tiếp đến, chúng ta cũng cần phải cài đặt một số các gói phần mềm cần thiết cho hệ điều hành để có thể biên dịch và khởi chạy được thư viện xác thực MQTT Broker, và đồng thời cũng cần phải cài đặt MQTT Broker từ mã nguồn để có được phiên bản mới nhất, tương thích tốt nhất với bộ thư viện xác thực trên. Chi tiết về cách cài đặt bộ thư viện xác thực cũng như cài đặt MQTT Broker, tác giả đã mô tả rất kĩ trong tài liệu đặc tả thư viện của mình trên Github.

Sau khi cài đặt thành công MQTT Broker và bộ thư viện xác thực, ta tiến hành cấu hình các tham số cho bộ thư viện xác thực MQTT Broker. Chi tiết về các tham số cấu hình như sau:

```
auth_plugin /home/uet/mosquitto-go-auth/go-auth.so

auth_opt_backends http, jwt
auth_opt_check_prefix false
allow_anonymous false

auth_opt_cache true
auth_opt_cache_reset true

auth_opt_cache_host localhost
auth_opt_cache_port 6379
auth_opt_cache_password 
auth_opt_cache_db 4
auth_opt_auth_cache_seconds 15
auth_opt_acl_cache_seconds 15

auth_opt_http_host 
auth_opt_http_port 3000
auth_opt_http_getuser_uri /api/auth/mqtt-users?type=http
auth_opt_http_superuser_uri /api/auth/mqtt-superusers?type=http
auth_opt_http_aclcheck_uri /api/auth/mqtt-acls?type=http
auth_opt_http_with_tls true
auth_opt_http_params_mode form

auth_opt_jwt_remote true
auth_opt_jwt_host 
auth_opt_jwt_port 3000
auth_opt_jwt_getuser_uri /api/auth/mqtt-users?type=jwt
auth_opt_jwt_superuser_uri /api/auth/mqtt-superusers?type=jwt
auth_opt_jwt_aclcheck_uri /api/auth/mqtt-acls?type=jwt
auth_opt_jwt_with_tls true
auth_opt_jwt_params_mode form
```

Hình 3.2: Cấu hình bộ thư viện xác thực MQTT Broker

Ở đây tôi lựa chọn hai phương thức xác thực MQTT Broker, đó là gửi gói tin HTTP POST lên server để xác thực trường hợp là người dùng kết nối tới MQTT Broker, và JWT để xác thực bằng JSON Web Token đối với các dịch vụ bên thứ ba truy cập vào MQTT Broker để lấy dữ liệu theo thời gian thực.

Do bộ thư viện xác thực này hỗ trợ khả năng lưu bộ nhớ đệm kết quả xác thực vào trong Redis (một cơ sở dữ liệu NoSQL, lưu dữ liệu vào trong RAM nên tốc

độ truy xuất rất nhanh), nên tôi cũng thiết lập các tham số khởi tạo kết nối tới Redis, bao gồm: thông tin về địa chỉ kết nối tới Redis database cùng với cổng, mật khẩu xác thực truy cập Redis và thời gian lưu dữ liệu tại bộ nhớ đệm.

Sau khi đã cài đặt Redis để lưu bộ nhớ đệm xác thực MQTT Broker, tôi tiếp tục cấu hình xác thực hai giao thức: HTTP và JWT. Đối với hai giao thức này, các tham số cần khai báo, bao gồm:

- Địa chỉ nơi đặt máy chủ xác thực và cổng dịch vụ. Trong trường hợp này, máy chủ xác thực chính là API Server.
- Đường dẫn để gửi yêu cầu kiểm tra xác thực. Cả 2 phương thức xác thực đều có 3 tính năng kiểm tra:
 - Kiểm tra xem người dùng đang truy cập đến có tồn tại hay không, hoặc token đang dùng để xác thực có hợp lệ hay không.
 - Kiểm tra người dùng đang truy cập tới có phải là người quản trị cấp cao của hệ thống hay không. Việc kiểm tra quyền quản trị cấp cao này diễn ra khi một người dùng cố gắng đăng kí lấy dữ liệu vào topic không thuộc quyền hạn của mình. Ở đây, do phương thức xác thực bằng JWT sẽ chỉ được dùng bởi các dịch vụ bên thứ ba, nên điều đó đồng nghĩa với việc sẽ không có trường hợp các token dùng để xác thực kết nối tới MQTT Broker có quyền quản trị cao nhất. Nhưng vì ràng buộc của bộ thư viện xác thực, nên tham số về đường dẫn kiểm tra quyền quản trị cấp cao là bắt buộc. Để giải quyết nhược điểm này, tôi vẫn cung cấp một API để kiểm tra token có phải người quản trị cấp cao hay không, nhưng kết quả trả về sẽ luôn là không hợp lệ.
 - Kiểm tra xem người dùng hoặc dịch vụ có được đăng kí vào một topic cụ thể lấy dữ liệu hay không.
- Một số các tham số phụ khác, như là tùy chọn sử dụng kết nối bảo mật SSL/TLS giữa thư viện xác thực và phương thức gửi dữ liệu lên máy chủ xác thực là dạng form hay JSON.

Cuối cùng, khi đã cấu hình xong bộ thư viện xác thực, ta tiến hành lưu tệp tin chứa các tham số được định nghĩa ở trên và khởi chạy MQTT Broker với tham số truyền vào là đường dẫn tên tệp tin cấu hình được lưu ở trên. Ở đây, tôi lưu tệp tin đó tại đường dẫn `/etc/mosquitto/conf.d/` với tên là `jwt_auth.conf`. Như vậy, câu lệnh để khởi chạy sẽ là:

“mosquitto -c /etc/mosquitto/conf.d/jwt_auth.conf”

Nếu khởi chạy thành công, hệ thống sẽ in ra thông báo như hình 3.3 dưới đây:

```

uet@srv-NguyenHoaiSon:~$ mosquitto -c /etc/mosquitto/mosquitto.conf
1589899641: Loading config file /etc/mosquitto/conf.d/default.conf
1589899641: Loading config file /etc/mosquitto/conf.d/jwt-auth.conf
INFO[2020-05-19T21:47:21+07:00] Backend registered: HTTP
INFO[2020-05-19T21:47:21+07:00] Backend registered: JWT
INFO[2020-05-19T21:47:21+07:00] Cache activated
INFO[2020-05-19T21:47:21+07:00] started cache redis client on DB 4
INFO[2020-05-19T21:47:21+07:00] flushed cache

```

Hình 3.3: Thông báo khởi chạy dịch vụ thành công

3.2. Quy trình đăng kí dịch vụ mới

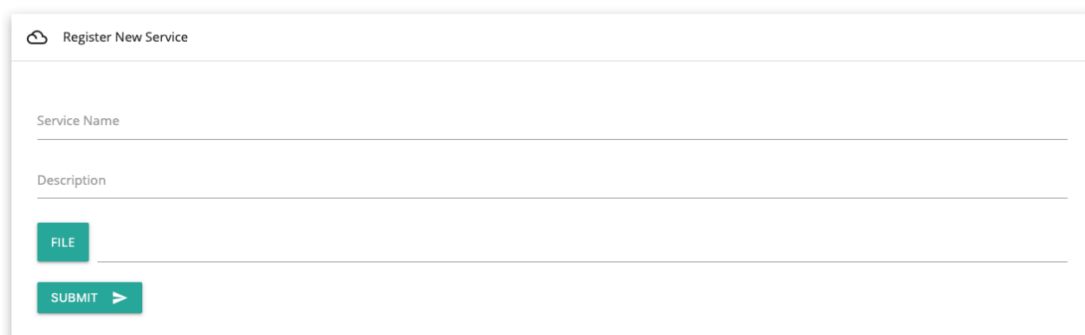
Để đăng kí một dịch vụ mới, trước hết người quản trị hệ thống sẽ phải cung cấp cho nhà phát triển dịch vụ một tài khoản với loại là nhà phát triển.

The screenshot shows a web interface for adding a new user. The header is 'Smart IoT' with navigation links 'Homes', 'Users', 'Services', and 'Log Out'. The main heading is 'Add User'. Below it, a box contains the form with the following fields: Username (filled with 'service_provider3'), Password (masked), Password Confirmation (masked), Full Name (filled with 'service_provider_3'), Phone Number, and Role (a dropdown menu currently showing 'Service Provider'). A green 'SUBMIT' button with a right-pointing arrow is at the bottom of the form.

Hình 3.4: Người quản trị đăng kí tài khoản cho nhà cung cấp dịch vụ

Sau khi đã được đăng kí một tài khoản, nhà cung cấp dịch vụ sẽ đăng nhập vào hệ thống với tên đăng nhập và mật khẩu được cung cấp bởi người quản trị hệ thống. Lúc này, nhà cung cấp dịch vụ có thể bắt đầu đăng kí dịch vụ mới của mình lên hệ thống thông qua biểu mẫu ở hình 3.5

Add Service



Hình 3.5: Biểu mẫu đăng kí dịch vụ mới lên hệ thống dành cho nhà cung cấp dịch vụ

Tại đây, nhà cung cấp dịch vụ sẽ cần phải đăng kí tên cho dịch vụ của mình, kèm theo mô tả dịch vụ cho người dùng khi họ lựa chọn đăng kí sử dụng, và cuối cùng là đính kèm một tập tin nén định dạng zip, bao gồm mã nguồn của của dịch vụ, kèm theo hướng dẫn thiết lập các tham số dịch vụ lần đầu và khởi chạy dịch vụ trên cloud.

Lúc này, người quản trị hệ thống sẽ nhận được thông báo về một dịch vụ mới được đăng kí trên hệ thống, đang chờ được phê duyệt và triển khai lên hệ thống

Smart IoT							Homes	Users	Services	Log Out
Un-Approved Service										
#	Service ID	Service Name	Description	Source Code	Publisher	Action				
1	tubfckmwzevu1989-s4a4hiip-kh5g	Air Conditional control	control AC using a smart algorithms	/Users/khaidq/Documents/Project/API IoT Platform/public/uploads/smartlight-api.zip	sp2	<div>APPROVE DELETE</div>				

Made by khaidq

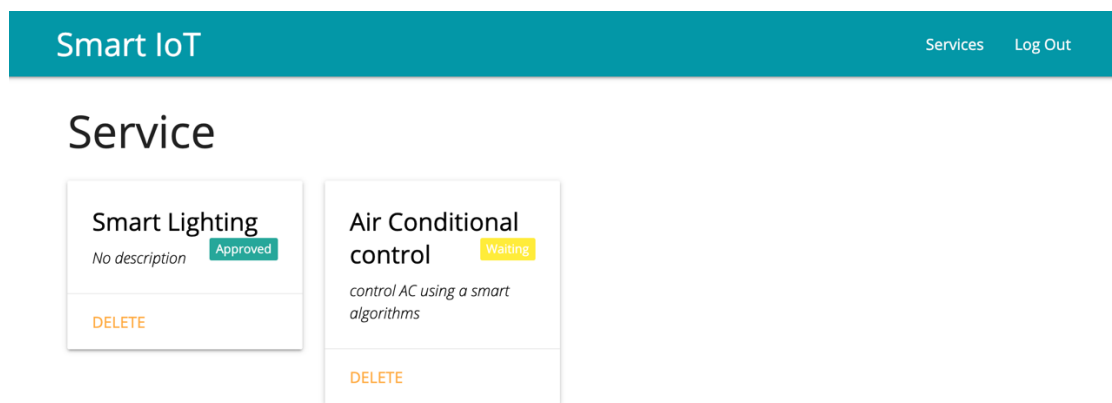
Hình 3.6: Danh sách các dịch vụ đang chờ được triển khai và phê duyệt

Thông qua giao diện như trong hình 3.6, người quản trị hệ thống có thể dễ dàng nắm bắt được danh sách các dịch vụ đang chờ phê duyệt, bao gồm các thông tin như:

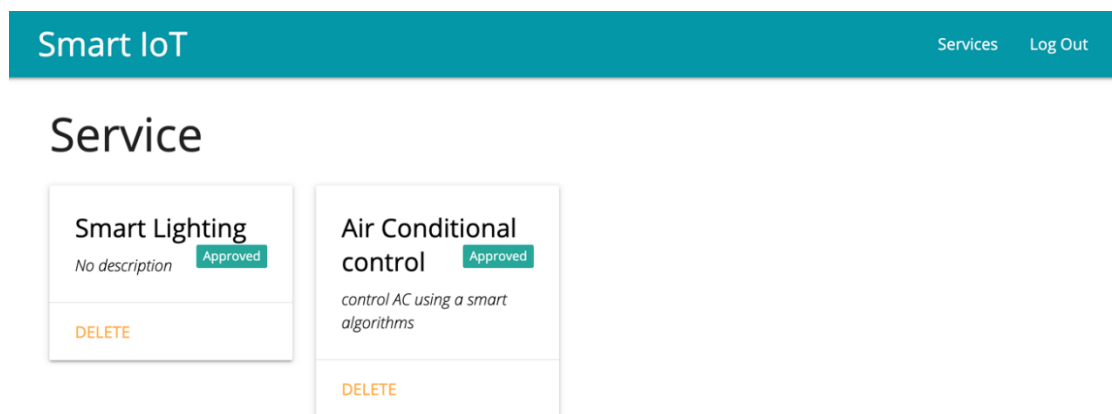
- Tên dịch vụ
- Mô tả dịch vụ
- Đường dẫn đến mã nguồn của dịch vụ trên hệ thống

- Tên nhà cung cấp đăng kí dịch vụ

Lúc này, người quản trị hệ thống sẽ tiến hành di chuyển tới đường dẫn chứa mã nguồn của dịch vụ để tiến hành kiểm duyệt mã nguồn tìm kiếm mã độc nếu có. Sau khi quá trình kiểm duyệt diễn ra thành công và dịch vụ được chấp nhận, người quản trị hệ thống sẽ triển khai dịch vụ đó trên hạ tầng đám mây cùng với API Server, thiết lập công dịch vụ cũng như URI để dịch vụ có thể truy cập được. Cuối cùng, người quản trị sẽ ấn nút “Approve” trên hệ thống, để thông báo rằng dịch vụ đã được phê duyệt và cho phép hoạt động trên hệ thống. Các nhà cung cấp dịch vụ có thể kiểm tra trạng thái của dịch vụ mình đăng kí ngay tại trang chủ dành cho nhà cung cấp dịch vụ.



Hình 3.7: Thông báo khi một dịch vụ đang chờ được phê duyệt



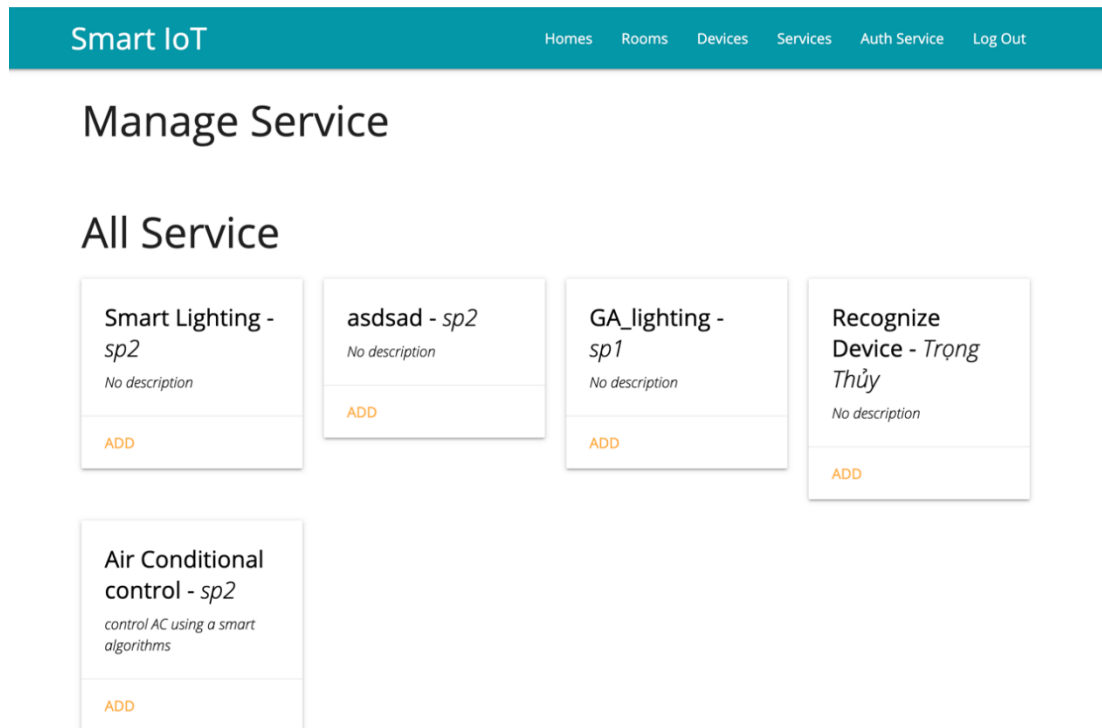
Hình 3.8: Thông báo khi dịch vụ đã được phê duyệt thành công

3.3. Quy trình quản lý dịch vụ trong hệ thống của người dùng

3.3.1. Quy trình thêm dịch vụ mới vào hệ thống

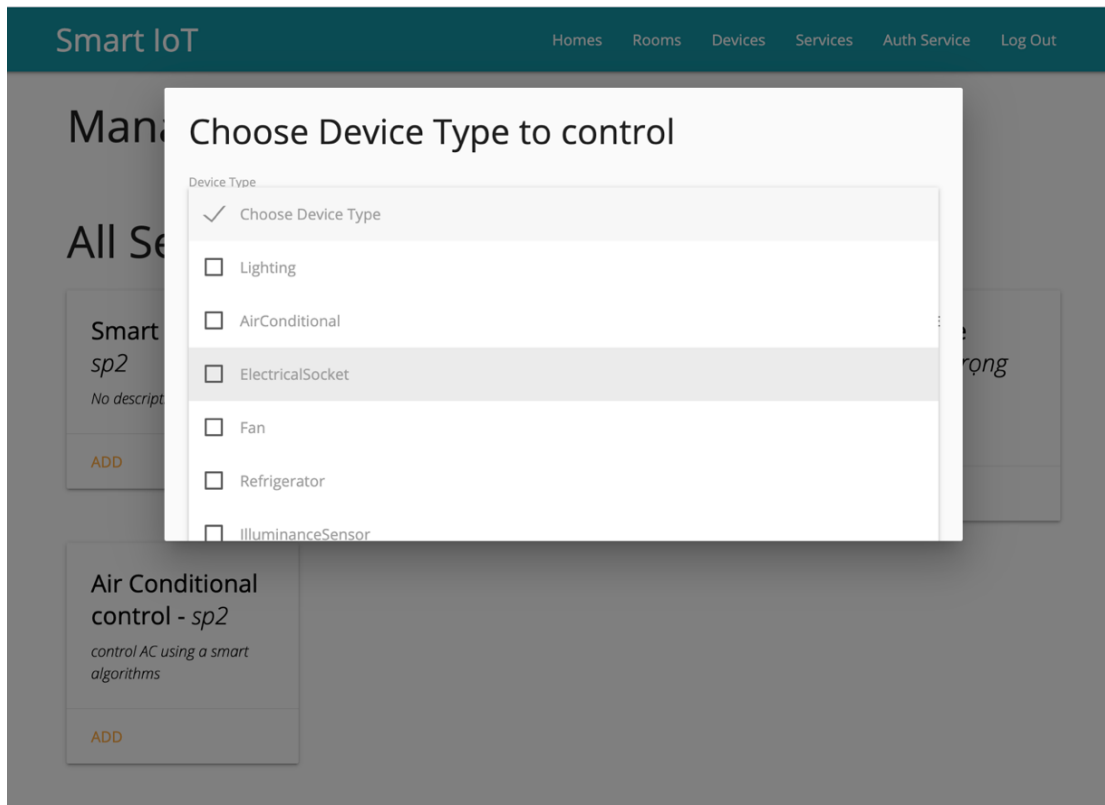
Để thêm được dịch vụ vào trong hệ thống của mình, trước hết người dùng cần đăng nhập vào hệ thống với tên đăng nhập và mật khẩu được cung cấp bởi người quản trị hệ thống.

Khi đã đăng nhập thành công vào hệ thống, người dùng sẽ tiến hành chuyển tới mục quản lý các dịch vụ (Service) trên giao diện web của mình.



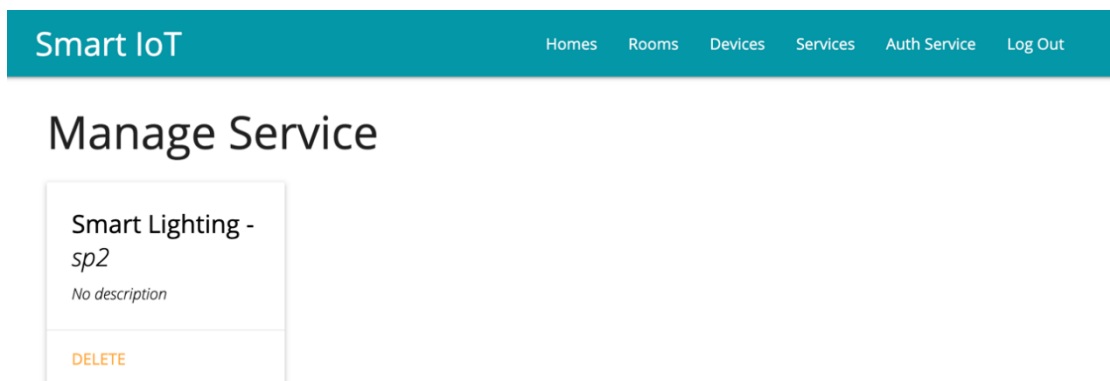
Hình 3.9: Giao diện quản lý dịch vụ tại hệ thống cho người dùng

Tại đây, hệ thống sẽ liệt kê tất cả các dịch vụ đã được phê duyệt và đang được hoạt động trên API Server. Người dùng sau khi đã lựa chọn được dịch vụ mong muốn được thêm vào hệ thống, sẽ tiến hành ấn nút “Add” dịch vụ đó. Một hộp thoại sẽ được hiện ra như trong hình 3.10, cho phép người dùng lựa chọn loại thiết bị mà dịch vụ đó có thể truy cập đến.



Hình 3.10: Hộp thoại cấp quyền loại thiết bị dịch vụ được truy cập vào hệ thống của người dùng

Sau khi đã lựa chọn xong các loại thiết bị mà dịch vụ có thể truy cập tới để lấy dữ liệu, dịch vụ đó giờ đây đã được thêm thành công vào hệ thống của người dùng. Người dùng có thể theo dõi các dịch vụ mà mình đã thêm vào hệ thống ở ngay tại giao diện liệt kê các dịch vụ trong hệ thống.



Hình 3.11: Giao diện liệt kê các dịch vụ đã được người dùng thêm vào hệ thống của mình

3.3.2. Quy trình khởi chạy dịch vụ

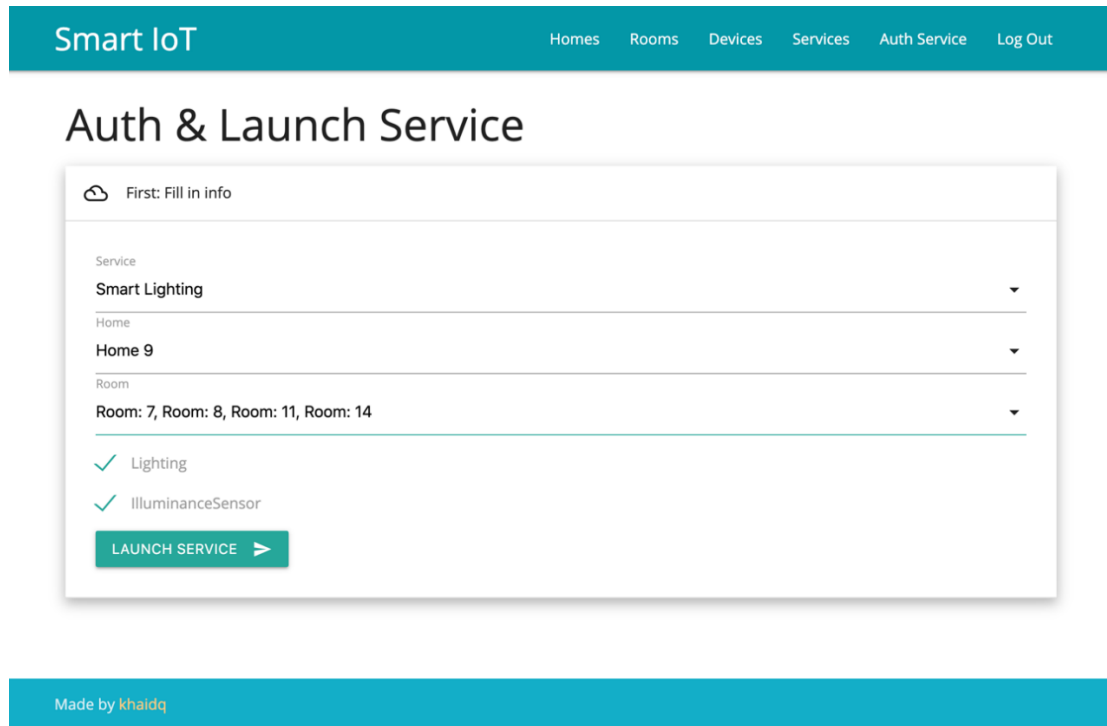
Để khởi chạy một dịch vụ, người dùng trước tiên sẽ điều hướng tới trang giao diện xác thực dịch vụ (Auth Service)

The screenshot shows a web application titled "Smart IoT" with a navigation bar containing links for "Homes", "Rooms", "Devices", "Services", "Auth Service", and "Log Out". The main heading is "Auth & Launch Service". Below this, a form titled "First: Fill in info" contains three dropdown menus labeled "Service", "Home", and "Room". The "Service" dropdown is currently open, showing "Choose Service". The "Home" dropdown shows "Choose Home", and the "Room" dropdown shows a blank space. At the bottom of the form is a green button labeled "LAUNCH SERVICE" with a right-pointing arrow. The footer of the page says "Made by khaidq".

Hình 3.12: Giao diện khởi chạy dịch vụ

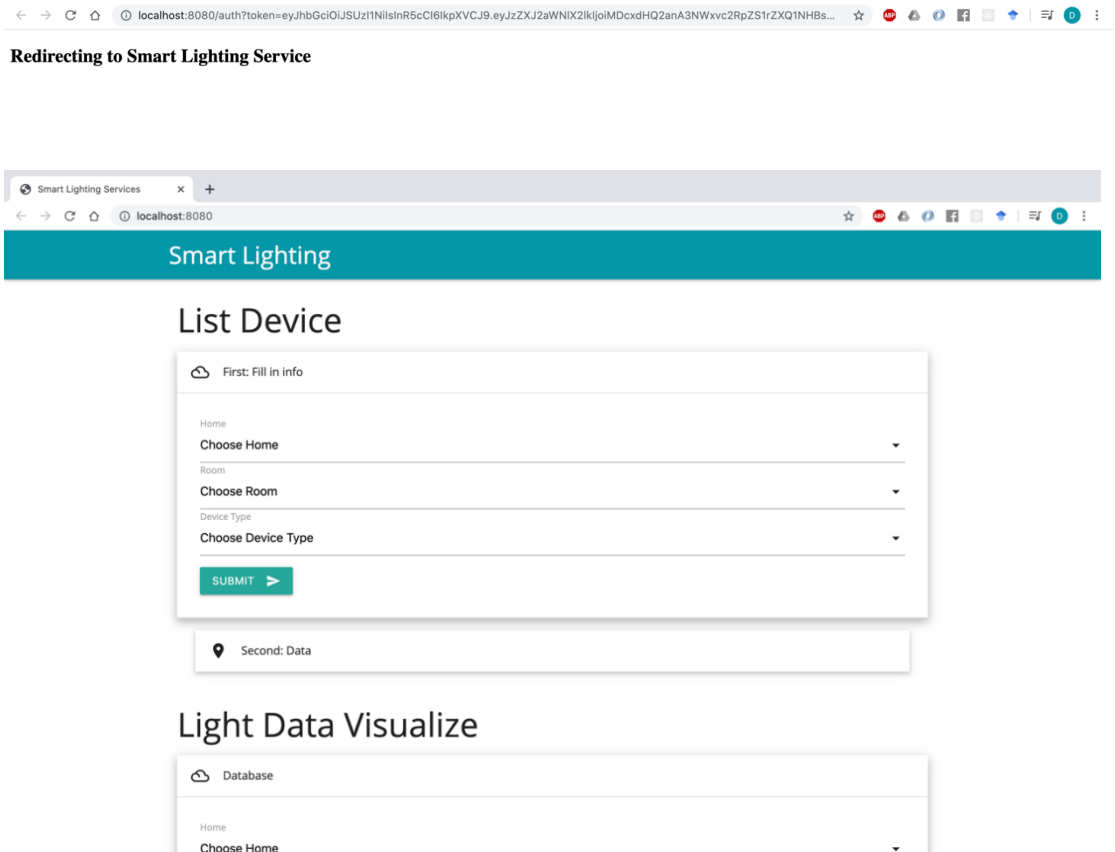
Tại đây, người dùng sẽ tiến hành lựa chọn các tùy chọn theo trình tự sau, bao gồm:

- **Bước 1:** Lựa chọn dịch vụ người dùng muốn khởi chạy, từ danh sách các dịch vụ người dùng đã đăng ký vào trong hệ thống của mình.
- **Bước 2:** Lựa chọn nhà mà người dùng muốn khởi chạy dịch vụ trong đó. Tại đây, người dùng chỉ có thể lựa chọn một nhà tại mỗi một lần khởi chạy.
- **Bước 3:** Lựa chọn danh sách các phòng thuộc nhà mà người dùng đã lựa chọn tại bước 2. Ở đây, người dùng có thể lựa chọn nhiều phòng để khởi chạy dịch vụ.
- **Bước 4:** Lựa chọn các loại thiết bị mà người dùng cho phép dịch vụ truy cập đến trong mỗi lần khởi chạy



Hình 3.13: Giao diện sau khi người dùng lựa chọn xong các tùy chọn để khởi chạy một dịch vụ

Sau khi đã lựa chọn xong các tùy chọn, người dùng sẽ tiến hành khởi chạy dịch vụ với các tùy chọn trên, bằng cách nhấn vào nút “Launch Service”. Hệ thống sẽ ghi nhận các tùy chọn của người dùng và đồng thời tạo JWT Token chứa thông tin về quyền truy cập của dịch vụ đó. Cuối cùng hệ thống sẽ điều hướng sang khởi chạy thông qua một GET request chứa token tới dịch vụ.



Hình 3.14: Giao diện API Server điều hướng để khởi chạy dịch vụ

3.4. Đánh giá kết quả thực hiện

3.4.1. Cài đặt các chức năng trên API Server

API Server được đề xuất trong khóa luận tốt nghiệp này đã được cài đặt các chức năng, bao gồm:

- Xác thực quyền truy cập của dịch vụ tới API Server, sử dụng JSON Web Token.
- Xác thực quyền truy cập vào MQTT Broker của dịch vụ, sử dụng bộ thư viện bên thứ ba và xử lý xác thực tại API Server.
- Khả năng đăng kí một dịch vụ mới lên hệ thống.
- Khả năng thêm một dịch vụ vào trong hệ thống nhà thông minh của người dùng. Ở đây cụ thể dịch vụ tôi đã lựa chọn để thử nghiệm là dịch vụ điều khiển đèn thông minh.

Môi trường được tôi sử dụng để tiến hành đánh giá hiệu năng của API Server là một máy chủ ảo hóa với 10 nhân CPU và 10GB RAM. Bảng thông thực tế đo được tại phía đặt máy chủ, sử dụng phiên bản dòng lệnh của phần mềm Speedtest phát triển bởi Ookla, thu được tốc độ là 46.51Mbps cho đường tải xuống và 41.98Mbps cho đường tải lên. Độ trễ đến máy chủ giao động trong khoảng từ 4 đến 5 ms với 2.2% số lượng gói tin mất mát trên đường truyền. Ngoài ra, phiên

bản của các phần mềm được cài đặt tại phía máy chủ để có thể chạy được API Server bao gồm:

- Hệ điều hành mã nguồn mở Ubuntu phiên bản 18.04.4 LTS.
- Node.js phiên bản 10.20.1.
- MySQL phiên bản 14.14, bản phân phối 5.7.30.
- Redis server phiên bản 4.0.9
- Mosquitto Broker phiên bản 1.6.9

Để thực hiện các bài đánh giá hiệu năng của API Server, tôi sử dụng một công cụ mã nguồn mở có tên gọi Apache JMeter để thực hiện các bài kiểm tra này. Phần mềm được chạy tại máy tính cá nhân, với các tham số cấu hình cho từng trường hợp thử nghiệm như sau:

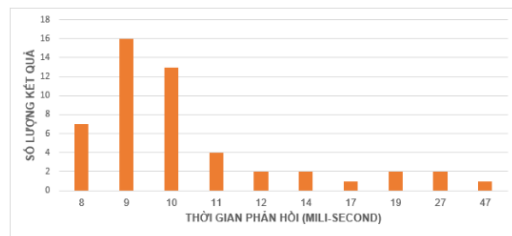
- **Trường hợp xác thực kết nối tới MQTT Broker:**
 - URL: <https://{host}:3000/api/auth/mqtt-users?type=jwt>
 - Phương thức: POST
 - Token được gửi trong header của gói tin.
 - Tốc độ xử lý: tương ứng với các mức tốc độ 1, 10, 50, 100, 500 và 1000 dịch vụ/giây được API Server xác thực trong 1 giây.
 - Số vòng lặp: 50 vòng với mỗi mức tốc độ xử lý.
- **Trường hợp xác thực dịch vụ thực hiện truy vấn tới API Server:**
 - URL: https://{host}:3000/api/services/devices?home_id=9&&room_id=7&&device_type=Lighting
 - Phương thức: GET.
 - Token được gửi trong header của gói tin.
 - Tốc độ xử lý: tương ứng với các mức tốc độ 1, 10, 50, 100, 500 và 1000 dịch vụ được API Server xác thực trong 1 giây.
 - Số vòng lặp: 50 vòng với mỗi mức tốc độ xử lý.
- **Trường hợp xác thực dịch vụ thực hiện truy vấn tới API Server:**
 - URL: https://{host}:3000/api/services/devices?home_id=9&&room_id=7&&device_type=Lighting
 - Phương thức: GET.
 - Token được gửi trong header của gói tin.
 - Tốc độ xử lý: tương ứng với các mức tốc độ xác thực 1, 10, 50, 100, 200 và 500 dịch vụ truy vấn lấy dữ liệu từ API Server trong 1 giây.
 - Số vòng lặp: 50 vòng với mỗi mức tốc độ xử lý.

3.4.2. Thời gian chạy xác thực kết nối tới MQTT Broker

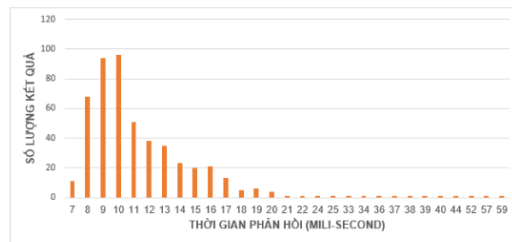
Để tiến hành bài đánh giá thời gian xác thực MQTT Broker của API Server, tôi thiết lập 6 kịch bản thử nghiệm, lần lượt theo thứ tự, bao gồm:

- **Trường hợp 1 (a):** API Server tiến hành xác thực dịch vụ kết nối tới MQTT Broker với tốc độ 1 dịch vụ/giây.
- **Trường hợp 2 (b):** API Server tiến hành xác thực dịch vụ kết nối tới MQTT Broker với tốc độ 10 dịch vụ/giây.
- **Trường hợp 3 (c):** API Server tiến hành xác thực dịch vụ kết nối tới MQTT Broker với tốc độ 50 dịch vụ/giây.
- **Trường hợp 4 (d):** API Server tiến hành xác thực dịch vụ kết nối tới MQTT Broker với tốc độ 100 dịch vụ/giây.
- **Trường hợp 5 (e):** API Server tiến hành xác thực dịch vụ kết nối tới MQTT Broker với tốc độ 500 dịch vụ/giây.
- **Trường hợp 6 (f):** API Server tiến hành xác thực dịch vụ kết nối tới MQTT Broker với tốc độ 1000 dịch vụ/giây.

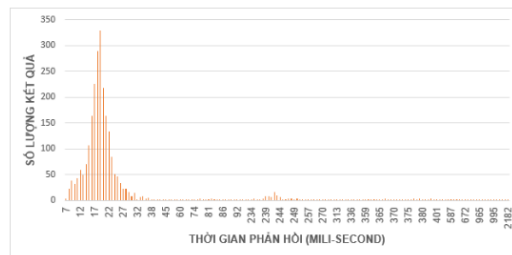
Kết quả của 6 trường hợp kiểm thử được mô tả trong biểu đồ 3.1 dưới đây.



a) API Server xử lý xác thực kết nối tới MQTT Broker với tốc độ 1 dịch vụ/giây

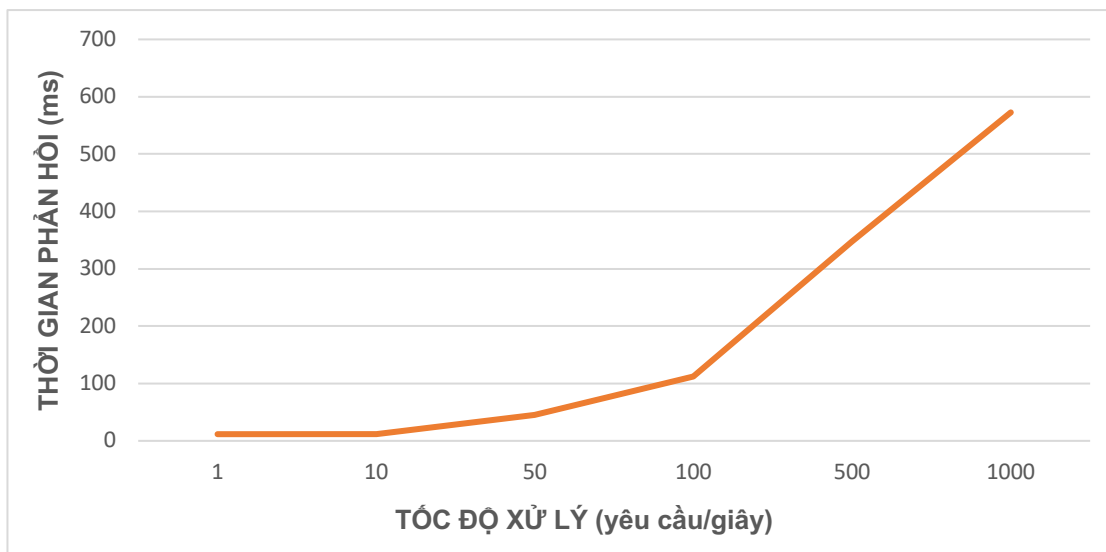


b) API Server xử lý xác thực kết nối tới MQTT Broker với tốc độ 10 dịch vụ/giây



Từ biểu đồ trên ta có thể thấy rằng với mỗi tốc độ xác thực dịch vụ kết nối tới MQTT Broker cùng lúc khác nhau sẽ có sự phân bố khá khác nhau về thời gian phản hồi, cụ thể:

- Khi tốc độ xác thực là 1 dịch vụ/giây, thời gian hệ thống phản hồi nhanh nhất là 8ms và lâu nhất là 42ms. Thời gian phản hồi chủ yếu trong khoảng từ 8 – 10 ms, và nhiều nhất có 16 kết quả được phản hồi sau 9ms. Ngoài ra các thời gian phản hồi khác có số lượng kết quả rải rác từ 1- 4 kết quả và thấp nhất với thời gian phản hồi là 17ms và 47 ms chỉ có 1 kết quả.
- Đối với tốc độ là 10 dịch vụ/giây, thời gian phản hồi dao động từ 7 – 59 ms. Trong đó phân bố nhiều nhất trong khoảng 8 – 13 ms, với cao nhất là 10ms với 96 kết quả.
- Trường hợp tốc độ là 50 dịch vụ/giây, thời gian phản hồi kết quả dao động trong khoảng từ 7 – 2182 ms. Trong đó phân bố nhiều nhất trong khoảng từ 15 – 22 ms, cao nhất là 19ms với 329 kết quả.
- Khi tăng số lượng dịch vụ được xác thực kết nối tới MQTT Broker lên 100 dịch vụ cùng lúc thì thời gian phản hồi chủ yếu rơi vào khoảng từ 16 – 25 ms, trong đó nhiều nhất là 19ms với 576 kết quả. Ngoài ra các kết quả khác được trả về sau khoảng từ 7 – 2182 ms.
- Đối với 500 dịch vụ/giây, ta thấy rằng kết quả được phản hồi trong khoảng thời gian từ 6 – 5232 ms. Trong đó phân bố nhiều nhất trong khoảng từ 22 – 40 ms. Và tập trung nhiều nhất là 1516 kết quả được phản hồi sau 23ms.
- Cuối cùng khi tăng tốc độ xác thực lên 1000 dịch vụ/giây, ta có thể thấy rằng thời gian phản hồi nằm trong khoảng từ 6 – 13399 ms. Trong đó tập trung nhiều nhất trong khoảng 7 – 57 ms với số lượng kết quả được phản hồi sau 21ms là nhiều nhất là 1052 kết quả.
- Bên cạnh đó, theo như quan sát thực tế, khi tốc độ xác thực là 1000 dịch vụ/giây, băng thông phía máy chủ được sử dụng vượt quá ngưỡng tối đa, dẫn đến số lượng gói tin mất mát trên đường truyền tăng cao. Từ đó dẫn đến nhiều yêu cầu xác thực sẽ không được ghi nhận và thực hiện.



Biểu đồ 3.2: Thời gian phản hồi tương ứng với số lượng dịch vụ cần xác thực với từng mức tốc độ/giây.

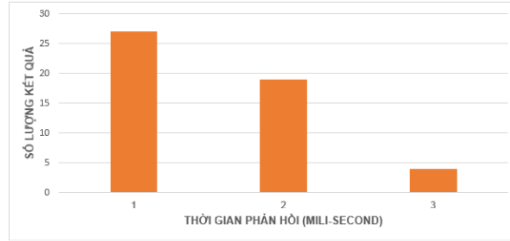
Qua biểu đồ 3.2, ta có thể thấy rằng thời gian phản hồi trung bình tăng dần theo số lượng dịch vụ cần được xác thực đồng thời. Khi tốc độ dưới 100 dịch vụ/giây, hệ thống cho thời gian chạy rất thấp ($\leq 100\text{ms}$). Khi tăng tốc độ xác thực lên 500 dịch vụ/giây cùng lúc và 1000 dịch vụ/giây cùng lúc, thời gian phản hồi trung bình cũng tăng lên ngưỡng 350ms và gần tiệm cận mức 600ms. Đây là một khoảng thời gian nằm trong ngưỡng chấp nhận được.

3.4.3. Thời gian chạy xác thực dịch vụ thực hiện truy vấn tới API Server

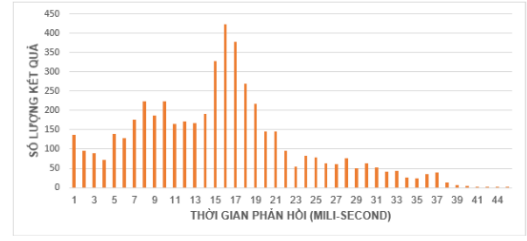
Để tiến hành bài đánh giá thời gian xác thực MQTT Broker của API Server, tôi thiết lập 6 kịch bản thử nghiệm, lần lượt theo thứ tự, bao gồm:

- **Trường hợp 1 (a):** API Server tiến hành xác thực dịch vụ truy vấn tới hệ thống với tốc độ 1 dịch vụ/giây.
- **Trường hợp 2 (b):** API Server tiến hành xác thực dịch vụ truy vấn tới hệ thống với tốc độ 10 dịch vụ/giây.
- **Trường hợp 3 (c):** API Server tiến hành xác thực dịch vụ truy vấn tới hệ thống với tốc độ 50 dịch vụ/giây.
- **Trường hợp 4 (d):** API Server tiến hành xác thực dịch vụ truy vấn tới hệ thống với tốc độ 100 dịch vụ/giây.
- **Trường hợp 5 (e):** API Server tiến hành xác thực dịch vụ truy vấn tới hệ thống với tốc độ 500 dịch vụ/giây.
- **Trường hợp 6 (f):** API Server tiến hành xác thực dịch vụ truy vấn tới hệ thống với tốc độ 1000 dịch vụ/giây.

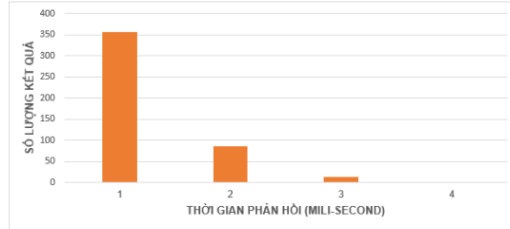
Kết quả của 6 trường hợp kiểm thử được mô tả trong biểu đồ 3.3 dưới đây.



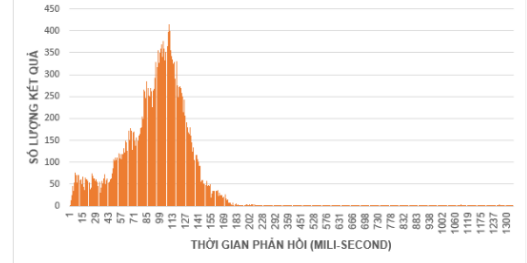
a) API Server xử lý xác thực khi truy cập hệ thống với tốc độ 1 dịch vụ/giây



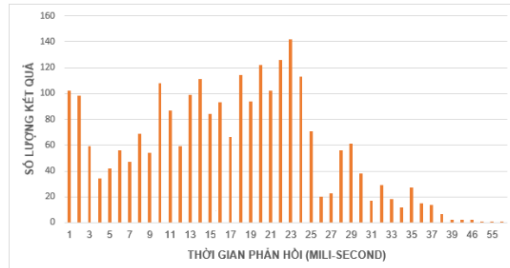
d) API Server xử lý xác thực khi truy cập hệ thống với tốc độ 100 dịch vụ/giây



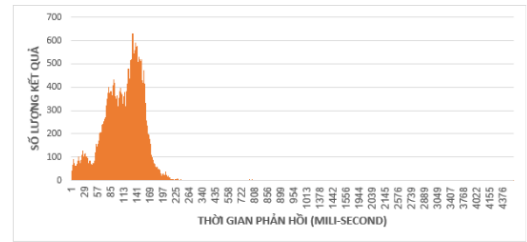
b) API Server xử lý xác thực khi truy cập hệ thống với tốc độ 10 dịch vụ/giây



e) API Server xử lý xác thực khi truy cập hệ thống với tốc độ 500 dịch vụ/giây



c) API Server xử lý xác thực khi truy cập hệ thống với tốc độ 50 dịch vụ/giây



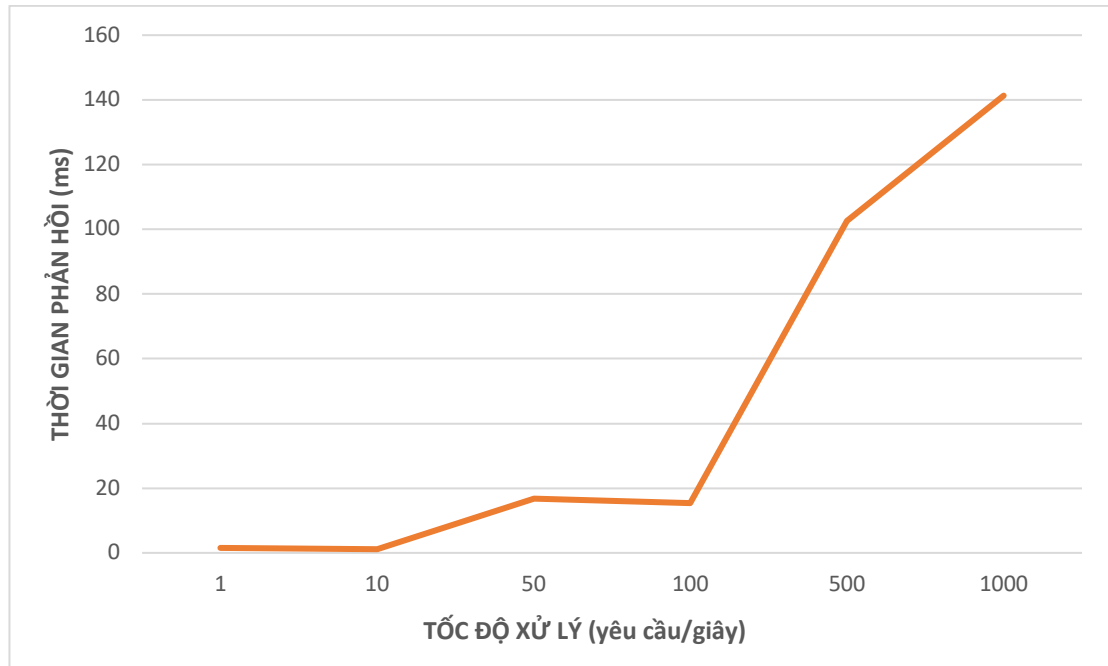
f) API Server xử lý xác thực khi truy cập hệ thống với tốc độ 1000 dịch vụ/giây

Biểu đồ 3.3: Phân bố thời gian chạy khi API Server thực hiện xác thực đồng thời các dịch vụ truy vấn hệ thống với từng mức tốc độ/giây

Theo dõi biểu đồ 3.3, ta có thể thấy với mỗi số lượng dịch vụ cần được đồng thời xác thực sẽ có sự phân bố khác nhau về thời gian phản hồi như sau:

- Với trường hợp tốc độ xác thực khi truy cập hệ thống là 1 dịch vụ/giây thì đa số kết quả cho thấy hệ thống mất 1ms để xử lý xác thực. Chỉ 4 trường hợp được phản hồi sau 3ms.
- Tương tự vậy, khi tốc độ được tăng lên 10 dịch vụ/giây, chủ yếu kết quả được phản hồi chỉ sau 1ms với 356 kết quả. Còn lại chỉ rải rác một số lượng nhỏ kết quả được phản hồi sau 2 – 4 ms với số lượng giảm dần.
- Khi tốc độ xác thực là 50 dịch vụ/giây, ta có thể thấy sự thay đổi rõ rệt thông qua sự phân bố không đồng đều trên biểu đồ. Thời gian phản hồi chủ yếu trong khoảng từ 1 – 2 ms, 10 – 11ms, 13 – 16 ms và 18 – 22 ms. Nhanh nhất là 1ms, chậm nhất là 61ms và nhiều nhất là 20ms với 120 kết quả.
- Với tốc độ là 100 dịch vụ/giây, thời gian phản hồi trong khoảng từ 1 – 48 ms. Trong đó tập trung nhiều nhất trong khoảng từ 7 – 19 ms. Cao nhất là 16ms với 424 kết quả.

- Khi tăng tốc độ lên 500 dịch vụ/giây, thời gian phản hồi trong khoảng từ 1 – 1391 ms. Trong đó tập trung rất ít từ sau 144ms. Ngoài ra thời gian phản hồi chủ yếu rơi vào khoảng từ 49 – 141 ms, và cao nhất là 109ms void 415 kết quả.
- Đối với 1000 dịch vụ/giây tại cùng một thời điểm, ta có thể thấy sự phân bố thời gian phản hồi khá tương đồng với 500 dịch vụ. Số kết quả được phản hồi từ 173ms thấp, thời gian phản hồi tập trung chủ yếu trong khoảng từ 52 – 172 ms và cao nhất là 131 – 132 ms với tổng cộng là 1285 kết quả.



Biểu đồ 3.4: Thời gian phản hồi tương ứng số lượng dịch vụ được xác thực khi truy vấn lấy dữ liệu tới API Server với từng mức tốc độ/giây.

Qua biểu đồ 3.4, nhìn chung ta có thể thấy rằng thời gian phản hồi tỷ lệ thuận với tốc độ xử lý yêu cầu của API Server. Với tốc độ từ 1 – 200 dịch vụ/giây đồng thời, thời gian phản hồi trung bình của API Server không tăng quá nhiều và không tăng đột ngột (tăng từ 40ms đến 640ms). Tuy nhiên khi tăng tốc độ lên 500 dịch vụ/giây thì thời gian phản hồi đã tăng vọt lên tới trên mức 2.6s.

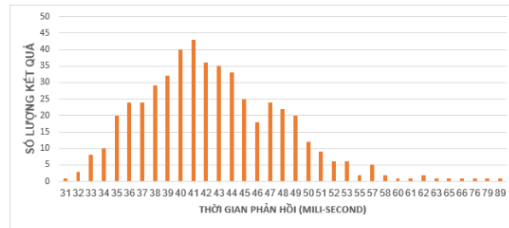
Lí giải cho sự tăng vọt về thời gian phản hồi này nằm ở lưu lượng băng thông sử dụng trong kịch bản thử nghiệm với tốc độ xử lý 500 dịch vụ/giây. Đó là do lượng băng thông cần thiết vượt quá ngưỡng có thể đáp ứng được của đường truyền tại phía server. Điều này làm cho tốc độ gửi/nhận dữ liệu giữa API Server và từng dịch vụ bị giảm đi rõ rệt, từ đó làm tăng thời gian cần thiết để hoàn thành truy vấn.

3.4.4. Thời gian chạy khi dịch vụ thực hiện truy vấn dữ liệu từ đến API Server

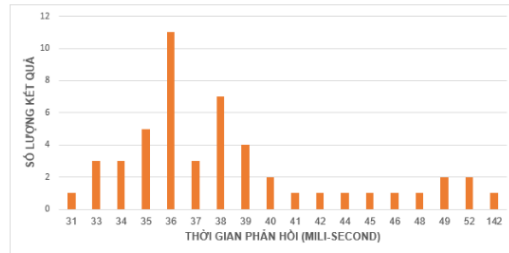
Để tiến hành bài đánh giá thời gian hệ thống phản hồi yêu cầu truy vấn lấy dữ liệu từ phía dịch vụ, tôi thiết lập 6 kịch bản thử nghiệm, lần lượt theo thứ tự, bao gồm:

- **Trường hợp 1 (a):** API Server tiến hành xử lý truy vấn lấy dữ liệu tới hệ thống với tốc độ 1 dịch vụ/giây.
- **Trường hợp 2 (b):** API Server tiến hành xử lý truy vấn lấy dữ liệu tới hệ thống với tốc độ 10 dịch vụ/giây.
- **Trường hợp 3 (c):** API Server tiến hành xử lý truy vấn lấy dữ liệu tới hệ thống với tốc độ 50 dịch vụ/giây.
- **Trường hợp 4 (d):** API Server tiến hành xử lý truy vấn lấy dữ liệu tới hệ thống với tốc độ 100 dịch vụ/giây.
- **Trường hợp 5 (e):** API Server tiến hành xử lý truy vấn lấy dữ liệu tới hệ thống với tốc độ 200 dịch vụ/giây.
- **Trường hợp 6 (f):** API Server tiến hành xử lý truy vấn lấy dữ liệu tới hệ thống với tốc độ 500 dịch vụ/giây.

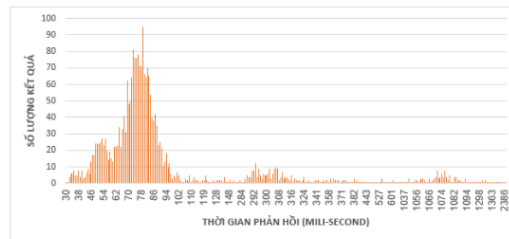
Kết quả của 6 trường hợp kiểm thử được mô tả trong biểu đồ 3.5 dưới đây.



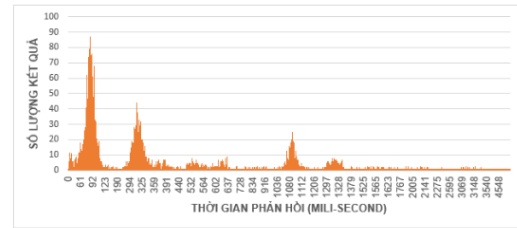
a) API Server xử lý yêu cầu lấy dữ liệu với tốc độ 1 dịch vụ/giây



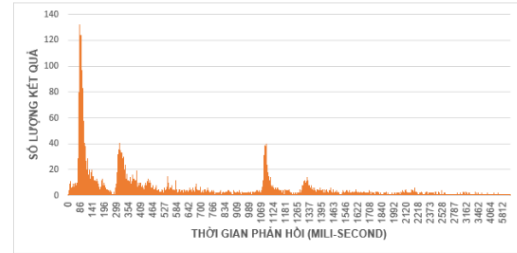
b) API Server xử lý yêu cầu lấy dữ liệu với tốc độ 10 dịch vụ/giây



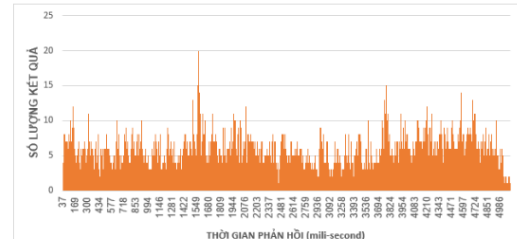
c) API Server xử lý yêu cầu lấy dữ liệu với tốc độ 50 dịch vụ/giây



d) API Server xử lý yêu cầu lấy dữ liệu với tốc độ 100 dịch vụ/giây



e) API Server xử lý yêu cầu lấy dữ liệu với tốc độ 200 dịch vụ/giây



f) API Server xử lý yêu cầu lấy dữ liệu với tốc độ 500 dịch vụ/giây

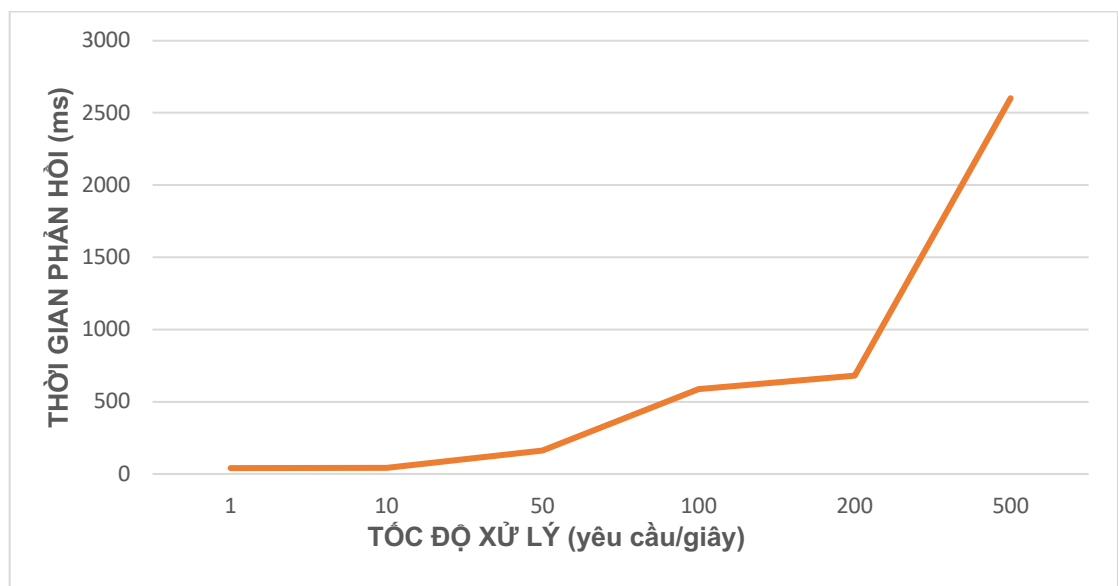
Biểu đồ 3.5: Phân bố thời gian phản hồi khi dịch vụ thực hiện truy vấn lấy dữ liệu từ API Server với từng mức tốc độ/giây

Khi dịch vụ tiến hành truy cập lấy dữ liệu tới API Server, cũng như khi truy cập hệ thống và kết nối tới MQTT Broker, với mỗi tốc độ xử lý yêu cầu lấy dữ liệu cùng lúc sẽ có sự phân bố về thời gian phản hồi nhất định:

- Khi tốc độ là 1 dịch vụ/giây tại cùng một thời điểm, ta có thể thấy rằng thời gian phản hồi nằm trong khoảng từ 31 – 142 ms. Tuy nhiên thời gian phản hồi chỉ tập trung chủ yếu trong khoảng từ 36 – 49 ms. Có thể thấy biểu đồ phân bố thời gian phản hồi khi tốc độ xử lý của API Server là 1 dịch vụ/giây khá tương đồng về khu vực phân bố với biểu đồ tốc độ xử lý 10 dịch vụ/giây và 50 dịch vụ/giây. Với tốc độ là 10 dịch vụ/giây thì thời gian phản hồi chủ yếu trong khoảng từ 35 – 39 ms. Và khi tốc độ tăng lên 100 dịch vụ/giây, thời gian phản hồi chủ yếu tập trung trong khoảng từ 49 – 91 ms.
- Với biểu đồ đánh giá về sự phân bố thời gian khi tốc độ xử lý yêu cầu lấy dữ liệu từ dịch vụ của API Server là 100 dịch vụ/giây và 200 dịch vụ/giây, ta cũng có thể thấy sự tương đồng về sự phân bố không đồng đều về thời gian phản hồi của 2 biểu đồ này. Với tốc độ xử lý là 100 dịch vụ/giây có

3 khoảng thời gian có sự tập trung phân bố, đó là 74 – 100 ms, 310 – 321 ms và 1082 – 1090 ms. Với tốc độ 200 dịch vụ/giây ta cũng thấy rằng có 3 khoảng thời gian phản hồi có sự tập trung phân bố là 73 – 118 ms, 303 – 310 ms và 1080 – 1095 ms.

- Với tốc độ xử lý là 500 dịch vụ/giây được xác thực truy cập tới API Server tại cùng một thời điểm, ta có thể thấy rằng sự phân bố về thời gian phản hồi khác so với các số lượng dịch vụ khác. Trong khi với các tốc độ xử lý thấp hơn, thời gian phản hồi chỉ tập trung trong một số khoảng nhất định, thì với tốc độ xử lý là 500 dịch vụ/giây, thời gian phản hồi lại phân bố tương đối đồng đều trên toàn bộ, không có quá nhiều sự khác biệt giữa các khoảng thời gian với nhau.

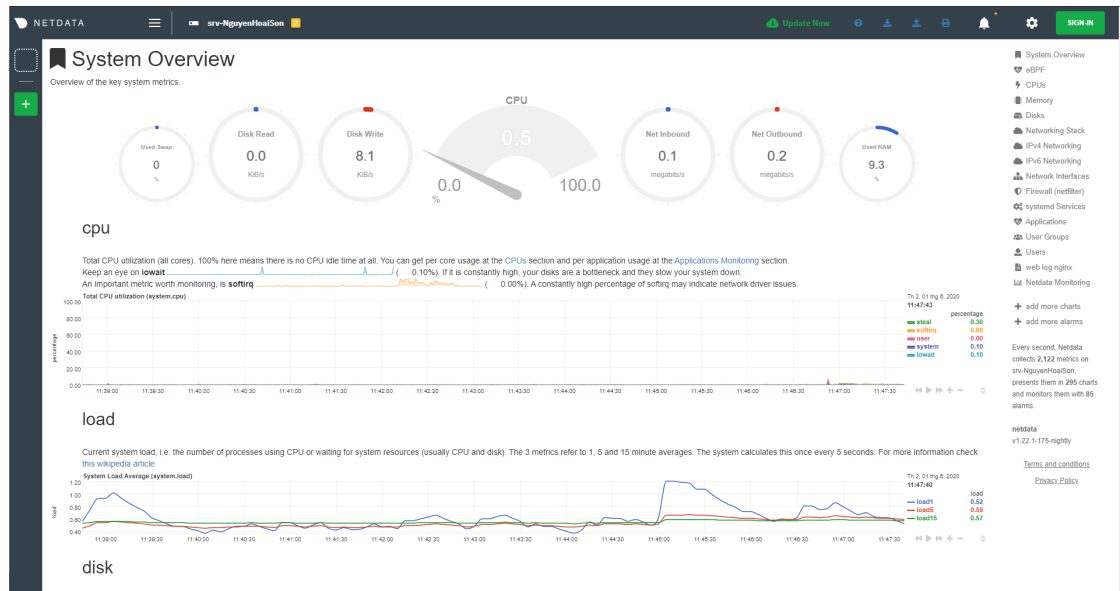


Biểu đồ 3.6: Thời gian phản hồi tương ứng số lượng dịch vụ truy vấn lấy dữ liệu đồng thời với từng mức tốc độ/giây

Tương đồng với Biểu đồ 3.4 và Biểu đồ 3.5, trong biểu đồ trên thời gian phản hồi trung bình tăng dần theo tốc độ xử lý yêu cầu. Điểm khác biệt ở đây, đó là thời gian phản hồi trung bình chỉ tăng nhẹ khi tốc độ xử lý yêu cầu tăng từ 1 đến 100 dịch vụ/giây đồng thời. Tuy nhiên khi tốc độ xử lý tăng từ 100 dịch vụ/giây lên đến 500 dịch vụ/giây và thậm chí là 1000 dịch vụ/giây, thời gian phản hồi trung bình tăng rõ rệt, lên tới trên 100ms. Trong khi tốc độ xử lý yêu cầu là 100 dịch vụ/giây đồng thời, thời gian phản hồi của API Server chỉ nằm dưới mức 20ms.

3.4.5. Tài nguyên hệ thống được sử dụng trong các tác vụ

Để xác định phần trăm CPU được sử dụng trong mỗi trường hợp kiểm thử, tôi sử dụng phần mềm theo dõi tài nguyên hệ thống tên là *netdata*.

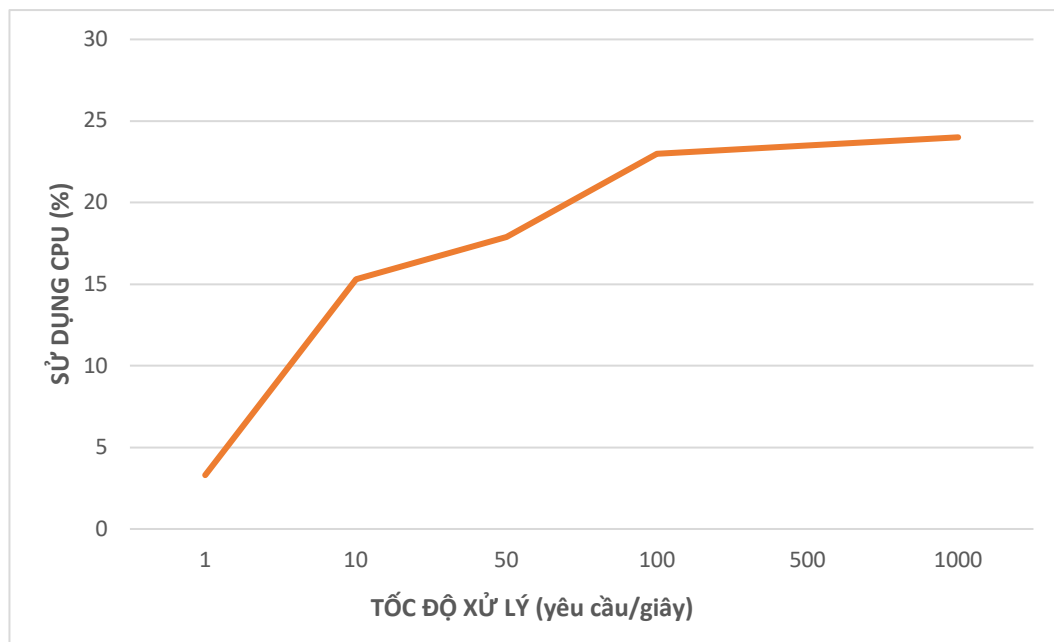


Hình 3.15: Giao diện ứng dụng netdata

Cách thức xác định phần trăm CPU được sử dụng trong các kịch bản thử nghiệm được tính theo công thức sau:

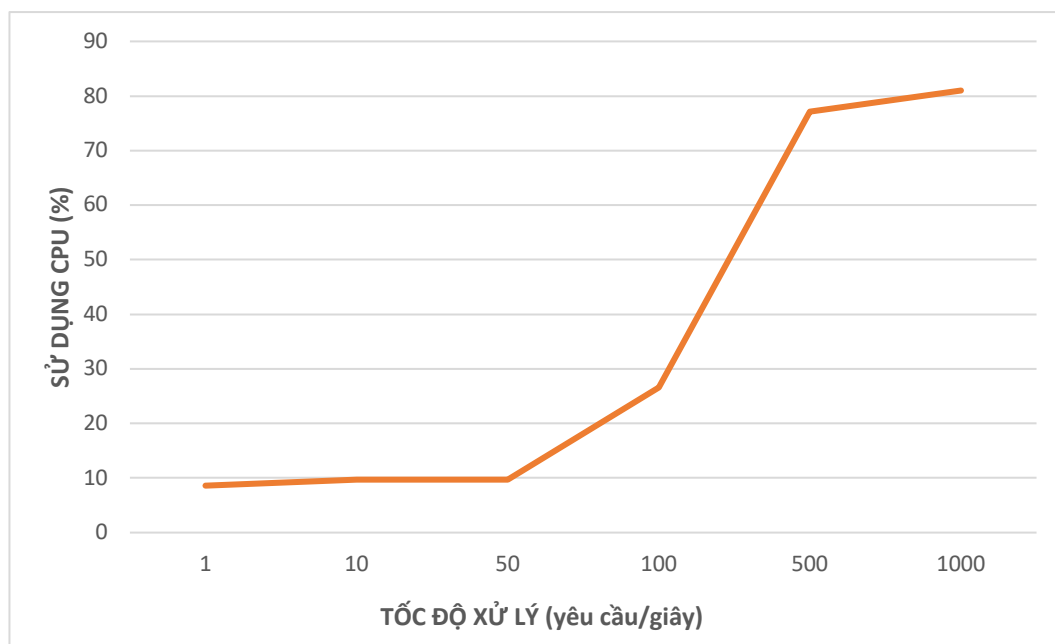
Phần trăm CPU sử dụng = Phần trăm CPU sau khi chạy kịch bản thử nghiệm - phần trăm CPU trước khi chạy kịch bản

- **Xác thực kết nối tới MQTT Broker:** Biểu đồ 3.7 thể hiện phần trăm CPU API Server sử dụng khi thực hiện tác vụ xác thực các kết nối tới MQTT Broker với tốc độ xử lý lần lượt là 1 dịch vụ/giây, 10 dịch vụ/giây, 50 dịch vụ/giây, 100 dịch vụ/giây, 500 dịch vụ/giây và 1000 dịch vụ/giây.



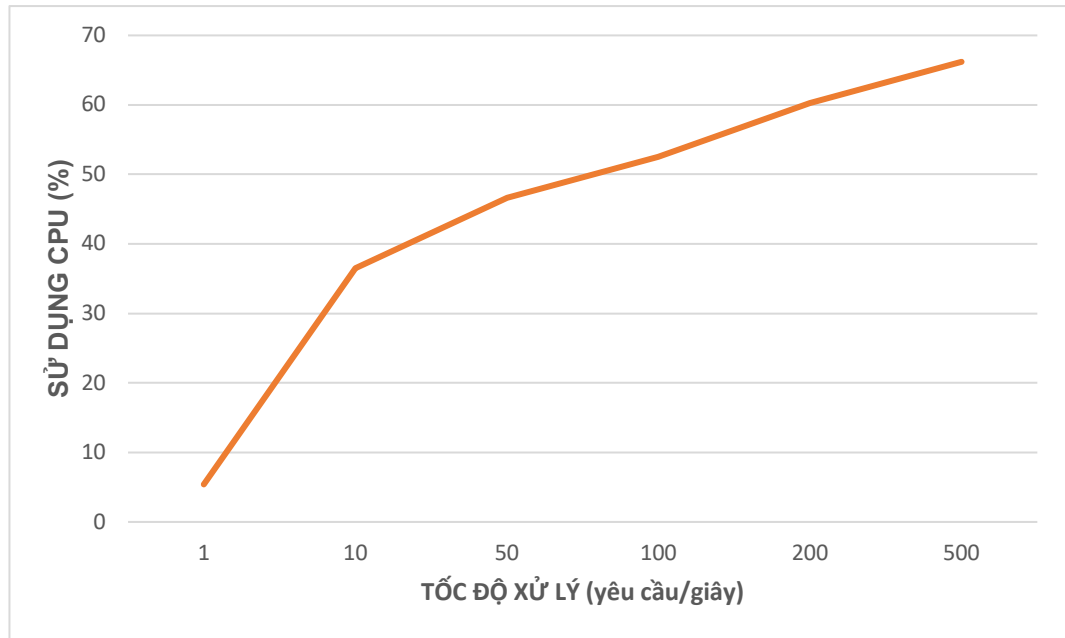
Biểu đồ 3.7: Phần trăm CPU sử dụng tương ứng với tốc độ xử lý xác thực kết nối tới MQTT Broker của API Server

- **Xác thực truy vấn tới hệ thống của dịch vụ:** Biểu đồ 3.8 thể hiện phần trăm CPU sử dụng bởi API Server trong tác vụ xác thực truy vấn của dịch vụ tới hệ thống với tốc độ xử lý tương ứng 1 dịch vụ/giây, 10 dịch vụ/giây, 50 dịch vụ/giây, 100 dịch vụ/giây, 500 dịch vụ/giây và 1000 dịch vụ/giây.



Biểu đồ 3.8: Phần trăm CPU sử dụng tương ứng với tốc độ xử lý xác thực dịch vụ đồng thời truy vấn tới API Server

- **Truy vấn lấy dữ liệu từ API Server:** Biểu đồ 3.9 thể hiện phần trăm CPU sử dụng bởi API Server trong tác vụ xử lý truy vấn lấy dữ liệu của dịch vụ tới hệ thống, trong các trường hợp tốc độ xử lý là 1 dịch vụ/giây, 10 dịch vụ/giây, 50 dịch vụ/giây, 100 dịch vụ/giây, 200 dịch vụ/giây và 500 dịch vụ/giây.



Biểu đồ 3.9: Phần trăm CPU sử dụng tương ứng với tốc độ xử lý dịch vụ truy vấn lấy dữ liệu từ API Server.

Qua 3 biểu đồ 3.7, 3.8 và 3.9, ta thấy rằng phần trăm CPU sử dụng đều tỷ lệ thuận với tốc độ xử lý yêu cầu đồng thời trên cả ba kịch bản thử nghiệm được đặt ra. Tuy nhiên có sự khác biệt giữa phần trăm CPU sử dụng tương ứng với tốc độ xử lý yêu cầu truy vấn tới API Server. Tại đây phần trăm CPU sử dụng tăng rất ít khi tốc độ xử lý chỉ có từ 1 dịch vụ/giây đến 50 dịch vụ/giây đồng thời, nhưng tăng mạnh khi tốc độ xử lý từ 50 đến 500 dịch vụ/giây tại cùng một thời điểm. Và khi tốc độ xử lý yêu cầu tăng từ 500 đến 1000 dịch vụ/giây, phần trăm CPU hệ thống sử dụng chỉ tăng nhẹ, tương đương với tốc độ xử lý yêu cầu truy cập vào hệ thống từ 1 đến 50 dịch vụ/giây. Điều đó cho thấy rằng mức tăng phần trăm CPU sử dụng tương ứng với số lượng dịch vụ đồng thời truy vấn đến API Server không đồng đều. Cụ thể:

- Trong biểu đồ 3.7 và 3.9, phần trăm CPU hệ thống sử dụng tăng mạnh khi tốc độ xử lý tăng từ 1 đến 10 dịch vụ/giây. Sau đó phần trăm CPU chỉ tăng nhẹ, không có mức chênh lệch quá cao giữa các mức tốc độ xử lý dịch vụ truy cập hệ thống đồng thời.
- Đối với trường hợp trong biểu đồ 3.8, khi tốc độ xử lý chỉ dưới 50 dịch vụ/giây đồng thời, phần trăm CPU hệ thống sử dụng tăng rất ít, không đáng kể (~ 1%). Nhưng sau đó tốc độ xử lý tăng từ 50 đến 500 dịch vụ/giây, phần trăm CPU tăng cao rõ rệt (~70%).

Thông qua các kịch bản sử dụng được thực hiện ở trên, ta có thể rút ra kết luận rằng API Server được xây dựng trong khóa luận tốt nghiệp này hoạt động ổn định với điều kiện tốc độ xử lý dịch vụ đồng thời gửi yêu cầu đến trong một giây không quá 500 dịch vụ. Một số cải tiến có thể áp dụng để có thể nâng cao tốc độ xử lý yêu cầu mà API Server có thể phục vụ được trong 1 giây, bao gồm:

- Nâng cấp cấu hình của máy chủ đám mây nơi cài đặt API Server
- Tăng băng thông đường truyền tại phía máy chủ.
- Cài đặt các bản sao của API Server trên nhiều cụm server (clustering) để giảm tải hệ thống
- Tối ưu hóa cơ sở dữ liệu bằng cách sử dụng đánh chỉ mục (indexing)
- Sử dụng bộ nhớ đệm để lưu các yêu cầu với phương thức GET tới API Server.

CHƯƠNG 4: TỔNG KẾT

1. Kết luận

Như vậy trong khóa luận tốt nghiệp này, tôi đã hoàn thiện việc xây dựng API Server tích hợp dịch vụ bên thứ ba cho nhà thông minh. Hệ thống đã giải quyết được một số nhược điểm của các IoT Platform hiện nay đang có trên thị trường.

Hệ thống cung cấp cho các nhà phát triển dịch vụ một tập các hàm thường dùng (hay còn được gọi là API), để nhà phát triển có thể nhúng vào trong sản phẩm dịch vụ của mình, từ đó các dịch vụ này có thể gửi truy vấn lấy dữ liệu lên hệ thống, hoặc gửi lệnh điều khiển tới các thiết bị trong nhà của người dùng. Bên cạnh đó, để quản lý quyền hạn truy cập dữ liệu của các dịch vụ lên hệ thống trong nhà thông minh của người dùng, API Server cũng được cài đặt cơ chế xác thực bằng JSON Web Token chứa quyền hạn của dịch vụ, bao gồm danh sách về nhà, về phòng và loại thiết bị mà dịch vụ có thể truy cập đến, ngay trong payload của token. Đồng thời, hệ thống cũng cung cấp một giao diện trang web để người dùng có thể lựa chọn quyền truy cập của dịch vụ và khởi chạy dịch vụ đó.

2. Hướng phát triển

Để hệ thống có thể hoạt động với hiệu quả cao hơn, tôi dự định phát triển tiếp hệ thống trong tương lai theo các hướng sau:

- Xây dựng quy trình kiểm duyệt mã nguồn dịch vụ khi mới được đăng kí, cùng với đó là quy trình triển khai dịch vụ lên hạ tầng đám mây cùng với API Server.
- Xây dựng thư viện để giao tiếp với IoT platform cho các nhà phát triển phần mềm bên thứ 3.
- Đảm bảo khả năng mở rộng của hệ thống khi số lượng nhà và dịch vụ tăng.

TÀI LIỆU THAM KHẢO

Tiếng Việt

- [1] Nguyễn Việt Bắc, *Xây dựng IoT Platform cho nhà thông minh*, Khóa luận tốt nghiệp, K58 – Ngành Công nghệ thông tin.
- [2] Nguyễn Hoài Sơn, Nguyễn Việt Bắc, *Xây dựng IoT Platform cho nhà thông minh tương thích chuẩn Echonet Lite*, Hội nghị Quốc gia về Điện tử, Truyền thông và Công nghệ Thông tin, 14 tháng 12 năm 2017, Thành Phố Hồ Chí Minh, Việt Nam.

Tiếng Anh

- [3] Minh Hoang Ngo, Xuan Viet Cuong Nguyen, Quang Khai Duong, Hoai Son Nguyen, *Adaptive Smart Lighting Control based on Genetic Algorithm*, 2019 25th Asia-Pacific Conference on Communication (APCC), Ho Chi Minh City, Vietnam, 2019, pp. 320-325, doi: 10.1109/APCC47188.2019.9026473..
- [4] Hoai Son Nguyen, Xuan Anh Do, Hoang Le, Van Hoang Nguyen, Quang Khai Duong, Xuan Viet Cuong Nguyen, Minh Hoang Ngo, *ECHONET Lite-based IoT Platform for Smart Homes*, accepted in IEEE RIVF 2020, public in October 2020.
- [5] Ignacio Gómez, *mosquito-go-auth*, Github repository, Available at: <https://github.com/iegomez/mosquito-go-auth>.