KHOA KỸ THUẬT VÀ CÔNG NGHỆ

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**

****

**THỰC TẬP ĐỒ ÁN CƠ SỞ NGÀNH**

**HỌC KỲ I, NĂM HỌC 2024-2025**

**XÂY DỰNG HỆ THỐNG ĐIỀU KHIỂN CÔNG TẮC TỪ XA CHO THIẾT BỊ ĐIỆN DÂN DỤNG**

|  |  |
| --- | --- |
| *Giáo viên hướng dẫn*  ThS. Nguyễn Ngọc Đan Thanh | *Sinh viên thực hiện:*  Họ tên: Nguyễn Đức Mạnh  Mã số sinh viên: 110121202  Lớp: DA21TTB |

***Trà Vinh, Tháng 12 Năm 2025***

KHOA KỸ THUẬT VÀ CÔNG NGHỆ

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**

****

**THỰC TẬP ĐỒ ÁN CƠ SỞ NGÀNH**

**HỌC KỲ I, NĂM HỌC 2024-2025**

**XÂY DỰNG HỆ THỐNG ĐIỀU KHIỂN CÔNG TẮC TỪ XA CHO THIẾT BỊ ĐIỆN DÂN DỤNG**

|  |  |
| --- | --- |
| *Giáo viên hướng dẫn*  ThS. Nguyễn Ngọc Đan Thanh | *Sinh viên thực hiện:*  Họ tên: *Nguyễn Đức Mạnh*  Mã số sinh viên: 110121202  Lớp: DA21TTB |

***Trà Vinh, Tháng 12 Năm 2025***

**NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN**

*Trà Vinh, ngày … tháng … năm 2025*

**Giáo viên hướng dẫn**

*(Ký và ghi rõ họ tên)*

**NHẬN XÉT CỦA THÀNH VIÊN HỘI ĐỒNG**

*Trà Vinh, ngày … tháng … năm 2024*

**Thành viên hội đồng**

*(Ký và ghi rõ họ tên)*

**LỜI CẢM ƠN**

Tôi xin gửi lời cảm ơn chân thành đến cô Nguyễn Ngọc Đan Thanh, đã không quản thời gian, quan tâm, nhắc nhở đôn đốc, động viên và hỗ trợ em trong suốt quá trình thực hiện đồ án. Nhờ sự quan tâm sát sao, những góp ý quý báu của cô mà tôi đã hoàn thành đồ án đúng tiến độ và đạt được chất lượng như mong đợi.

Xin bày tỏ lòng biết ơn sâu sắc đến toàn thể quý thầy cô bộ môn Công nghệ Thông tin. Nhờ sự quan tâm, tạo điều kiện và tạo không khí làm việc tích cực của bộ môn mà chúng tôi, những sinh viên năm cuối, đã có cơ hội được trải nghiệm thực tế, rèn luyện kỹ năng và khẳng định bản thân qua các đồ án mà chúng tôi đã lựa chọn.

Gửi đến những người bạn, người đồng đội bên cạnh một lời cảm ơn thân tình. Cảm ơn đã chia sẻ những bí quyết, kinh nghiệm, kiến thức vô cùng quý giá đã giúp tôi vượt qua các giai đoạn khó khăn để hoàn thành tốt đồ án này.

Tôi cũng gửi đến cô Đan Thanh, đến bạn bè và các bạn cùng nhóm lời xin lỗi chân thành, trong quá trình thực hiện đồ án có những lúc bối rối áp lực tôi đã có những sự chậm trễ, phiền hà gây ảnh hưởng đến mọi người.

Cuối cùng xin kính chúc các bạn có một kỳ báo cáo đồ án thành công như mong đợi, kính chúc quý thầy cô hoàn thành tốt nhiệm vụ. Kính mong nhận được sự góp ý của quý thầy cô và các bạn để cho đồ án này có thể được tiếp tục phát triển và hoàn thiện hơn trong tương lai.

**MỤC LỤC**

[CHƯƠNG 1. TỔNG QUAN NGHIÊN CỨU 1](#_Toc187278721)

[1.1. Giới thiệu đề tài 1](#_Toc187278722)

[1.2. Mục đích nghiên cứu 2](#_Toc187278723)

[1.3. Đối tượng nghiên cứu 3](#_Toc187278724)

[1.4. Phương pháp nghiên cứu 3](#_Toc187278725)

[CHƯƠNG 2. NGHIÊN CỨU LÝ THUYẾT 4](#_Toc187278726)

[2.1. Công tắc 4](#_Toc187278727)

[2.1.1 Module ESP-01 4](#_Toc187278728)

[2.1.2 Relay 6](#_Toc187278729)

[2.1.3 Module giảm áp 7](#_Toc187278730)

[2.1.4 Nguồn Hi-link 8](#_Toc187278731)

[2.1.5 Phần mềm lập trình 9](#_Toc187278732)

[2.1.6 Ngôn ngữ lập trình C/C++ với Arduino Framework 10](#_Toc187278733)

[2.2. Server 10](#_Toc187278734)

[2.2.1 NestJS 10](#_Toc187278735)

[2.2.2 Kiến trúc Microservices 11](#_Toc187278736)

[2.2.3 Giao thức gRPC 12](#_Toc187278737)

[2.3. Cơ sở dữ liệu 13](#_Toc187278738)

[2.4. Ứng dụng Android “Smartome” với React Native 13](#_Toc187278739)

[2.5. Hệ thống quản lý giao tiếp IOT với EMQX 15](#_Toc187278740)

[2.5.1 Các cổng kết nối 16](#_Toc187278741)

[2.5.2 Công cụ hỗ trợ tương tác với MQTT Broker 17](#_Toc187278742)

[CHƯƠNG 3. HIỆN THỰC HÓA NGHIÊN CỨU 19](#_Toc187278743)

[3.1. Mô tả bài toán 19](#_Toc187278744)

[3.2. Phân tích thiết kế hệ thống 22](#_Toc187278745)

[3.2.1 Đặc tả yêu cầu hệ thống 22](#_Toc187278746)

[3.2.2 Kiến trúc hệ thống 25](#_Toc187278747)

[3.2.3 Thiết kế dữ liệu 27](#_Toc187278748)

[3.2.4 Thiết kế công tắc 37](#_Toc187278749)

[3.2.5 Thiết kế server 43](#_Toc187278750)

[3.2.6 Thiết kế ứng dụng di động 51](#_Toc187278751)

[3.2.7 Thiết lập MQTT Broker 58](#_Toc187278752)

[CHƯƠNG 4. KẾT QUẢ NGHIÊN CỨU 60](#_Toc187278753)

[4.1. Công tắc 60](#_Toc187278754)

[4.1.1 Sản phẩm 60](#_Toc187278755)

[4.1.2 Hoạt động 60](#_Toc187278756)

[4.2. Server 61](#_Toc187278757)

[4.2.1 Ứng dụng thành công mô hình microservices 61](#_Toc187278758)

[4.2.2 Logic xử lý 62](#_Toc187278759)

[4.2.3 Các API quan trọng 63](#_Toc187278760)

[4.3. Ứng dụng di động 66](#_Toc187278761)

[4.3.1 Giao diện 66](#_Toc187278762)

[4.3.2 Logic xử lý 68](#_Toc187278763)

[4.4. MQTT Broker 69](#_Toc187278764)

[4.4.1 Triển khai 69](#_Toc187278765)

[4.4.2 Cấu hình và hoạt động 69](#_Toc187278766)

[CHƯƠNG 5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 71](#_Toc187278767)

[5.1. Kết luận 71](#_Toc187278768)

[5.2. Hướng phát triển 71](#_Toc187278769)

**DANH MỤC HÌNH ẢNH – BẢNG BIỂU**

[Hình 2.1 ESP-01 5](#_Toc187242581)

[Hình 2.2 Module CH340 Chuyển Đổi USB – TTL 6](#_Toc187242582)

[Hình 2.3 Sơ đồ cấu tạo của relay 7](#_Toc187242583)

[Hình 2.4 Module AMS1117-3.3VDC 8](#_Toc187242584)

[Hình 2.5 Nguồn Hi-link 5V DC 8](#_Toc187242585)

[Hình 2.6 PlatformIO 9](#_Toc187242586)

[Hình 2.7 Arduino framework 10](#_Toc187242587)

[Hình 2.8 NestJS framwork 10](#_Toc187242588)

[Hình 2.9 Kiến trúc microservices tiêu biểu 12](#_Toc187242589)

[Hình 2.10 React Native framework 13](#_Toc187242590)

[Hình 2.11 EMQX 15](#_Toc187242591)

[Hình 2.12 Giao diện phần mềm MQTTX 17](#_Toc187242592)

[Hình 3.1 Kiến trúc hệ thống tổng quát 25](#_Toc187242593)

[Hình 3.2 Sơ đồ dữ liệu EER 29](#_Toc187242594)

[Hình 3.3 Sơ đồ dữ liệu EER 32](#_Toc187242595)

[Hình 3.4 Sơ đồ dữ liệu EER 33](#_Toc187242596)

[Hình 3.5 Sơ đồ mạch module relay kích ở mức cao 37](#_Toc187242597)

[Hình 3.6 Sơ đồ mạch công tắc 38](#_Toc187242598)

[Hình 3.7 Sơ đồ hoạt động của công tắc 40](#_Toc187242599)

[Hình 3.8 Sơ đồ tuần tự xử lý tin nhắn của công tắc 42](#_Toc187242600)

[Hình 3.9 Kiến trúc server theo mô hình microservices 43](#_Toc187242601)

[Hình 3.10 Mô hình tuần tự đăng ký người dùng 45](#_Toc187242602)

[Hình 3.11 Sơ đồ tuần tự xử lý đăng nhập 47](#_Toc187242603)

[Hình 3.12 Cấu trúc code của một Service điển hình 48](#_Toc187242604)

[Hình 3.13 Sơ đồ hoạt động của ứng dụng Smartome 51](#_Toc187242605)

[Hình 3.14 Bản phát thảo màn hình đăng nhập và màn hình đăng ký 52](#_Toc187242606)

[Hình 3.15 Bản phác thảo các giao diện tại màn hình chính ứng dụng Smartome 54](#_Toc187242607)

[Hình 3.16 Bản phác thảo giao diện màn hình thêm công tắc 55](#_Toc187242608)

[Hình 3.17 Cấu trúc code ứng dụng di động Smartome 56](#_Toc187242609)

[Hình 3.18 Form tạo người dùng trên MQTT Broker 59](#_Toc187242610)

[Hình 4.1 Công tắc sau khi lắp ráp các module và hoàn thiện chức năng 60](#_Toc187242611)

[Hình 4.2 Các dịch vụ đang chạy theo mô hình microservices 61](#_Toc187242612)

[Hình 4.3 Kiểm tra API đăng nhập bằng Postman 63](#_Toc187242613)

[Hình 4.4 Kiểm tra API lấy access-token mới bằng Postman 65](#_Toc187242614)

[Hình 4.5 Màn hình đăng nhập và màn hình đăng ký 66](#_Toc187242615)

[Hình 4.6 Các giao diện màn hình chính ở 3 trạng thái 67](#_Toc187242616)

[Hình 4.7 Màn hình thêm công tắc 68](#_Toc187242617)

[Hình 4.8 MQTT Broker chạy thành công trên Docker container 69](#_Toc187242618)

[Hình 4.9 Giao diện trang đăng nhập trang quản lý MQTT Broker 69](#_Toc187242619)

[Hình 4.10 Danh sách người dùng xác thực với MQTT Broker 70](#_Toc187242620)

[Hình 4.11 Trang thông tin các kết nối và chủ đề của MQTT Broker 70](#_Toc187242621)

[Hình 4.12 Trang danh sách các kết nối của MQTT Broker 70](#_Toc187242622)

[Bảng 3.1 Danh sách các thực thể 30](#_Toc187016012)

[Bảng 3.2 Thực thể users 30](#_Toc187016013)

[Bảng 3.3 Thực thể login\_history 31](#_Toc187016014)

[Bảng 3.4 Thực thể password\_history 31](#_Toc187016015)

[Bảng 3.5 Danh sách thực thể 32](#_Toc187016016)

[Bảng 3.6 Thực thể devices 32](#_Toc187016017)

[Bảng 3.7 Thực thể device\_ap 33](#_Toc187016018)

[Bảng 3.8 Danh sách các thực thể 33](#_Toc187016019)

[Bảng 3.9 Thực thể Houses 35](#_Toc187016020)

[Bảng 3.10 Thực thể Areas 35](#_Toc187016021)

[Bảng 3.11 Thực thể Setting 36](#_Toc187016022)

[Bảng 3.12 Thực thể Own\_deivce 37](#_Toc187016023)

**DANH MỤC CÁC TỪ VIẾT TẮT**

|  |  |  |
| --- | --- | --- |
| **STT** | **Ký hiệu chữ viết tắt** | **Chữ viết đầy đủ** |
| 1 | MQTT | Message Queuing Telemetry Transport |
| 2 | EEPROM | Electrically Erasable Programmable Read-Only Memory |
| 3 | AP | Access Point |
| 4 | STA | Station |
| 5 | UART | Universal Asynchronous Receiver – Transmitter |
| 6 | IC | Integrated Circuit |
| 7 | RPC | Remote Procedure Call |
| 8 | Protobuf | **Protocol Buffers** |
| 9 | EMQX | Erlang MQTT Broker |
| 10 | PCB | Printed Circuit Board |

# TỔNG QUAN NGHIÊN CỨU

## Giới thiệu đề tài

Từ ngày 19/11/1997, khi dịch vụ Internet chính thức được triển khai tại Việt Nam, mạng lưới này đã trải qua một quá trình phát triển vượt bậc. Đến năm 2022, theo báo cáo của Hiệp hội Internet Việt Nam (VIA) và Trung tâm Internet Việt Nam (VNNIC), hạ tầng Internet đã phủ sóng rộng khắp cả nước với tỷ lệ 99,7% số thôn. Hệ thống cáp quang đã được triển khai đến hầu hết các đơn vị hành chính cấp xã và các cơ sở giáo dục, đạt tỷ lệ 91% tại các thôn bản. Bên cạnh đó, độ phủ sóng của mạng 3G/4G đã đạt 95% dân số, đưa Việt Nam vào nhóm các quốc gia có tỷ lệ sử dụng Internet cao. [1]

Sự phát triển nhanh chóng của hạ tầng Internet đã tạo điều kiện thuận lợi cho việc tiếp cận thông tin và các dịch vụ trực tuyến. Internet không chỉ đáp ứng các nhu cầu truyền thống như tra cứu thông tin, liên lạc từ xa và giải trí mà còn thúc đẩy sự ra đời và phát triển của các ứng dụng mới, điển hình là “nhà thông minh”. Ứng dụng này cho phép người dùng điều khiển, theo dõi và quản lý các thiết bị trong ngôi nhà từ xa, mang lại nhiều tiện ích và nâng cao chất lượng cuộc sống. Tiềm năng ứng dụng của “nhà thông minh” còn mở rộng ra các lĩnh vực khác như quản lý doanh nghiệp và cơ sở hạ tầng.

Thị trường thiết bị nhà thông minh tại Việt Nam hiện nay đang trong giai đoạn phát triển ban đầu, với quy mô còn nhỏ và sự đa dạng của các nhà cung cấp. Các sản phẩm hiện có thường gặp phải những hạn chế như: phụ thuộc vào nhà cung cấp, chi phí cao, tính năng thừa và khó khăn trong việc tích hợp với các hệ thống khác. Bên cạnh đó, việc sử dụng các giải pháp nhà thông minh của nước ngoài cũng tiềm ẩn nhiều rủi ro về bảo mật, đặc biệt khi áp dụng cho các thiết bị liên quan đến an ninh và an toàn.

Từ những thực trạng trên, đề tài “**Xây dựng hệ thống điều khiển công tắc từ xa cho thiết bị điện dân dụng**” được khởi động nhằm mục tiêu khai thác tối đa tiềm năng của Internet, tạo ra một giải pháp nhà thông minh an toàn cho cá nhân. Hệ thống này sẽ được thiết kế đặc biệt để đáp ứng nhu cầu sử dụng cụ thể của người dùng, giúp nâng cao chất lượng cuộc sống và tiết kiệm chi phí. Được xây dựng từ các công nghệ hiện đại dễ tiếp cận, dễ xây dựng và bảo trì như React Native cho app Android, Arduino framework, NestJS, MQTT và MQTT Broker.

## Mục đích nghiên cứu

Nghiên cứu này nhằm thiết kế và triển khai một hệ thống điều khiển công tắc điện dân dụng từ xa, phục vụ mục tiêu đa dạng. Thứ nhất, hệ thống được xây dựng như một nền tảng nghiên cứu, cung cấp một môi trường thực tế để khám phá và ứng dụng các công nghệ liên quan đến Internet of Things (IoT), điều khiển từ xa và giao thức truyền thông. Thứ hai, hệ thống hướng đến việc đáp ứng nhu cầu cá nhân hóa của người dùng, cho phép họ tự động hóa các thiết bị điện trong gia đình một cách linh hoạt và tiện lợi. Cuối cùng, nghiên cứu này mong muốn cung cấp một giải pháp thay thế cho các dịch vụ thương mại hiện có, nhằm giải quyết các vấn đề về bảo mật thông tin và tùy biến.

Cụ thể, nghiên cứu tập trung vào các nội dung sau:

Thiết kế mạch và lập trình: Xây dựng mạch điện tử dựa trên các module phổ biến, kết hợp với lập trình vi điều khiển để tạo ra các công tắc thông minh có khả năng kết nối Internet.

Giao thức MQTT: Nghiên cứu và ứng dụng giao thức MQTT và MQTT Broker để thiết lập kết nối giữa các công tắc thông minh, ứng dụng di động và máy chủ trung gian.

Phát triển ứng dụng Android: Thiết kế và phát triển một ứng dụng di động thân thiện với người dùng, cho phép điều khiển các công tắc một cách trực quan và dễ dàng.

Qua đó, nghiên cứu này đóng góp vào việc phát triển các giải pháp IoT an toàn, đáng tin cậy và tùy biến cao, đồng thời cung cấp một nguồn tài liệu tham khảo hữu ích cho cộng đồng nghiên cứu và các nhà phát triển.

## Đối tượng nghiên cứu

Người dùng: Tập trung vào việc phân tích và xác định các kịch bản sử dụng phổ biến của người dùng không chuyên đối với hệ thống nhà thông minh, từ đó xây dựng mô hình kiến trúc hệ thống đáp ứng các yêu cầu về hiệu năng, khả năng mở rộng và bảo mật.

Phần cứng: Nghiên cứu sẽ thiết kế mạch điện tối ưu về kích thước và số lượng linh kiện, lựa chọn các module điện tử có công suất phù hợp, độ tin cậy cao và chi phí hợp lý. Đặc biệt, vấn đề an toàn điện sẽ được đặt lên hàng đầu trong quá trình thiết kế và triển khai.

Phần mềm: Thiết kế giao diện người dùng trực quan, dễ sử dụng, đồng thời đảm bảo tính linh hoạt để thích ứng với các thiết bị và tính năng mới.

Máy chủ (Server): Kiến trúc hệ thống máy chủ sẽ được thiết kế để đảm bảo khả năng xử lý lượng lớn dữ liệu, thời gian đáp ứng nhanh, khả năng phục hồi cao và khả năng mở rộng linh hoạt.

## Phương pháp nghiên cứu

Với đặc điểm của dự án, sẽ có hai phương pháp chính được áp dụng:

Nghiên cứu lý thuyết: Tìm hiểu công nghệ được sử dụng phổ biến cho từng đối tượng có trong hệ thống. Tìm hiểu các công nghệ giao tiếp giữa các cặp đối tượng trong hệ thống với nhau. Tìm hiểu các công cụ hỗ trợ trong quá trình xây dựng hệ thống.

Thực nghiệm: Xây dựng các đối tượng trong hệ thống ở mức cơ sở, triển khai và chạy thử với nhiều kịch bản tình huống thực tế nhằm kiểm giao tiếp thực tế giữa các đối tượng với nhau.

# NGHIÊN CỨU LÝ THUYẾT

## Công tắc

### Module ESP-01

Module ESP-01 có kích thước nhỏ gọn nhất trong module ESP, nhưng vẫn giữ khả năng kết nối Wi-Fi và lập trình linh hoạt, module ESP-01 là một lựa chọn lý tưởng cho vai trò bộ não của hệ thống này vì số lượng chân I/O đủ để kết nối với các cảm biến, thiết bị điều khiển và module truyền thông khác. Cụ thể, module ESP-01 là sự lựa chọn tối ưu nhất khi nó có đến 8 chân trong đó có 2 chân IO, 1 chân TX, 1 chân RX. Module này là sự lựa chọn hoàn hảo đáp ứng đủ số chân tín hiệu cho việc điều khiển led và nhận tín hiệu nút nhấn ở 2 chân IO, chân TX sẽ dùng cho việc điều khiển thiết bị đóng mở dòng điện (Relay). Module Wi-Fi ESP8266 được tích hợp bên trong nên ESP-01 có hỗ trợ các chuẩn giao tiếp không dây (802.11 b/g/n), giúp thiết bị dễ dàng tham gia mạng cục bộ hoặc kết nối trực tiếp với các dịch vụ đám mây. Tốc độ truyền tải dữ liệu cao và khả năng hoạt động ổn định trong các môi trường nhiễu sóng khiến ESP-01 trở thành lựa chọn tối ưu cho các hệ thống IoT thời gian thực. [2]

**Thông số kỹ thuật:**

*Điện áp đầu vào:* Điện áp hoạt động từ 3.0V – 3.6V DC.

*Dòng tiêu thụ:* Khi hoạt động liên tục trung bình 71mA tối đa 300mA. Dòng điện tiêu thụ khi ở chế độ ngủ giao động từ ~0,02mA đến ~20mA.

*Chân giao tiếp quan trọng:* 3 chân GPIO (1 chân GPIO1 tích hợp vào chân TX, nên chân này dùng cho tín hiệu ra), 1 RST (reset), 1 GND, 1 VCC.

*Nhiệt độ hoạt động:* -20°C đến 86°C.

A computer chip with a chip attached to it

Description automatically generated

Hình 2.1 ESP-01

**EEPROM lưu trữ dữ liệu động:** Là một loại bộ nhớ cho phép lưu trữ dữ liệu một cách bền vững ngay cả khi thiết bị mất nguồn. Tuy nhiên, trong trường hợp của ESP-01, một phiên bản thu gọn của module ESP8266, bộ nhớ EEPROM không được tích hợp. Thay vào đó, ESP-01 tận dụng flash memory để thực hiện các chức năng lưu trữ dữ liệu bền vững có thể thay thế được cho EEPROM. Kỹ thuật trên được hỗ trợ bởi thư viện “EEPROM.h”.

**Hai chế độ hoạt động:**

***AP mode:* ESP-01 hoạt động như một điểm truy cập WiFi, cho phép các thiết bị hoặc cảm biến kết nối trực tiếp thậm chí là giữa các ESP với nhau. Trong chế độ này, ESP-01 tạo ra một mạng WiFi riêng (SSID), cung cấp IP (Thường là 192.168.4.1) qua DHCP.**

*STA mode:* ESP-01 kết nối vào một mạng WiFi sẵn có, đóng vai trò như một thiết bị khách (station). Trong chế độ này, ESP-01 nhận IP từ router hoặc hotspot và có thể giao tiếp với server, cloud, hoặc các thiết bị khác trên cùng mạng hoặc bất cứ đâu qua internet.

**Phương pháp nạp firmware:** Có nhiều giải pháp khác nhau để nạp firmware, bao gồm sử dụng Arduino Uno, ESP-32, và các thiết bị tương tự. Tuy nhiên, nhằm đơn giản hóa quy trình và đảm bảo tính nhanh chóng, giải pháp tối ưu là sử dụng **“Mạch nạp và giao tiếp USB UART ESP-01”**. Thiết bị này đóng vai trò làm cầu nối giao tiếp giữa module ESP và máy tính thông qua IC CH340. Người dùng chỉ cần kết nối mạch với máy tính qua cổng USB và thực hiện nạp code dễ dàng, tương tự như thao tác nạp code qua cáp cho Arduino Uno.

A close-up of a computer chip

Description automatically generated

Hình 2.2 Module CH340 Chuyển Đổi USB – TTL

### Relay

Relay được xem là một thành phần gần như không thể thiếu trong các hệ thống điều khiển liên quan với dòng điện dân dụng. Lựa chọn relay cho hệ thống là giải pháp đơn giản nhưng ưu điểm vượt trội như đáp ứng khả năng chịu tải cao dễ dàng xử lý dòng điện dân dụng với công suất lớn, thường dao động trong khoảng 10A hoặc hơn, phù hợp cho hầu hết các thiết bị điện gia dụng như bóng đèn, quạt, và máy bơm. Do nguyên lý hoạt động sử dụng cuộn dây và tiếp điểm, relay đảm bảo không có sự kết nối trực tiếp giữa mạch điều khiển và mạch tải do đó đảm bảo cách ly hoàn toàn giữa mạch điều khiển và mạch tải, tăng độ an toàn và độ bền cho thiết bị. Relay hoạt động như một công tắc điện từ, sử dụng tín hiệu điện áp thấp từ mạch điều khiển để kích hoạt hoặc ngắt mạch tải. [3]

A diagram of electrical components

Description automatically generated

Hình 2.3 Sơ đồ cấu tạo của relay

**Thông số kỹ thuật:**

*Dòng điện – hiệu điện thế tối đa ở các tiếp điểm: 10A-250VAC, 10A-30VAC,...*

*Điện áp hoạt động: 5V.*

Dòng điện tiêu thụ: 150mA. [4]

**Module relay kích ở mức cao:** Khi tích hợp relay với ESP-01, việc chỉ sử dụng relay đơn thuần là không đủ. Relay cần được thiết kế thành một module hoàn chỉnh, trong đó yếu tố quan trọng nhất là bổ sung một chân nhận tín hiệu điều khiển. Module này phải được tối ưu hóa để hệ thống có thể tự động đóng/ngắt dòng điện vào cuộn dây relay khi nhận tín hiệu điều khiển ở mức cao. Các linh kiện điện tử cần thiết phải có gồm: Transistor 2N2222 NPN, điện trở 1K, diot 1N4007.

### Module giảm áp

Module AMS1117-3.3V được thiết kế nhằm chuyển đổi điện áp đầu vào cao hơn (thường nằm trong khoảng 4.5V - 12V) xuống mức 3.3V ổn định, phù hợp cho nhiều vi điều khiển và module nhúng, trong đó có ESP-01. Thiết kế dựa trên bộ điều chỉnh điện áp tuyến tính, cung cấp đầu ra 3.3V cường độ dòng điện tối đa đạt 800mA với độ dao động rất thấp. Khả năng này đặc biệt quan trọng đối với các thiết bị nhúng như ESP-01, vốn nhạy cảm với các biến đổi điện áp nhỏ, ả [5]nh hưởng đến hiệu năng và độ ổn định trong quá trình giao tiếp không dây. Module AMS1117-3.3VDC có thiết kế nhỏ gọn, tích hợp sẵn các linh kiện phụ trợ như tụ điện để giảm thiểu nhiễu và ổn định nguồn. Điều này không chỉ giúp giảm tải công việc thiết kế mạch mà còn tiết kiệm không gian trên bảng mạch PCB, lý tưởng cho các hệ thống có không gian hạn chế. [4]

A close-up of a blue circuit board

Description automatically generated

Hình 2.4 Module AMS1117-3.3VDC

### Nguồn Hi-link

Thiết bị nguồn Hi-Link chuyển đổi nguồn điện AC sang DC và cung cấp điện áp một chiều 5V ổn định. Thiết kế nhỏ gọn, Hi-Link giúp đảm bảo sự an toàn khi được tích hợp các tính năng bảo vệ như chống quá tải, quá áp, và ngắn mạch, giúp thiết bị vận hành ổn định trước các điều kiện khắc nghiệt của nguồn điện xoay chiều đầu vào. Khả năng cung cấp điện áp DC chính xác và liên tục, Hi-Link giảm thiểu các hiện tượng nhiễu và dao động là nhân tố có hại đối các ứng dụng yêu cầu độ chính [6]xác cao như IoT và hệ thống nhúng. [5]

A black electronic device with white text

Description automatically generated

Hình 2.5 Nguồn Hi-link 5V DC

Việc sử dụng Hi-Link làm nguồn cung cấp giúp loại bỏ các bước thiết kế phức tạp liên quan đến bộ chuyển đổi AC/DC truyền thống. Không cần phải tự xây dựng các thành phần chuyển đổi điện như biến áp, mạch chỉnh lưu, và mạch lọc. Điều này không chỉ tiết kiệm thời gian và công sức trong giai đoạn thiết kế mà còn giảm thiểu rủi ro từ các lỗi liên quan đến mạch điện phức tạp.

### Phần mềm lập trình

PlatformIO là một hệ sinh thái phát triển Internet vạn vật (IoT) mã nguồn mở, được xây dựng trên nền tảng Python, hoạt động như một Môi trường Phát triển Tích hợp (IDE) đa nền tảng với khả năng gỡ lỗi hợp nhất, tương thích với các hệ điều hành Windows, macOS và Linux. Hệ thống này tích hợp trình quản lý thư viện hỗ trợ các nền tảng phổ biến như Arduino và MBED, đồng thời cung cấp các công cụ kiểm thử phần mềm và cập nhật firmware.

A logo of an alien

Description automatically generated

Hình 2.6 PlatformIO

Khả năng hỗ trợ đa dạng các nền tảng, framework và board mạch, bao gồm Arduino, ESP32 và ESP8266, cùng với việc cung cấp các ví dụ và thư viện phong phú, là một đặc điểm nổi bật của PlatformIO. Tính độc lập với nền tảng vận hành, chỉ yêu cầu cài đặt Python, cho phép PlatformIO hoạt động trên nhiều hệ thống.

PlatformIO tối ưu hóa quy trình phát triển phần mềm thông qua các tính năng hỗ trợ lập trình C/C++ như tự động hoàn thành mã (IntelliSense) và công cụ kiểm tra cú pháp và lỗi tiềm ẩn (Linter), giúp nâng cao hiệu suất và chất lượng mã nguồn so với Arduino IDE. Bên cạnh đó, giao diện người dùng có thể tùy chỉnh với các chủ đề sáng và tối, cùng với khả năng điều hướng và định dạng mã thông minh, góp phần cải thiện trải nghiệm người dùng. Các tính năng cốt lõi của PlatformIO bao gồm hệ thống xây dựng đa nền tảng, quản lý thư viện, giám sát cổng nối tiếp và nhiều hơn nữa.

### Ngôn ngữ lập trình C/C++ với Arduino Framework

Trong lĩnh vực phát triển phần mềm nhúng nói chung và vi điều khiển như ESP8266 và ESP32 nói riêng, **Arduino Framework khi được ra mắt vào năm XXXX, không lâu sau đó** đã nhanh chóng trở thành một lựa chọn hàng đầu nhờ tính trực quan và khả năng dễ tiếp cận của nó. Arduino Framework có bộ API và thư viện C/C++ mạnh mẽ, framework này được thiết kế để đơn giản hóa quá trình lập trình phần cứng, giúp lập trình viên tập trung hơn vào việc phát triển logic ứng dụng mà không cần am hiểu sâu về cấu trúc nội bộ của vi điều khiển.

A blue and black logo

Description automatically generated

Hình 2.7 Arduino framework

Điểm nổi bật của framework này không chỉ nằm ở khả năng trừu tượng hóa những phức tạp của phần cứng, mà còn ở sự phong phú của hệ sinh thái thư viện và cộng đồng hỗ trợ. Việc tích hợp sẵn các thư viện phần cứng và phần mềm mở rộng tạo điều kiện thuận lợi để các nhà phát triển nhanh chóng hiện thực hóa ý tưởng và xây dựng các ứng dụng IoT một cách hiệu quả. Từ đó, Arduino Framework trở thành nền tảng lý tưởng, đặc biệt với những người muốn tiếp cận và khai thác sức mạnh của các vi điều khiển hiện đại.

## Server

### NestJS



Hình 2.8 NestJS framwork

NestJS là một framework backend được xây dựng trên nền tảng Node.js và sử dụng TypeScript, một phiên bản mở rộng của JavaScript, giúp tăng độ chính xác và khả năng bảo trì mã nguồn. Khác với các framework khác trong hệ sinh thái Node.js như Express, NestJS tập trung vào việc thiết lập cấu trúc dự án nhất quán thông qua các mẫu thiết kế như d**ependency injection** và ***MVC***.

NestJS không chỉ là một framework linh hoạt mà còn có thể được coi như một **“opinionated framework”**, tức là định hướng lập trình viên theo các quy tắc và cấu trúc cụ thể để duy trì chất lượng và tính dễ đọc của dự án. Tích hợp chặt chẽ với hệ sinh thái Node.js, NestJS cung cấp nhiều công cụ hiệu quả để phát triển các ứng dụng backend hiện đại.

**Ưu điểm của NestJS:**

***Sử dụng Typescript:*** Việc sử dụng Typescript giúp NestJS trở nên mạnh mẽ trong việc phát hiện lỗi sớm và cải thiện khả năng đọc, tái sử dụng mã nguồn. Đây là yếu tố quan trọng trong việc đảm bảo chất lượng phần mềm.

***Hỗ trợ modular và microservice:*** NestJS khuyến khích chia dự án thành các module nhỏ, độc lập rõ ràng và dễ bảo trì, chẳng hạn như MVC và Dependency Injection, điều này phù hợp với kiến trúc mà microservice yêu cầu. Hơn nữa NestJS hỗ trợ rất tốt hầu hết các giao thức có thể thấy ở các microservices như các giao thức **gRPC**, **MQTT**, và **Kafka**.

***Hệ sinh thái phong phú và cộng đồng hỗ trợ tốt:*** Nhờ tận dụng sức mạnh của Node.js, NestJS cung cấp một hệ sinh thái phong phú với thư viện và công cụ hỗ trợ đa dạng. Cộng đồng người dùng đông đảo cũng giúp dễ dàng tìm kiếm tài liệu và giải pháp cho các vấn đề phát sinh trong quá trình phát triển.

### Kiến trúc Microservices

Microservice là một kiến trúc phát triển hệ thống backend trong đó ứng dụng được chia thành nhiều dịch vụ nhỏ, độc lập với nhau. Mỗi microservice được thiết kế để thực hiện một chức năng cụ thể và hoạt động như một thực thể riêng biệt. Một điểm nổi bật trong kiến trúc microservice là **sử dụng cơ sở dữ liệu độc lập**: mỗi dịch vụ quản lý dữ liệu riêng của mình. Các microservice có thể **chạy độc lập trên các máy chủ vật lý khác nhau hoặc trên các container**. Vì thế, các nhóm phát triển có thể làm việc song song trên các dịch vụ khác nhau với ngôn ngữ lập trình, framework, hoặc công nghệ riêng biệt, tạo ra một môi trường phát triển nhanh chóng, linh hoạt, và dễ bảo trì. Đây là một lựa chọn tối ưu cho các hệ thống lớn, phức tạp, yêu cầu khả năng mở rộng và khả dụng cao.

A diagram of a software application

Description automatically generated with medium confidence

Hình 2.9 Kiến trúc microservices tiêu biểu

### Giao thức gRPC

Giao thức gRPC được phân tách thành 2 phần để làm rõ ý nghĩa là “g” trong gRPC và “RPC”.

***RPC*** là một cơ chế giao tiếp trong đó một chương trình có thể gọi hàm hoặc thủ tục trên một máy khác, giống như cách nó gọi hàm cục bộ. RPC ẩn đi sự phức tạp của việc trao đổi dữ liệu qua mạng, giúp các nhà phát triển tập trung vào logic ứng dụng thay vì xử lý các chi tiết mạng.

***gRPC*** là một framework hiện đại được Google phát triển dựa trên RPC, sử dụng **HTTP/2** để truyền dữ liệu và **Protobuf** làm định dạng tuần tự hóa. Khả năng hỗ trợ đa ngôn ngữ, hiệu suất cao, truyền dữ liệu song công (bi-directional streaming), và quản lý hiệu quả kết nối, gRPC đã trở thành một lựa chọn phổ biến cho các hệ thống phân tán, đặc biệt trong kiến trúc microservice.

## Cơ sở dữ liệu

Cơ sở dữ liệu quan hệ như MySQL, PostgreSQL, SQL Server,…. Dữ liệu được tổ chức thành các bảng với các hàng và cột. Mối quan hệ giữa các bảng được thiết lập thông qua khóa. Ưu điểm của loại cơ sở dữ liệu này là tính toàn vẹn dữ liệu cao, hỗ trợ ACID (Atomicity, Consistency, Isolation, Durability) đảm bảo tính nhất quán của dữ liệu. Phù hợp với các ứng dụng yêu cầu tính chính xác và cấu trúc dữ liệu rõ ràng.

Cơ sở dữ liệu NoSQL điển hình như MongoDB, Cassandra, Redis,…. Linh hoạt hơn trong việc lưu trữ dữ liệu, không nhất thiết tuân theo cấu trúc bảng. Dữ liệu có thể được lưu trữ dưới dạng JSON, BSON (Binary JSON), key-value, graph... Ưu điểm của NoSQL là khả năng mở rộng tốt, xử lý dữ liệu lớn và tốc độ truy vấn nhanh. Phù hợp với các ứng dụng có dữ liệu biến động, khối lượng dữ liệu lớn và yêu cầu hiệu suất cao.

## Ứng dụng Android “Smartome” với React Native

React Native là một framework mã nguồn mở do Facebook phát triển, cho phép xây dựng các ứng dụng di động đa nền tảng bằng cách sử dụng JavaScript. Thay vì tạo ra các ứng dụng web được gói trong một container di động, React Native sử dụng một cách tiếp cận đặc biệt: chuyển đổi mã JavaScript sang thành phần native thực sự của Android và iOS. Điều này mang lại trải nghiệm người dùng mượt mà và hiệu suất tương đương với ứng dụng native.

A blue atom symbol with a black background

Description automatically generated

Hình 2.10 React Native framework

Cốt lõi của React Native nằm ở kiến trúc **Bridge**, cho phép giao tiếp giữa mã JavaScript và các thành phần native. Thông qua công nghệ này, lập trình viên có thể viết mã logic ứng dụng bằng JavaScript và vẫn truy cập được các API hoặc thành phần giao diện native mà không phải học thêm ngôn ngữ đặc thù như Swift hoặc Kotlin.

Khi kết hợp với Expo, một bộ công cụ mạnh mẽ và thân thiện, React Native trở thành lựa chọn lý tưởng để tăng tốc phát triển. Expo cung cấp các API tích hợp sẵn, môi trường giả lập nhanh, và khả năng dễ dàng triển khai ứng dụng mà không cần cài đặt hoặc cấu hình phức tạp.

**Hiệu suất gần tương đương với native:** Các thành phần được biên dịch thành mã native thay vì chạy trong một web view như các framework khác. Điều này giúp React Native đạt được hiệu suất gần tương đương với ứng dụng được phát triển trực tiếp bằng Swift hoặc Kotlin.

**Tích hợp với hệ sinh thái Javascript:** Là một phần mở rộng của React, React Native tận dụng sức mạnh của hệ sinh thái JavaScript và React, bao gồm các thư viện nổi tiếng, state management tools (Redux, MobX), và một cộng đồng phát triển rộng lớn.

**Hot reloading:** Tính năng **này** giúp nhà phát triển quan sát ngay lập tức những thay đổi trong mã nguồn mà không cần khởi động lại ứng dụng. Điều này giúp tăng tốc độ phát triển và cải thiện trải nghiệm lập trình.

**Khả năng mở rộng với thành phần native:** React Native cho phép lập trình viên viết hoặc tích hợp các module native bằng Swift, Objective-C (trên iOS) hoặc Java, Kotlin (trên Android). Điều này giúp ứng dụng duy trì hiệu năng tối đa khi làm việc với các tính năng đặc thù hoặc yêu cầu tài nguyên cao.

**Hệ sinh thái và cộng đồng hỗ trợ tốt:** Với hàng triệu lập trình viên trên toàn cầu, React Native sở hữu một hệ sinh thái đa dạng gồm hàng ngàn thư viện, công cụ và giải pháp sẵn có. Cộng đồng phát triển năng động giúp giải quyết nhanh chóng các vấn đề trong quá trình triển khai, cùng lượng tài liệu học tập phong phú dành cho cả người mới bắt đầu và chuyên gia.

**Ứng dụng thời gian thực:** React Native phù hợp với các ứng dụng yêu cầu xử lý thời gian thực như tin nhắn, theo dõi vị trí GPS, hoặc điều khiển các thiết bị IoT. Khả năng tích hợp WebSocket và API native giúp nó xử lý dữ liệu nhanh chóng và hiệu quả.

## Hệ thống quản lý giao tiếp IOT với EMQX

EMQX (Erlang MQTT Broker) là một nền tảng MQTT broker mã nguồn mở, được xây dựng bằng ngôn ngữ lập trình **Erlang/OTP**, vốn nổi tiếng với khả năng xử lý đồng thời và độ ổn định cao. EMQX được thiết kế để cung cấp giải pháp truyền thông mạnh mẽ, phù hợp với các hệ thống IoT lớn, kết nối hàng triệu thiết bị và xử lý hàng tỷ tin nhắn mỗi ngày.

A green hexagon with text in it

Description automatically generated

Hình 2.11 EMQX

EMQX hỗ trợ các phiên bản MQTT tiêu chuẩn (3.x và 5.0), giúp tối đa hóa khả năng tương thích với các ứng dụng và thiết bị IoT. Nền tảng này không chỉ giới hạn ở MQTT mà còn tích hợp thêm các giao thức truyền thông khác như **CoAP, HTTP, WebSocket**,và các công nghệ như **Kafka, PostgreSQL**, đảm bảo tính linh hoạt trong các hệ thống đa dạng.

**Hỗ trợ đa giao thức:** Ngoài MQTT, EMQX hỗ trợ giao tiếp qua WebSocket, CoAP và HTTP, giúp tích hợp với nhiều thiết bị và dịch vụ IoT khác nhau. Điều này tạo ra sự linh hoạt trong việc xây dựng các hệ thống giao tiếp đa dạng và phức tạp. [6]

**Bảo mật cao:** EMQX cung cấp cơ chế xác thực và ủy quyền mạnh mẽ thông qua các giao thức như **TLS/SSL, OAuth 2.0**, LDAP, và nhiều tích hợp khác. Điều này giúp bảo vệ thông tin truyền tải giữa các thiết bị, đảm bảo an toàn trước các mối đe dọa an ninh mạng.

**Hỗ trợ MQTT 5.0:** Là một trong những broker đầu tiên hỗ trợ chuẩn **MQTT 5.0**, EMQX mang lại nhiều tính năng nâng cao như: chia sẻ subscription, cải thiện hiệu suất routing message, và bổ sung thuộc tính metadata cho tin nhắn, giúp tăng khả năng kiểm soát và tối ưu luồng dữ liệu.

### Các cổng kết nối

Thông thường giao thức MQTT dựa vào TCP làm giao thức tầng giao vận, nghĩa là mặc định kết nối giữa broker và client là kết nối không bảo mật. Để tăng tính bảo mật và có khả năng mã hóa các kênh giao tiếp, hầu hết các MQTT broker hiện nay như Mosquitto, HiveMQ đều cho phép sử dụng TLS và sử dụng một số cổng sau cho từng đặc tính kết nối khác nhau.

*Cổng 80:* Cổng mặc định của giao thức HTTP dùng để truy cập trang quản trị MQTT qua nền tảng Web.

*Cổng chuẩn 1883 TCP:* Cổng này được sử dụng cho các kết nối MQTT không được mã hóa. Đây là cổng MQTT được sử dụng phổ biến nhất và là cổng mặc định cho hầu hết các nhà môi giới MQTT. Sử dụng cổng này thường là các thiết bị phần cứng nhỏ có tài nguyên hạn chế, thiết bị kết nối có thể xuất bản tin nhắn, đăng ký chủ đề và nhận tin nhắn đã xuất bản.

*Cổng 8883 SSL:* Được chuẩn hóa để sử dụng cho kết nối MQTT có bảo mật. Tên được đặt theo tổ chức IANA là “secure-mqtt” và port 8883 được dành riêng cho giao thức MQTT sử dụng TLS. Nhược điểm khi sử dụng TLS cho giao thức MQTT đó là tăng tính bảo mật đồng nghĩa với tiêu tốn năng lượng cho CPU nhiều hơn và quá trình giao tiếp sẽ không mượt mà so với cổng không sử dụng TSL.

*Cổng 8083 WS:* Cổng này cho phép các client kết nối và giao tiếp với broker thông qua giao thức WebSocket. Đây là giao thức lý tưởng để tích hợp MQTT vào ứng dụng web, đặc biệt là khi cần giao tiếp thời gian thực giữa thiết bị IoT và giao diện người dùng dựa trên trình duyệt.

*Cổng 8084 WSS:* Tương tự như cổng 8083 nhưng được sử dụng thêm TSL/SSL tăng tính bảo mật khi truyền tải dữ liệu, điển hình là nghe lén và tấn công trung gian.

### Công cụ hỗ trợ tương tác với MQTT Broker

EMQX, một trong những nhà cung cấp hàng đầu về MQTT Broker và cũng chính là nhà phát hành MQTTX– một phần mềm hỗ trợ tuyệt vời cho giao thức MQTT, phần mềm này cho phép ta kết nối với MQTT Broker để kiểm tra, gửi và nhận tin nhắn ở chủ đề mong muốn, trực quan hoá bằng giao diện được thiết kế tối ưu cho giao tiếp MQTT.

Giao diện của MQTTX được tối ưu hóa để phục vụ nhu cầu tương tác nhanh chóng và hiệu quả trong môi trường giao tiếp MQTT. Người dùng không chỉ quản lý các phiên kết nối mà còn có thể giám sát và trực quan hóa luồng dữ liệu theo thời gian thực, từ đó đơn giản hóa việc phát triển, thử nghiệm, và phân tích các hệ thống IoT hoặc ứng dụng liên quan.

Với sự kết hợp giữa **MQTTX** và **EMQX Broker**, người dùng không chỉ có một hệ thống mạnh mẽ, đáng tin cậy để vận hành giao thức MQTT mà còn được cung cấp các công cụ hiện đại để phát triển, gỡ lỗi, và giám sát các ứng dụng IoT một cách dễ dàng và hiệu quả.

A screenshot of a computer

Description automatically generated

Hình 2.12 Giao diện phần mềm MQTTX

# HIỆN THỰC HÓA NGHIÊN CỨU

## Mô tả bài toán

Xây dựng các hệ thống điều khiển công tắc từ xa yêu cầu tiên quyết là khả năng kết nối và quản lý thông minh. Bên cạnh đó vấn đề cần được quan tâm và liên quan trực tiếp tới trải nghiệm người dùng là tính tối ưu, dễ sử dụng và linh hoạt để phục vụ đa dạng nhu cầu cho hầu hết các phân khúc người dùng. Đề tài “**Xây dựng hệ thống điều khiển công tắc từ xa cho thiết bị điện dân dụng”** tái hiện lại những gì đã có và tập trung giải quyết những thách thức cốt lõi trong việc thiết kế và triển khai một hệ thống công tắc điện thông minh đáp ứng các yêu cầu đó.

Hệ thống này yêu cầu khả tương tác giữa **công tắc và ứng dụng theo thời gian thực**. Ứng dụng trên Android là nơi duy nhất và đầy đủ nhất xác thực và cung cấp cho người dùng có toàn quyền quản lý các công tắc thuộc sở hữu của mình. Người dùng không chỉ điều khiển bật/tắt các công tắc mà còn có thể dễ dàng cấu hình các công tắc mới mà không cần đến sự hỗ trợ từ nhà sản xuất. Có nghĩa là khi muốn sử dụng công tắc cho ngôi nhà của mình thì người dùng có thể mua thiết bị về tự mình lắp đặt và kết nối công tắc với mạng Wi-Fi có sẵn (tại nơi họ bố trí công tắc đấy), thiết lập các cài đặt cần thiết thông qua một giao diện ứng dụng thân thiện được cung cấp sẵn mà không yêu cầu thêm phần mềm hay công cụ đặc biệt nào khác. Điều này nhằm tạo ra một **quy trình đơn giản và tự động hóa cao cho những người tham gia hệ thống này**, hướng đến việc tối ưu hóa trải nghiệm người dùng ngay từ lần sử dụng đầu tiên.

Bài toán đặt ra yêu cầu tích hợp một cách tối ưu giữa **phần cứng tối giản, giao thức truyền thông hiệu quả và giao diện người dùng thân thiện**, đồng thời đảm bảo khả năng xử lý dữ liệu ổn định và mở rộng cho tương lai. Hệ thống không chỉ giải quyết các yêu cầu trước mắt mà còn xây dựng một nền tảng vững chắc để hỗ trợ các cải tiến và mở rộng chức năng theo nhu cầu người dùng trong những giai đoạn tiếp theo.

**Về mặt phần cứng:**

Công tắc phải được thiết kế sao cho khi hoạt động **không phụ thuộc vào nguồn năng lượng riêng biệt như pin hoặc nguồn rời**. Thay vào đó, công tắc cần tận dụng hiệu quả dòng điện từ nguồn điện hiện có của thiết bị gia dụng mà nó điều khiển. Thiết kế giúp người dùng nhàn rỗi khi không phải bận tâm về việc mất nguồn đột ngột và thời gian kiểm tra nguồn điện hoạt động của công tắc định kỳ,… tất cả điều đó đảm bảo một giải pháp **gọn nhẹ và tiện lợi**, sẵn sàng hoạt động liên tục trong mọi điều kiện mà không yêu cầu bảo dưỡng thường xuyên. Đảm bảo tiêu chí “Chỉ lắp và dùng”.

Ngoài tính năng điều khiển từ xa, nút nhấn vật lý trên công tắc cũng phải được thiết kế sao cho có thể hoạt động như một nút nhấn cơ học truyền thống mặc dù tín hiệu từ nút nhấn vẫn được thu nhận và xử lý bởi vi xử lý bên trong công tắc. Điều này có nghĩa rằng, trong trường hợp xảy ra sự cố kết nối mạng hoặc server gặp trục trặc, người dùng vẫn có thể điều khiển công tắc của mình thủ công như một công tắc cơ học để ngắt/cung cấp điện cho thiết bị gia dụng của họ. Tính năng này đảm bảo sự ổn định và tính tin cậy, được thể hiện đặc biệt quan trọng trong các tình huống khẩn cấp.

Để gia tăng tính linh hoạt và tính tái sử dụng, công tắc cũng cung cấp tính năng **khôi phục cài đặt gốc (reset)**. Người dùng có thể sử dụng ứng dụng Android để thực hiện quá trình reset hoặc sử dụng nút nhấn vật lý để trên công tắc để reset chính nó (tính năng này có thể yêu cầu xác thực từ ứng dụng trên Android), trong đó giao diện ứng dụng sẽ hiển thị rõ ràng thông tin xác nhận và hướng dẫn cụ thể. Sau khi reset thành công, công tắc sẽ trở về trạng thái mặc định như lúc ban đầu, sẵn sàng để được cài đặt lại tại một vị trí hoặc môi trường mới. Tính năng này giúp tối đa hóa vòng đời của thiết bị và hỗ trợ người dùng linh hoạt trong việc thay đổi hoặc mở rộng hệ thống mà không cần đầu tư thêm các công tắc mới.

**Về mặt ứng dụng trên Android:**

Ứng dụng Android là đối tượng chủ đạo vì nó là nơi người dùng tiếp xúc và thực hiện toàn bộ các hoạt động điều khiển, cấu hình thiết bị của họ cho nên cần thiết kế giao diện người dùng (UI) hiện đại, trực quan và tránh làm khó người dùng. Tính năng bật/tắt thiết bị phải được thiết kế sao cho rõ ràng và dễ tiếp cận nhất. Cho phép tùy chỉnh và cá nhân hóa thông tin của từng công tắc để phù hợp với môi trường sử dụng. cho phép phân nhóm các công tắc theo không gian (phòng khách, phòng bếp...) hoặc chức năng (chiếu sáng, làm mát...) để giúp người dùng quản lý dễ dàng trong trường hợp có nhiều công tắc được kết nối (chẳng hạn như việc sử dụng công tắc tại công ty, công xưởng,…). Giao diện cần hỗ trợ khả năng phản hồi nhanh chóng, hiển thị thông tin rõ ràng và trực tiếp giúp người dùng thao tác mà không gặp khó khăn. Những nguyên tắc **UI/UX** như sử dụng biểu tượng (icon) dễ hiểu, màu sắc gợi nhắc trạng thái thiết bị (chẳng hạn như xanh khi bật và đỏ khi tắt), và tối ưu hóa số bước thao tác cho các chức năng thường dùng được áp dụng triệt để nhằm mang đến trải nghiệm thân thiện, hiện đại.

**Về mặt hệ thống server:**

Tất cả thông tin liên quan đến hệ thống đều được lưu trữ tại server, bao gồm: dữ liệu người dùng, dữ liệu thiết bị, cấu hình mạng, trạng thái hiện tại (bật/tắt), và các thông số kỹ thuật khác. Lịch sử hoạt động, Các cài đặt riêng của công tắc,… Việc lưu trữ theo hình thức này giúp bảo đảm an toàn, bảo mật và đảm bảo tính đồng bộ cao giữa các thiết bị. Người dùng có thể truy cập dữ liệu hoặc điều khiển thiết bị từ bất kỳ ứng dụng Android nào được liên kết, ngay cả khi họ sử dụng một thiết bị mới.

Khả năng ghi nhận và lưu trữ toàn bộ lịch sử hoạt động của hệ thống. Server luôn lắng nghe và ghi nhận các lệnh điều khiển hoặc thay đổi trạng thái từ thiết bị, ngay cả khi phần mềm ở thiết bị của người dùng không thể gửi hoặc nhận dữ liệu do mất kết nối mạng. Điều này đảm bảo cho l**ịch sử được lưu đầy đủ:** Mọi thay đổi của công tắc đều được ghi nhận chính xác tại server, đảm bảo tính toàn vẹn của dữ liệu lịch sử. Khi ứng dụng Android khôi phục kết nối, người dùng vẫn có thể xem đầy đủ các hoạt động diễn ra trong thời gian họ offline.

## Phân tích thiết kế hệ thống

### Đặc tả yêu cầu hệ thống

#### Yêu cầu chức năng

**Đối với người dùng:**

***Đăng ký và xác thực tài khoản:*** Cung cấp khả năng đăng ký tài khoản người dùng, trong đó địa chỉ email sẽ là định danh chính. Quá trình đăng ký yêu cầu người dùng xác thực email nhằm đảm bảo tính hợp lệ và bảo mật.

***Đăng nhập và quản lý quyền truy cập:*** Người dùng có thể đăng nhập vào hệ thống thông qua tên đăng nhập và mật khẩu. Hệ thống sử dụng **JWT (JSON Web Token)** để xác thực người dùng, cho phép truy cập an toàn và kiểm soát quyền hạn đối với các tính năng tiếp theo.

***Quản lý nhiều ngôi nhà:*** Hệ thống cho phép người dùng quản lý độc lập nhiều ngôi nhà nếu có từ hai ngôi nhà trở lên. Điều này giúp tối ưu hóa việc kiểm soát và phân tách quản lý một cách rõ ràng, trực quan.

***Phân khu vực quản lý trong ngôi nhà:*** Người dùng có thể chia ngôi nhà thành các khu vực nhỏ hơn (phòng khách, phòng ngủ, nhà bếp, v.v.) để quản lý thiết bị theo từng khu vực, thuận tiện hơn trong việc theo dõi và điều khiển.

***Thêm mới thiết bị:*** Hệ thống hỗ trợ người dùng thêm và kết nối thiết bị mới bằng cách cấu hình thông số mạng Wi-Fi trực tiếp qua ứng dụng, đảm bảo quá trình thiết lập nhanh chóng và thuận tiện.

***Điều khiển thiết bị:*** Người dùng được phép toàn quyền điều khiển các thiết bị đã kết nối và sở hữu, bao gồm bật/tắt công tắc, thiết lập chế độ hoạt động, và kiểm tra trạng thái thiết bị theo thời gian thực.

***Khôi phục cài đặt thiết bị:*** Cung cấp khả năng đặt lại thiết bị công tắc về trạng thái gốc ban đầu (trạng thái xuất xưởng), cho phép người dùng tự do sử dụng lại hoặc tái kết nối với hệ thống khác.

**Đối với công tắc:**

*Cấu hình qua ứng dụng:* Công tắc có khả năng tự phát mạng Wi-Fi cục bộ (Access Point) để ứng dụng kết nối, gửi các thông tin cấu hình cần thiết như SSID và mật khẩu mạng.

*Kết nối Internet và MQTT Broker:* Công tắc kết nối vào mạng Wi-Fi và truy cập internet để tương tác với hệ thống MQTT Broker. Hỗ trợ các giao thức WS, WSS, và TCP nhằm đảm bảo kết nối an toàn và hiệu quả.

*Tiếp nhận và xử lý tín hiệu điều khiển từ xa:* Công tắc phải có khả năng xử lý chính xác các lệnh điều khiển từ ứng dụng qua MQTT, đồng thời phản hồi trạng thái hoạt động để đồng bộ với server và ứng dụng.

*Hoạt động độc lập:* Trong trường hợp mất kết nối mạng, công tắc vẫn hoạt động như một công tắc vật lý thông thường, đảm bảo không làm gián đoạn nhu cầu sử dụng của người dùng.

*Thông báo trạng thái:* Công tắc sử dụng đèn LED tích hợp để hiển thị trạng thái hiện tại (bật/tắt, kết nối hoặc lỗi). Đồng thời, trạng thái cũng được gửi đến server để cập nhật lên ứng dụng cho người dùng.

**Đối với server:**

***API quản lý:*** Server cung cấp các API RESTful với đầy đủ các chức năng: lưu trữ và quản lý thông tin người dùng, xử lý đăng ký, xác thực tài khoản, lưu trữ thông tin và trạng thái thiết bị, cũng như quản lý mối quan hệ giữa người dùng và công tắc.

***Giám sát hoạt động người dùng và công tắc:*** Server liên tục lắng nghe và ghi nhận tín hiệu từ công tắc cũng như yêu cầu từ người dùng. Tất cả các hoạt động như thay đổi trạng thái công tắc, sự bất thường trong hoạt động đều được ghi lại để phân tích và theo dõi.

#### Yêu cầu phi chức năng

**Hiệu năng:**

Hệ thống phải đảm bảo độ trễ điều khiển dưới **500ms** từ lúc gửi yêu cầu từ ứng dụng đến khi công tắc phản hồi trạng thái.

MQTT broker phải hỗ trợ xử lý tối thiểu **1000 kết nối đồng thời** mà không gây hiện tượng gián đoạn.

Server NestJS phải có khả năng xử lý tối thiểu **100 yêu cầu/giây**, bao gồm cả các tác vụ xử lý logic và truy vấn cơ sở dữ liệu.

**Tính ổn định và tin cậy:**

Hệ thống phải hoạt động liên tục 24/7, với thời gian ngừng hoạt động tối đa không quá **0.5%/năm**.

Trong trường hợp mất kết nối internet, ESP-01 cần lưu trạng thái hiện tại của công tắc và tự đồng bộ trạng thái khi kết nối được khôi phục.

**Bảo mật:**

Tất cả các giao tiếp giữa app, server, và ESP-01 phải được mã hóa bằng **TLS** để bảo vệ dữ liệu trước các cuộc tấn công trung gian (man-in-the-middle).

Chỉ những thiết bị ESP-01 được đăng ký hợp lệ với hệ thống mới có thể giao tiếp với MQTT broker, hạn chế truy cập không mong muốn.

**Khả năng tương thích:**

Ứng dụng React Native phải hỗ trợ trên ít nhất hai hệ điều hành phổ biến: **Android 8.0** trở lên.

ESP-01 phải tương thích với các module relay điện áp khác nhau (5V, 12V) và có thể kết nối hiệu quả với mạng Wi-Fi tiêu chuẩn IEEE **802.11 b/g/n**.

**Khả năng bảo trì:**

NestJS server phải hỗ trợ **Hot Reload** để phát triển và cập nhật hệ thống nhanh chóng mà không cần tắt server.

Log hoạt động phải được lưu lại trên server với độ chi tiết cao, bao gồm các sự kiện quan trọng như thay đổi trạng thái công tắc, lỗi kết nối và đăng nhập người dùng.

**Dễ sử dụng:**

Giao diện ứng dụng React Native phải thân thiện với người dùng, tối thiểu hóa số lần thao tác để điều khiển công tắc từ xa.

Cung cấp các hướng dẫn dễ hiểu để người dùng thêm thiết bị mới hoặc xử lý sự cố phổ biến.

**Tiết kiệm năng lượng:**

ESP-01 phải tiêu thụ điện năng ở mức tối thiểu trong khi hoạt động, không vượt quá **0.3W** khi ở chế độ standby.

Ứng dụng React Native nên hạn chế tối đa việc tiêu hao pin thiết bị di động, đặc biệt khi chạy nền.

**Khả năng chịu lỗi:**

Hệ thống phải tự động khởi động lại dịch vụ cần thiết nếu xảy ra lỗi (server NestJS, MQTT broker).

MQTT broker và server phải có cơ chế retry khi gặp lỗi kết nối. ESP-01 phải thực hiện các thao tác tương tự khi không thể gửi hoặc nhận tín hiệu.

### Kiến trúc hệ thống

A diagram of a computer

Description automatically generated

Hình 3.1 Kiến trúc hệ thống tổng quát

Tâm điểm của hệ thống là **MQTT Broker**, đóng vai trò trung gian giữa các thiết bị công tắc thông minh và các ứng dụng di động của người dùng (được xem như và người vận chuyển tin). Hệ thống tận dụng giao thức MQTT và các phương thức giao tiếp như TCP (gọn nhẹ, ít tốn tài nguyên) và WebSocket (hoạt động theo thời gian thực) để đảm bảo tốc độ và tính ổn định của việc điều khiển.

Các công tắc thông minh được kết nối với internet thông qua Wi-Fi. Khi được cài đặt, mỗi công tắc sẽ kết nối vào MQTT Broker, sau đó đăng ký vào một chủ đề (topic) cụ thể và sẵn sàng nhận lệnh từ ứng dụng di động hoặc gửi trạng thái hiện tại của công tắc. Ví dụ, khi công tắc chuyển trạng thái (bật/tắt), thông tin này được gửi (**publish**) tới MQTT Broker, từ đó được truyền tải tới mọi thiết bị (ứng dụng) đang đăng ký (subscribe) vào cùng một topic. Các công tắc sử dụng giao thức TCP dựa trên Websocket để duy trì kết nối ổn định với Broker, đảm bảo mọi thay đổi được cập nhật ngay lập tức.

Trên ứng dụng di động, giao thức WebSocket được sử dụng để liên lạc với MQTT Broker, cho phép người dùng nhận thông tin theo thời gian thực mà không cần làm mới ứng dụng. Khi người dùng gửi lệnh điều khiển qua app, lệnh này được publish đến topic tương ứng trên MQTT Broker. Công tắc sau đó nhận lệnh từ topic này, thực thi tác vụ (bật/tắt thiết bị) và gửi phản hồi để cập nhật trạng thái. Đồng thời ứng dụng di động cũng kết nối đến server qua giao thức HTTP để lấy các thông tin liên quan khác như thông tin người dùng, thông tin các thiết bị, thông tin nhà đang quản lý,…

Một điểm nổi bật khác của hệ thống là khả năng lưu trữ và giám sát. MQTT Broker có thể truyền dữ liệu về một server hỗ trợ HTTP, nơi thông tin được ghi lại và sử dụng cho các mục đích như phân tích dữ liệu hoặc tích hợp với các hệ thống quản lý lớn hơn. Server này đăng ký vào tất cả các topic trên MQTT Broker (ký hiệu wildcard \*), do đó, có thể theo dõi toàn bộ các thiết bị trong hệ thống.

#### Kiến trúc server

Theo kiến trúc tổng quát của hệ thống yêu cầu server phải có hai giao thức kết nối là HTTP và MQTT do đó server sẽ phải xử lý nhiều loại dữ liệu có đặc tính khác biệt lớn nên mô hình microservice sẽ được ứng dụng thay thế mô hình MVC thông thường. Server sẽ được chia thành ba service chính và một gateway phục vụ các yêu cầu HTTP (riêng service phục vụ MQTT thì không cần gateway).

**Auth-service:**Chuyên xử lý các yêu cầu về xác thực người dùng và đăng ký quản lý thông tin người dùng.

**Device-service:** Chuyên xử lý các yêu cầu về thông tin các công tắc đang tồn tại trong hệ thống.

**House-service:** Chuyên xử lý các yêu cầu về thêm – sửa – xoá các đối tượng mà người dùng sẽ quản lý như “nhà”, “khu vực”, “công tắc”. Bên cạnh đó là lưu trữ các thiết lập của người dùng cho các đối tượng trên.

**Sys-operation:** Chuyên tiếp nhận kết nối đến MQTT Broker lắng nghe tất cả các sự kiện nhằm phân loại – xử lý cho từng xử kiện được tiếp nhận. Chủ yếu là lắng nghe tương tác giữa người dùng và thiết bị của họ, ghi lại lịch sử hỗ trợ người dùng lúc họ không có internet.

**Gateway:** Cổng điều phối và tiếp nhận các yêu cầu HTTP để chuyển đến các service mục tiêu trong hệ thống kể cả “Sys-operation”.

### Thiết kế dữ liệu

Với đặc điểm của server được xây dựng theo kiến trúc microservices, dữ liệu sẽ không còn tập trung trong một hệ quản trị cơ sở dữ liệu đơn lẻ mà thay vào đó được phân tán ra nhiều cơ sở dữ liệu và các loại cơ sở dữ liệu khác nhau. Trong dự án này, cấu trúc dữ liệu được chia thành ba cơ sở dữ liệu chính: hai cơ sở dữ liệu MySQL và một cơ sở dữ liệu MongoDB.

Cơ sở dữ liệu MySQL có khả năng tổ chức dữ liệu có cấu trúc, đảm bảo mối quan hệ chặt chẽ giữa các bảng, hỗ trợ các truy vấn phức tạp và tính toàn vẹn dữ liệu. Lý tưởng cho việc lưu trữ các thông tin mang tính chính quy, liên quan đến hoạt động quản lý cốt lõi, bao gồm: Thông tin xác thực người dùng, để đảm bảo dữ liệu liên quan đến danh tính và quyền truy cập luôn được bảo mật và có tổ chức rõ ràng. Thông tin quản lý nhà và khu vực, nơi mỗi nhà hoặc khu vực được liên kết với một nhóm thiết bị cụ thể, cho phép kiểm soát logic rõ ràng. Dữ liệu về thiết bị được quản lý, cung cấp chi tiết về trạng thái, loại thiết bị, và các thông số kỹ thuật.

Ngược lại, MongoDB, với thiết kế không ràng buộc chặt chẽ giữa các bảng và khả năng xử lý linh hoạt dữ liệu không có cấu trúc, lý tưởng cho các tình huống yêu cầu lưu trữ khối lượng lớn dữ liệu và tốc độ truy xuất cao. Trong dự án này, MongoDB được sử dụng để lưu trữ lịch sử hoạt động của thiết bị. Đây là loại dữ liệu thường xuyên được đọc và ghi với tần suất cao, ví dụ: các sự kiện bật/tắt thiết bị, thay đổi trạng thái, và thông tin thời gian thực. Việc sử dụng MongoDB giúp giảm tải cho hệ thống và tăng tốc độ xử lý các giao dịch lớn nhờ cấu trúc tài liệu (document-oriented), đồng thời hỗ trợ truy vấn linh hoạt trên dữ liệu đa dạng.

Sự kết hợp giữa MySQL và MongoDB tạo nên một hệ thống cơ sở dữ liệu lai (hybrid database), khai thác tối đa khả năng của từng loại cơ sở dữ liệu để phục vụ nhu cầu khác nhau trong cùng một hệ thống. Các thành phần dữ liệu có cấu trúc và yêu cầu tính toàn vẹn cao sẽ được xử lý tập trung tại MySQL, trong khi MongoDB sẽ đảm nhận những thành phần dữ liệu linh hoạt, mang tính chất phi cấu trúc, cần tốc độ truy cập cao hơn. Nhờ đó, hệ thống sẽ đáp ứng được tính chính xác trong lưu trữ mà, duy trì được hiệu năng trong xử lý các tác vụ ở quy mô lớn và phức tạp. Điều này còn dự phòng cho khả năng mở rộng khi có thể thêm các loại thiết bị khác thay vì chỉ mỗi “công tắc” trong tương lai.

#### Danh sách các thực thể và mối kết hợp ở Auth-service

A screenshot of a computer

Description automatically generated

Hình 3.2 Sơ đồ dữ liệu EER

Bảng 3.1 Danh sách các thực thể

|  |  |  |
| --- | --- | --- |
| *STT* | *Tên thực thể/mối kết hợp* | *Diễn giải* |
| 1 | Users | Thông tin người dùng |
| 2 | Password\_history | Thông tin mật khẩu và lịch sử thay đổi |
| 3 | Login\_history | Thông tin lịch sử đăng nhập và quản lý đăng nhập |
| 4 | User\_register | Thông tin người dùng đăng ký và đang chờ xác thực |
| 5 | Otp | Mã xác thực thông tin đăng ký OTP |

**Chi tiết các thực thể và mối kết hợp quan trọng**

Tên thực thể: users

Mô tả: Lưu trữ thông tin của người dùng

Chi tiết thực thể:

Bảng 3.2 Thực thể users

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *STT* | *Tên thuộc tính* | *Diễn giải* | *Kiểu dữ liệu* | *Ràng buộc toàn vẹn* |
| 1 | id | Mã người dùng | Varchar(6) | Khoá chính |
| 2 | lastname | Tên người dùng | Varchar(10) |  |
| 3 | firstname | Họ và tên lót người dùng | Varchar(20) |  |
| 4 | username | Tên đăng nhập | Varchar(25) |  |
| 5 | email | Địa chỉ email | Varchar(254) |  |
| 6 | avatar\_path | Địa chỉ ảnh đại diện | Varchar(2500) |  |
| 7 | role | Quyền của người dùng | Enum |  |
| 8 | is\_delete | Tài khoản đã xoá? | Tinyint |  |
| 9 | created\_at | Thời gian dữ liệu được ghi | Timestamp(6) |  |
| 10 | updated\_at | Thời gian dữ liệu được thay đổi lần cuối | Timestamp(6) |  |

Tên thực thể: Login\_history

Mô tả: Lưu trữ token đăng nhập, đồng thời lưu trữ thông tin lịch sử của người dùng và các thông tin vị trí người dùng đăng nhập.

Chi tiết thực thể:

Bảng 3.3 Thực thể login\_history

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *STT* | *Tên thuộc tính* | *Diễn giải* | *Kiểu dữ liệu* | *Ràng buộc toàn vẹn* |
| 1 | id | Mã sinh viên | INT | Khoá chính |
| 2 | id\_user | Mã người dùng | Varchar(6) | Khoá ngoại |
| 3 | token | Refresh token | Varchar(2500) |  |
| 4 | latitude | Vĩ độ lúc đăng nhập | Float |  |
| 5 | longtitude | Kinh độ lúc đăng nhập | Float |  |
| 6 | is\_expired | Hết hạn? | Tinyint |  |
| 7 | created\_at | Thời gian dữ liệu được ghi | Timestamp(6) |  |
| 8 | updated\_at | Thời gian dữ liệu được thay đổi lần cuối | Timestamp(6) |  |

Tên thực thể: Password\_history

Mô tả: Lưu trữ mật khẩu của người dùng kể cả các mật khẩu đã không còn sử dụng nữa.

Chi tiết thực thể:

Bảng 3.4 Thực thể password\_history

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *STT* | *Tên thuộc tính* | *Diễn giải* | *Kiểu dữ liệu* | *Ràng buộc toàn vẹn* |
| 1 | id | Mã sinh viên | INT | Khoá chính |
| 2 | id\_user | Mã người dùng | Varchar(6) | Khoá ngoại |
| 3 | password | Mật khẩu đã được hash | Varchar(500) |  |
| 4 | is\_active | Đang được sử dụng? | Tinyint |  |
| 5 | created\_at | Thời gian dữ liệu được ghi | Timestamp(6) |  |

#### Danh sách các thực thể và mối kết hợp ở Device-service

A screenshot of a computer

Description automatically generated

Hình 3.3 Sơ đồ dữ liệu EER

Bảng 3.5 Danh sách thực thể

|  |  |  |
| --- | --- | --- |
| *STT* | *Tên thực thể/mối kết hợp* | *Diễn giải* |
| 1 | Devices | Thông tin công tắc |
| 2 | Device\_ap | Thông tin công tắc ở chế độ Access Point |

**Chi tiết các thực thể và mối kết hợp quan trọng**

Tên thực thể: Devices

Mô tả: Lưu trữ thông tin của công tắc.

Chi tiết thực thể:

Bảng 3.6 Thực thể devices

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *STT* | *Tên thuộc tính* | *Diễn giải* | *Kiểu dữ liệu* | *Ràng buộc toàn vẹn* |
| 1 | id | Mã công tắc | Varchar(6) | Khoá chính |
| 2 | type | Loại thiết bị | Enum |  |
| 3 | name | Tên công tắc | Varchar(30) |  |
| 4 | desc | Mô tả công tắc | Varchar(150) |  |
| 5 | created\_at | Thời gian dữ liệu được ghi | Timestamp(6) |  |
| 6 | updated\_at | Thời gian dữ liệu được thay đổi lần cuối | Timestamp(6) |  |

Tên thực thể: Device\_ap

Mô tả: Lưu trữ thông tin của công tắc khi ở chế độ access point.

Chi tiết thực thể:

Bảng 3.7 Thực thể device\_ap

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *STT* | *Tên thuộc tính* | *Diễn giải* | *Kiểu dữ liệu* | *Ràng buộc toàn vẹn* |
| 1 | id\_device | Mã công tắc | Varchar(6) | Khoá chính, khoá ngoại |
| 2 | ap\_ssid | Tên wifi của công tắc | Varchar(30) |  |
| 3 | ap\_password | Mật khẩu của công tắc | Varchar(10) |  |

#### Danh sách các thực thể và mối kết hợp ở House-service

A diagram of a computer

Description automatically generated with medium confidence

Hình 3.4 Sơ đồ dữ liệu EER

Bảng 3.8 Danh sách các thực thể

|  |  |  |
| --- | --- | --- |
| *STT* | *Tên thực thể/mối kết hợp* | *Diễn giải* |
| 1 | Houses | Thông tin các nhà của người dùng |
| 2 | Areas | Thông tin các khu vực thuộc nhà |
| 3 | Setting | Thông tin thiết lập của nhà |
| 4 | Own\_deivce | Thông tin về các thiết bị mà người dùng sở hữu |

**Chi tiết các thực thể và mối kết hợp quan trọng**

Tên thực thể: Houses

Mô tả: Lưu trữ thông tin “nhà” của người dùng

Chi tiết thực thể:

Bảng 3.9 Thực thể Houses

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *STT* | *Tên thuộc tính* | *Diễn giải* | *Kiểu dữ liệu* | *Ràng buộc toàn vẹn* |
| 1 | id | Mã nhà | Varchar(6) | Khoá chính |
| 2 | id\_user | Mã người dùng | Varchar(6) |  |
| 3 | name | Tên nhà | Varchar(25) |  |
| 4 | desc | Mô tả nhà | Varchar(150) |  |
| 5 | is\_delete | Nhà đã xoá? | Tinyint |  |
| 6 | created\_at | Thời gian dữ liệu được ghi | Timestamp(6) |  |
| 7 | updated\_at | Thời gian dữ liệu được thay đổi lần cuối | Timestamp(6) |  |

Tên thực thể: Areas

Mô tả: Lưu trữ thông tin “khu vực” thuộc “nhà” của người dùng.

Chi tiết thực thể:

Bảng 3.10 Thực thể Areas

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *STT* | *Tên thuộc tính* | *Diễn giải* | *Kiểu dữ liệu* | *Ràng buộc toàn vẹn* |
| 1 | id | Mã khu vực | Varchar(6) | Khoá chính |
| 2 | id\_house | Mã nhà | Varchar(6) | Khoá ngoại |
| 3 | name | Tên khu vực | Varchar(25) |  |
| 4 | desc | Mô tả | Varchar(150) |  |
| 5 | is\_delete | Khu vực đã xoá? | Tinyint |  |
| 6 | created\_at | Thời gian dữ liệu được ghi | Timestamp(6) |  |
| 7 | updated\_at | Thời gian dữ liệu được thay đổi lần cuối | Timestamp(6) |  |

Tên thực thể: Setting

Mô tả: Lưu trữ thông tin thiết lập của “nhà”

Chi tiết thực thể:

Bảng 3.11 Thực thể Setting

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *STT* | *Tên thuộc tính* | *Diễn giải* | *Kiểu dữ liệu* | *Ràng buộc toàn vẹn* |
| 1 | id\_house | Mã nhà | Varchar(6) | Khoá chính |
| 2 | index\_show | Vị trí hiển thị trên giao diện | Int |  |
| 3 | wallpaper\_path | Đường dẫn ảnh nền | Varchar(3000) |  |
| 4 | wallpaper\_blur | Chế độ blur cho ảnh nền? | Tinyint |  |
| 5 | is\_main\_house | Có phải là nhà chính? | Tinyint |  |
| 6 | created\_at | Thời gian dữ liệu được ghi | Timestamp(6) |  |
| 7 | updated\_at | Thời gian dữ liệu được thay đổi lần cuối | Timestamp(6) |  |

Tên thực thể: Own\_device

Mô tả: Lưu trữ thông tin các công tắc mà người dùng đang sở hữu và quản lý

Chi tiết thực thể:

Bảng 3.12 Thực thể Own\_deivce

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *STT* | *Tên thuộc tính* | *Diễn giải* | *Kiểu dữ liệu* | *Ràng buộc toàn vẹn* |
| 1 | id | Mã thực thể | Int | Khoá chính |
| 2 | id\_device | Mã công tắc | Varchar(6) | Khoá ngoại |
| 3 | id\_house | Mã nhà | Varchar(6) | Khoá ngoại |
| 4 | id\_area | Mã khu vực | Varchar(6) | Khoá ngoại |
| 5 | name | Tên công tắc được người dùng đặt lại | Varchar(25) |  |
| 6 | desc | Mô tả | Varchar(150) |  |
| 7 | is\_delete | Tài khoản đã xoá? | Tinyint |  |
| 8 | created\_at | Thời gian dữ liệu được ghi | Timestamp(6) |  |
| 9 | updated\_at | Thời gian dữ liệu được thay đổi lần cuối | Timestamp(6) |  |

#### Danh sách các thực thể và mối kết hợp ở Sys-operation-service

**Chi tiết các thực thể và mối kết hợp quan trọng**

### Thiết kế công tắc

#### Thiết kế mạch

**Module Relay:**

A diagram of a circuit

Description automatically generated

Hình 3.5 Sơ đồ mạch module relay kích ở mức cao

Module relay được thiết kế với các thành phần gồm relay 5V, transistor 2N2222, điện trở 1kΩ và diode 1N4007, hoạt động hiệu quả để điều khiển tải thông qua tín hiệu từ vi điều khiển như ESP-01. Trong mạch, relay đóng vai trò chính là công tắc điện, được kích hoạt khi transistor dẫn dòng. Tín hiệu điều khiển từ GPIO của vi điều khiển đi qua điện trở 1kΩ để bảo vệ chân GPIO và cung cấp dòng vừa đủ để kích hoạt base của transistor 2N2222. Khi đó, transistor dẫn dòng từ cuộn dây relay xuống GND, cho phép relay hoạt động và đóng tải.

Diode 1N4007 được lắp song song với cuộn dây relay nhằm bảo vệ transistor khỏi dòng ngược phát sinh khi relay ngắt. Cách bố trí này giúp đảm bảo sự bền bỉ của mạch, bảo vệ linh kiện và giữ cho tín hiệu điều khiển ổn định. Nguồn điện 5V riêng được sử dụng để cung cấp năng lượng cho relay, đảm bảo không làm ảnh hưởng đến nguồn cấp cho vi điều khiển. Thiết kế module này đáng tin cậy, phù hợp cho các ứng dụng điều khiển từ xa, tự động hóa gia đình hoặc tích hợp trong các dự án IoT như điều khiển thiết bị qua WiFi với ESP-01.

**Công tắc:**

A diagram of a circuit

Description automatically generated

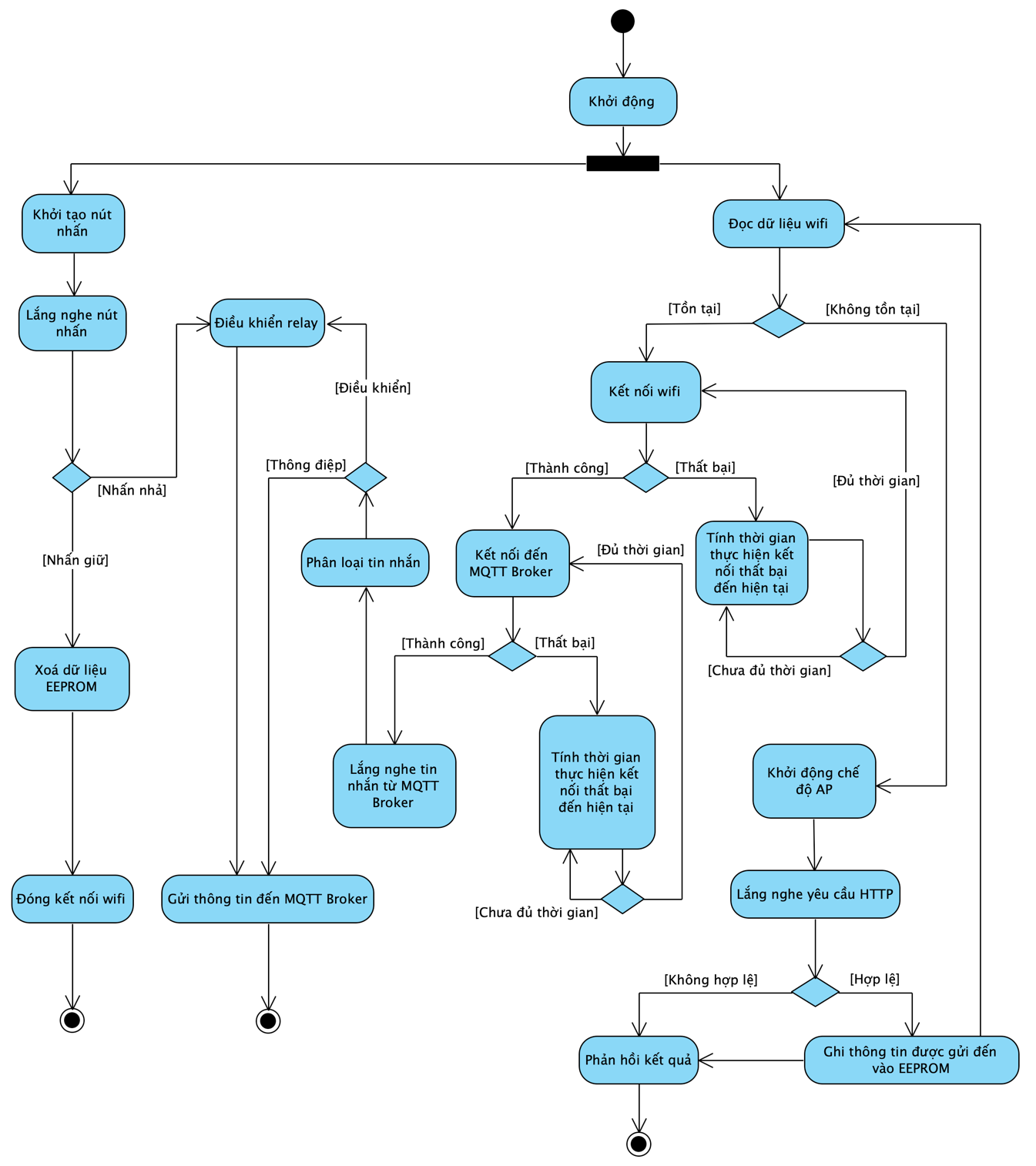
Hình 3.6 Sơ đồ mạch công tắc

Trong thiết kế này, module relay được tích hợp nhằm đảm bảo tính an toàn và hiệu quả cao khi điều khiển các tải sử dụng điện áp cao. Tuy nhiên, có một vấn đề mâu thuẫn cần giải quyết: relay yêu cầu điện áp hoạt động 5V DC, trong khi ESP-01 lại vận hành ổn định ở mức 3.3V DC. Việc giảm điện áp cấp cho relay xuống 3.3V là không khả thi, bởi điều này sẽ ảnh hưởng trực tiếp đến khả năng đóng cắt của relay. Để giải quyết vấn đề này, module AMS1117 được sử dụng để hạ áp từ 5V xuống 3.3V, cung cấp nguồn điện chính xác và ổn định cho ESP-01 mà không làm ảnh hưởng đến hoạt động của relay.

Hơn nữa, theo yêu cầu của hệ thống, việc sử dụng pin để cấp nguồn cho mạch không phải là giải pháp phù hợp, do yêu cầu về sự ổn định và liên tục trong cấp điện. Do đó, nguồn Hi-Link được chọn để chuyển đổi trực tiếp điện áp AC 220V từ lưới điện sang DC, đảm bảo cung cấp đủ năng lượng cho cả relay và ESP-01. Với cách bố trí này, hệ thống vừa đảm bảo an toàn khi làm việc với điện áp cao, vừa duy trì khả năng hoạt động ổn định của các thành phần quan trọng trong mạch.

#### Thiết kế xử lý

**Sơ đồ hoạt động tổng quát:**



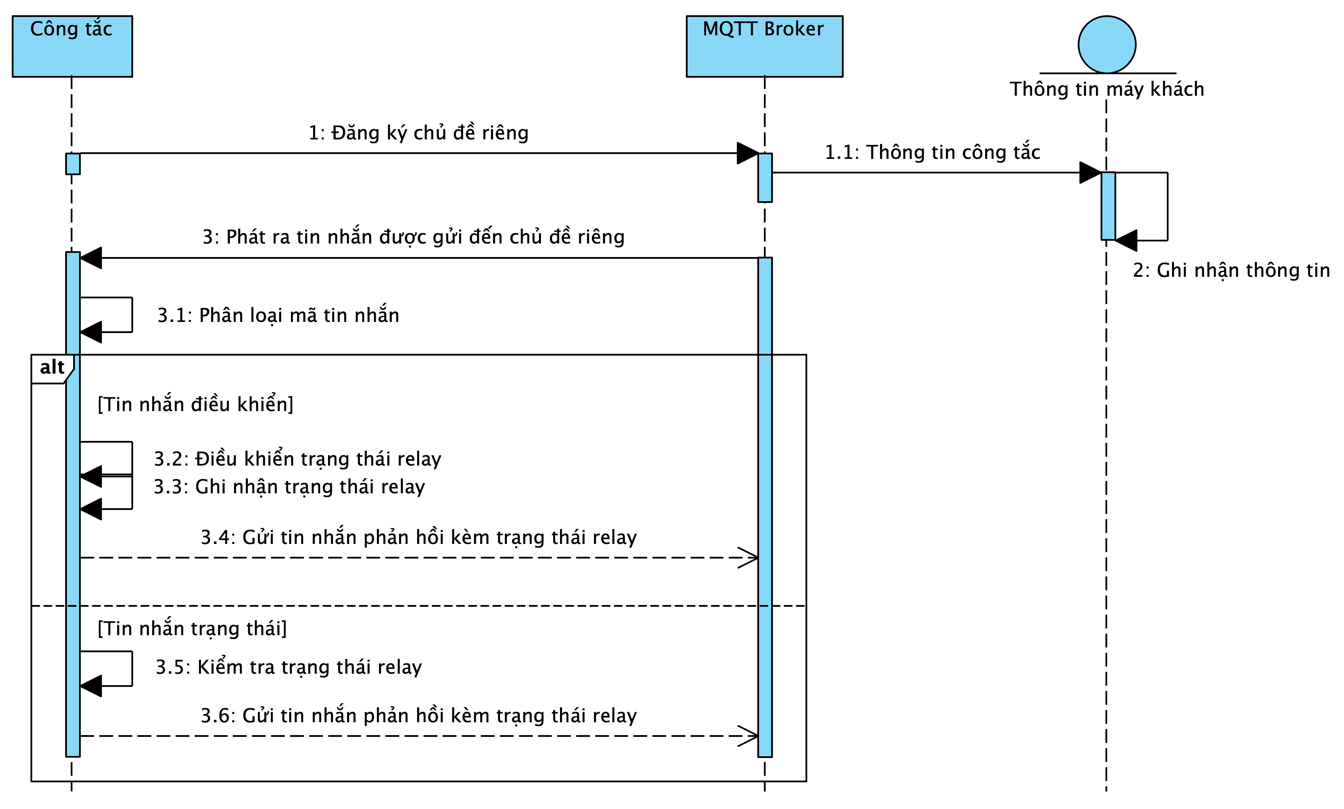
Hình 3.7 Sơ đồ hoạt động của công tắc

Khi công tắc được cấp nguồn để khởi động, nó sẽ hình thành 2 luồng hoạt động, một dành riêng cho lắng nghe và xử lý tín hiệu điều khiển từ nút nhấn, một còn lại dành riêng cho việc kết nối và tương tác với hệ thống máy chủ.

Trong luồng dành cho việc kết nối thì công tắc sẽ đọc các thông tin về ssid và mật khẩu wifi trong bộ nhớ. Nếu không tồn tại ssid thì lập tức công tắc sẽ chuyển sang chế độ AP để cho phép ứng dụng Smartome kết nối đến và cấu hình. Nếu tìm thấy ssid thì công tắc sẽ cố gắng kết nối vào mạng wifi tương ứng, lặp đi lặp lại việc kết nối cho đến khi kết nối được (mỗi lần thực hiện lại kết nối sẽ được quyết định bởi thời gian chờ được cấu hình sẵn). Nếu kết nối vào wifi thành công thì công tắc sẽ cố gắng kết nối vào MQTT Broker tương tự như cách nó kết nối vào mạng wifi. Nếu kết nối vào MQTT Broker thành công thì công tắc sẽ đăng ký một chủ đề riêng giữa ứng dụng Smartome và công tắc (Thường là mã công tắc kết hợp với mã nhà mà nó thuộc về) sau đó công tắc sẽ lắng nghe các tin nhắn được phát ra từ chủ đề mà nó đã đăng ký với MQTT Broker. Đồng thời cũng cho phép gửi tin nhắn đến MQTT Broker với chủ đề đó.

Trong khi luồng kết nối đang cố gắng thực hiện thì luồng dành riêng cho nút nhấn cũng khởi tạo và lắng nghe các tín hiệu “nhấn nhả” hoặc “nhấn giữ” từ nút nhấn vật lý, nếu là nhấn nhả thì sẽ đảo trạng thái của relay nhằm đóng/mở dòng điện. Nếu tín hiệu là nhấn giữ thì công tắc sẽ xoá toàn bộ dữ liệu cấu hình của người dùng mà công tắc đã lưu trữ để trở về trạng thái gốc.

**Sơ đồ tuần tự xử lý tin nhắn:**



Hình 3.8 Sơ đồ tuần tự xử lý tin nhắn của công tắc

Đăng ký vào một chủ đề và kèm theo thông tin công tắc là điều bắt buộc khi muốn nhận tin nhắn từ MQTT Broker. Sau khi đăng ký chủ đề công tắc sẽ lắng nghe những tin nhắn từ chủ đề đó được phát ra bởi MQTT Broker. Theo kiến trúc được xây dựng thì tin nhắn gửi đến công tắc sẽ có 2 dạng chính gồm: Tin nhắn điều khiển, tin nhắn trạng thái.

Tin nhắn điều khiển là những tin nhắn được gửi đến chủ yếu từ ứng dụng Smartome để điều khiển đóng mở điện, khi nhận được tin nhắn này công tắc sẽ chuyển trạng thái relay sang trạng thái được yêu cầu từ tin nhắn, sau khi chuyển trạng thái thành công công tắc sẽ phản hồi lại bằng cách gửi tin nhắn trạng thái hiện tại đến chủ đề đã đăng ký trên MQTT Broker.

Tin nhắn trạng thái là những tin nhắn được gửi đến công tắc nhằm mục đích kiểm trang trạng thái trực tuyến/ngoại tuyến của công tắc đồng thời lấy trạng thái relay của công tắc. Khi nhận được tin nhắn này công tắc sẽ đọc trạng thái của relay và phản hồi tin nhắn tương tự phản hồi tin nhắn điều khiển.

### Thiết kế server

#### Ứng dụng mô hình microservices

A diagram of a service

Description automatically generated

Hình 3.9 Kiến trúc server theo mô hình microservices

Hệ thống server được thiết kế theo hướng phân tách chức năng thành các dịch vụ độc lập, dựa trên quá trình phân tích chi tiết các đối tượng và luồng xử lý trong hệ thống. Cốt lõi của thiết kế này là khả năng giải quyết hiệu quả các tác vụ liên quan đến xác thực người dùng, quản lý thiết bị công tắc, xử lý dữ liệu người dùng, và điều phối tương tác giữa người dùng với các công tắc trong môi trường IoT.

Kiến trúc này kế thừa những ưu điểm nổi bật của mô hình microservice, trong đó các thành phần được xây dựng để đảm nhận các chức năng chuyên biệt, đồng thời giao tiếp với nhau qua giao thức nhẹ và hiệu năng cao. Đáp ứng yêu cầu điều phối các luồng xử lý đa dạng, một **Gateway** được triển khai để tiếp nhận toàn bộ yêu cầu từ phía người dùng, thực hiện các kiểm tra ban đầu và chuyển tiếp thông minh đến các dịch vụ liên quan.

***Gateway:* được xem là cửa ngõ của server**, đóng vai trò làm cổng trung gian giữa người dùng và các microservice. Gateway xử lý tất cả các yêu cầu HTTP từ phía client, thực hiện kiểm tra ban đầu như xác thực cơ bản và sau đó điều phối các yêu cầu này tới microservice tương ứng. Việc sử dụng gRPC trong kết nối giữa Gateway và các microservice giúp tối ưu hóa hiệu năng nhờ giao tiếp nhanh và hiệu quả trên nền tảng kết nối RPC và HTTP/2.

***Auth-Service:*** tập trung giải quyết các nhiệm vụ liên quan đến xác thực người dùng. Dịch vụ này chịu trách nhiệm xử lý quy trình đăng nhập, tạo mới tài khoản, và quản lý quyền truy cập. Ngoài ra, Auth-Service đảm bảo dữ liệu nhạy cảm được bảo mật trong quá trình giao tiếp với các dịch vụ khác bằng cách phát hành và xác nhận các token xác thực.

***House-Service:*** được xây dựng để xử lý thông tin quản lý nhà ở, tập trung vào việc tạo mới các thông tin liên quan như nhà, khu vực bên trong nhà, và việc thêm mới các thiết bị điều khiển (công tắc) cho từng không gian. Dịch vụ này sử dụng mô hình dữ liệu linh hoạt nhằm đáp ứng các yêu cầu cấu trúc nhà phức tạp từ phía người dùng.

***Device-Service:*** Chịu trách nhiệm lưu trữ và quản lý các thông tin chi tiết về công tắc, bao gồm cấu hình, trạng thái và các thuộc tính liên quan. Bằng cách cung cấp giao diện API chuyên dụng, Device-Service hỗ trợ tích hợp dễ dàng giữa các thiết bị IoT và hệ thống quản lý trung tâm.

***Sys-Operation-Service***: Dịch vụ chuyên xử lý và giám sát giao tiếp qua giao thức MQTT. Dịch vụ này liên tục lắng nghe các thông điệp từ các chủ đề tại MQTT Broker, đồng thời thực hiện các xử lý cần thiết, bao gồm việc ghi nhận thông tin tương tác giữa người dùng và các thiết bị công tắc. Các thông tin lịch sử này không chỉ đóng vai trò lưu trữ mà còn hỗ trợ phân tích hành vi sử dụng thiết bị theo thời gian.

#### Thiết kế xử lý

**Mô hình tuần tự đăng ký người dùng:**

**A diagram of a project

Description automatically generated**

Hình 3.10 Mô hình tuần tự đăng ký người dùng

Gateway là service đặc biệt đón và điều hướng tất cả các yêu cầu http của hệ thống. Gateway sẽ chuyển thông tin đăng ký đến Auth-Service, Auth-Service sẽ gửi thông tin quan trọng nhất gồm email và tên đăng nhập đến cơ sở dữ liệu để kiểm tra xem đã có người dùng nào sử dụng một trong hai thông tin trên hay cả hai hay chưa. Sẽ có 2 trường hợp xảy ra gồm: Thông tin đã tồn tại hoặc thông tin chưa tồn tại.

Khi thông tin tạo mới người dùng gồm email và tên đăng nhập đã tồn tại trong cơ sở dữ liệu thì cơ sở dữ liệu sẽ trả về toàn bộ thông tin người dùng đó đến Auth-Service, tại Auth-Service sẽ xử lý để gắn các thông báo và mã lỗi phù hợp sau đó phản hồi về cho Gateway.

Khi thông tin tạo mới người dùng chưa từng tồn tại trong cơ sở dữ liệu thì kết quả gửi về Auth-Service sẽ là rỗng, khi đó Auth-Service sẽ hiểu rằng thông tin tạo mới này là hợp lệ thì sẽ gửi thông tin đó đến cơ sở dữ liệu lần nữa kèm theo mã OTP (Có thời gian sống là 2 phút) để lưu lại. Ngay lập tức Auth-Service sẽ yêu cầu máy chủ mail gửi mã OTP đó đến email mà người dùng vừa đăng ký. Gửi mail thành công Auth-Service sẽ phản hồi cho Gateway là đăng ký người dùng thành công.

Xác thực OTP là bước mà Gateway sẽ gửi thông tin OTP đến Auth-Service để kiểm tra, Auth-Service sẽ gửi OTP kèm email đến cơ sở dữ liệu để kiểm tra, tại đây cũng sẽ xảy ra 2 trường hợp gồm: Mã OTP khớp hoặc mã OTP không khớp.

Nếu mã OTP không khớp thì Auth-Service sẽ nhận được một giá trị rỗng từ cơ sở dữ liệu phản hồi về và cũng sẽ xử lý để có thông báo lỗi và mã lỗi phù hợp. Cuối cùng là gửi đến Gateway đề phản hồi về cho người dùng.

Nếu mà OTP là khớp thì Auth-Service sẽ nhận được dữ liệu mã OTP kèm theo các thông tin mà người dùng đã đăng ký và được ghi nhận xuống cơ sở dữ liệu trước đó. Auth-Service sẽ xử lý để cấu trúc dữ liệu phù hợp và sẽ gửi đến cơ sở dữ liệu để lưu trữ thông tin người dùng chính thức. Kết quả sau khi hoàn tất các giai đoạn trên Auth-Service cũng sẽ phản hồi về cho Gateway thông điệp xác thực người dùng thành công.

**Mô hình tuần tự đăng nhập và xác thực đăng nhập:**

**A diagram of a diagram

Description automatically generated**

Hình 3.11 Sơ đồ tuần tự xử lý đăng nhập

Thông tin đăng nhập gồm tên đăng nhập và mật khẩu gửi đến Auth-Service từ Gateway, Auth-Service sẽ gửi tên đăng nhập đến cơ sở dữ liệu để tìm kiếm người dùng đó. Cơ sở sẽ trả về cho Auth-Service một người dùng phù hợp hoặc một dữ liệu rỗng. Auth-Service sẽ xử lý, nếu là dữ liệu rỗng Auth-Service sẽ xử lý tạo các thông báo lỗi và mã lỗi phù hợp để phản hồi về cho Gateway. Nếu dữ liệu trả về từ cơ sở dữ liệu là một người dùng thì Auth-Service sẽ kiểm tra đến mật khẩu đã khớp hay chưa, nếu không khớp thì lặp lại quá trình như lúc nhận một dữ liệu rỗng. Nếu các thông tin tên đăng nhập và mật khẩu đều khớp thì Auth-Service sẽ xử lý các dữ liệu cho phù hợp và tạo token từ dữ liệu đó. Token sẽ được gửi đến cơ sở dữ liệu để lưu trữ và đồng thời cũng được gửi đến Gateway.

#### Cấu trúc code

Server được chia thành nhiều Service nhưng vẫn đồng nhất về công nghệ sử dụng cho nên cấu trúc code ở các Service đều không có quá nhiều khác biệt, cấu trúc code vẫn tuân thủ theo nguyên tắc và tối thiểu phải có ở mỗi service là cấu trúc sau:

A screenshot of a computer

Description automatically generated

Hình 3.12 Cấu trúc code của một Service điển hình

*scripts:* Chứa các file lệnh (script) để đọc giấy phép TLS gốc (RootCA.crt) để tạo ra các TLS dành riêng cho service này.

*mTLS:* Thư mục chứa các giấy phép mTLS được tạo ra khi thực thi các lệnh bên trong thư mục scripts.

*src:* Thư mục chính chứa toàn bộ mã nguồn của dịch vụ.

src/config: Thư mục chứa các class cấu hình chính cho service như cấu hình các **exception filters** dùng để xử lý lỗi phát sinh trong ứng dụng và gửi thông tin lỗi về client, cấu hình các interceptor (xử lý các yêu cầu trước khi vào/ra controller), cấu hình các phản hồi về máy khách (response).

*src/DTO:* Chứa các **Data Transfer Objects (DTO)** để định nghĩa các kiểu dữ liệu mà ứng dụng nhận từ các request hoặc gửi trả về cho client.

*src/entity:* Chứa các **entity**, đại diện cho các bảng (table) trong cơ sở dữ liệu. Mỗi entity được định nghĩa bằng TypeScript class với các decorator của TypeORM.

*src/modules:* Thư mục chứa các service nhỏ hơn phục vụ các nhiệm vụ nhỏ nhưng cần phối hợp với nhau để hoàn chỉnh.

*src/modules/auth:* Là một service chuyên xử lý các vấn đề về xác thực và tạo mới người dùng. Bên trong một service này là các tập tin controller, repository, service.

*src/modules/common:* Có cấu trúc tương tự “src/modules/auth”, là service có nhiệm vụ xử lý các tác vụ chung như mã hoá, gửi email, tạo id, kết nối cơ sở dữ liệu.

*src/app.module.ts:* File này được gọi là **root module, l**à entry point chính của ứng dụng NestJS**, nơi** khai báo các module con, các provider (dịch vụ), và các controller mà ứng dụng sử dụng.

*src/main.ts:* Tệp khởi chạy chính của ứng dụng. Chứa lệnh khởi tạo và chạy server (bootstrap()).

*.env:* Chứa các biến môi trường, các dữ liệu nhạy cảm (secret).

*.eslintrc.js:* Tệp cấu hình cho **ESLint** dùng để kiểm tra và chuẩn hóa code theo quy tắc cụ thể. Đảm bảo mã nguồn đồng nhất, dễ bảo trì. Ngăn chặn các lỗi thông thường (syntax error, bad practices,...).

*nest-cli.json:* Tệp cấu hình dành riêng cho **Nest CLI**, hỗ trợ bạn khi sử dụng các lệnh CLI để tạo module, controller, service,...

*tsconfig.json:* Tệp cấu hình **TypeScript** chính.

### Thiết kế ứng dụng di động

#### Thiết kế xử lý

**Sơ đồ hoạt động:**

**A diagram of a flowchart

Description automatically generated**

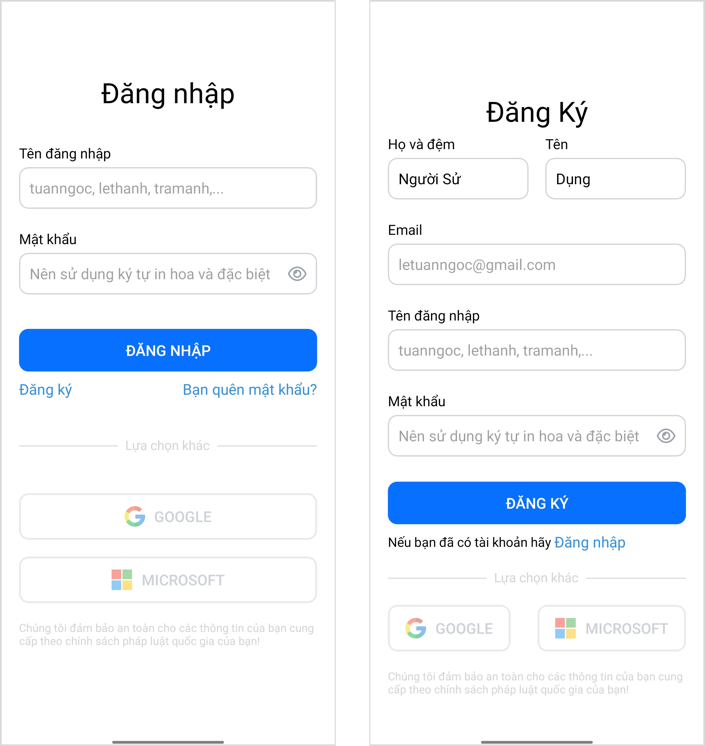
Hình 3.13 Sơ đồ hoạt động của ứng dụng Smartome

Kiểm tra các thông tin xác thực là hoạt động đầu tiên của ứng dụng Smartome, thông tin về token sẽ được lấy từ bộ nhớ bảo mật, nếu không tìm thấy thì ứng dụng sẽ chuyển đến màn hình đăng nhập, đăng nhập không thành công ứng dụng sẽ chuyển đến màn hình đăng ký. Đăng ký thành công sẽ quay lại màn hình đăng nhập, thực hiện lại đăng nhập sau khi đăng nhập thành công sẽ nhận được token từ server thì ứng dụng sẽ lưu token vào bộ nhớ và thực hiện lấy thông tin người dùng từ token đó bằng cách gửi yêu cầu đến server.

Sau khi có thông tin người dùng, ứng dụng sẽ gọi đến server để lấy thông tin về danh sách nhà và các khu vực trong từng nhà. Tiếp đến ứng dụng sẽ dựa vào thông tin nhà đã lấy để gọi đến server lấy dữ liệu về các công tắc mà người dùng đang sở hữu. Sau khi đã lấy thông tin các công tắc thành công và tồn tại ít nhất một công tắc thì ứng dụng sẽ kết nối đến MQTT Broker để thiết lập kết nối từ ứng dụng đến các công tắc. Tại đây sẽ có 2 trường hợp xảy ra, trường hợp kết nối không thành công thì sẽ tạm ngưng trong một khoản thời gian được cấu hình để cố gắng thực hiện lại việc kết nối. Trường hợp kết nối thành công, ứng dụng sẽ đăng ký vào các chủ đề thuộc về công tắc và ứng dụng. Tiếp sau đó là hoạt động gửi tin nhắn lấy trạng thái công tắc đến hàng loạt đến các chủ đề đã đăng ký ở trên. Ngay sau đó ứng dụng sẽ lắng nghe các tin nhắn phản hồi và xử lý để hiển thị lên giao diện trạng thái công tắc.

#### Thiết kế giao diện

**Giao diện đăng nhập:**



Hình 3.14 Bản phác thảo màn hình đăng nhập và màn hình đăng ký

Giao diện màn hình đăng nhập chú trọng cho người dùng đăng nhập bằng tài khoản riêng của hệ thống với yêu cầu đăng nhập gồm tên đăng nhập (Tên tài khoản) và mật khẩu. Ngoài ra còn hỗ trợ đăng nhập bằng Google hoặc Microsoft.

Giao diện màn hình đăng ký người dùng chủ đạo vẫn là các ô nhập liệu các thông tin người dùng nhằm hướng đến người dùng sử dụng tài khoản của hệ thống. Các yêu cầu không bắt buộc gồm “Họ và đệm” và “Tên”, tên này hệ thống đã chọn mặc định cho người dùng, người dùng có thể để mặc định hoặc sửa đổi tuỳ thích. Kế tiếp là các thông tin bắc buộc gồm “Email”, “Tên đăng nhập” và “Mật khẩu”. Người dùng phải sử dụng email chưa từng được đăng ký tài khoản với hệ thống và tên đăng nhập không trùng với các người dùng đã đăng ký trước đó, yêu cầu về mật khẩu cũng được đảm bảo khi người dùng phải đặt mật khẩu phải đảm bảo đủ chữ số, chữ cái, ký tự đặc biệt. Giao diện cũng hỗ trợ cảnh báo lỗi cụ thể với từng thông tin mà người dùng nhập vào, giúp cho việc đăng ký trở nên rõ ràng và thuận tiện hơn cho người dùng. Từ đó giúp người dùng nhanh chóng tiếp cận những tiện ích mà hệ thống mang lại sau khi đăng nhập thành công.

**Các giao diện trang chính:**

Screens screenshot of a phone

Description automatically generated

Hình 3.15 Bản phác thảo các giao diện tại màn hình chính ứng dụng Smartome

Giao diện màn hình chính có thiết kế tập trung giúp người dùng dễ dàng tiếp cận các công tắt của mình ngay khi mở app, nhưng vẫn đảm bảo phân cụm công tắc theo khu vực mà người dùng đã thiết lập.

Ngay tại màn hình chính người dùng có thể thêm nóng các công tắc ngay lập tức, nút nhấn này được bố trí ngay dưới tên nhà kèm dòng mô tả bên trong nút “Thêm thiết bị” (Được mô tả ở ảnh A).

Để mở rộng khả năng xem và quản lý cho người dùng nhưng đảm bảo giao diện được tối giản thì ứng dụng bố trí hai nút nhấn ở góc trên bên phải màn hình cho người dùng dễ dàng tiếp cận. Nút đầu tiên từ phải sang trái là nút dùng mở bảng chọn (Khi nhấn vào sẽ có giao diện như ảnh B) cài đặt nhà và cho phép truy cập đến nhà đang có trực tiếp, hoặc người dùng có thể truy cập vào khu vực cụ thể trong nhà ngay tại bảng chọn này. Nút thứ 2 từ phải sang trái là nút dùng mở bảng chọn dành riêng cho việc thêm thiết bị, tạo mới nhà, tạo mới khu vực trong nhà (Giao diện của bảng chọn này được mô tả qua ảnh C).

**Giao diện trang thêm công tắc:**

A screenshot of a phone

Description automatically generated

Hình 3.16 Bản phác thảo giao diện màn hình thêm công tắc

Giao diện màn hình thêm thiết công tắc sẽ có hai dạng gồm: Thêm công tắc với mạng wifi 2.4GHz, thêm công tắc với mạng wifi phát hai băng tần 2.4GHz và 5GHz.

Khi ở trường hợp mạng wifi chỉ có băng tần 2.4GHz người dùng chỉ cần điền thông tin của công tắc và thông tin mạng wifi mà công tắc được phép kết nối vào. Nhấn kết nối ứng dụng sẽ xử lý để gửi thông tin đó đến công tắc để ghép nối và cấu hình công tắc đó.

Khi ở trường hợp mạng wifi phát hai băng tần thì người dùng chỉ cần điền thông tin mạng wifi mà công tắc được phép kết nối vào. Sau đó mở cài đặt hệ thống, đi đến phần kết nối wifi để tìm và kết nối đến wifi của công tắc phát ra. Kết nối thành công và quay lại ứng dụng Smartome để thực hiện việc xác nhận cấu hình. Lúc này ứng dụng sẽ tự ghép nối và cấu hình công tắc đó.

#### Cấu trúc code

A screenshot of a computer

Description automatically generated

Hình 3.17 Cấu trúc code ứng dụng di động Smartome

*app:* Đây là thư mục chứa các tập tin lập trình cho các màn hình giao diện.

*assets:* Chứa các tài nguyên tĩnh như hình ảnh, biểu tượng, font chữ và âm thanh.

*components:* Chứa các thành phần giao diện tái sử dụng như button, card, modal, và các thành phần giao diện khác.

*config:* Thư mục chứa các tệp tin cấu hình gồm cấu hình API, theme.

*constants:* Chứa các giá trị cố định được sử dụng trong ứng dụng như màu sắc, các giá trị kích thước, thời gian.

*fetch:* Được sử dụng để xử lý các yêu cầu HTTP tới server. Tất cả các logic liên quan đến API sẽ được giữ trong thư mục này.

*hooks:* Lưu trữ các hooks tuỳ chỉnh như hook để xác thực, hook để kết nối đến MQTT Broker, .

*interfaces:* Chứa các định nghĩa interface trong TypeScript để cấu trúc dữ liệu trong ứng dụng cho các trường hợp yêu cầu cấu trúc chặt chẽ.

*.env:* Chứa các biến môi trường, các dữ liệu nhạy cảm (secret).

*app.json:* Tệp cấu hình của Expo, nơi định nghĩa thông tin ứng dụng như tên, biểu tượng, slug, hoặc quyền hạn cần thiết (permissions) khi đóng gói ứng dụng.

*babel.config.js:* Cấu hình Babel, giúp chuyển đổi mã ES6/ES7+ sang mã tương thích với môi trường cũ hơn như React Native hoặc các trình duyệt khác.

*eas.json:* Tệp cấu hình của Expo Application Services (EAS) để tùy chỉnh quá trình build hoặc publish ứng dụng lên các nền tảng như App Store hay Google Play.

*expo-env.d.ts:* File định nghĩa kiểu (type definition) của các biến môi trường trong TypeScript khi làm việc với Expo.

*tsconfig.json:* Tệp cấu hình cho TypeScript, định nghĩa các quy tắc kiểm tra kiểu, phạm vi module, và cách biên dịch TypeScript sang JavaScript.

#### Xây dựng tập tin APK

Bước 1: Đảm bảo máy tính đã cài đặt Expo, kiểm tra bằng cách chạy lệnh “expo -v” hoặc “expo –version” để lấy thông tin phiên bản cài đặt expo. Nếu đã cài đặt Expo thì chuyển đến bước 3.

Bước 2: Chạy lệnh sau để cài đặt expo:

npm install --global expo-cli eas-cli

Bước 3: Mở terminal tại thư mục gốc dự án, chạy lệnh sau để build ứng dụng:

eas build:run -p android

Bước 4: Chờ đợi quá trình build hoàn tất, sau khi build thành công Expo sẽ cung cấp một địa chỉ HTTP để tải tập tin .apk của ứng dụng.

### Thiết lập MQTT Broker

#### Tải mã nguồn và cài đặt

Truy cập vào trang cung cấp MQTT Broker mã nguồn mở được EMQX chính thức phát hành tại địa chỉ <https://www.emqx.com/en/downloads-and-install/broker> để tìm hiểu và đọc các yêu cầu từ nhà cung cấp.

Để cài đặt MQTT Broker với Docker ta thực hiện lần lượt các bước sau:

Bước 1: Chạy lệnh sau để kéo MQTT Broker image về máy.

docker pull emqx/emqx:5.8.4

Bước 2: Chạy docker container từ image vừa kéo về.

docker run -d --name emqx -p 1883:1883 -p 8083:8083 -p 8084:8084 -p 8883:8883 -p 18083:18083 emqx/emqx:5.8.4

#### Cấu hình

Truy cập địa chỉ [http://localhost:18083/ - /login](http://localhost:18083/#/login) để vào trang quản trị MQTT Broker. Đăng nhập với “admin” cho username và “public” cho mật khẩu. Để tạo tài khoản cho công tắc, ứng dụng Smartome và server:

Bước 1: Truy cập vào địa chỉ [http://localhost:18083/ - /users](http://localhost:18083/#/users)

Bước 2: Nhấn vào nút “+ Create” ở góc trên bên phải để mở form tạo. Điền các thông tin và nhấn nút “Create”.

A screenshot of a chat

Description automatically generated

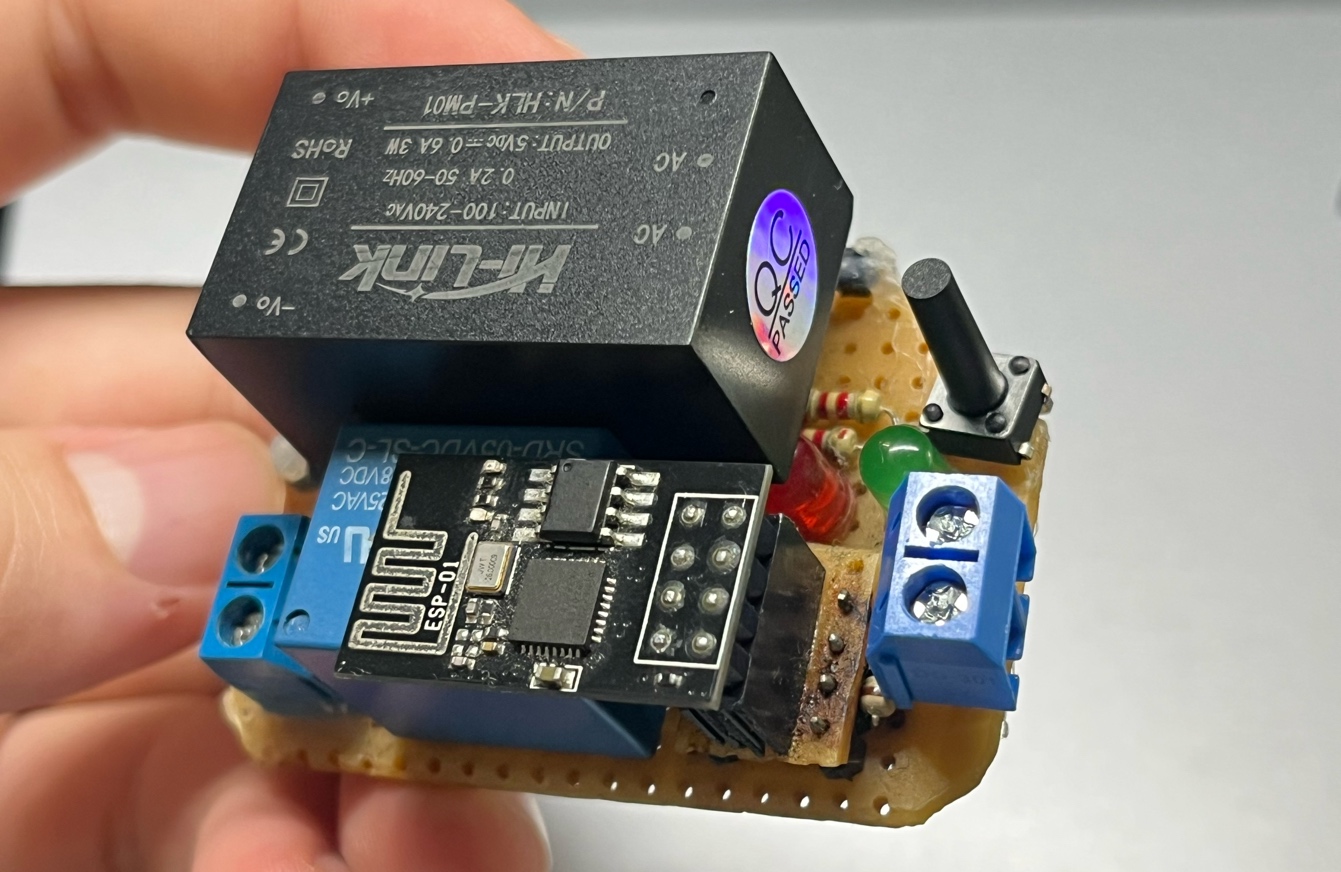
Hình 3.18 Form tạo người dùng trên MQTT Broker

# KẾT QUẢ NGHIÊN CỨU

## Công tắc

### Sản phẩm

Công tắc được lắp ráp từ các module có sẵn và tự thiết kế tương thích với nhau tốt, bố trí các linh kiện chủ động đã cho ra công tắc có kích thước gọn gàng. Sử dụng các bảng mạch đục lỗ (PCB) có sẵn, kết nối các linh kiện qua dây dẫn, điều này đảm bảo an toàn, dễ sửa chữa, và linh kiện sẵn có nếu làm với quy mô nhỏ việc sử dụng PCB có sẵn này còn giúp tối ưu chi phí rất đáng kể.



Hình 4.1 Công tắc sau khi lắp ráp các module và hoàn thiện chức năng

### Hoạt động

Khi nạp code cho công tắc ở chế độ debug để theo dõi hoạt động của công tắc khi tương tác với MQTT Broker thì thu được các giá trị đúng theo sơ đồ hoạt động đã thiết kế.

Trong các log có hai vị trí cần chú ý “Message arrived” là tin nhắn điều khiển được mà công tắc nhận được. “JSON Message published” là tin nhắn phản hồi mà công tắc gửi đi.

Log khi có tín hiệu điều khiển từ MQTT Broker:

Message arrived [P7XGWU/7C3Q9S/receive] {"type":"CONTROLL","id":"1736311146064","value":true}

Id: 1736311146064

Type: CONTROLL

Value: 1

Turning ON

JSON Message published: {"id":"1736311146064","type":"CONTROLL\_RES","value":true}

Log khi công tắc được điều khiển bởi nút nhấn vật lý trên công tắc:

Button pressed

JSON Message published: {"id":"","type":"NOTI","value":false}

## Server

### Ứng dụng thành công mô hình microservices

Screens screenshot of a computer screen

Description automatically generated

Hình 4.2 Các dịch vụ đang chạy theo mô hình microservices

Server hoạt động tốt với các API đã xây dựng, các dịch vụ trao đổi dữ liệu với nhau tương thích tốt, hoạt động trơn tru.

*Tối ưu hóa hiệu năng và khả năng đáp ứng:* Thời gian xử lý các thao tác như xác thực người dùng hay điều khiển công tắc đã được giảm đáng kể so với mô hình monolithic. Sự kết hợp giữa Gateway và gRPC đã tối ưu hóa băng thông và đảm bảo khả năng giao tiếp liền mạch giữa các thành phần.

*Đảm bảo tính ổn định trong vận hành:* Việc phân tách hệ thống thành các microservice độc lập đã chứng minh hiệu quả vượt trội trong việc duy trì sự ổn định của server. Các lỗi tại từng microservice, nếu xảy ra, đều được cách ly và khắc phục mà không ảnh hưởng đến toàn bộ hệ thống. Điều này giúp hệ thống luôn duy trì trạng thái hoạt động liên tục, phục vụ người dùng ngay cả trong những trường hợp bất ngờ.

*Hiệu quả quản lý và tương tác:* Gateway đã thực hiện xuất sắc vai trò điều phối, giúp tôi dễ dàng kiểm soát mọi yêu cầu từ phía người dùng, xác thực và phân phối các tác vụ đến đúng microservice liên quan. Ví dụ, Auth-Service đã đảm bảo toàn bộ quá trình đăng nhập và cấp quyền được diễn ra an toàn, trong khi Sys-Operation-Service đã xử lý hàng triệu thông điệp MQT, hỗ trợ ghi nhận chính xác mọi tương tác giữa người dùng và các công tắc.

*Ứng dụng thành công vào thực tiễn:* Hệ thống không chỉ vận hành trơn tru mà còn hỗ trợ hiệu quả quá trình phát triển. Trong các lần triển khai tính năng mới, việc cập nhật từng microservice độc lập đã giảm đáng kể thời gian bảo trì và hạn chế tối đa lỗi không mong muốn.

### Logic xử lý

Logic xử lý đã được xây dựng đúng theo sơ đồ hoạt động, đặc biệt logic xử lý ở việc xử lý đăng ký người dùng, đăng nhập và xác thực người dùng trong quá trình xây dựng đã phát sinh nhiều vấn đề về bảo mật đáng quan ngại. Việc chỉ sử dụng một token cho người dùng vẫn đáp ứng được vấn đề xác thực bảo mật nhưng nó không tối ưu và an toàn cho dữ liệu khi chẳng may token này bị lộ và không thể làm gì hơn cho tới khi chờ token này hết hạn. Vì thế thay vì sử dụng một token thì server đã tạo ra hai loại token sau khi đăng nhập thành công. Một là access-token có thời gian sống ngắn đơn vị tính bằng phút, hai là refresh-token, token này được lưu ở máy người dùng và cả ở server. Ở máy người dùng refresh-token chỉ có nhiệm vụ duy nhất là gửi lên server để lấy access-token mới mỗi khi access-token gần hết hạn. Theo thiết kế này thì mỗi khi người dùng phát hiện token của mình bị lộ và bị khai thác thì người dùng hoàn toàn có thể dừng việc đó lại ngay lập tức bằng cách yêu cầu server huỷ refresh-token này.

Cơ chế này mang lại nhiều lợi ích nổi bật:

*An toàn hơn khi bị lộ token:* Nếu một access-token bị lộ, tác động tiêu cực sẽ bị giới hạn nhờ thời gian sống ngắn của nó.

*Khả năng xử lý linh hoạt khi phát hiện xâm phạm:* Trong trường hợp refresh-token bị lộ, người dùng có thể ngay lập tức yêu cầu server hủy token này, qua đó ngăn chặn mọi hành vi khai thác trái phép.

*Nâng cao trải nghiệm người dùng:* Với việc duy trì refresh-token, người dùng không cần phải đăng nhập lại thường xuyên, mang đến sự liền mạch trong trải nghiệm mà vẫn đảm bảo tính bảo mật.

### Các API quan trọng

#### Đăng nhập

Sử dụng Postman để kiểm tra:

A screenshot of a computer

Description automatically generated

Hình 4.3 Kiểm tra API đăng nhập bằng Postman

Kết quả:

{

"code": 200,

"status": "Sign-in successfully",

"message": [],

"data": {

"access\_token": {

"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6Ik5UUzRTOSIsImxhc3RuYW1lIjpudWxsLCJmaXJzdG5hbWUiOm51bGwsInVzZXJuYW1lIjoibmRtIiwiZW1haWwiOiJuZ3V5ZW5kdWNtYW5oc3R0dnVAZ21haWwuY29tIiwiYXZhdGFyUGF0aCI6bnVsbCwicm9sZSI6IlVTRVIiLCJ1cGRhdGVkQXQiOiIyMDI1LTAxLTA2VDEzOjUwOjMxLjkyNFoiLCJpYXQiOjE3MzYzMDQ3OTYsImV4cCI6MTczNjMwNjU5Nn0.T4zHFuIbKOwUyxOsw5gXwdjy65GOcFIuKpBBUBmLzn8",

"expires\_in": 1800

},

"refresh\_token": {

"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6Ik5UUzRTOSIsImxhc3RuYW1lIjpudWxsLCJmaXJzdG5hbWUiOm51bGwsInVzZXJuYW1lIjoibmRtIiwiZW1haWwiOiJuZ3V5ZW5kdWNtYW5oc3R0dnVAZ21haWwuY29tIiwiYXZhdGFyUGF0aCI6bnVsbCwicm9sZSI6IlVTRVIiLCJ1cGRhdGVkQXQiOiIyMDI1LTAxLTA2VDEzOjUwOjMxLjkyNFoiLCJpYXQiOjE3MzYzMDQ3OTYsImV4cCI6MTczNjkwOTU5Nn0.OmEQwHOc9LuhTWrFCak\_5i2RQDBnO5vxJCxd7UpN2no",

"expires\_in": 604800

}

}

}

#### Lấy access-token mới

Sử dụng Postman để kiểm tra:

A screenshot of a computer

Description automatically generated

Hình 4.4 Kiểm tra API lấy access-token mới bằng Postman

Kết quả:

{

"code": 200,

"status": "Get new access token is successfully",

"message": [],

"data": {

"access\_token": {

"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6Ik5UUzRTOSIsImxhc3RuYW1lIjpudWxsLCJmaXJzdG5hbWUiOm51bGwsInVzZXJuYW1lIjoibmRtIiwiZW1haWwiOiJuZ3V5ZW5kdWNtYW5oc3R0dnVAZ21haWwuY29tIiwiYXZhdGFyUGF0aCI6bnVsbCwicm9sZSI6IlVTRVIiLCJpYXQiOjE3MzYyNjkzNDYsImV4cCI6MTczNjI3MTE0Nn0.6RS7llWTrAZclxeBGJOPtMgACsoXiTDHtmIvHh8thH0",

"expires\_in": 1800

}

}

}

## Ứng dụng di động

### Giao diện

**Màn hình đăng nhập và đăng ký:**

Screens screenshot of a phone

Description automatically generated

Hình 4.5 Màn hình đăng nhập và màn hình đăng ký

Xây dựng thành công hơn mức thiết kế khi các ô nhập dữ liệu có hỗ trợ hiển thị lỗi cụ thể cho người dùng dễ dàng kiểm xoát thông tin của mình, đồng thời hỗ trợ tốt việc xác định lỗi khi lần đầu tiếp xúc với ứng dụng.

**Màn hình chính với nhiều trạng thái:**

Screens screenshot of a phone

Description automatically generated

Hình 4.6 Các giao diện màn hình chính ở 3 trạng thái

Màn hình chính, màn hình giao diện đầu tiên khi truy cập ứng dụng (Hình A)

Màn hình chính khi tương tác với nút nhấn đầu tiên từ phải sang trái ở góc trên bên phải màn hình, bảng chọn mở ra cho phép vào màn hình cài đặt nhà tại lựa chọn đầu tiền, tiếp đến là cụm cho phép chuyển đổi nhanh vào các nhà đang quản lý, cụm dưới cùng là các lựa chọn truy cập nhanh về khu vực trong nhà đang xem (Hình B).

Tại cụm nút góc trên bên phải, khi tương tác với nút thứ hai từ phải sáng trái thì một bảng chọn mở ra cho phép truy cập trang thêm thiết bị, thêm phòng, thêm nhà mới (Hình C).

**Màn hình thêm công tắc:**

**A screenshot of a phone

Description automatically generated**

Hình 4.7 Màn hình thêm công tắc

Màn hình công tắc được xây dựng theo bảng thiết kế đồng thời có thêm nhiều cải tiến hỗ trợ người dùng trong nhiều điều kiện mạng wifi khác nhau gồm việc vô hiệu hoá nút kết nối khi chưa nhập đầy đủ thông tin, thông tin không hợp lệ hoặc không thể kết nối đến công tắc tự động thì việc điền thông tin công tắc sẽ được che đi, việc kết quả ứng dụng tại màn hình này vượt mức thiết kế khi trong quá trình xây dựng ứng dụng xuất hiện các lỗi chủ quan khi thao tác với ứng dụng.

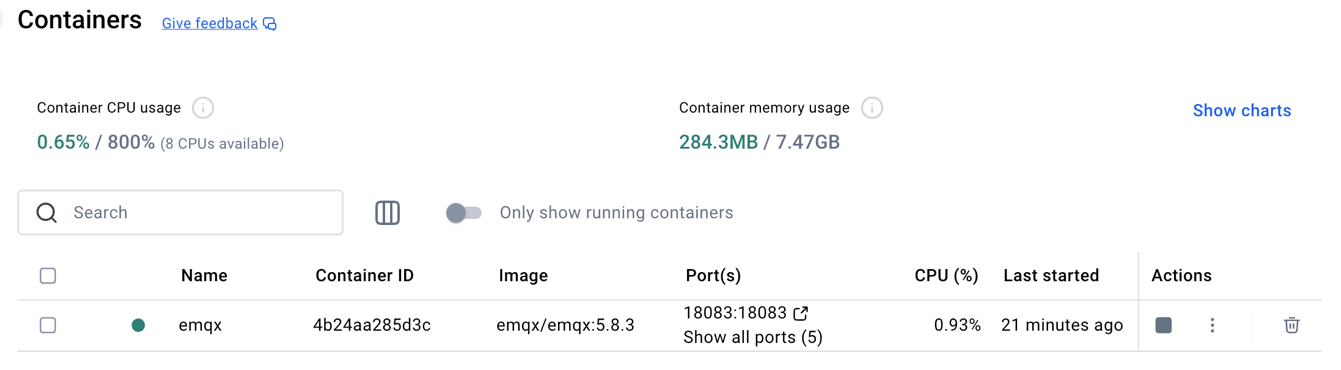
### Logic xử lý

Logic xử lý dữ liệu người dùng đảm bảo tuân thủ chặt chẽ sơ đồ hoạt động đã thiết kế, thậm chí còn hơn thế nữa khi các bước xử lý được chia nhỏ, được quản lý với nhiều hook khác nhau. Điều này đã làm mềm dẻo logic hơn trong mô hình thiết kế, qua đó cũng dễ dàng xử lý bảo mật và kiểm soát tốt hơn khi việc sử dụng token được chia nhỏ ra thành access-token và refresh-token thay vì chỉ sử dụng một token như trong sơ đồ hoạt động đã thiết kế.

## MQTT Broker

### Triển khai

Triển khai MQTT Broker thành công với Docker Container, cấu hình theo yêu cầu từ tài liệu của EMQX:



Hình 4.8 MQTT Broker chạy thành công trên Docker container

Truy cập thành công trang quản trị MQTT Broker bằng trình duyệt qua cổng mặc định 18083:

A screenshot of a computer

Description automatically generated

Hình 4.9 Giao diện trang đăng nhập trang quản lý MQTT Broker

### Cấu hình và hoạt động

Tạo thành công các tài khoản xác thực với MQTT Broker để sử dụng cho các đối tượng trong có trong hệ thống:

A screenshot of a computer

Description automatically generated

Hình 4.10 Danh sách người dùng xác thực với MQTT Broker

Các đối tượng trong hệ thống kết nối đến MQTT Broker thành công, đăng ký các chủ đề thành công, gửi và nhận các tin nhắn giữa các đối tượng trong hệ thống thành công và được giám sát rõ ràng qua trang quản trị.

A screenshot of a computer

Description automatically generated

Hình 4.11 Trang thông tin các kết nối và chủ đề của MQTT Broker

A screenshot of a computer

Description automatically generated

Hình 4.12 Trang danh sách các kết nối của MQTT Broker

# KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

## Kết luận

Qua quá trình nghiên cứu tổng thể và nghiên cứu cụ thể từng đối, lựa chọn các giải pháp, thiết kế hệ thống đảm bảo theo các yêu cầu đặt ra và yêu cầu thực tế từ cuộc sống, dự án “Xây dựng hệ thống điều khiển công tắc từ xa cho thiết bị điện dân dụng” đã đạt được yêu cầu theo thiết kế và của một hệ thống điều khiển thực tế ở quy mô nhỏ.

Việc nghiên cứu và xây dựng hệ thống giao tiếp giữa server và thiết bị IOT (công tắc) gặp rất nhiều trở ngại khi sử dụng các giao thức truyền thống (HTTP, WS) như về giới hạn tài nguyên phần cứng (ở thiết bị IOT) trên mức chi phí cho việc xây dựng thiết bị. Khi nghiên cứu và ứng dụng thành công MQTT cho giao tiếp trung gian đã mở ra một bước ngoặt mới cho việc giao tiếp giữa chúng, xoá bỏ được rào cản về mức chi phí đầu tư cho thiết bị IOT cao vì giao thức MQTT là một giao thức nhẹ - nhanh – hiệu quả. Trước đó một phần cứng giá rẻ trên thị trường phải nặng nhọc trong việc sử dụng HTTP thì giờ đây rất linh hoạt và đủ sức sử dụng giao thức MQTT.

Về mặt công nghệ thì việc ứng dụng mô hình microservices cho server là một lợi ích cao về mặt sản phẩm và giá trị cao trong công việc nghiên cứu công nghệ phần mềm. Đây là công nghệ không mới nhưng đang dần thu hút được nhiều sự quan tâm trong giới công nghệ phần mềm khi số lượng người dùng internet và các dịch vụ internet trở nên khổng lồ.

Đây là một đồ án không chỉ có giá trị về mặt thực tiễn mà còn có giá trị về mặt nghiên cứu công nghệ phần mềm khi nó được ứng dụng các công nghệ tiên tiến, công nghệ được quan tâm nhiều. Về mặt thực tiễn thì nó cho ra một sản phẩm có tính ứng dụng cao giúp đơn giản hoá các công việc của cá nhân, cho hộ gia đình nhỏ, công xưởng nhỏ một cách đột phá.

## Hướng phát triển

Dự án này đơn thuần chỉ là tạo ra một hệ thống nền tảng cho việc sử dụng ứng dụng trên các thiết bị di động Android để điều khiển một phần cứng có chức năng như một công tắc qua internet. Vì thế phạm vi mở rộng của dự án này còn rất nhiều tiềm năng phát triển có thể kể đến chia sẻ việc quản lý – giám sát công tắc, xem lịch sử hoạt động, ngoài công tắc có thể thêm nhiều thiết bị điện dân dụng khác như ổ cắm, quản lý chất lượng không khí, cảnh báo, ….

Việc nâng cấp công nghệ cho việc thiết lập kết nối giữa ứng dụng di động với công tắc là điều đáng cân nhắc khi giá thành của các linh kiện IOT ngày càng rẻ và phổ biến. Có thể sử dụng các module hỗ trợ Bluetooth hoặc sử dụng công nghệ Smart Config để thực hiện phần này.

**TÀI LIỆU THAM KHẢO**

|  |  |
| --- | --- |
| [1] | VTV, "Những con số ấn tượng sau 25 năm Internet tại Việt Nam," 07 12 2022. [Online]. Available: https://vtv.vn/cong-nghe/nhung-con-so-an-tuong-sau-25-nam-internet-tai-viet-nam-20221207181529187.htm. |
| [2] | Shenzhen Ai-Thinker Technology Co., Ltd, "esp-01 product specification," 2017, 2017. [Online]. Available: https://docs.ai-thinker.com/\_media/esp8266/docs/esp-01\_product\_specification\_en.pdf. |
| [3] | CTY TNHH Kỹ thuật Công nghệ Tín Việt, "thietbihienthi," [Online]. Available: https://thietbihienthi.com/relay-la-gi/. |
| [4] | A. Monolithic, "AMS1117-3.3," [Online]. Available: http://www.advanced-monolithic.com/pdf/ds1117.pdf. |
| [5] | Hi-Link, "Nguồn Hi-link," [Online]. Available: https://www.lcsc.com/datasheet/lcsc\_datasheet\_1909111105\_HI-LINK-HLK-PM24\_C399250.pdf. |
| [6] | EMQX, "MQTT Protocol Explained," [Online]. Available: https://www.emqx.com/en/blog/the-easiest-guide-to-getting-started-with-mqtt. |