A white rectangular frame with black border

Description automatically generated**TRƯỜNG ĐẠI HỌC BÌNH DƯƠNG**

**KHOA CÔNG NGHỆ THÔNG TIN, ROBOT**

**VÀ TRÍ TUỆ NHÂN TẠO**

**---oOo---**

****

**TIỂU LUẬN MÔN HỌC**

**CÔNG NGHỆ IOT**

**Tên đề tài:**

**TRIỂN KHAI HỆ THỐNG IOT VỚI PLATFORM OPENREMOTE**

|  |  |
| --- | --- |
| Giảng viên hướng dẫn: | **Lê Duy Hùng** |
| Nhóm sinh viên thực hiện: | **Nguyễn Quốc Duy Khang - 22050006**  **Bùi Công Nghĩa - 22050021** |
|  |  |

**Bình Dương 2024**

**TRƯỜNG ĐẠI HỌC BÌNH DƯƠNG**

**KHOA CÔNG NGHỆ THÔNG TIN, ROBOT**

**VÀ TRÍ TUỆ NHÂN TẠO**

**---oOo---**

****

**TIỂU LUẬN MÔN HỌC**

**CÔNG NGHỆ IOT**

**Tên đề tài:**

**TRIỂN KHAI HỆ THỐNG IOT VỚI PLATFORM OPENREMOTE**

|  |  |
| --- | --- |
| Giảng viên hướng dẫn: | **Lê Duy Hùng** |
| Nhóm sinh viên thực hiện: | **Nguyễn Quốc Duy Khang – 22050006**  **Bùi Công Nghĩa - 22050021** |

**Bình Dương 2024**

**NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN**

**GIẢNG VIÊN HƯỚNG DẪN**

**(Ký và ghi rõ họ tên)**

MỤC LỤC

[MỤC LỤC iv](#_Toc175515405)

[LỜI CẢM ƠN vi](#_Toc175515406)

[LỜI MỞ ĐẦU viii](#_Toc175515407)

[CHƯƠNG I: GIỚI THIỆU ĐỀ TÀI 1](#_Toc175515408)

[I. GIỚI THIỆU 1](#_Toc175515409)

[1. THÔNG TIN CHUNG 1](#_Toc175515410)

[2. HỆ THỐNG TRIỂN KHAI 1](#_Toc175515411)

[3. THIẾT BỊ SỬ DỤNG 1](#_Toc175515412)

[4. GIỚI THIỆU VỀ PLATFORM OPENREMOTE 2](#_Toc175515413)

[II. KẾ HOẠCH THỰC HIỆN 3](#_Toc175515414)

[CHƯƠNG II: TỔNG QUAN VỀ HỆ THỐNG 4](#_Toc175515415)

[I. GIỚI THIỆU HỆ THỐNG: 4](#_Toc175515416)

[II. MÔ HÌNH KIẾN TRÚC VÀ CHỨC NĂNG 4](#_Toc175515417)

[1. MÔ HÌNH KIẾN TRÚC 4](#_Toc175515418)

[2. MÔ HÌNH HOẠT ĐỘNG 5](#_Toc175515419)

[III. SƠ ĐỒ KẾT NỐI 6](#_Toc175515420)

[CHƯƠNG III: KẾT QUẢ TRIỂN KHAI 7](#_Toc175515421)

[I. ĐÁNH GIÁ KẾT QUẢ ĐẠT ĐƯỢC 7](#_Toc175515422)

[II. SO SÁNH VỚI HỆ THỐNG KHÁC 7](#_Toc175515423)

[1. DOCKER-BASED VS. SERVERLESS: 7](#_Toc175515424)

[2. OPENREMOTE PLATFORM VS. SERVERLESS PLATFORM: 8](#_Toc175515425)

[III. TRIỂN KHAI HỆ THỐNG 9](#_Toc175515426)

[CHƯƠNG IV: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 19](#_Toc175515427)

[I. KẾT LUẬN 19](#_Toc175515428)

[1. KẾT QUẢ ĐỐI CHIẾU VỚI MỤC TIÊU 19](#_Toc175515429)

[2. CÁC HẠN CHẾ 19](#_Toc175515430)

[II. HƯỚNG PHÁT TRIỂN 20](#_Toc175515431)

[PHỤ LỤC 21](#_Toc175515432)

[TÀI LIỆU THAM KHẢO 27](#_Toc175515433)

**MỤC LỤC HÌNH**

[Hình 1. Sơ đồ kết nối DHT11 6](#_Toc175505509)

[Hình 2. Sơ đồ kết nối đèn LED 6](#_Toc175505510)

[Hình 3. Sao chép đường link phía trên để lấy URL 9](#_Toc175505511)

[Hình 4. Tải và sử dụng Github Desktop để clone file về máy 9](#_Toc175505512)

[Hình 5. Mở file đã clone về và mở powershell hoặc cmd 10](#_Toc175505513)

[Hình 6. Nhập lệnh docker compose pull 10](#_Toc175505514)

[Hình 7. Nhập lệnh docker compose -p openremote up để chạy mạng local 11](#_Toc175505515)

[Hình 8. Giao diện đăng nhập của OpenRemote 12](#_Toc175505516)

[Hình 9. Giao diện chính của OpenRemote 12](#_Toc175505517)

[Hình 10. Thêm Agent 13](#_Toc175505518)

[Hình 11. Thêm Asset (Thing Asset) 13](#_Toc175505519)

[Hình 12. Thêm Asset (Light Asset) 14](#_Toc175505520)

[Hình 13. Tùy chỉnh thuộc tính trong Thing Asset 14](#_Toc175505521)

[Hình 14. Thêm vào trong 3 thuộc tính như sau 15](#_Toc175505522)

[Hình 15. Light Asset 15](#_Toc175505523)

[Hình 16. Sử dụng MQTT Explorer để test truy cập 16](#_Toc175505524)

[Hình 17. ID của thiết bị nhận Topic được Publish 16](#_Toc175505525)

[Hình 18. Thay đổi ID đích cho thiết bị muốn Publish 17](#_Toc175505526)

[Hình 19. Publish dữ liệu giả lập từ broker 17](#_Toc175505527)

[Hình 20. Cập nhật dữ liệu sau khi publish 18](#_Toc175505528)

# LỜI CẢM ƠN

Với lòng biết ơn sâu sắc và tình cảm chân thành nhất, nhóm em muốn bày tỏ lòng biết ơn đến thầy về những bài giảng và cũng như sự giúp đỡ của thầy trong suốt quá trình nghiên cứu và hoàn thiện đề tài này.

Nhóm em xin gửi lời cảm ơn chân thành nhất đến thầy, Lê Duy Hùng, người đã dành thời gian và công sức để hướng dẫn và giúp đỡ để em có thể hoàn thành đề tài tiểu luận này một cách tốt nhất.

Trong quá trình thực hiện đề tài, em đã cố gắng hết sức và dồn hết tâm huyết, tuy nhiên, vẫn không tránh khỏi những thiếu sót và hạn chế. Do đó, em mong nhận được sự góp ý và đánh giá công tâm từ thầy để đề tài của em được hoàn thiện và cải thiện hơn.

**LỜI CAM ĐOAN**

Chúng em, nhóm sinh viên thực hiện bài tiểu luận này, xin cam đoan rằng tất cả nội dung nghiên cứu, phân tích và kết luận được trình bày trong bài đều là kết quả độc lập của nhóm. Đã xem và tham khảo và có trích dẫn đầy đủ các nguồn thông tin theo đúng quy định.

Bình Dương, ngày.... tháng.... năm....

**Chữ ký từng thành viên**

# LỜI MỞ ĐẦU

Trong kỷ nguyên số, Internet of Things (IoT) đã trở thành một xu hướng công nghệ không thể thiếu, biến đổi cách chúng ta sống và làm việc. Việc ứng dụng IoT vào các thiết bị thông minh, như cảm biến nhiệt độ và độ ẩm, đã mở ra nhiều tiềm năng mới trong việc tự động hóa và tối ưu hóa các quá trình trong cuộc sống.

Nghiên cứu này tập trung vào việc thử nghiệm xây dựng một hệ thống giám sát nhiệt độ và độ ẩm dựa trên nền tảng mã nguồn mở OpenRemote, nhằm khám phá khả năng ứng dụng của IoT trong việc quản lý môi trường sống một cách hiệu quả.

# CHƯƠNG I: GIỚI THIỆU ĐỀ TÀI

## I. GIỚI THIỆU

### 1. THÔNG TIN CHUNG

* **Tên đề tài:** Hiển thị Nhiệt độ - Độ ẩm và điều khiển đèn LED thông qua Platform OpenRemote
* **Danh sách thành viên của nhóm:**
  + Nguyễn Quốc Duy Khang – MSSV: 22050006
  + Bùi Công Nghĩa – MSSV: 22050021
* **Ngày hoàn thành:** 26/8/2024
* **Link Github:** [khangksp/OpenRemote (github.com)](https://github.com/khangksp/OpenRemote)

**2. HỆ THỐNG TRIỂN KHAI**

* **Arduino IDE:** Dùng để viết và nạp code vào thiết bị.
* **Visual Studio Code:** Môi trường phát triển mã nguồn cho các thành phần liên quan như server, giao diện web.
* **Docker Window:** Đóng gói các dịch vụ server, MQTT broker trong các container Docker để đảm bảo tính nhất quán và dễ triển khai.
* **OpenRemote:** Nền tảng mã nguồn mở cho việc quản lý và giám sát các thiết bị IoT

### 3. THIẾT BỊ SỬ DỤNG

* **ESP32 và ESP8266:** Module Wi-Fi được sử dụng để kết nối với các cảm biến DHT11 và gửi dữ liệu nhiệt độ, độ ẩm lên nền tảng OpenRemote thông qua giao thức MQTT.
* **DHT11:** Cảm biến nhiệt độ và độ ẩm được sử dụng để thu thập dữ liệu môi trường.
* **LED:** Sử dụng để biểu thị trạng thái (ví dụ: bật/tắt) dựa trên dữ liệu nhận từ nền tảng OpenRemote.
* **Màn hình LCD:** Sử dụng để hiển thị trực tiếp các thông tin về nhiệt độ và độ ẩm từ cảm biến DHT11.

### 4. GIỚI THIỆU VỀ PLATFORM OPENREMOTE

Bắt đầu vào năm 2015, OpenRemote là một nền tảng mã nguồn mở mạnh mẽ và linh hoạt, được thiết kế để quản lý và kiểm soát các hệ thống IoT (Internet of Things).   
Với OpenRemote, người dùng có thể dễ dàng tạo, triển khai, xây dựng, phát triển và quản lý các ứng dụng IoT từ đơn giản đến phức tạp, phục vụ nhiều mục đích khác nhau như giám sát môi trường, quản lý năng lượng, tự động hóa tòa nhà, và nhiều ứng dụng khác.

**Kiến trúc và Cấu trúc của OpenRemote**

OpenRemote được xây dựng với kiến trúc module, cho phép tích hợp dễ dàng với các thiết bị và hệ thống hiện có. Nền tảng này bao gồm ba thành phần chính:

* **Manager**: Là trung tâm điều khiển, nơi quản lý và cấu hình các thiết bị, luồng dữ liệu và các quy tắc hoạt động. Manager cũng cung cấp giao diện người dùng (UI) để giám sát và kiểm soát hệ thống từ xa.
* **Agent**: Là thành phần trung gian giúp thu thập dữ liệu từ các thiết bị IoT và gửi về cho Manager để xử lý. Agent cũng có thể thực hiện các lệnh điều khiển từ Manager đến các thiết bị.
* **Protocol Adapters**: Cho phép OpenRemote kết nối với nhiều loại thiết bị khác nhau, sử dụng các giao thức tiêu chuẩn như MQTT, HTTP, Modbus, Zigbee, và nhiều giao thức khác.

**Tính năng nổi bật của OpenRemote**

* **Tùy chỉnh linh hoạt**: OpenRemote cho phép người dùng tùy chỉnh giao diện và logic hoạt động theo yêu cầu cụ thể, từ đó tạo ra các giải pháp IoT tối ưu và độc đáo.
* **Khả năng mở rộng**: Với kiến trúc module, OpenRemote có thể dễ dàng mở rộng để quản lý hàng ngàn thiết bị IoT, đáp ứng nhu cầu từ những dự án nhỏ đến các hệ thống quy mô lớn.
* **Hỗ trợ đa nền tảng**: OpenRemote có thể chạy trên nhiều nền tảng khác nhau, từ các thiết bị nhúng nhỏ gọn đến các máy chủ đám mây, giúp người dùng triển khai hệ thống IoT ở bất kỳ đâu.
* **Mã nguồn mở**: Là một nền tảng mã nguồn mở, OpenRemote không chỉ miễn phí mà còn có một cộng đồng phát triển lớn mạnh, cung cấp tài liệu và hỗ trợ kỹ thuật phong phú.

**Lợi ích khi sử dụng OpenRemote**

* **Tiết kiệm chi phí**: Là một nền tảng mã nguồn mở, OpenRemote không yêu cầu người dùng trả phí bản quyền, giúp giảm chi phí đầu tư cho các giải pháp IoT.
* **Tính bảo mật cao**: OpenRemote cho phép quản lý quyền truy cập và đảm bảo an toàn thông tin trong hệ thống IoT.
* **Khả năng tích hợp cao**: Với hỗ trợ đa giao thức, OpenRemote dễ dàng tích hợp với các hệ thống và thiết bị hiện có, giúp tối ưu hóa các giải pháp IoT.

**II. KẾ HOẠCH THỰC HIỆN**

| **Thành viên** | **Công việc được phân công** | **Tỉ lệ đóng góp** |
| --- | --- | --- |
| Bùi Công Nghĩa – MSSV 22050021 | Publish data của DHT11 lên platform thông qua giao thức MQTT | 100% |
| Nguyễn Quốc Duy Khang – MSSV 22050006 | Subscribe, đọc dữ liệu bật tắt từ platform thông qua giao thức MQTT | 100% |

# CHƯƠNG II: TỔNG QUAN VỀ HỆ THỐNG

**I. GIỚI THIỆU HỆ THỐNG:**

Quản lý và giám sát môi trường sống là điều quan trọng. Đặc biệt, theo dõi các yếu tố như nhiệt độ và độ ẩm còn giúp cải thiện chất lượng cuộc sống, hỗ trợ trong nhiều lĩnh vực khác như nông nghiệp, công nghiệp và chăm sóc sức khỏe. Hệ thống giám sát nhiệt độ và độ ẩm này có khả năng theo dõi, lưu trữ và phản hồi thông tin một cách hiệu quả. Sử dụng các thiết bị phần cứng như ESP32, cảm biến DHT11 và màn hình LCD và ứng dụng vào trong đó giao thức MQTT và nền tảng OpenRemote. Với sự kết hợp này, hệ thống có thể thu thập dữ liệu từ môi trường phòng học hoặc nhà cửa, truyền tải dữ liệu một cách nhanh chóng và hiển thị thông tin trực quan cho người dùng, giúp họ có thể đưa ra các quyết định kịp thời và chính xác.

**II. MÔ HÌNH KIẾN TRÚC VÀ CHỨC NĂNG**

1. **MÔ HÌNH KIẾN TRÚC**

Hệ thống giám sát nhiệt độ và độ ẩm được xây dựng dựa trên một kiến trúc phân tầng, bao gồm ba lớp chính: lớp cảm biến, lớp truyền thông và lớp quản lý. Mỗi lớp đóng vai trò quan trọng trong việc đảm bảo hệ thống hoạt động hiệu quả và liên tục.

* **Lớp cảm biến:** Tại lớp này, cảm biến DHT11 sẽ đảm nhận nhiệm vụ thu thập dữ liệu về nhiệt độ và độ ẩm từ môi trường. DHT11 là một cảm biến phổ biến với độ chính xác cao và khả năng tương thích tốt với các vi điều khiển như ESP8266 hoặc ESP32.
* **Lớp truyền thông:** Sau khi dữ liệu được thu thập, ESP32 sẽ đóng vai trò xử lý và truyền tải dữ liệu. Với khả năng kết nối Wi-Fi mạnh mẽ, ESP32 sẽ gửi dữ liệu thông qua giao thức MQTT đến nền tảng OpenRemote. MQTT là giao thức nhẹ, phù hợp cho các ứng dụng IoT, giúp giảm thiểu băng thông và tăng cường tốc độ truyền tải dữ liệu.
* **Lớp quản lý:** Cuối cùng, dữ liệu sẽ được gửi đến nền tảng OpenRemote, nơi nó được lưu trữ, quản lý và hiển thị. OpenRemote cung cấp một giao diện người dùng trực quan, cho phép người dùng theo dõi và kiểm soát các thông số nhiệt độ và độ ẩm theo thời gian thực. Ngoài ra, màn hình LCD kết nối với ESP32 sẽ hiển thị trực tiếp thông tin nhiệt độ và độ ẩm, giúp người dùng quan sát nhanh chóng mà không cần truy cập vào giao diện web.

1. **MÔ HÌNH HOẠT ĐỘNG**

Hệ thống hoạt động theo quy trình tuần tự từ việc thu thập dữ liệu đến khi hiển thị và phản hồi thông tin. Quá trình này có thể được chia thành các bước chính sau:

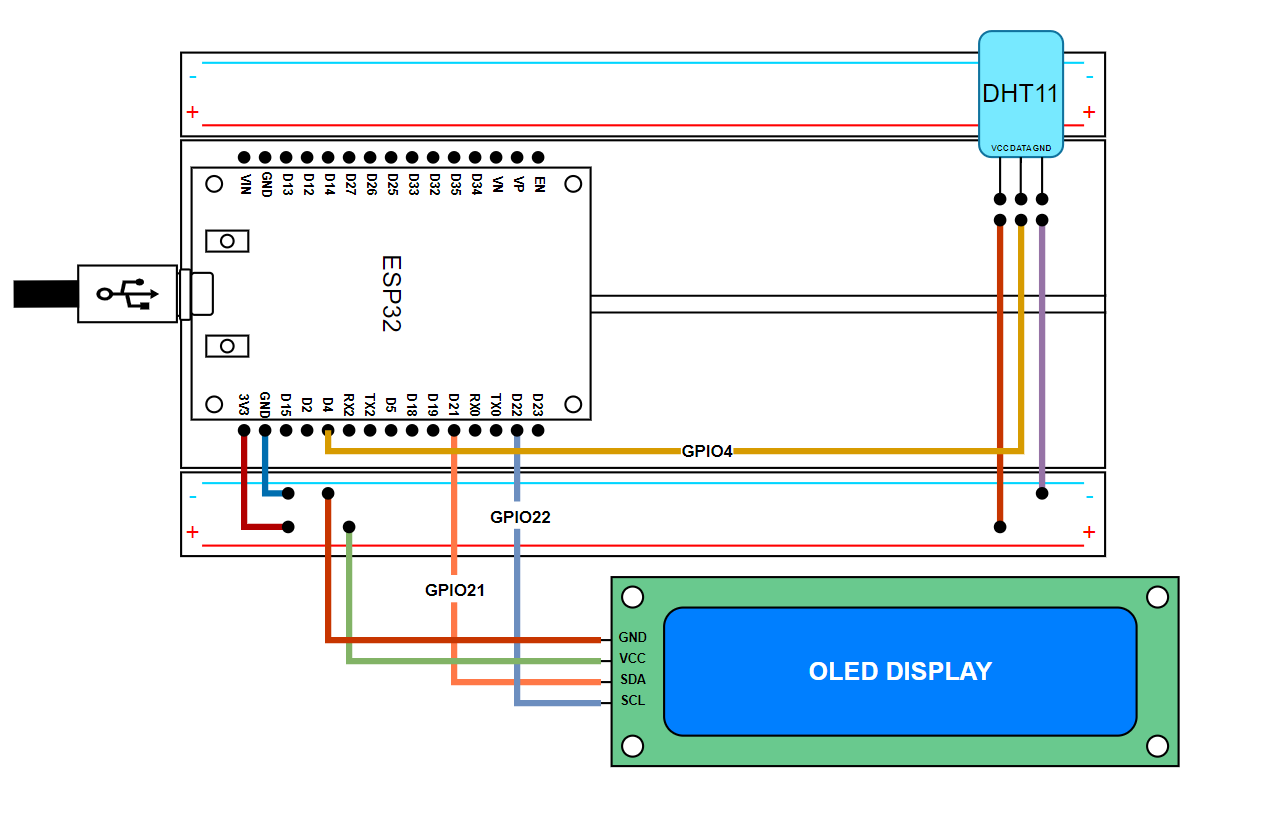
**Với DHT 11:**

1. Thu thập dữ liệu: Cảm biến DHT11 liên tục đo nhiệt độ và độ ẩm của môi trường xung quanh. Các giá trị này được truyền đến ESP32 để xử lý.
2. Truyền tải dữ liệu: ESP32 sau khi nhận được dữ liệu từ cảm biến, sẽ sử dụng giao thức MQTT để truyền dữ liệu này đến nền tảng OpenRemote. Đồng thời, các giá trị này cũng được hiển thị trên màn hình LCD kết nối với ESP32.
3. Quản lý và hiển thị dữ liệu: Tại server, dữ liệu được lưu trữ và quản lý. Người dùng có thể truy cập vào giao diện quản lý của server để xem các thông số nhiệt độ và độ ẩm theo thời gian thực. Nếu có bất kỳ thay đổi nào trong dữ liệu, server sẽ phản hồi ngay lập tức, giúp người dùng nắm bắt tình hình nhanh chóng.

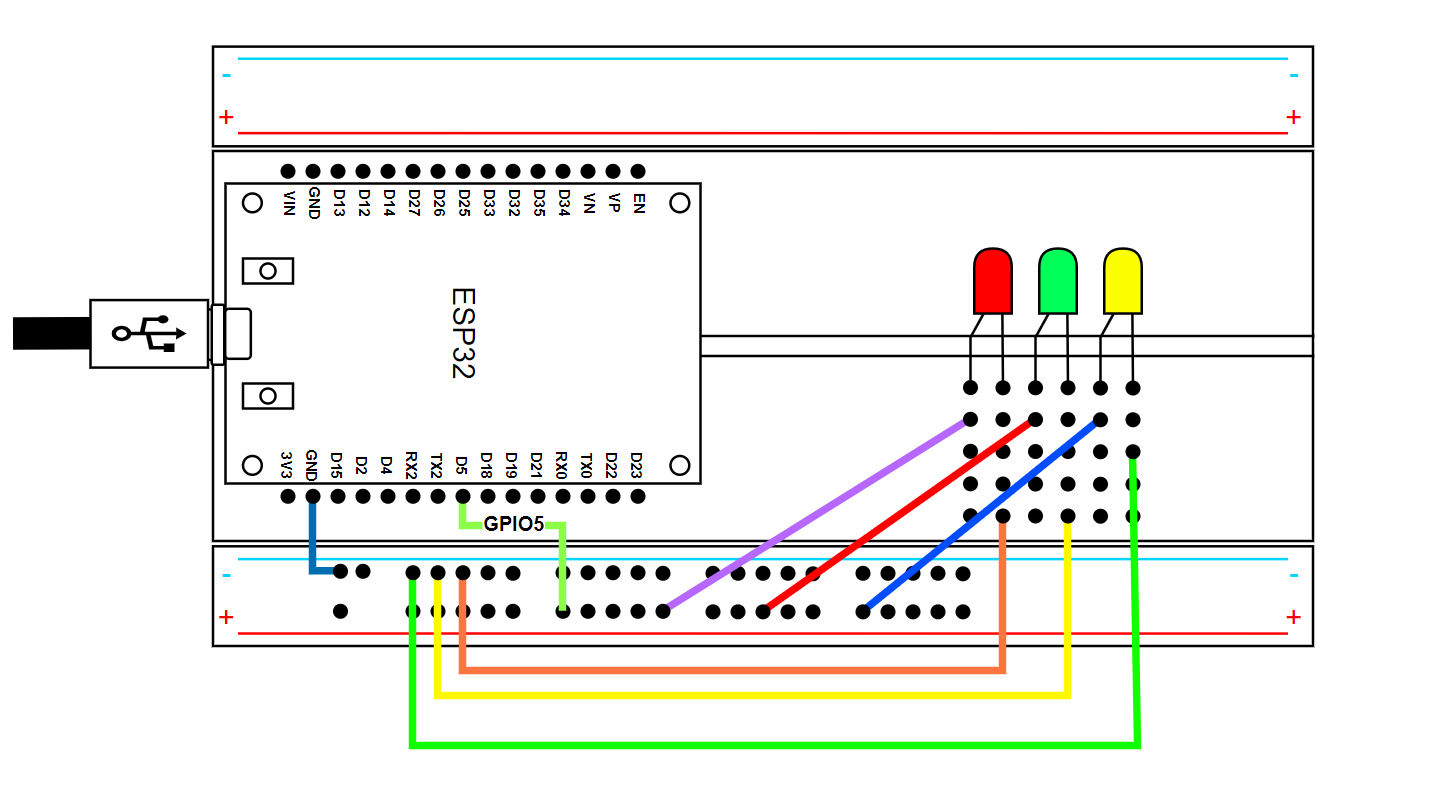
**Với điều khiển LED:**

1. Thu thập dữ liệu: người dùng thao tác bật tắt thông qua hệ thống mạng.
2. Truyền tải dữ liệu: ESP32 sử dụng giao thức MQTT sẽ duy trì kết nối đến server và nhận lệnh từ hệ thống.
3. Quản lý và hiển thị dữ liệu: Tại server, dữ liệu bật tắt được lưu trữ và quản lý, dù cho thiết bị offline thì khi có kết nối mạng vẫn sẽ nhận và thực hiện theo lệnh mới nhất từ server.

**III. SƠ ĐỒ KẾT NỐI**



**Hình 1. Sơ đồ kết nối DHT11**



**Hình 2. Sơ đồ kết nối đèn LED**

**CHƯƠNG III: KẾT QUẢ TRIỂN KHAI**

**I. ĐÁNH GIÁ KẾT QUẢ ĐẠT ĐƯỢC**

Hệ thống giám sát nhiệt độ và độ ẩm đã được triển khai thành công với các thành phần chính bao gồm ESP32, cảm biến DHT11, màn hình LCD, đèn LED và kết nối MQTT đến OpenRemote platform. Dưới đây là một số điểm đáng chú ý trong quá trình triển khai:

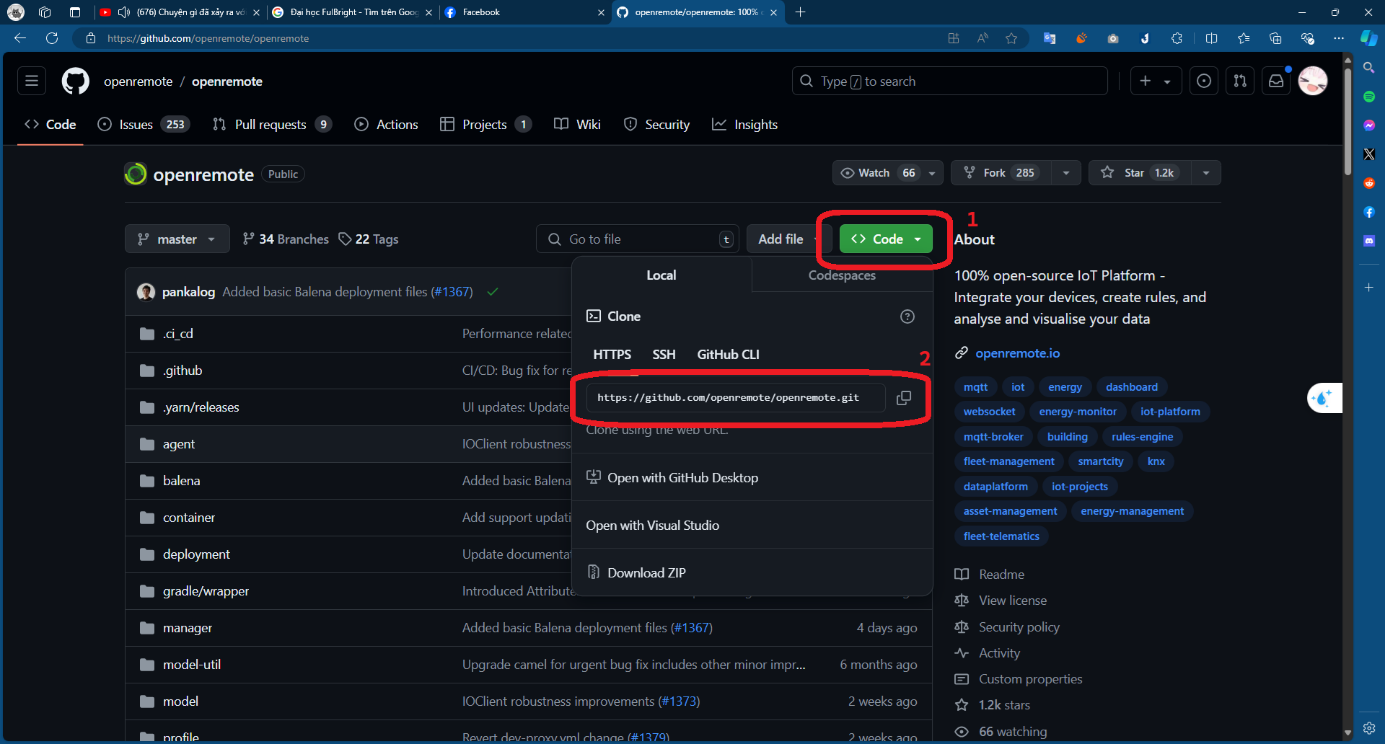
* Hiệu quả hoạt động: Hệ thống hoạt động ổn định, dữ liệu nhiệt độ và độ ẩm được cập nhật và hiển thị trên màn hình LCD một cách chính xác. Đồng thời, dữ liệu này được gửi lên server qua giao thức MQTT và có thể được quản lý và giám sát từ xa.
* Tính tương thích: Các thành phần được sử dụng trong dự án hoàn toàn tương thích với nhau, từ phần cứng (ESP32, DHT11, LCD, LED) cho đến các phần mềm (Arduino IDE, Visual Studio Code, Docker).
* Khả năng mở rộng: Với nền tảng OpenRemote, hệ thống có thể dễ dàng mở rộng thêm các cảm biến hoặc thiết bị khác, cũng như tùy chỉnh để phù hợp với các yêu cầu cụ thể hơn của người sử dụng.

**II. SO SÁNH VỚI HỆ THỐNG KHÁC**

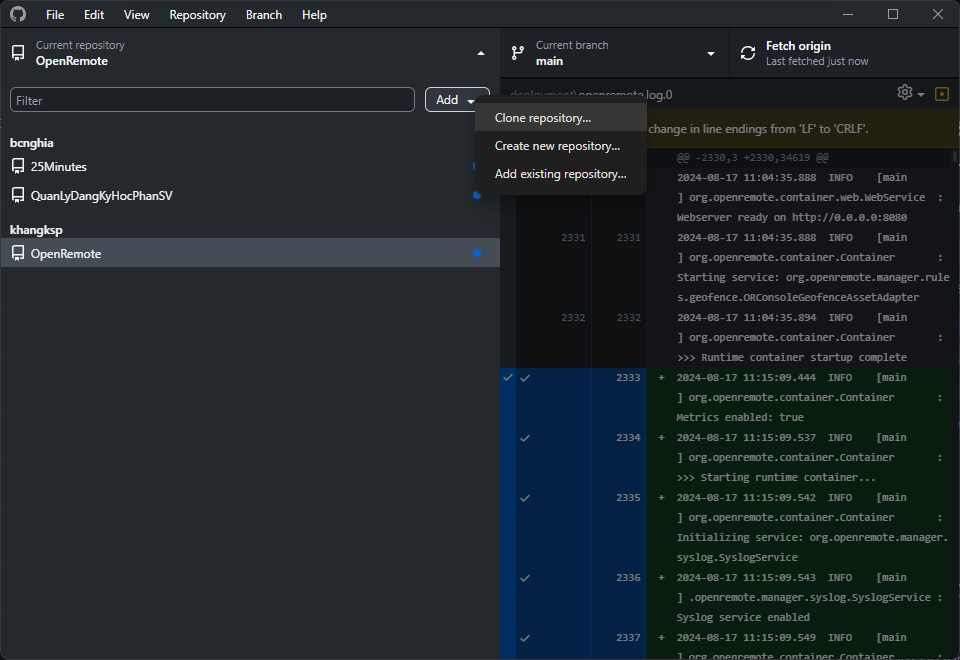
* 1. **DOCKER-BASED VS. SERVERLESS:**
  + Docker-based: Hệ thống hiện tại sử dụng Docker để triển khai OpenRemote. Điều này cho phép dễ dàng quản lý và triển khai các dịch vụ trong các container riêng biệt, đảm bảo tính nhất quán và dễ dàng mở rộng hệ thống.
  + Serverless: Trong khi đó, nếu sử dụng một nền tảng không cần máy chủ (serverless), hệ thống sẽ giảm bớt được khối lượng công việc liên quan đến quản lý server. Tuy nhiên, khả năng kiểm soát và tuỳ chỉnh sẽ bị hạn chế hơn so với việc sử dụng Docker.
  + Ưu điểm của Docker: Docker cung cấp môi trường triển khai linh hoạt và dễ dàng kiểm soát hơn so với nền tảng serverless, đặc biệt trong các dự án yêu cầu sự kiểm soát chặt chẽ về môi trường triển khai và dễ dàng di chuyển giữa các máy chủ khác nhau.
  1. **OPENREMOTE PLATFORM VS. SERVERLESS PLATFORM:**
  + OpenRemote Platform: dễ dàng xây dựng ứng dụng, phần mềm quản lý hay web server riêng, phù hợp tạo các ứng dụng quản lý năng lượng chuyên nghiệp (theo công bố của Platform). Bảng điều khiển Mobile cho phép kết nối tới các dịch vụ của điện thoại: đẩy thông báo, vị trí, ... Cửa sổ thiết kế cho xử lý dữ liệu, thiết lập các quy tắc (Role) dựa trên sự kiện. Hỗ trợ Dashboard chuẩn cho quá trình chuẩn bị, tự động hóa, điều khiển và giám sát ứng dụng cũng như các thành phần WEB UI để xây dựng các ứng dụng khác. Tuy nhiên hiện tại Dashboard của platform vẫn chưa hỗ trợ tùy chỉnh nhiều loại bảng, biểu đồ dữ liệu do tiêu chí của platform chủ yếu tập trung vào các thiết bị về năng lượng. Đồng thời cộng đồng của platform này khá hạn chế và các nguồn tài liệu hướng dẫn chủ yếu từ trang web chính chủ nên gây khó khăn cho người mới tiếp cận.
  + ThingSpeak Platform: cho phép phân tích và trực quan hóa dữ liệu trong MATLAB. Được sử dụng chủ yếu để ghi nhật ký cảm biến, theo dõi vị trí, cảnh báo và phân tích. Không cung cấp khả năng tự xây dựng phần mềm hoặc web server riêng như một giải pháp độc lập, nó chủ yếu được cung cấp như một dịch vụ đám mây bởi MathWorks. Tài liệu hỗ trợ và cộng đồng lớn trên toàn thế giới, có nhiều tài liệu được người dùng chia sẻ kinh nghiệm cá nhân nên rất hữu ích cho việc nghiên cứu, tìm hiểu.

## III. TRIỂN KHAI HỆ THỐNG

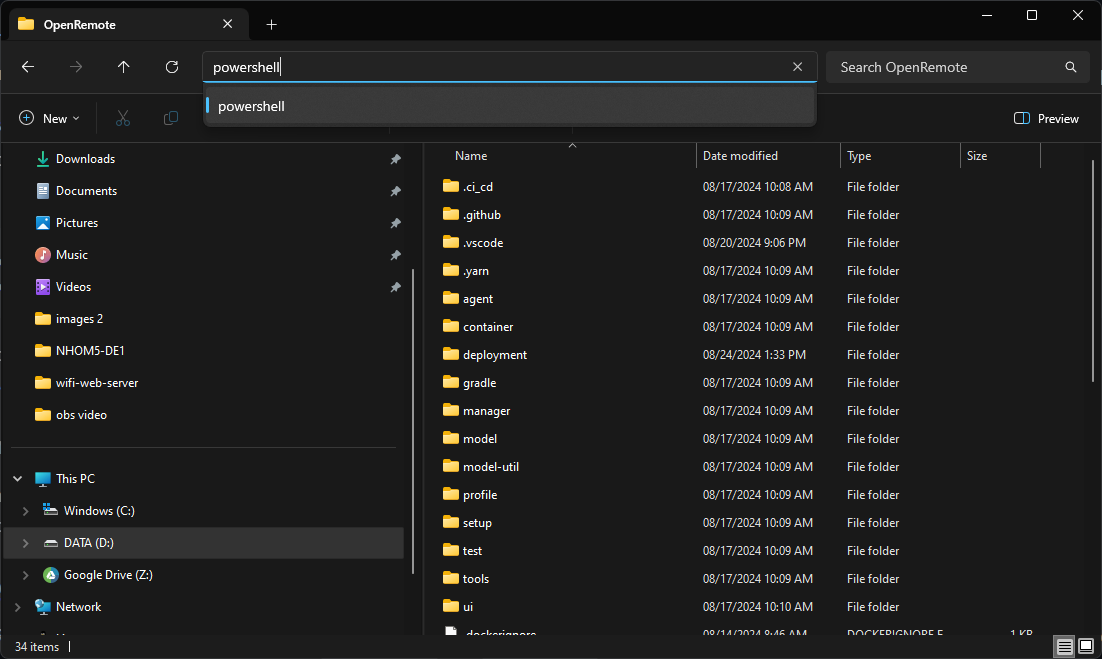
Để sử dụng và triển khai hệ thống trên trên nền tảng mã nguồn mở của OpenRemote, trước tiên phải truy cập vào đường link github [openremote/openremote: 100% open-source IoT Platform - Integrate your devices, create rules, and analyse and visualise your data (github.com)](https://github.com/openremote/openremote) sau đó copy URL và clone file về máy để bắt đầu sử dụng.

****

**Hình 3. Sao chép đường link phía trên để lấy URL**

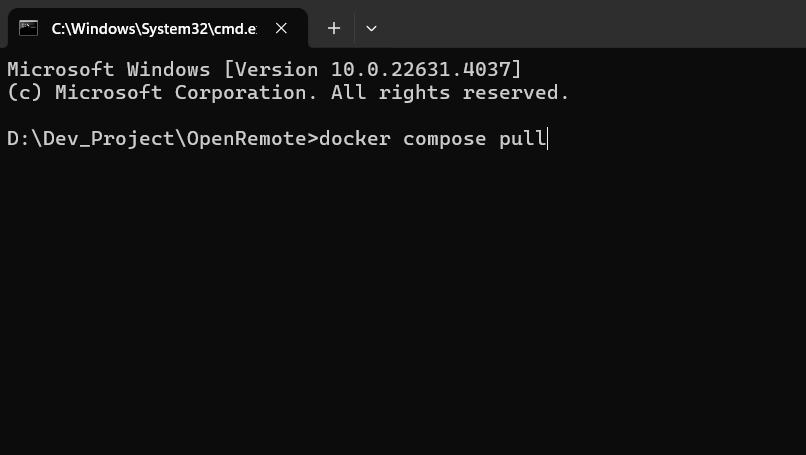


**Hình 4. Tải và sử dụng Github Desktop để clone file về máy**



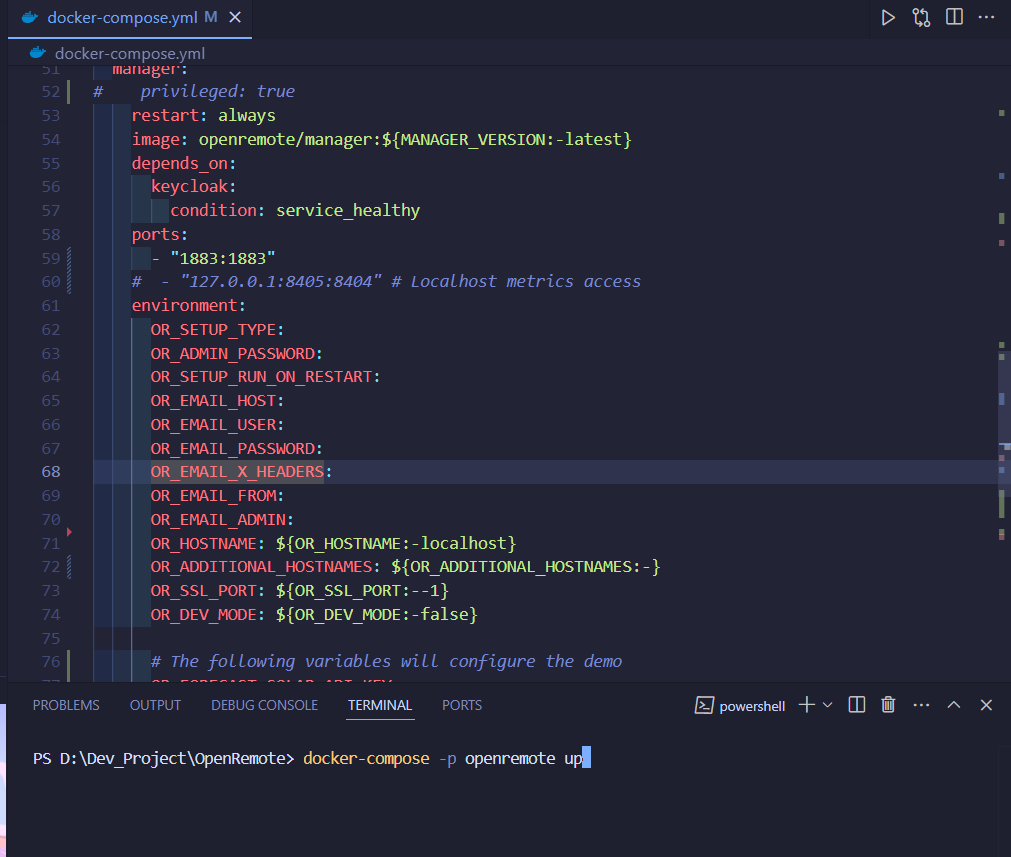
**Hình 5. Mở file đã clone về và mở powershell hoặc cmd**

Truy cập folder chứa dự án → sử dụng lệnh “docker compose pull” để tải về docker các tài nguyên cần thiết



**Hình 6. Nhập lệnh docker compose pull**

Chạy phần mềm docker window → vào terminal và sử dụng lệnh “docker compose -p openremote up “ để chạy dự án trong mạng local

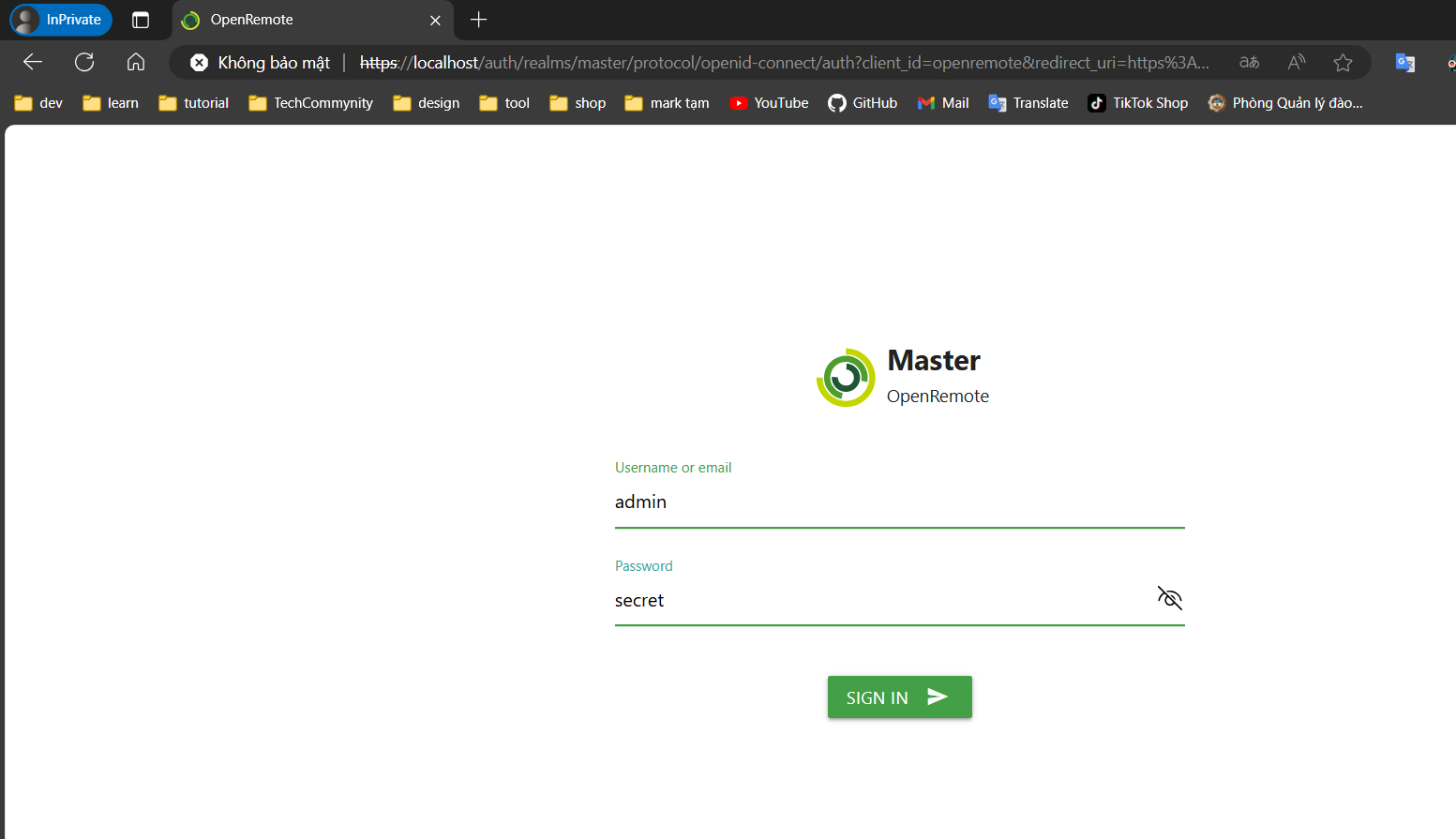


**Hình 7. Nhập lệnh docker compose -p openremote up để chạy mạng local**

Sử dụng tài khoản mặc định được cấp của platform để truy cập vào hệ thống

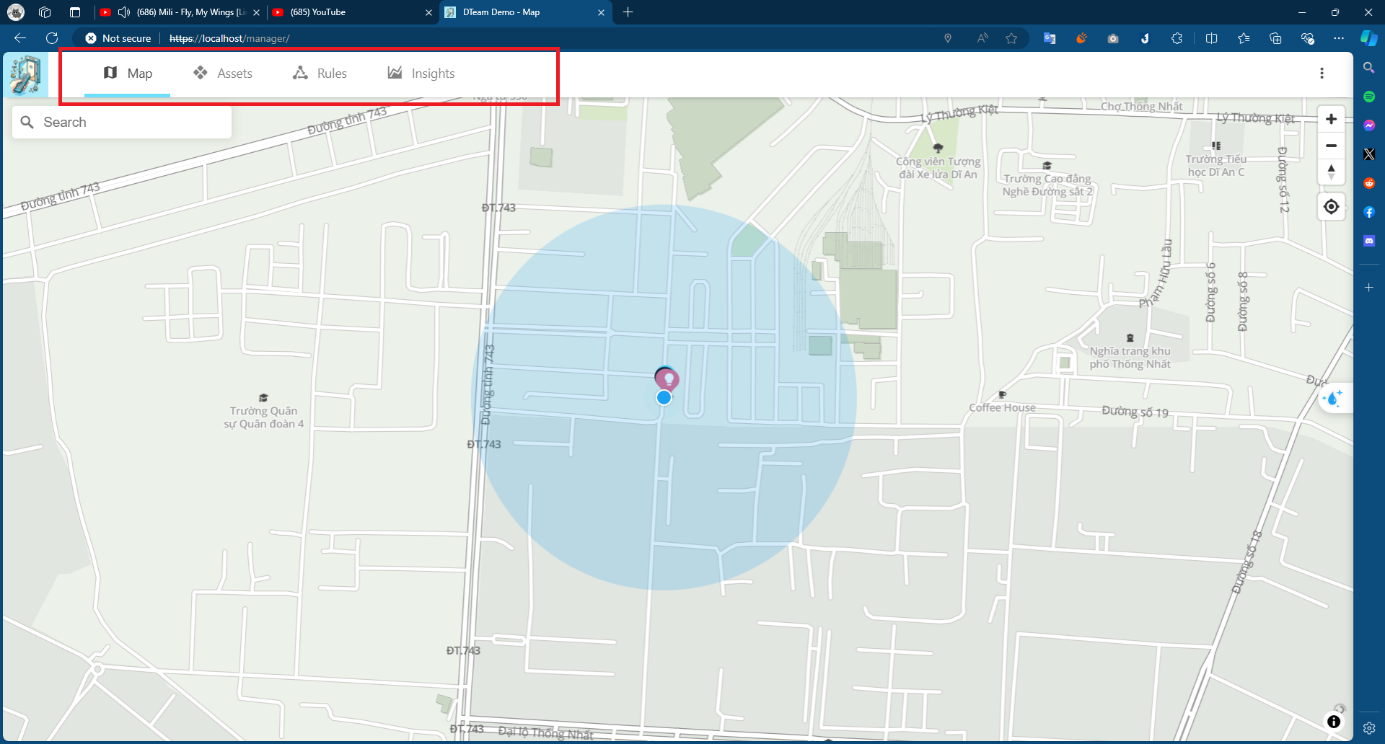
**Username: admin**

**Password: secret**



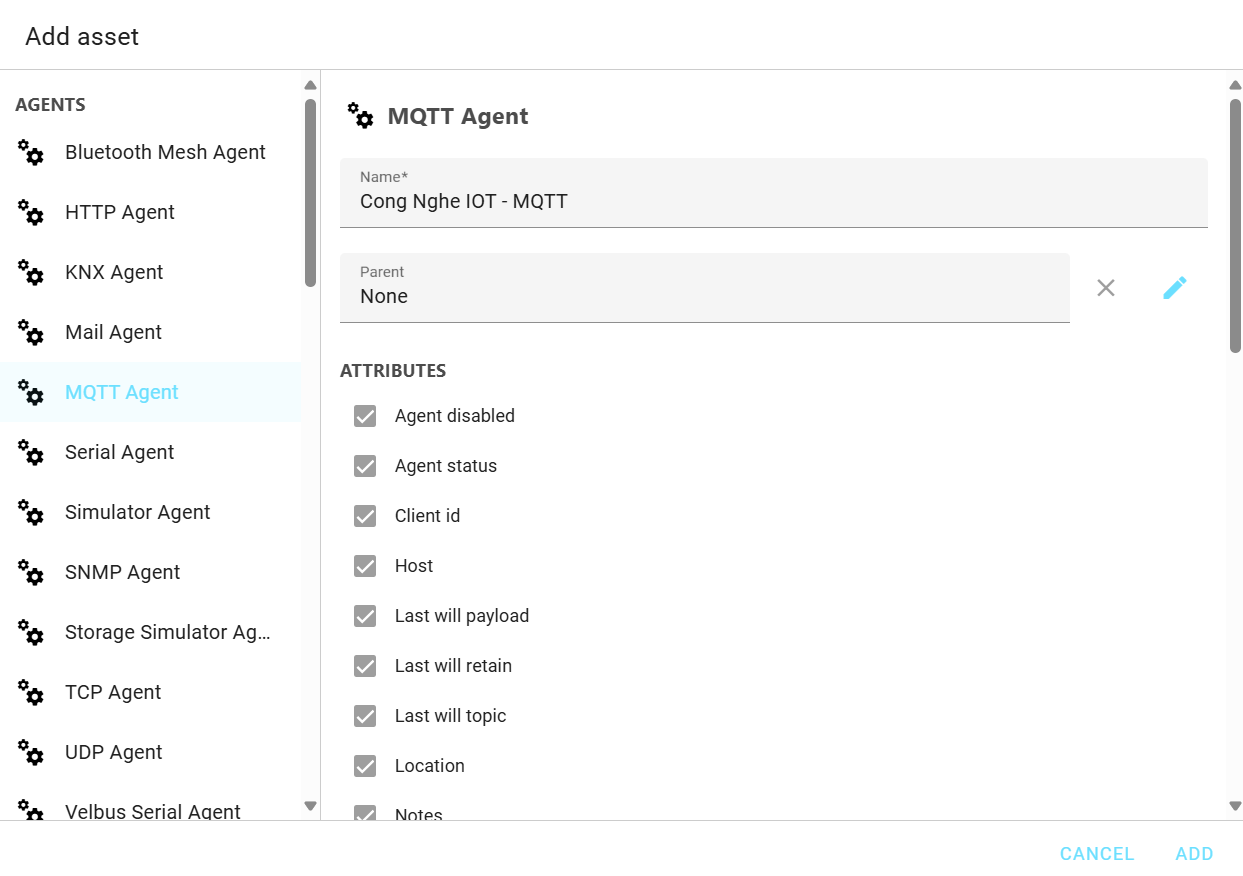
**Hình 8. Giao diện đăng nhập của OpenRemote**

Trang web hiển thị với 4 Tab làm việc chính (giao diện admin)



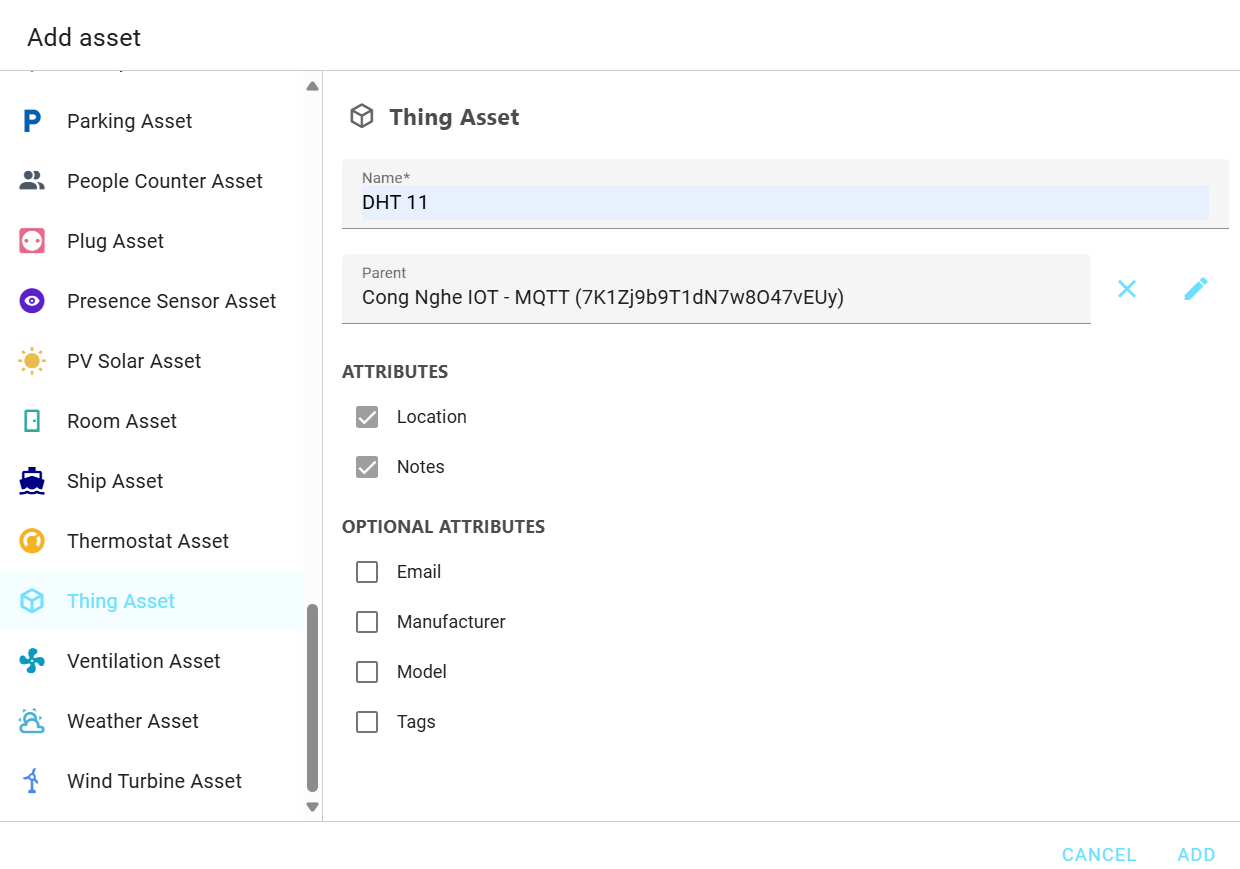
**Hình 9. Giao diện chính của OpenRemote**

Truy cập “Assets” → Click “+” (Add) để thêm thiết bị hoặc đại lý quản lý thiết bị → Click chọn “MQTT Agent” để thêm đại lý quản lý các thiết bị sử dụng giao thức MQTT → Đặt tên cho đại lý và click “Add”

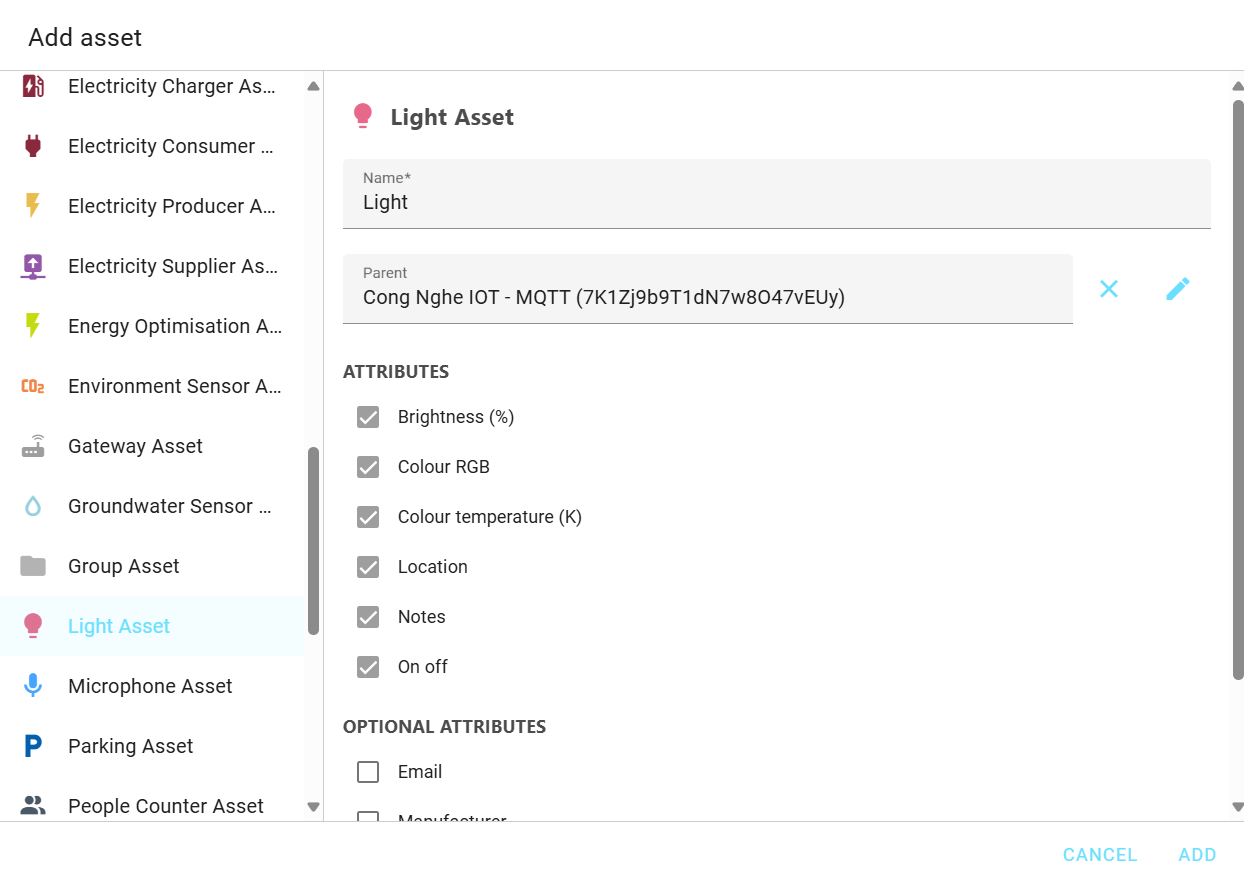


**Hình 10. Thêm Agent**

Trong mục MQTT Agent click “**+**” (add) để thêm thiết bị hoặc đại lý quản lý thiết bị → sau đó click vào “**Thing Asset**” để thêm một thiết bị trống (trắng thông tin) và “**Light Asset**” để thêm 1 thiết bị quản lý bóng đèn → rồi click chọn icon edit (bên cạnh “Parent”) và chọn đại lý vừa tạo → Đặt tên cho thiết bị và click “Add”.

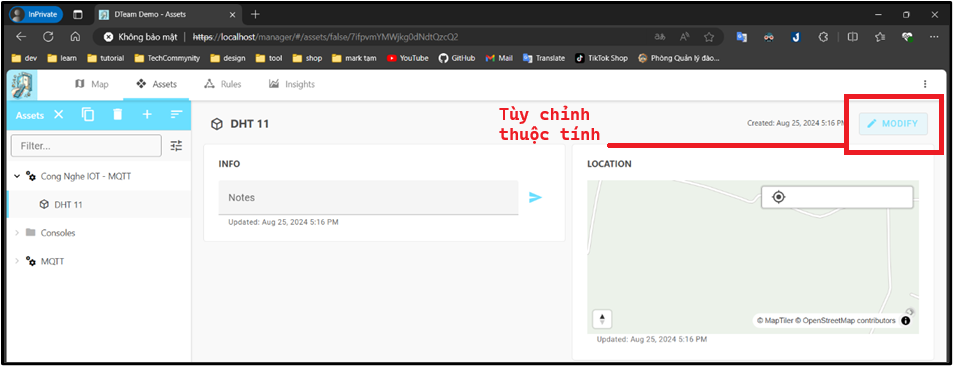


**Hình 11. Thêm Asset (Thing Asset)**



**Hình 12. Thêm Asset (Light Asset)**

Trong trang thông tin thiết bị click “Modify” để tùy chỉnh các thuộc tính



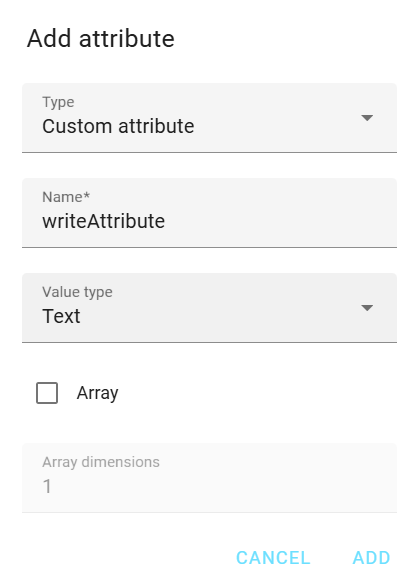
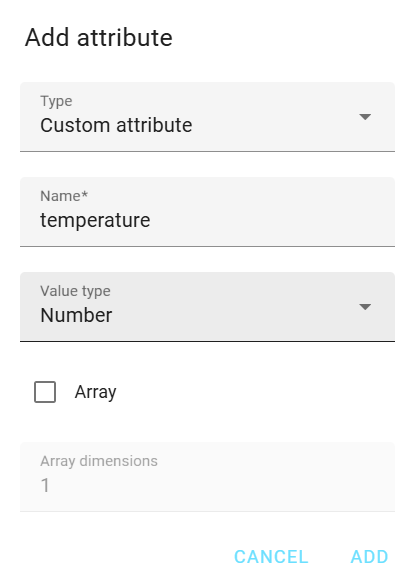
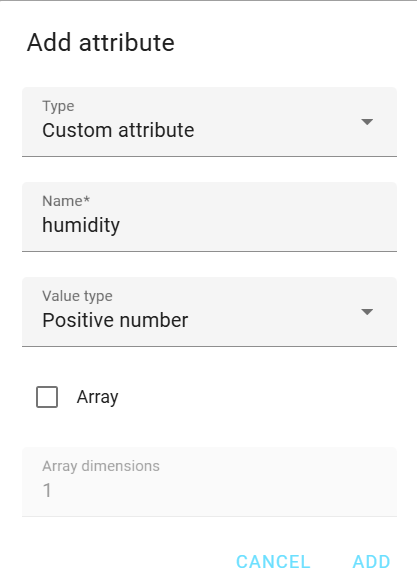
**Hình 13. Tùy chỉnh thuộc tính trong Thing Asset**

Trường hợp DHT11: thiết bị triển khai trong dự án này chỉ cần nhận thông tin của cảm biến và GỬI thông tin (PUBLISH) đã nhận được về SERVER → chỉ cần tạo 3 trường chính để nhận thông tin về nhiệt độ (temperature), độ ẩm (humidity), và trường write để test và nhận lỗi trong môi trường phát triển (writeAttribute)

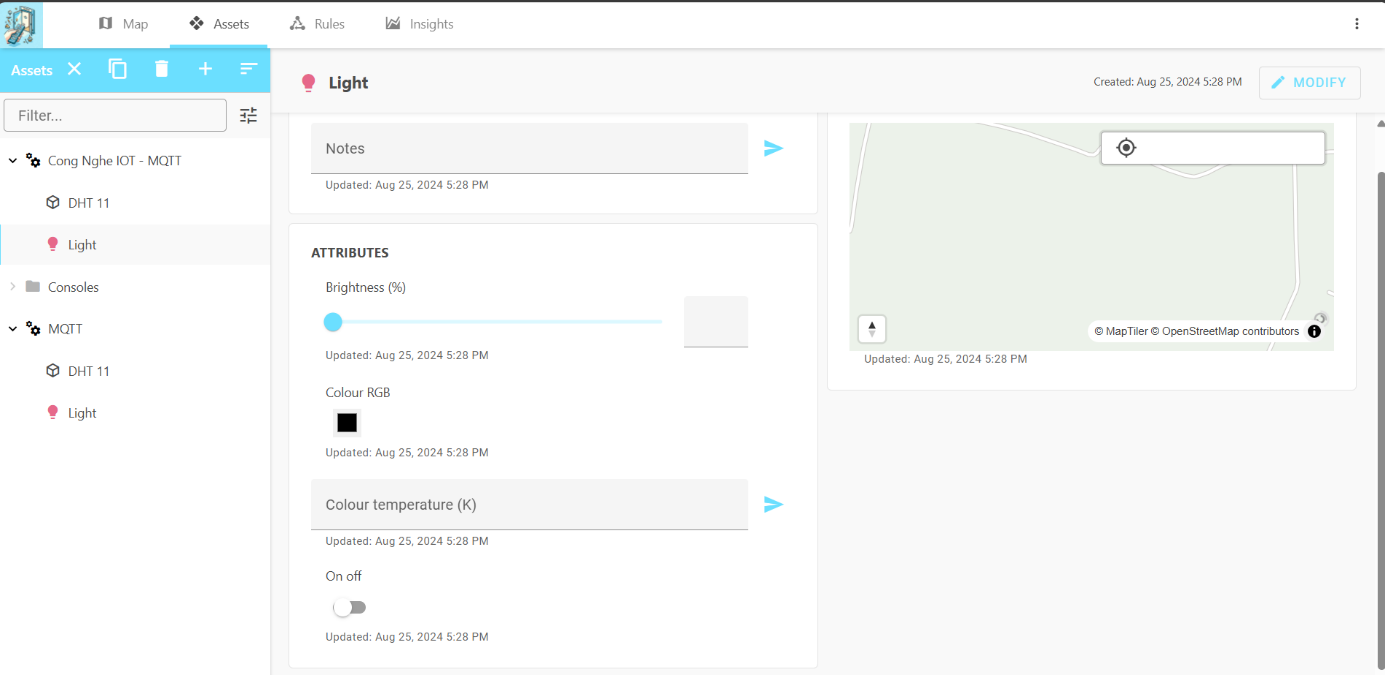
Click “Add Attribute” để thêm các giá trị mới và tùy biến giá trị theo 3 trường đã nêu và nhấn “Save” để lưu.

Trường hợp điều khiển LED: thiết bị triển khai trong dự án này chỉ cần BẬT/TẮ

**Hình 14. Thêm vào trong 3 thuộc tính như sau**



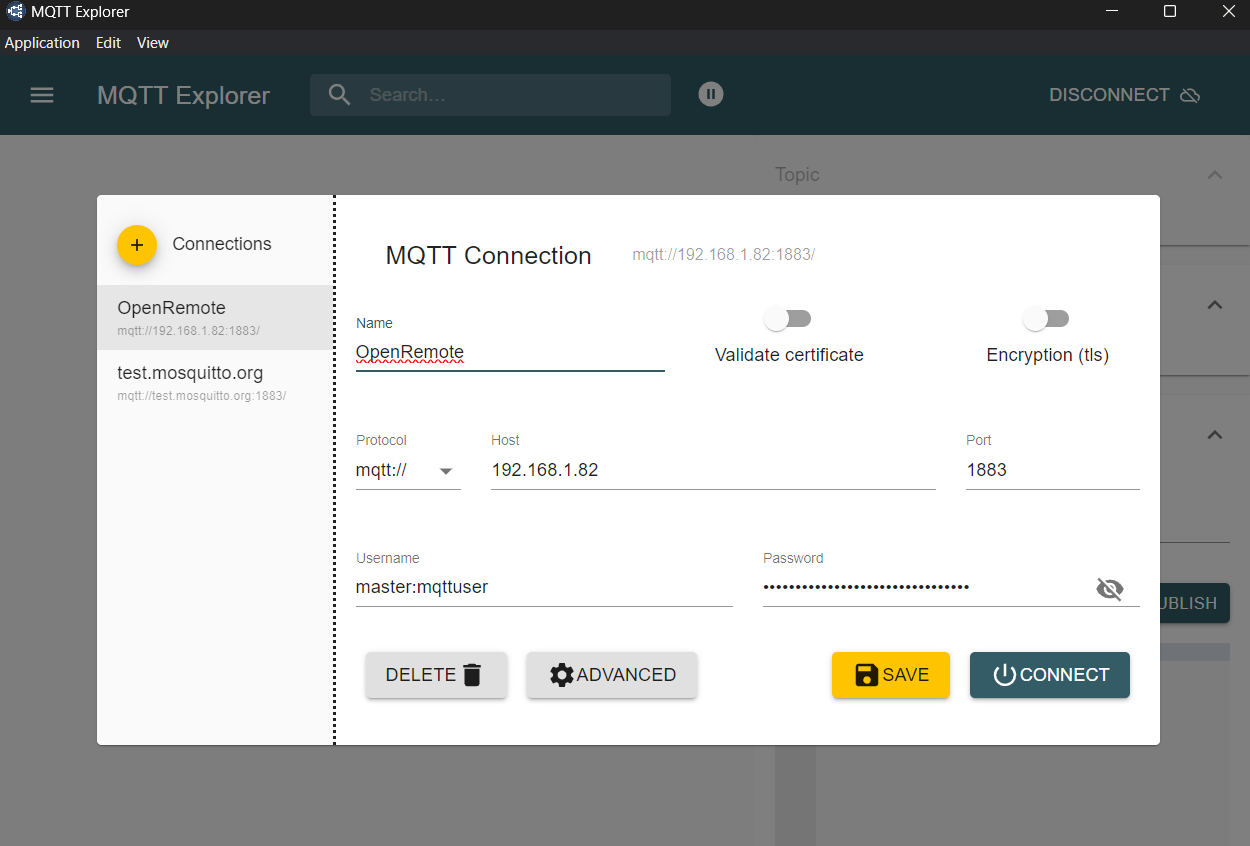
Trường hợp LED: thông qua server (không dùng button vật lý) nên thiết bị vật lý cần ĐỌC thông tin (SUBSCRIBE) từ SERVER và điều khiển đèn led dựa theo thông tin nhận được → “Light Asset” đã hỗ trợ sẵn “OnOff” là một phương thức Subscribe nên không cần làm gì thêm trong trường hợp này.



**Hình 15. Light Asset**

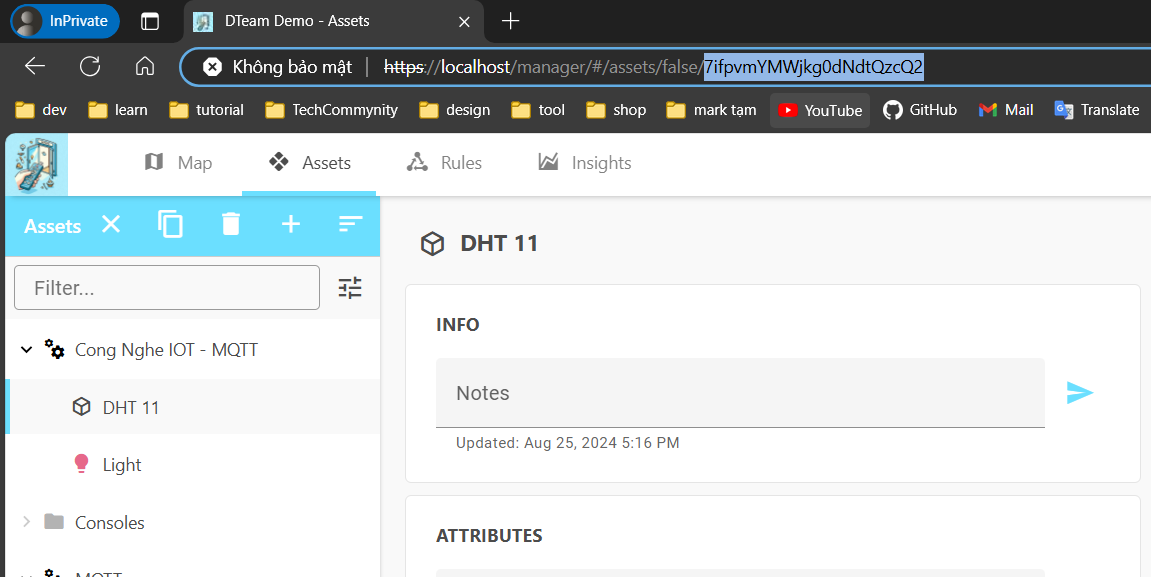
Sử dụng các phần mềm MQTT broker để giả lập kết nối đến server trước khi áp dụng trên thiết bị thật. Trong dự án này nhóm sử dụng MQTT Explorer.

Tạo một connections mới → Đặt tên, host: địa chỉ ip máy server, port: 1883/8883, username, password: nhập thông tin user để kết nối => Click “Connect”



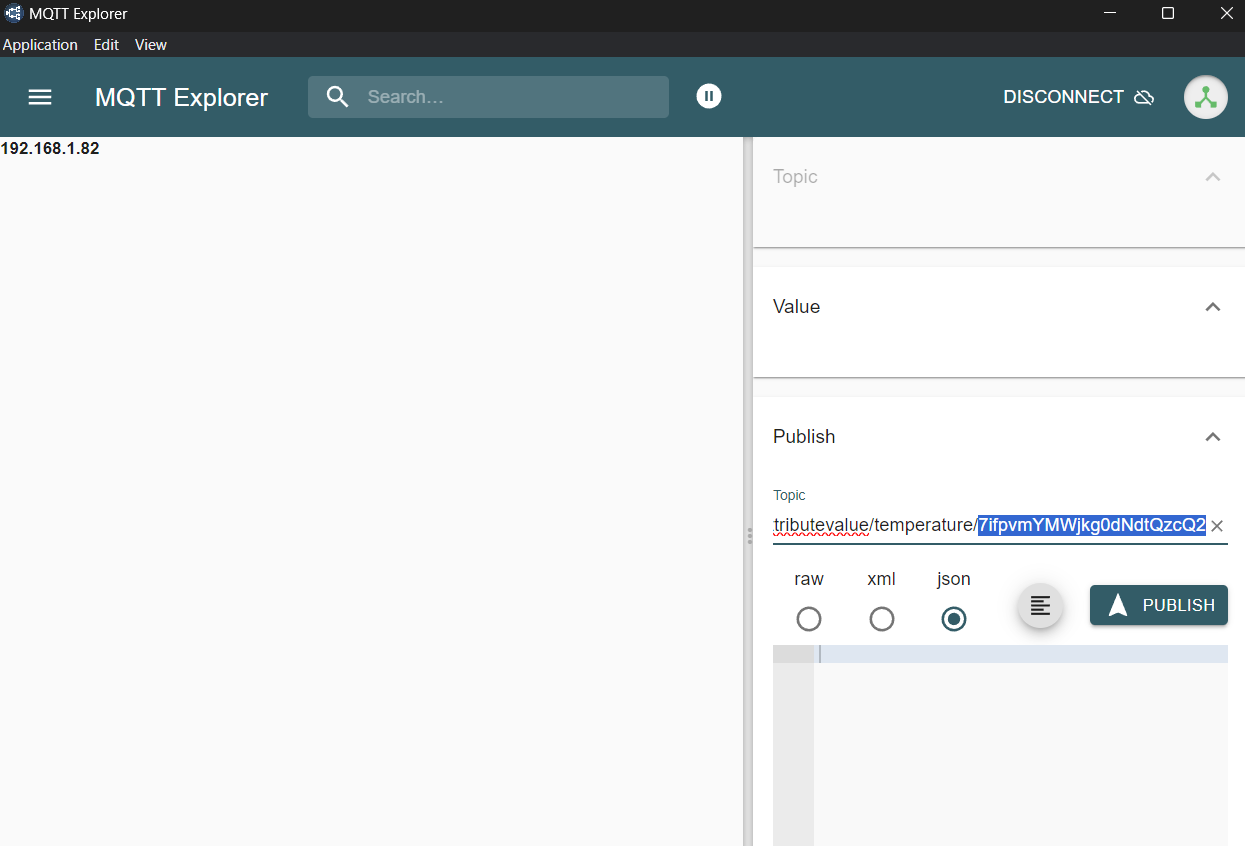
**Hình 16. Sử dụng MQTT Explorer để test truy cập**

Copy Id thiết bị cần publish/subscribe tại thanh đường dẫn của trình duyệt.



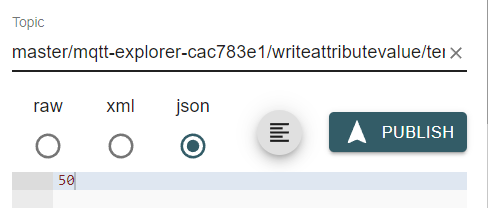
**Hình 17. ID của thiết bị nhận Topic được Publish**

Tại giao diện chính khi connect, nhập vào Topic mà thiết bị sẽ gửi thông tin và chỉnh sửa thông tin id thiết bị (Cú pháp topic phải đúng yêu cầu trong tài liệu hướng dẫn mà Platform đặt ra)



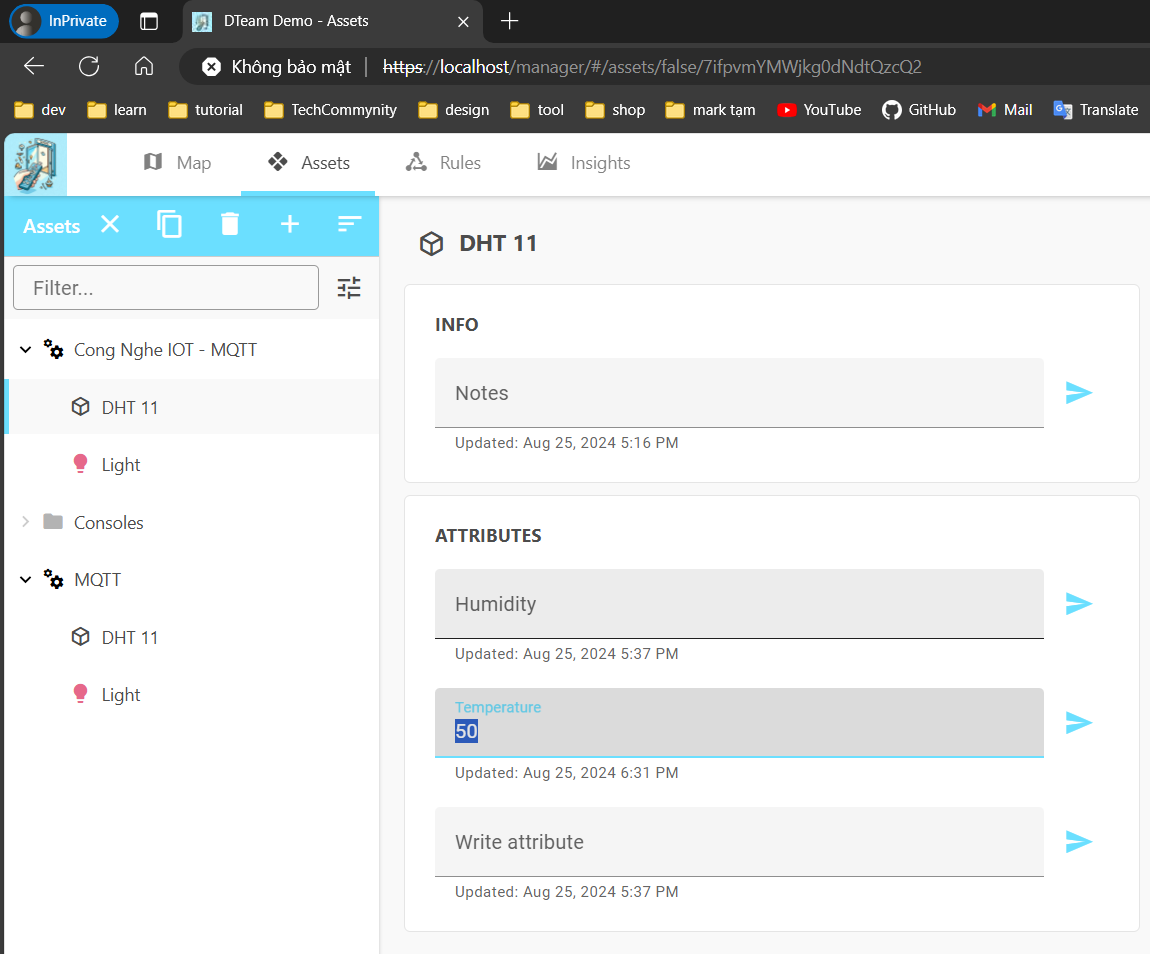
**Hình 18. Thay đổi ID đích cho thiết bị muốn Publish**

Nhập vào dữ liệu giả định cần truyền và click “Publish” để truyền dữ liệu đến server, với " mqtt-explorer-cac783e1" là thông tin ID của mỗi thiết bị, nó có thể được cài đặt tùy ý nhưng nên thay đổi để nó là duy nhất khi sau này hệ thống mở rộng (ở đây id này là id của MQTT Explorer).



**Hình 19. Publish dữ liệu giả lập từ broker**

Kiểm tra lại thông tin mà server đã nhận được sau khi publish



**Hình 20. Cập nhật dữ liệu sau khi publish**

Trong trường hợp gặp lỗi, vấn đề phát sinh thì kiểm tra lại các kết nối (địa chỉ mạng, port, protocol, ...), thông tin thiết bị “Asset”, đường dẫn “Topic” có đúng với cài đặt không và khắc phục.

Khi đã có thể truyền được dữ liệu từ Broker hay thiết bị giả lập thành công thì có thể trực tiếp tiến hành với code và thiết bị thật.

**CHƯƠNG IV: KẾT LUẬN VÀ   
HƯỚNG PHÁT TRIỂN**

**I. KẾT LUẬN**

Hệ thống đã đáp ứng được các yêu cầu đặt ra ban đầu, từ việc giám sát nhiệt độ và độ ẩm trong thời gian thực cho đến khả năng hiển thị và quản lý dữ liệu thông qua OpenRemote. Sự kết hợp giữa các công nghệ phần cứng và phần mềm đã tạo nên một giải pháp IoT hoàn chỉnh, hiệu quả trong việc quản lý môi trường sống.

* 1. **KẾT QUẢ ĐỐI CHIẾU VỚI MỤC TIÊU**

|  |  |  |  |
| --- | --- | --- | --- |
| **Mục tiêu** | **Kết quả thực hiện** | **Đánh giá** | **Giải thích (nếu không đạt)** |
| Xây dựng hệ thống giám sát nhiệt độ và độ ẩm sử dụng ESP32 và DHT11. | Hoàn thành. Hệ thống hoạt động ổn định. | Đạt |  |
| Hiển thị dữ liệu lên màn hình LCD và gửi dữ liệu đến OpenRemote thông qua MQTT. | Dữ liệu được hiển thị trên LCD và gửi đi thành công. | Đạt |  |
| Tích hợp hệ thống với nền tảng OpenRemote để quản lý và giám sát dữ liệu. | Đã tích hợp thành công với OpenRemote. | Đạt |  |
| Đảm bảo hệ thống hoạt động liên tục và ổn định trong môi trường thực tế. | Hệ thống hoạt động liên tục, tuy nhiên có một số lỗi nhỏ về kết nối Wi-Fi không ổn định. | Không đạt | Do môi trường Wi-Fi có nhiễu, làm giảm tính ổn định. |

**2. CÁC HẠN CHẾ**

* **Kết nối Wi-Fi chưa ổn định**: Trong một số trường hợp, kết nối Wi-Fi có thể bị gián đoạn do môi trường có nhiễu hoặc tín hiệu yếu, ảnh hưởng đến khả năng gửi dữ liệu từ ESP32 lên server.
* **Khả năng mở rộng**: Hệ thống hiện tại chỉ sử dụng trong mạng local (LAN), chưa thể điều khiển và giám sát từ xa và phụ thuộc vào ip server. Nếu muốn mở rộng để giám sát từ nhiều khu vực khác nhau, cần phải điều chỉnh lại hệ thống mạng ra internet để cấp tên miền server thay vì địa chỉ ip như hiện tại.
* **Thời gian phản hồi**: Thời gian cập nhật dữ liệu từ cảm biến lên server có thể bị chậm nếu số lượng thiết bị kết nối lớn hoặc khi hệ thống phải xử lý nhiều tác vụ cùng lúc.
* **Khả năng tùy chỉnh:** Quá phụ thuộc vào cấu hình mặc định của Platform, chưa thể tùy chỉnh chuyên sâu vào giao diện, cách thức quản lý của hệ thống.

## II. HƯỚNG PHÁT TRIỂN

* **Cải thiện ổn định kết nối Wi-Fi**: Tìm hiểu và áp dụng các phương pháp để tăng cường tín hiệu Wi-Fi, bao gồm sử dụng các bộ phát tín hiệu mạnh hơn, hoặc chuyển sang sử dụng giao thức mạng khác như LoRa cho các ứng dụng IoT.
* **Tùy biến hệ thống:** Thiết kế, tích hợp lại các giao diện mặc định theo đúng chủ đề, định hướng hoặc yêu cầu của doanh nghiệp, khách hàng đề ra. Tăng cường chức năng hỗ trợ cho người dùng.
* **Mở rộng hệ thống**: Đưa hệ thống ra mạng internet để kết nối, điều khiển, giám sát thiết bị từ xa nhằm đáp ứng nhu cầu quản lý các thiết bị IOT toàn diện hơn.
* **Tối ưu hóa thời gian phản hồi**: Cải thiện thuật toán xử lý và truyền tải dữ liệu để giảm thời gian chờ, đồng thời tăng cường khả năng xử lý đồng thời nhiều tín hiệu từ các cảm biến khác nhau.
* **Bảo mật hệ thống**: Tăng cường các biện pháp bảo mật như mã hóa dữ liệu khi truyền tải, sử dụng các phương pháp xác thực mạnh để ngăn chặn truy cập trái phép vào hệ thống.

**PHỤ LỤC**

**Code trong file docker-compose.yml**

version: '2.4'

volumes:

proxy-data:

manager-data:

postgresql-data:

services:

proxy:

image: openremote/proxy:${PROXY\_VERSION:-latest}

restart: always

depends\_on:

manager:

condition: service\_healthy

ports:

- "80:80" # Needed for SSL generation using letsencrypt

- "${OR\_SSL\_PORT:-443}:443"

- "8883:8883"

#- "127.0.0.1:8404:8404" # Localhost metrics access

volumes:

- proxy-data:/deployment

environment:

LE\_EMAIL: ${OR\_EMAIL\_ADMIN:-}

DOMAINNAME: ${OR\_HOSTNAME:-localhost}

DOMAINNAMES: ${OR\_ADDITIONAL\_HOSTNAMES:-}

# USE A CUSTOM PROXY CONFIG - COPY FROM <https://raw.githubusercontent.com/openremote/proxy/main/haproxy.cfg>

#HAPROXY\_CONFIG: '/data/proxy/haproxy.cfg'

postgresql:

restart: always

image: openremote/postgresql:${POSTGRESQL\_VERSION:-latest}

volumes:

- postgresql-data:/var/lib/postgresql/data

- manager-data:/storage

keycloak:

restart: always

image: openremote/keycloak:${KEYCLOAK\_VERSION:-latest}

depends\_on:

postgresql:

condition: service\_healthy

volumes:

- ./deployment:/deployment

environment:

KEYCLOAK\_ADMIN\_PASSWORD: ${OR\_ADMIN\_PASSWORD:-secret}

KC\_HOSTNAME: ${OR\_HOSTNAME:-localhost}

KC\_HOSTNAME\_PORT: ${OR\_SSL\_PORT:--1}

manager:

# privileged: true

restart: always

image: openremote/manager:${MANAGER\_VERSION:-latest}

depends\_on:

keycloak:

condition: service\_healthy

ports:

- "127.0.0.1:8405:8404" # Localhost metrics access

environment:

OR\_SETUP\_TYPE:

OR\_ADMIN\_PASSWORD:

OR\_SETUP\_RUN\_ON\_RESTART:

OR\_EMAIL\_HOST:

OR\_EMAIL\_USER:

OR\_EMAIL\_PASSWORD:

OR\_EMAIL\_X\_HEADERS:

OR\_EMAIL\_FROM:

OR\_EMAIL\_ADMIN:

OR\_HOSTNAME: ${OR\_HOSTNAME:-localhost}

OR\_ADDITIONAL\_HOSTNAMES: ${OR\_ADDITIONAL\_HOSTNAMES:-}

OR\_SSL\_PORT: ${OR\_SSL\_PORT:--1}

OR\_DEV\_MODE: ${OR\_DEV\_MODE:-false}

# The following variables will configure the demo

OR\_FORECAST\_SOLAR\_API\_KEY:

OR\_OPEN\_WEATHER\_API\_APP\_ID:

OR\_SETUP\_IMPORT\_DEMO\_AGENT\_KNX:

OR\_SETUP\_IMPORT\_DEMO\_AGENT\_VELBUS:

volumes:

- manager-data:/storage

- ./deployment:/deployment # Ánh xạ thư mục deployment

**Code trong file ino version DHT 11**

#include <WiFi.h>

#include <WiFiClientSecure.h>

#include <PubSubClient.h>

#include <DHT.h> // Thêm thư viện DHT

#include <LiquidCrystal\_I2C.h>

// WiFi

const char\* ssid = "admin"; // Wifi SSID

const char\* password = "ChoQRxemPass"; // Wifi Password

// MQTT Broker

const char\* mqtt\_server = "192.168.126.169";

unsigned int mqtt\_port = 1883; //SSL 8883 NoneSSL 1883

const char\* username = "master:mqttuser"; // Service User Realm:Serviceuser

const char\* mqttpass = "jrEeOQJfUCuIPOUZXajr9Iz8p3udYYw8"; // Service User Secret

const char\* ClientID = "client123";

// LastWill

const char\* lastwill = "master/client123/writeattributevalue/writeAttribute/3dmEu8zvK3Eseb0C0fzNWd";

const char\* lastwillmsg = "0";

// Subscribing Topic

const char \*topic = "master/client123/attribute/subscribeAttribute/3dmEu8zvK3Eseb0C0fzNWd";

// Localhost Certificate

// const char\* local\_root\_ca

// DHT Sensor Settings

#define DHTPIN 4

#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);

// Objects

WiFiClient askClient; // SSL Client

PubSubClient client(askClient);

LiquidCrystal\_I2C lcd(0x27, 16, 2);

void setup() {

Serial.begin(115200);

Serial.println(ssid);

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

}

Serial.println(WiFi.localIP());

// askClient.setCACert(local\_root\_ca);

client.setServer(mqtt\_server, mqtt\_port);

client.setCallback(callback);

dht.begin();

// Khởi tạo màn hình LCD

lcd.begin();

lcd.backlight();

lcd.setCursor(0, 0);

lcd.print("DFox...");

lcd.clear();

lcd.setCursor(0, 0);

lcd.print("IP:");

lcd.setCursor(3, 0);

lcd.print(WiFi.localIP());

}

void loop() {

if (!client.connected()) {

reconnect();

}

client.loop();

// Đọc dữ liệu từ cảm biến DHT11

float h = dht.readHumidity();

float t = dht.readTemperature();

// Kiểm tra xem dữ liệu có hợp lệ không

if (isnan(h) || isnan(t)) {

Serial.println("Failed to read from DHT sensor!");

return;

}

// Chuyển đổi dữ liệu thành chuỗi để publish lên MQTT

String temp = String(t);

String hum = String(h);

// Publish dữ liệu lên các topic MQTT

client.publish("master/client123/writeattributevalue/temperature/3dmEu8zvK3Eseb0C0fzNWd", temp.c\_str());

client.publish("master/client123/writeattributevalue/humidity/3dmEu8zvK3Eseb0C0fzNWd", hum.c\_str());

Serial.print("Temp: ");

Serial.println(temp.c\_str());

Serial.print("Hum: ");

Serial.println(hum.c\_str());

Serial.println("- - - - -");

lcd.clear();

lcd.setCursor(0, 0);

lcd.print("Temp: ");

lcd.print(temp);

lcd.print("C");

lcd.setCursor(0, 1);

lcd.print("Hum: ");

lcd.print(hum);

lcd.print("%");

delay(10000); // Delay 10 giây trước khi gửi dữ liệu tiếp theo

}

// MQTT callback

void callback(char\* topic, byte\* payload, unsigned int length) {

Serial.print("Message arrived in topic: ");

Serial.println(topic);

Serial.print("Message: ");

for (int i = 0; i < length; i++) {

Serial.print((char)payload[i]);

}

Serial.println();

}

// MQTT reconnect

void reconnect() {

while (!client.connected()) {

Serial.print("Attempting MQTT connection...");

if (client.connect(ClientID, username, mqttpass, lastwill, 1, 1, lastwillmsg)) {

Serial.println("connected");

client.subscribe(topic);

} else {

Serial.print("failed, rc=");

Serial.print(client.state());

Serial.println(" try again in 5 seconds");

delay(5000);

}

}

}

**Code trong file ino version LED**

#include <WiFi.h>

#include <WiFiClientSecure.h>

#include <PubSubClient.h>

#include <ArduinoJson.h>

// WiFi

const char\* ssid = "admin"; // Wifi SSID

const char\* password = "ChoQRxemPassCaiLol"; // Wifi Password

// MQTT Broker

const char\* mqtt\_server = "192.168.126.169";

unsigned int mqtt\_port = 1883; //SSL 8883 NoneSSL 1883

const char\* username = "master:mqttuser"; // Service User Realm:Serviceuser

const char\* mqttpass = "jrEeOQJfUCuIPOUZXajr9Iz8p3udYYw8"; // Service User Secret

const char\* ClientID = "client123";

const char\* lastwill = "master/client123/writeattributevalue/writeAttribute/3dmEu8zvK3Eseb0C0fzNWd";

const char\* lastwillmsg = "0";

// Subscribing Topic

const char \*topic = "master/client123/attribute/onOff/5a0O9cHSR4bYLfrvmqWiqB";

// LED

const int ledPin = 5;

int ledState = LOW;

WiFiClient askClient; // SSL Client

PubSubClient client(askClient);

void setup() {

Serial.begin(115200);

Serial.println(ssid);

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

}

Serial.println(WiFi.localIP());

client.setServer(mqtt\_server, mqtt\_port);

client.setCallback(callback);

pinMode(ledPin, OUTPUT);

digitalWrite(ledPin, ledState);

}

void loop() {

if (!client.connected()) {

reconnect();

}

client.loop();

delay(10); // Delay nhỏ để tránh xử lý quá nhanh

}

// MQTT callback

void callback(char\* receivedTopic, byte\* payload, unsigned int length) {

Serial.print("Message arrived in topic: ");

Serial.println(receivedTopic);

// Kiểm tra xem topic nhận được có khớp với topic mong muốn

if (String(receivedTopic) == topic) {

payload[length] = '\0'; // Đảm bảo payload là một chuỗi kết thúc đúng

Serial.print("Message: ");

Serial.println((char\*)payload);

// Phân tích JSON từ payload

StaticJsonDocument<256> doc;

DeserializationError error = deserializeJson(doc, payload);

// Kiểm tra lỗi khi phân tích JSON

if (!error) {

// Trích xuất giá trị của trường "value"

bool value = doc["value"];

Serial.print("Parsed value: ");

Serial.println(value);

// Kiểm tra giá trị "true" hoặc "false" và điều khiển LED

if (value) {

digitalWrite(ledPin, HIGH);

ledState = HIGH;

Serial.println("LED ON");

} else {

digitalWrite(ledPin, LOW);

ledState = LOW;

Serial.println("LED OFF");

}

} else {

Serial.print("deserializeJson() failed: ");

Serial.println(error.c\_str());

}

}

}

// MQTT reconnect

void reconnect() {

while (!client.connected()) {

Serial.print("Attempting MQTT connection...");

if (client.connect(ClientID, username, mqttpass, lastwill, 1, 1, lastwillmsg)) {

Serial.println("connected");

client.subscribe(topic);

} else {

Serial.print("failed, rc=");

Serial.print(client.state());

Serial.println(" try again in 5 seconds");

delay(5000);

}

}

}

**TÀI LIỆU THAM KHẢO**

[1] *OpenRemote Documents*;  [Introduction | OpenRemote Documentation;](https://docs.openremote.io/docs/introduction/;) 20/08/2024.

[2] Rich, Apurba, Peeter; 2022; *OpenRemote Forum*; [OpenRemote - OpenRemote forum](https://forum.openremote.io/); 20/08/2024.

[3] Conectar MQTT en OpenRemote; *Notion*; <https://prickly-skull-7f5.notion.site/Conectar-MQTT-en-OpenRemote-8625d0042f614a08af5b39a9a7cf6ff5>; 20/08/2024.

[4] All Readme of OpenRemote; *Github*; <https://github.com/openremote/openremote?tab=readme-ov-file>; 20/08/2024.