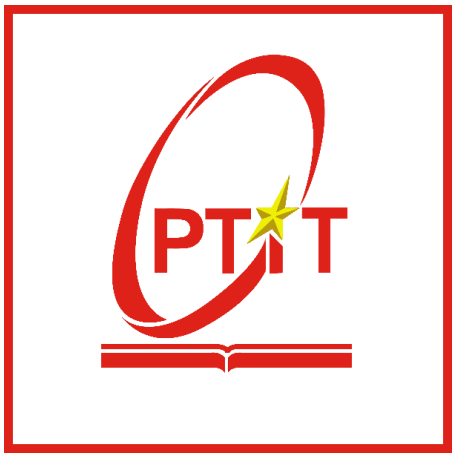
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

**KHOA CÔNG NGHỆ THÔNG TIN**

---




# BÁO CÁO BÀI TẬP LỚN

# MÔN HỌC: IOT VÀ ỨNG DỤNG

**Giảng viên : Nguyễn Quốc Uy**

**Sinh viên : Lê Quốc Trung**

**Lớp : D21CQCN10-B**

**Mã sinh viên : B21DCCN730**

**Số điện thoại : 0333982632**

**Tháng 8/2024**

# Mục lục

## Mục lục

**Lời cảm ơn**

Đầu tiên, em xin gửi lời cảm ơn sâu sắc đến Học viện Công nghệ Bưu chính Viễn thông và khoa CNTT1 đã đưa môn học Nhập môn trí tuệ nhân tạo vào trong chương trình giảng dạy. Đặc biệt, em xin gửi lời cảm ơn sâu sắc đến giảng viên bộ môn thầy Nguyễn Quốc Uy đã dạy dỗ, rèn luyện và truyền đạt những kiến thức quý báu cho em trong suốt thời gian học tập vừa qua.

Trong thời gian tham dự lớp học của thầy, em đã được tiếp thu thêm nhiều kiến thức bổ ích, học tập được tinh thần làm việc hiệu quả, nghiêm túc. Đây thực sự là những điều rất cần thiết cho quá trình học tập và công tác sau này của em. Thêm vào đó, nhờ sự dẫn dắt và chỉ bảo của thầy, em đã thực hiện được một đề tài bài tập lớn hoàn chỉnh cho môn học này, em rất biết ơn điều đó.

Em xin chân thành cảm ơn, chúc thầy luôn mạnh khoẻ và tiếp tục đạt được nhiều thành công trong cuộc sống!

# Chương 1. Giới thiệu

## 1.1. Đặt vấn đề

Trong bối cảnh phát triển của công nghệ, IoT (Internet of Things) đang trở thành xu hướng phổ biến trong việc kết nối và điều khiển các thiết bị điện tử thông qua mạng Internet. Việc ứng dụng IoT vào đo lường các chỉ số môi trường như nhiệt độ, độ ẩm, ánh sáng và điều khiển các thiết bị điện ngày càng trở nên cần thiết, đặc biệt là trong các hệ thống nhà thông minh.

Hệ thống đo lường và điều khiển thiết bị điện hiện tại thường gặp nhiều hạn chế về khả năng giám sát từ xa, hiệu suất hoạt động, và khả năng tích hợp với các thiết bị khác. Do đó, đề tài này nhằm mục đích xây dựng một hệ thống IoT hoàn chỉnh để theo dõi các chỉ số môi trường và điều khiển các thiết bị điện một cách tự động và hiệu quả.

## 1.2. Mục tiêu và phạm vi đề tài

### 1.2.1. Mục tiêu:

- Xây dựng hệ thống IoT có khả năng đo lường các chỉ số như nhiệt độ, độ ẩm, ánh sáng.
- Điều khiển các thiết bị điện (như quạt, điều hòa, đèn) dựa trên các chỉ số đo được.
- Tạo giao diện người dùng để theo dõi và điều khiển hệ thống từ xa.

### 1.2.2. Phạm vi:

- Đề tài tập trung vào việc thiết kế và triển khai hệ thống trong môi trường nhà ở thông minh.
- Hệ thống sẽ sử dụng các cảm biến phổ biến và kết nối với một nền tảng IoT để lưu trữ và xử lý dữ liệu.

## 1.3. Định hướng giải pháp

### 1.3.1. Phân Tích Yêu Cầu

Giai đoạn đầu tiên của quy trình phát triển hệ thống là phân tích yêu cầu. Trong bước này, cần đánh giá các yêu cầu hệ thống một cách chi tiết. Đầu tiên, xác định các loại cảm biến cần thiết để thu thập dữ liệu môi trường, chẳng hạn như cảm biến nhiệt độ, độ ẩm và ánh sáng.

Đồng thời, xác định các thiết bị điều khiển cần thiết để điều chỉnh môi trường dựa trên dữ liệu cảm biến, chẳng hạn như quạt, điều hòa không khí hoặc đèn chiếu sáng.

Tiếp theo, cần phân tích phương thức kết nối giữa các thành phần trong hệ thống. Điều này bao gồm việc lựa chọn giao thức truyền thông phù hợp (như MQTT, HTTP) và xác định cách các thiết bị sẽ giao tiếp với nhau và với hệ thống trung tâm. Cần xem xét các yếu tố quan trọng như độ tin cậy của kết nối, băng thông cần thiết và khả năng mở rộng của hệ thống khi số lượng cảm biến và thiết bị tăng lên.

### 1.3.2. Thiết Kế Hệ Thống

Giai đoạn thiết kế hệ thống bao gồm việc lập kế hoạch chi tiết cho thiết kế tổng thể hệ thống, bao gồm cả phần cứng và phần mềm. Trong bước này, cần phát triển sơ đồ tổng quan của hệ thống, phân chia các thành phần thành các module phần cứng và phần mềm, và xác định cách các module này sẽ tương tác với nhau.

Đối với phần cứng, cần lựa chọn các thành phần điện tử và cảm biến phù hợp, thiết kế mạch điện và bố trí các linh kiện trên bảng mạch. Đối với phần mềm, cần phát triển giao diện người dùng, các thuật toán xử lý dữ liệu và hệ thống lưu trữ và phân tích dữ liệu. Đặc biệt, cần đảm bảo tính tương thích giữa các thành phần phần cứng và phần mềm, kiểm tra tính tương thích của các giao thức truyền thông, điện áp hoạt động của các linh kiện và khả năng tích hợp của các phần mềm. Đồng thời, cần xem xét khả năng mở rộng của hệ thống để dễ dàng bổ sung thêm cảm biến hoặc thiết bị trong tương lai mà không làm gián đoạn hoạt động hiện tại.

### 1.3.3. Phát Triển và Thử Nghiệm

Giai đoạn phát triển bao gồm việc xây dựng hệ thống dựa trên thiết kế đã được xác định. Việc này bao gồm lắp ráp phần cứng, viết mã phần mềm và tích hợp các thành phần lại với nhau. Trong quá trình phát triển, cần liên tục kiểm tra và điều chỉnh các phần của hệ thống để đảm bảo mọi thành phần hoạt động đúng theo yêu cầu.

Sau khi hệ thống được xây dựng, cần tiến hành thử nghiệm toàn diện để đảm bảo hệ thống hoạt động ổn định và đáp ứng các yêu cầu đã đề ra. Thử nghiệm nên bao gồm các tình huống sử dụng thực tế, kiểm tra các trường hợp lỗi và khả năng xử lý các điều kiện ngoài dự kiến.

Cần đảm bảo rằng hệ thống có thể xử lý dữ liệu từ các cảm biến một cách chính xác, điều khiển các thiết bị đúng cách và duy trì hiệu suất ổn định trong thời gian dài.

Cuối cùng, dựa trên kết quả thử nghiệm, cần thực hiện các cải tiến cần thiết để tối ưu hóa hiệu suất và độ tin cậy của hệ thống. Sau khi hoàn tất, việc chuẩn bị tài liệu hướng dẫn sử dụng và bảo trì là bước quan trọng để hỗ trợ người dùng cuối trong việc vận hành và duy trì hệ thống.

## 1.4. Bố cục bài tập lớn

**Chương 2: Giao diện, thiết kế tổng thể**

Trình bày các giao diện người dùng, cách thiết kế tổng thể hệ thống, sơ đồ khối, và sơ đồ kết nối các thành phần.

**Chương 3: Chi tiết hệ thống**

Mô tả chi tiết từng thành phần của hệ thống, từ cảm biến, bộ điều khiển đến các thành phần như broker mqtt, backend, frontend, …

**Chương 4: Code**

Trình bày mã nguồn của hệ thống, các thư mục và file, và cách thức hoạt động của phần mềm.

**Chương 5: Kết quả**

Đánh giá kết quả thử nghiệm hệ thống, so sánh với mục tiêu đề ra và đưa ra nhận xét về tính hiệu quả của hệ thống.

# Chương 2. Giao diện, thiết kế tổng thể

## 2.1. Giao diện

Giao diện, chúng ta sẽ mô tả chi tiết các màn hình chính của hệ thống IoT, bao gồm các chức năng và thiết kế của từng màn hình:

**Màn hình Dashboard** là trung tâm của giao diện, nơi người dùng có thể theo dõi và quản lý các chỉ số môi trường trong thời gian thực. Trên màn hình này, các thông số như nhiệt độ, độ ẩm và ánh sáng được hiển thị rõ ràng, cho phép người dùng theo dõi sự thay đổi của chúng theo thời gian. Màn hình còn bao gồm các biểu đồ thể hiện sự biến động của các thông số này, giúp người dùng dễ dàng nắm bắt xu hướng và nhận biết các biến đổi quan trọng. Bên cạnh đó, màn hình Dashboard còn cung cấp các nút điều khiển để bật/tắt các thiết bị điện như quạt, điều hòa và đèn, cho phép người dùng quản lý các thiết bị này một cách nhanh chóng và thuận tiện.

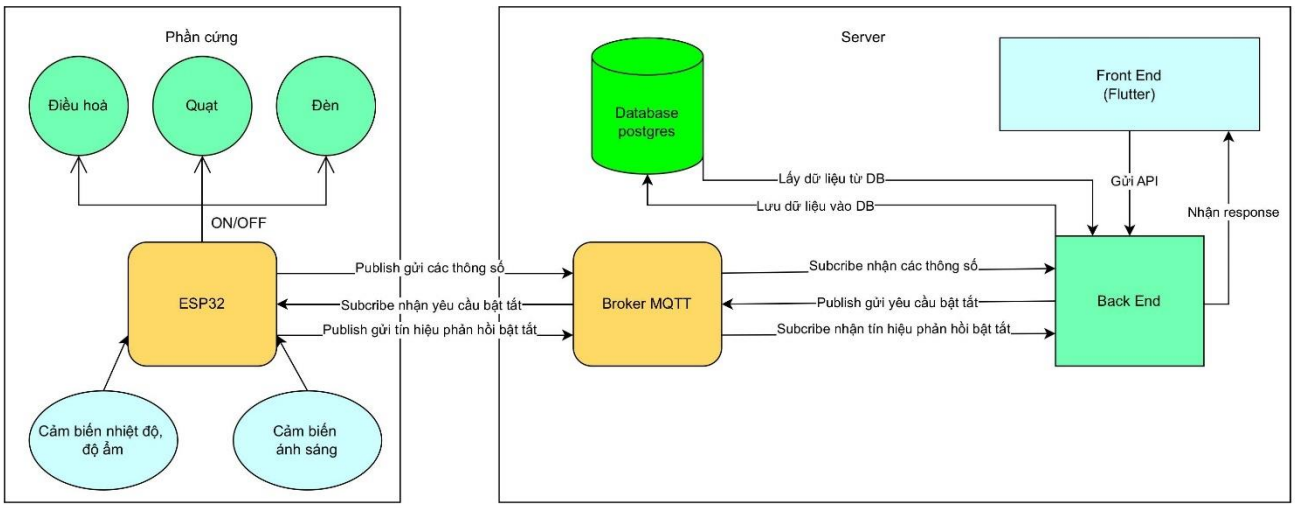
**Màn hình Data Sensor** cung cấp cái nhìn tổng quan về tất cả các thông số đo lường từ trước đến nay. Đây là nơi người dùng có thể xem và quản lý dữ liệu lịch sử của các cảm biến. Màn hình Data Sensor bao gồm các chức năng phân trang để dễ dàng điều hướng qua nhiều dữ liệu, cùng với các công cụ tìm kiếm và sắp xếp để người dùng có thể nhanh chóng tìm thấy thông tin cần thiết. Các tùy chọn sắp xếp có thể được cấu hình để hiển thị dữ liệu theo các tiêu chí khác nhau như thời gian, loại cảm biến hoặc giá trị đo lường.

**Màn hình Action History** tập trung vào việc lưu trữ và quản lý lịch sử các hành động liên quan đến việc bật/tắt các thiết bị điện. Màn hình này cho phép người dùng xem lại tất cả các hành động đã được thực hiện, với các tùy chọn phân trang và tìm kiếm theo khoảng thời gian để dễ dàng theo dõi các sự kiện cụ thể. Bên cạnh đó, người dùng cũng có thể lọc dữ liệu theo từng thiết bị, giúp dễ dàng xác định các thay đổi và hoạt động của từng thiết bị trong hệ thống.

**Màn hình Profile** chứa thông tin chi tiết của sinh viên sử dụng hệ thống. Màn hình này cung cấp một cái nhìn tổng quan về thông tin cá nhân, bao gồm tên, ảnh đại diện, và các thông tin liên quan khác như github và báo cáo. Đây là nơi người dùng có thể cập nhật và

quản lý thông tin cá nhân của mình, đảm bảo rằng các dữ liệu liên quan đến người sử dụng được giữ cập nhật và chính xác.

## 2.2. Thiết kế tổng thể




***Ảnh thiết kế tổng thể của hệ thống***

# Chương 3. Chi tiết về đề tài

## 3.1. Các thiết bị phần cứng

Trong hệ thống IoT được phát triển, các thành phần phần cứng đóng vai trò quan trọng trong việc thu thập dữ liệu, xử lý và điều khiển các thiết bị điện. Các thành phần phần cứng chính được sử dụng trong dự án bao gồm:

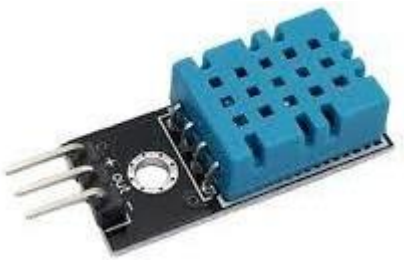
### 3.1.1. ES32:

**ESP32** là một vi điều khiển tích hợp module WiFi, cho phép kết nối mạng không dây để gửi và nhận dữ liệu. Đây là bộ điều khiển trung tâm trong hệ thống, đảm nhận nhiệm vụ nhận tín hiệu từ các cảm biến và điều khiển các thiết bị điện (như đèn LED) dựa trên các tín hiệu đó. Ngoài ra, ESP32 còn chịu trách nhiệm giao tiếp với server qua giao thức MQTT để gửi dữ liệu thu thập được từ các cảm biến lên hệ thống và nhận lệnh điều khiển từ server để thực hiện các tác vụ điều khiển.

*Ảnh về module ESP32*

### 3.1.2. Cảm biến nhiệt độ và độ ẩm:

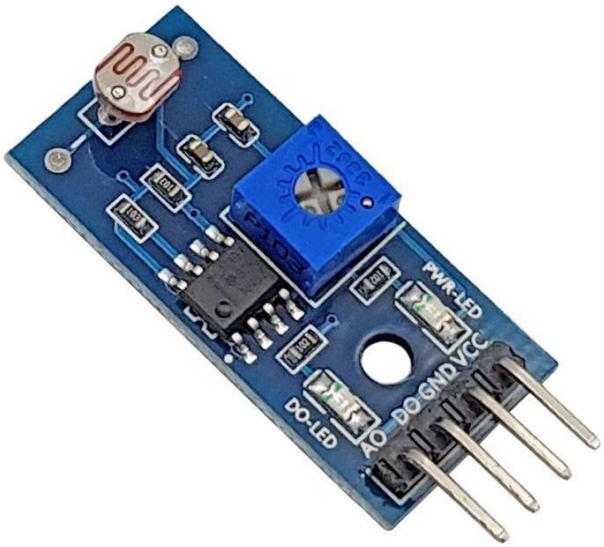
Cảm biến nhiệt độ và độ ẩm (DHT11) được sử dụng để đo lường hai thông số môi trường quan trọng là nhiệt độ và độ ẩm. Cảm biến này có nhiệm vụ gửi dữ liệu thu thập được về ESP32 để xử lý và truyền dữ liệu đó đến server hoặc sử dụng trực tiếp cho việc điều khiển các thiết bị khác.



*Ảnh về cảm biến DHT11*

### 3.1.3. Cảm biến ánh sáng:

Cảm biến ánh sáng (ví dụ: LDR - Light Dependent Resistor) được sử dụng để đo cường độ ánh sáng trong môi trường. Khi cường độ ánh sáng thay đổi, điện trở của LDR thay đổi theo, tạo ra tín hiệu mà ESP32 có thể đọc và xử lý. Thông tin về mức độ ánh sáng sẽ được sử dụng để điều khiển các thiết bị liên quan hoặc gửi về server để lưu trữ và phân tích.




*Ảnh cảm biến ánh sáng LDR – TH078*

### 3.1.4. Đèn LED:

Ba đèn LED được sử dụng để tượng trưng cho ba thiết bị điện khác nhau: **quạt, điều hòa và đèn chiếu sáng**. ESP32 sẽ điều khiển trạng thái bật/tắt của các đèn LED này dựa trên các tín hiệu nhận được từ server hoặc từ các cảm biến. Đây là cách mô phỏng đơn giản cho việc điều khiển các thiết bị điện trong thực tế.

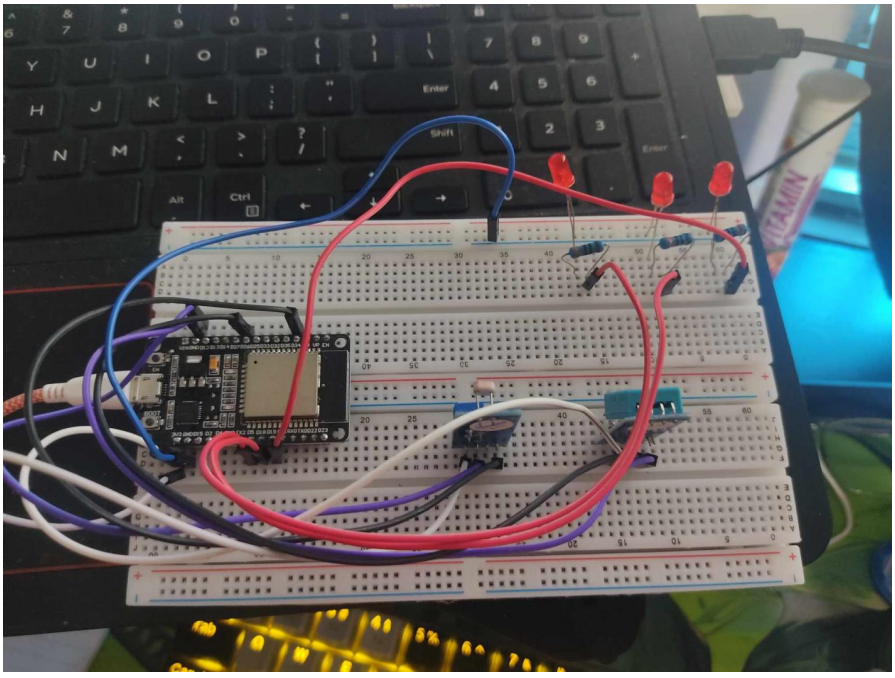
*Ảnh đèn LED*

### 3.1.5. Các thiết bị phụ trợ khác:

- **Dây nối**: Các dây nối (jumper wires) được sử dụng để kết nối các thành phần khác nhau trên breadboard, đảm bảo việc truyền tín hiệu giữa ESP32, cảm biến và đèn LED.
- **Breadboard**: Là bảng mạch thử nghiệm không hàn, breadboard được sử dụng để lắp ráp các mạch đ iện một cách dễ dàng và linh hoạt. Đây là nơi các thiết bị phần cứng được kết nối với nhau mà không cần hàn, giúp dễ dàng điều chỉnh mạch trong quá trình phát triển.
- **Điện trở**: Điện trở được sử dụng để hạn chế dòng điện trong mạch, đặc biệt là khi kết nối đèn LED để tránh tình trạng cháy hỏng do quá dòng.

Các thành phần phần cứng này kết hợp với nhau tạo thành một hệ thống IoT hoàn chỉnh, cho phép thu thập dữ liệu từ môi trường, xử lý thông tin và điều khiển các thiết bị điện từ xa thông qua mạng internet.

Dưới đây là hệ thống phần cứng hoàn chỉnh:



*Ảnh hệ thống phần cứng hoàn chỉnh*

## 3.2. Broker MQTT

### 3.2.1. Lý thuyết về MQTT và Broker

**MQTT** (Message Queuing Telemetry Transport) là một giao thức nhắn tin đơn giản và nhẹ, được thiết kế để truyền tải dữ liệu giữa các thiết bị có tài nguyên hạn chế như các cảm biến hoặc thiết bị nhúng. MQTT hoạt động dựa trên mô hình publish/subscribe, trong đó:

- **Publisher**: Thiết bị hoặc ứng dụng gửi (publish) thông tin lên một topic cụ thể.
- **Subscriber**: Thiết bị hoặc ứng dụng đăng ký (subscribe) với một hoặc nhiều topic để nhận thông tin từ các publisher.
- **Broker**: Là thành phần trung tâm trong hệ thống MQTT, broker nhận các tin nhắn từ các publisher và phân phối chúng đến các subscriber đã đăng ký với các topic tương ứng. Broker chịu trách nhiệm duy trì các kết nối với các client (publisher/subscriber), quản lý quyền truy cập vào các topic, và đảm bảo rằng các tin nhắn được gửi đến đúng các subscriber.

**MQTT** được thiết kế để hoạt động hiệu quả trong các môi trường có băng thông thấp, độ trễ cao hoặc mạng không ổn định, làm cho nó trở nên lý tưởng cho các ứng dụng IoT.

### 3.2.2. Sử dụng Broker MQTT trong dự án

Trong dự án của em, broker MQTT là thành phần không thể thiếu trong việc quản lý và phân phối các thông điệp giữa các thiết bị IoT. Để đơn giản hóa việc phát triển và thử nghiệm, em sử dụng server broker công khai tại địa chỉ ***test.mosquitto.org***, được cung cấp bởi dự án Eclipse Mosquitto.

- **Địa chỉ broker MQTT**: test.mosquitto.org là một broker MQTT công khai, cho phép người dùng kết nối và thử nghiệm các ứng dụng MQTT mà không cần phải tự triển khai một broker riêng.
- **Cổng kết nối MQTT**: Sử dụng cổng 1993 là cổng mặc định cho các kết nối MQTT không mã hóa. Điều này giúp đơn giản hóa quá trình kết nối và phát triển, đặc biệt khi không cần thiết phải bảo mật dữ liệu.

Broker tại ***test.mosquitto.org*** hỗ trợ nhiều cổng và cấu hình khác nhau, bao gồm cả các kết nối mã hóa TLS, nhưng với mục tiêu thử nghiệm và phát triển ban đầu, việc sử dụng cổng 1993 là đủ. Broker này cho phép các thiết bị trong hệ thống IoT của em gửi và nhận các thông điệp, thực hiện điều khiển các thiết bị điện từ xa, và thu thập dữ liệu từ các cảm biến để phân tích.

## 3.3. Backend

### 3.3.1. Giới thiệu

Trong dự án này, backend được xây dựng bằng **Java Spring**.

**Java Spring** cung cấp một bộ công cụ mạnh mẽ nhưng đơn giản để phát triển các API và xử lý yêu cầu HTTP. Trong hệ thống IoT, backend đóng vai trò quan trọng trong việc quản lý dữ liệu và điều khiển các thiết bị thông qua giao thức MQTT và cơ sở dữ liệu Postgre.

### 3.3.2. Chức năng của Backend

Backend thực hiện các nhiệm vụ chính sau:

**Trao đổi dữ liệu với hardware thông qua broker MQTT**:

- **Đóng vai trò client**: Backend hoạt động như một client trong hệ thống MQTT, liên kết với broker MQTT để gửi và nhận các thông điệp. Backend nhận các thông số từ các thiết bị phần cứng như cảm biến nhiệt độ, độ ẩm, và ánh sáng, hoặc gửi yêu cầu điều khiển từ frontend (chẳng hạn như bật/tắt quạt, đèn, điều hòa) đến phần cứng.
- **Xử lý dữ liệu**: Các dữ liệu từ hardware sau khi được backend nhận sẽ được xử lý và có thể được lưu trữ vào cơ sở dữ liệu hoặc phản hồi lại cho frontend.

**Giao tiếp với cơ sở dữ liệu (Postgre)**:

- **Lưu trữ dữ liệu**: Backend giao tiếp với Postgre để lưu trữ các thông số từ các cảm biến và trạng thái thiết bị. Điều này giúp duy trì một bản ghi liên tục của các giá trị môi trường cũng như lịch sử điều khiển thiết bị. Đồng thời cũng lưu dữ liệu lịch sử bật/tắt các thiết bị.

- **Truy xuất dữ liệu**: Backend cũng cho phép truy xuất dữ liệu từ Postgre theo yêu cầu từ frontend, chẳng hạn như lấy các thông số đã lưu trong một khoảng thời gian cụ thể để phân tích hoặc hiển thị biểu đồ.

**Nhận request từ frontend và trả lại response tương ứng**:

- **Xử lý request HTTP**: Backend nhận các request từ frontend, như yêu cầu lấy dữ liệu cảm biến, yêu cầu điều khiển thiết bị, hoặc truy vấn lịch sử dữ liệu.
- **Trả lại response**: Dựa trên request nhận được, backend xử lý và gửi lại response tương ứng cho frontend, đảm bảo rằng người dùng có thể theo dõi các thông số môi trường hoặc điều khiển các thiết bị một cách chính xác và kịp thời.

Backend trong hệ thống IoT của em được xây dựng đóng vai trò then chốt trong việc quản lý và điều phối dữ liệu giữa phần cứng, cơ sở dữ liệu, và giao diện người dùng. Bằng cách sử dụng MQTT để giao tiếp với phần cứng và Postgre để lưu trữ dữ liệu, backend đảm bảo rằng hệ thống hoạt động hiệu quả và ổn định, đáp ứng tốt các yêu cầu của ứng dụng IoT.

## 3.4. Frontend

### 3.4.1. Giới thiệu

Frontend của hệ thống IoT này được xây dựng bằng **Flutter**, một Framework mới nổi tiếng gần đây dùng để phát triển giao diện người dùng (UI). **Flutter** cung cấp khả năng tạo ra các ứng dụng di động và phản hồi nhanh, lý tưởng cho việc hiển thị dữ liệu thời gian thực từ các thiết bị IoT. Frontend được thiết kế để người dùng có thể dễ dàng theo dõi các thông số môi trường, quản lý thiết bị và xem lại lịch sử hành động thông qua các trang giao diện khác nhau.

**Các trang giao diện chính**

Frontend bao gồm bốn trang giao diện chính, mỗi trang thực hiện các chức năng cụ thể:

**Dashboard**:

- **Chức năng**: Trang Dashboard cung cấp một cái nhìn tổng quan về toàn bộ hệ thống, hiển thị các thông số chính như nhiệt độ, độ ẩm, ánh sáng theo thời gian thực.

Dashboard cũng có thể hiển thị trạng thái hiện tại của các thiết bị như quạt, đèn, điều hòa. Đồng thời là biểu đồ thay đổi các thông số.

- **Tương tác**: Người dùng có thể thực hiện các thao tác bật/tắt thiết bị trực tiếp từ Dashboard và nhận phản hồi ngay lập tức.

**Data Sensor**:

- **Chức năng**: Trang Data Sensor hiển thị chi tiết các dữ liệu từ các cảm biến như nhiệt độ, độ ẩm, và ánh sáng. Các dữ liệu này có thể được hiển thị dưới dạng bảng, cho phép người dùng theo dõi xu hướng và biến động của các thông số môi trường theo thời gian.
- **Tương tác**: Người dùng có thể chọn khoảng thời gian để xem dữ liệu lịch sử hoặc tìm kiếm theo các thông số.

**History Action**:

- **Chức năng**: Trang History Action ghi lại và hiển thị lịch sử các hành động điều khiển thiết bị, như việc bật/tắt quạt, đèn, hoặc điều hòa. Người dùng có thể theo dõi những thay đổi đã được thực hiện và thời gian thực hiện.
- **Tương tác**: Lịch sử hành động được phân loại theo thời gian, cho phép người dùng dễ dàng tìm kiếm và xem lại các hành động đã thực hiện.

**Profile**:

- **Chức năng**: Trang Profile cung cấp thông tin cá nhân của người dùng, bao gồm thông tin về tài khoản, các tùy chọn cài đặt và khả năng quản lý thông tin cá nhân.

### 3.4.2. Chức năng tổng quan của Frontend

Frontend đảm nhận vai trò giao tiếp với backend để xử lý và hiển thị dữ liệu cho người dùng. Các chức năng chính bao gồm:

**Gửi request cho backend**:

- Frontend gửi các request HTTP tới backend để lấy dữ liệu thời gian thực từ các cảm biến, dữ liệu lịch sử (history action), và thông tin từ trang Data Sensor.
- Ngoài ra, frontend cũng gửi các yêu cầu bật/tắt thiết bị từ giao diện người dùng đến backend để điều khiển phần cứng.

**Nhận và hiển thị response**:

- Sau khi nhận được phản hồi từ backend, frontend cập nhật giao diện người dùng với các thông tin mới nhất, như giá trị cảm biến hiện tại, trạng thái thiết bị, và thông tin lịch sử hành động.
- Dữ liệu được hiển thị một cách trực quan và dễ hiểu, giúp người dùng có thể theo dõi và điều khiển hệ thống một cách hiệu quả.

# Chương 4. Code

## 4.1. Embedded Code

### 4.1.1. Khai báo thư viện và định nghĩa các biến

- Ở phần này, code định nghĩa các thư viện và các thành phần cần thiết như sau:
- Khai báo các thư viện cần thiết cho cảm biến DHT, MQTT, Wi-Fi, và NTP.
- Khai báo chân GPIO cho cảm biến DHT và các thiết bị điều khiển như quạt, điều hòa, và đèn.
- Khai báo các thông tin kết nối Wi-Fi bao gồm SSID và mật khẩu.
- Định nghĩa địa chỉ và cổng của broker MQTT, cùng với các chủ đề (topics) để điều khiển các thiết bị.
- Khởi tạo đối tượng WiFiClient và PubSubClient để xử lý kết nối MQTT.

final_iot.ino

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <DHT.h>

#define DHT_PIN 26
#define LDR_PIN 34
#define DHT_TYPE DHT11

DHT dht(DHT_PIN, DHT_TYPE);

const char* ssid = "esp32";
const char* password = "12344321";
const char* mqtt_user = "trunglq";
const char* mqtt_pass = "b21dccn730";

//#define MQTT_SERVER "192.168.178.156"
#define MQTT_SERVER "192.168.114.156"

// #define MQTT_PORT 1883
#define MQTT_PORT 1993

#define MQTT_TEMP_TOPIC "final_iot"
#define MQTT_OPS_0_TOPIC "ops/0"
#define MQTT_OPS_1_TOPIC "ops/1"
#define MQTT_OPS_2_TOPIC "ops/2"
#define MQTT_OPS_3_TOPIC "ops/3"
#define ledPin0 5
#define ledPin1 18
#define ledPin2 19

unsigned long previousMillis = 0;
const long interval = 5000;

WiFiClient wifiClient;
PubSubClient client(wifiClient);


```

*Ảnh khai báo thư viện và các biến*

### 4.1.2. Kết nối Wifi

Hàm **setup_wifi()** kết nối ESP32 với mạng Wi-Fi bằng cách sử dụng SSID và mật khẩu đã được khai báo. Hàm sẽ liên tục kiểm tra trạng thái kết nối và chỉ dừng khi kết nối thành công. Sau khi kết nối, nó sẽ in địa chỉ IP cục bộ của ESP32 ra serial monitor.

```
void setup_wifi() {
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}
```

***Ảnh hàm kết nối Wifi***

### 4.1.3. Kết nối MQTT

Hàm **connect_to_broker()** thiết lập kết nối giữa ESP32 và MQTT broker. Nó cấu hình địa chỉ của broker, thiết lập hàm callback, và cố gắng kết nối với broker bằng cách sử dụng một client ID duy nhất dựa trên địa chỉ MAC của thiết bị. Nếu kết nối thành công, nó in thông báo thành công; nếu thất bại, nó in mã lỗi và thử lại sau một khoảng thời gian ngắn.

```
void connect_to_broker() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    String clientId = "ESP32";
    clientId += String(random(0xffff), HEX);
    if (client.connect(clientId.c_str(), mqtt_user, mqtt_pass)) {
      Serial.println("connected");
      client.subscribe(MQTT_OPS_0_TOPIC);
      client.subscribe(MQTT_OPS_1_TOPIC);
      client.subscribe(MQTT_OPS_2_TOPIC);
      client.subscribe(MQTT_OPS_3_TOPIC);
      client.setCallback(callback);
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 2 seconds");
      delay(2000);
    }
  }
}
```

***Ảnh hàm kết nối với MQTT Broker***

### 4.1.4. Đọc dữ liệu từ cảm biến

Hàm **sendSensorData()** đọc dữ liệu từ cảm biến DHT (nhiệt độ, độ ẩm) và cảm biến ánh sáng LDR, kiểm tra tính hợp lệ của các giá trị đo được. Nếu dữ liệu hợp lệ, hàm sẽ tạo một chuỗi JSON chứa các giá trị này và gửi chuỗi đó đến chủ đề MQTT sensor/datas để truyền dữ liệu lên server hoặc các thiết bị khác thông qua MQTT.

```
void sendSensorData() {
  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();
  float light = 4095 - analogRead(LDR_PIN);
  float dust = random(1, 1000);

  if (isnan(temperature) || isnan(humidity)) {
    Serial.println("Failed to read from DHT sensor");
    return;
  }
  if (isnan(light)) {
    Serial.println("Failed to read from Light sensor");
    return;
  }
  String payload = "{\"temperature\": " + String(temperature) + ", \"humidity\": " + String(humidity) + ", \"light\": " + String(light) + "}";
  client.publish(MQTT_TEMP_TOPIC, payload.c_str());
}
```

***Ảnh về hàm đọc và gửi dữ liệu***

### 4.1.5. Điều khiển các thiết bị

```
if (String(topic) == MQTT_OPS_0_TOPIC) {
  if(messageTemp == "true"){
    digitalWrite(ledPin0, HIGH);
  }
  else if(messageTemp == "false"){
    digitalWrite(ledPin0, LOW);
  }
}
if (String(topic) == MQTT_OPS_1_TOPIC) {
  if(messageTemp == "true"){
    digitalWrite(ledPin1, HIGH);
  }
  else if(messageTemp == "false"){
    digitalWrite(ledPin1, LOW);
  }
}
Serial.print(String(topic));
if (String(topic) == MQTT_OPS_2_TOPIC) {
  Serial.print(messageTemp);
  if(messageTemp == "true"){
    digitalWrite(ledPin2, HIGH);
  }
  else if(messageTemp == "false"){
    digitalWrite(ledPin2, LOW);
  }
}
```

***Ảnh hàm điều khiển các thiết bị***

### 4.1.6. Hàm callback

Hàm **callback** xử lý các tin nhắn **MQTT** được gửi đến từ các chủ đề mà ESP32 đã đăng ký. Khi một tin nhắn đến, hàm sẽ kiểm tra chủ đề (topic) để xác định thiết bị liên quan (quạt, điều hòa, hoặc đèn).

- Đầu tiên, hàm in ra chủ đề và nội dung tin nhắn nhận được.
- Sau đó, hàm kiểm tra chủ đề để xác định thiết bị cần điều khiển. Kiểm tra tên topic tương ứng để bật hoặc tắt thiết bị dựa trên nội dung tin nhắn (on hoặc off).
- Cuối cùng, hàm gửi lại một tin nhắn xác nhận về trạng thái hiện tại của thiết bị lên một chủ đề khác (ví dụ: iot/fan/status).

Hàm này đảm bảo rằng các thiết bị được điều khiển từ xa thông qua MQTT và gửi lại trạng thái xác nhận để đảm bảo rằng hành động đã được thực hiện thành công.

```
void callback(char* topic, byte* message, unsigned int length) {
  Serial.print("Message arrived on topic: ");
  Serial.print(topic);
  Serial.print(". Message: ");
  String messageTemp;

  for (int i = 0; i < length; i++) {
    Serial.print((char)message[i]);
    messageTemp += (char)message[i];
  }
  Serial.println();

  if (String(topic) == MQTT_OPS_0_TOPIC) {
    if(messageTemp == "true"){
      digitalWrite(ledPin0, HIGH);
    }
    else if(messageTemp == "false"){
      digitalWrite(ledPin0, LOW);
    }
  }
  if (String(topic) == MQTT_OPS_1_TOPIC) {
    if(messageTemp == "true"){
      digitalWrite(ledPin1, HIGH);
    }
    else if(messageTemp == "false"){
      digitalWrite(ledPin1, LOW);
    }
  }
  Serial.print(String(topic));
  if (String(topic) == MQTT_OPS_2_TOPIC) {
    Serial.print(messageTemp);
    if(messageTemp == "true"){
      digitalWrite(ledPin2, HIGH);
    }
    else if(messageTemp == "false"){
      digitalWrite(ledPin2, LOW);
    }
  }
}
```

***Ảnh hàm callback***

### 4.1.7. Hàm setup() và loop()

Hàm setup() và loop() là hai hàm chính trong chương trình Arduino, được sử dụng để khởi tạo và điều khiển các hoạt động của vi điều khiển ESP32.

**Hàm setup()**

- **Khởi tạo giao tiếp Serial**: Bắt đầu giao tiếp Serial với tốc độ baud là 9600, cho phép in thông tin lên Serial Monitor.
- **Kết nối WiFi**: Gọi hàm setup_wifi() để kết nối ESP32 với mạng WiFi đã định nghĩa.

- **Khởi động NTP Client**: Khởi tạo timeClient để lấy thời gian từ NTP server.
- **Cấu hình các chân GPIO**: Cài đặt các chân ledPin0, ledPin1, và ledPin2 làm đầu ra (OUTPUT) để điều khiển quạt, điều hòa, và đèn.

- **Kết nối với MQTT Broker**: Gọi hàm connect_to_broker() để kết nối ESP32 với MQTT broker.
- **Đăng ký các chủ đề MQTT**: So sánh điều kiện trong hàm connect_to_broker() để đăng ký các chủ đề mà ESP32 cần theo dõi.
- **Khởi tạo cảm biến DHT11**: Khởi tạo cảm biến DHT11 để bắt đầu đọc dữ liệu nhiệt độ và độ ẩm.

**Hàm loop()**

- **Duy trì kết nối MQTT**: Gọi hàm client.loop() để duy trì kết nối và xử lý các tin nhắn đến từ MQTT broker.
- **Đọc dữ liệu cảm biến**: Gọi hàm readDht() để đọc dữ liệu từ cảm biến DHT11 và gửi dữ liệu này lên chủ đề MQTT.
- **Chờ 2 giây**: Thêm một khoảng trễ 2 giây giữa các lần đọc cảm biến và gửi dữ liệu lên MQTT.

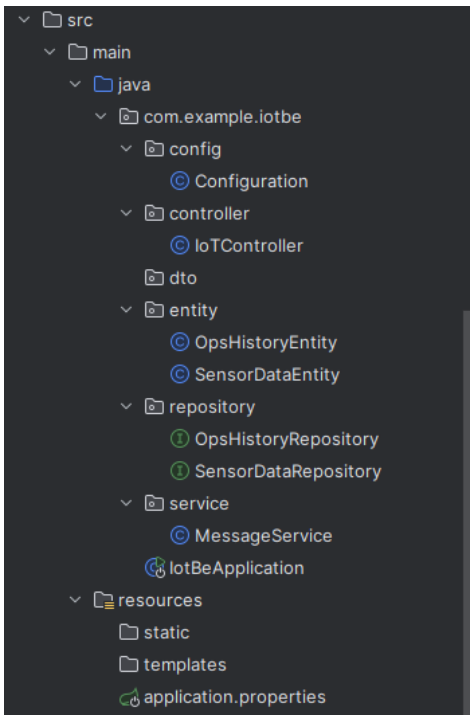
```
void setup() {
  Serial.begin(9600);
  Serial.setTimeout(500);
  setup_wifi();
  client.setServer(MQTT_SERVER, MQTT_PORT );
  connect_to_broker();
  Serial.println("Start transfer");
  pinMode(ledPin0,OUTPUT);
  pinMode(ledPin1,OUTPUT);
  pinMode(ledPin2,OUTPUT);
  pinMode(LDR_PIN, INPUT);
  dht.begin();
}

void loop() {
  if (!client.connected()) {
    connect_to_broker();
    Serial.print('trung');
  }
  client.loop();
  sendSensorData();
  delay(2000);
}
```

*Ảnh hàm setup và loop*

## 4.2. Backend Code

### 4.2.1. Tổng quan về cấu trúc code

Dưới đây là cấu trúc chính của backend:




***Ảnh về cấu trúc thư mục backend***

Phần Backend được tổ chức thành các thư mục chính:

**a. Controllers:**

- **Chức năng:** Thư mục này chứa các tệp tin xử lý logic của ứng dụng, bao gồm các hành động liên quan đến việc quản lý yêu cầu từ phía người dùng hoặc các client khác. Các controller sẽ nhận và xử lý các yêu cầu HTTP từ các route, sau đó gọi đến các service hoặc model tương ứng để thực hiện các tác vụ cần thiết.

**b. Entity:**

- **Chức năng:** Định nghĩa các trường thuộc tính mà Server trả về và client nhận để xử lí.

**c. Service:**

- **Chức năng:** Đăng ký topic và nhận message, data từ server trả về và xử lý.

**d. Config**

- **Chức năng:** Kết nối với MQTT qua user và password.

**e. Repository:**

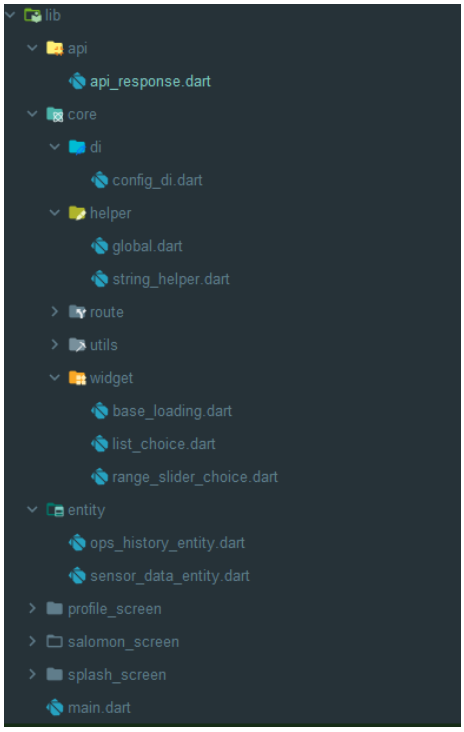
- **Chức năng:** Lớp trung gian giúp truy xuất và thao tác dữ liệu từ cơ sở dữ liệu, tách biệt logic với cách lưu trữ

#### 4.2.2. Luồng hoạt động:

- Khi hệ thống nhận một yêu cầu HTTP (ví dụ: bật quạt), yêu cầu này sẽ được xử lý bởi một controller tương ứng trong thư mục Controllers.
- Controller sẽ gọi đến các hàm trong Repossitory nếu cần truy xuất hoặc cập nhật dữ liệu trong Postgre.
- Nếu yêu cầu liên quan đến việc giao tiếp với các thiết bị IoT, controller sẽ tương tác với các module trong Mqtt để gửi thông điệp qua MQTT hoặc nhận phản hồi từ các thiết bị.

### 4.3. Frontend Code

Dưới đây là tổng quan chính về cấu trúc code Frontend:




*Ảnh về tổng quan cấu trúc FE*

### 4.3.1. Tổng Quan

Frontend của dự án được xây dựng bằng Flutter, một Famework mới nổi gần đây để xây dựng giao diện người dùng. Code chính của frontend được tổ chức trong thư mục lib, với cấu trúc thư mục rõ ràng giúp dễ dàng quản lý và phát triển.

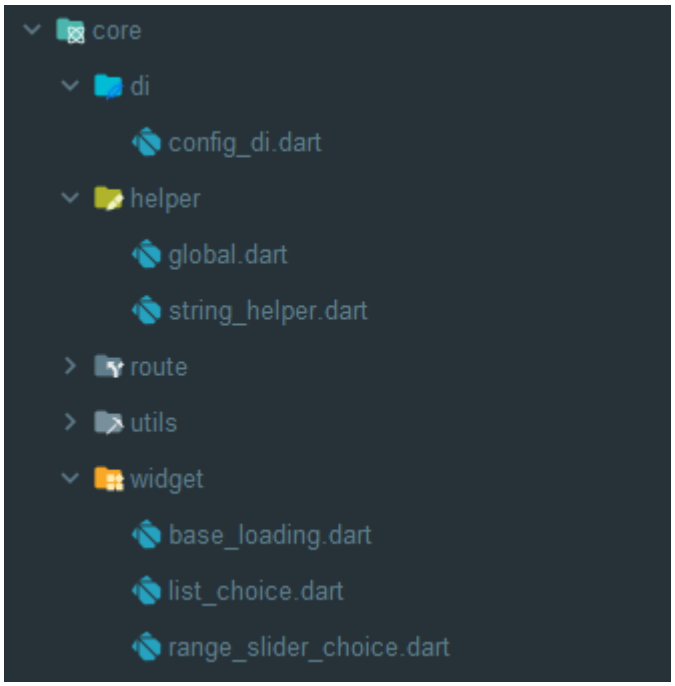
### 4.3.2. Thư Mục Api

- **Mục đích**: Thư mục api để xử lý call API lên server, gửi request và nhận response từ server.


*Ảnh thư mục api*

### 4.3.3. Thư Mục core

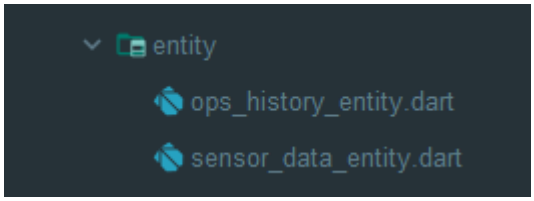
- **Mục đích**: Định nghĩa các widget, UI, component, … sẽ tái sử dụng nhiều lần trong màn hình, thư mục này được coi là lõi.




***Ảnh thư mục core***

### 4.3.4. Thư Mục entity:

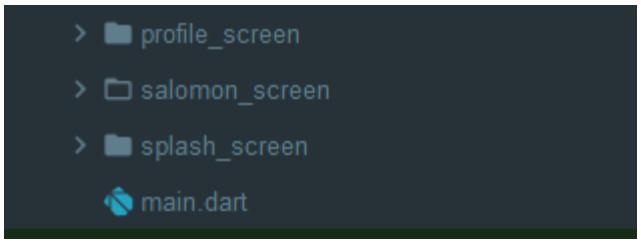
- **Vai trò**: Định nghĩa các thuộc tính của dữ liệu mà server gửi về.




***Ảnh thư mục css***

### 4.3.5. Các thư mục screen

- **Vai trò**: App.js là file chính của ứng dụng React, là nơi tất cả các component khác được kết hợp lại để tạo ra ứng dụng hoàn chỉnh.
- **Cấu trúc**:
    - **Cubit**: Quản lí state để render lại màn hình mỗi khi server gửi dữ liệu về
    - **View**: Giao diện tương ứng của từng màn hình.



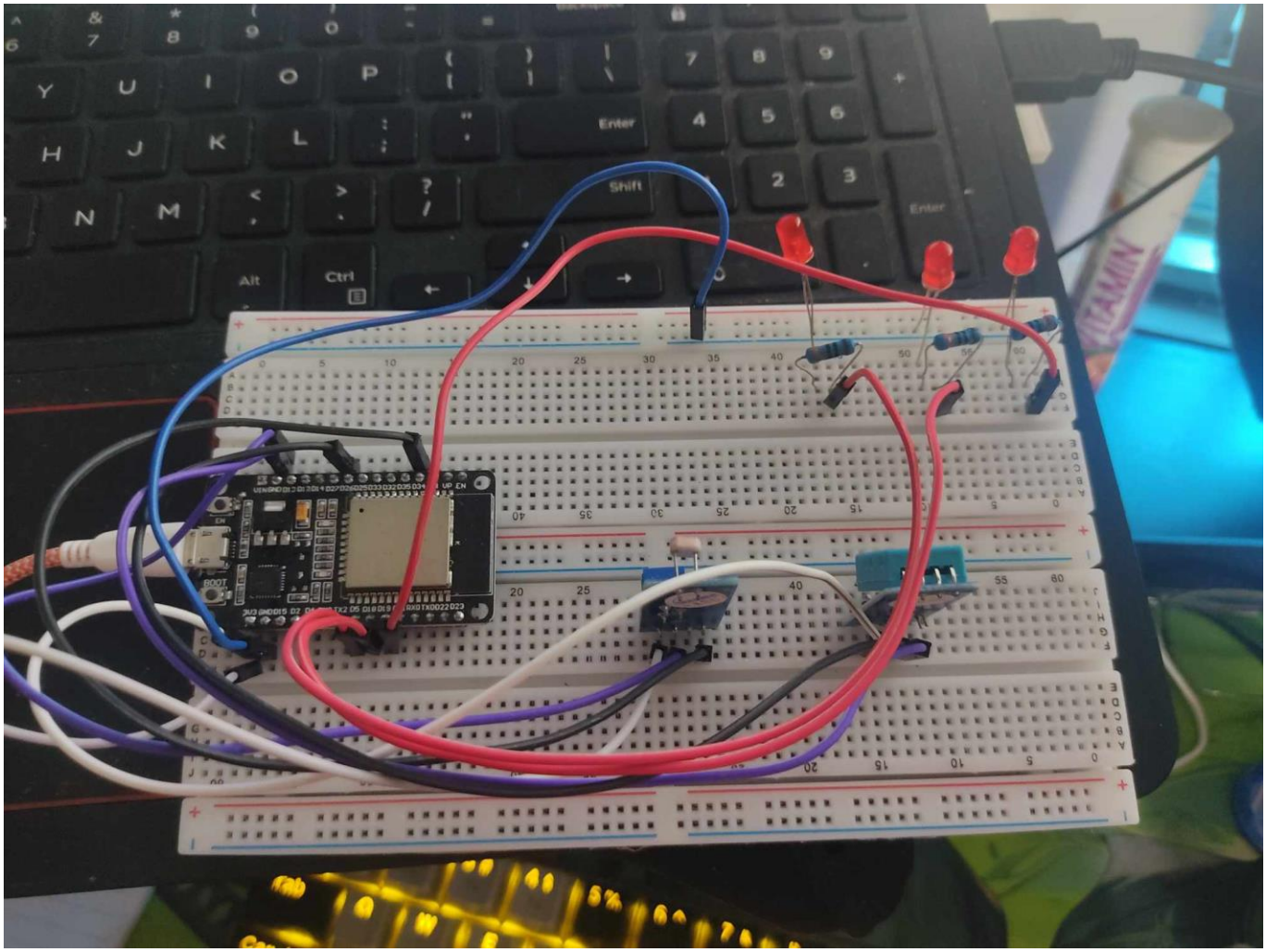

*Ảnh các thư mục screen*

# Chương 5. Kết quả thực nghiệm

## 5.1. Tổng quan

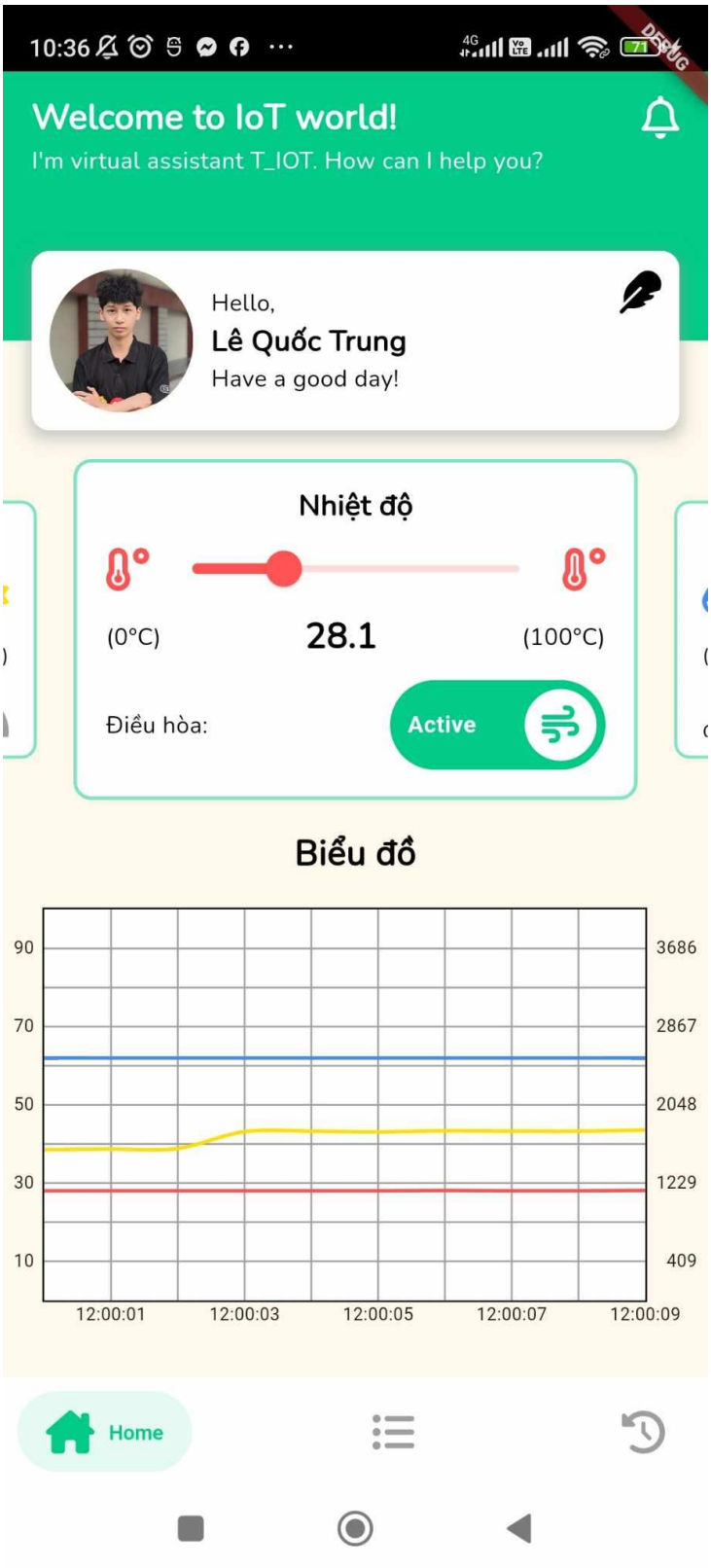
Trong quá trình phát triển hệ thống IoT, em đã thực hiện thành công các chức năng đo lường thời gian thực, điều khiển thiết bị điện từ xa, và lưu trữ dữ liệu cho người dùng. Hệ thống đã được triển khai và thử nghiệm, đáp ứng tốt các yêu cầu đề ra.

**Tổng quan về phần cứng:**



*Ảnh về toàn bộ hệ thống phần cứng*

**Tổng quan về trang DashBoard:**




*Ảnh tổng quan về trang Dashboard*

## 5.2. Đo lường các chỉ số như nhiệt độ, độ ẩm, ánh sáng thời gian thực

Em đã hoàn thành việc đo lường và thu thập các chỉ số môi trường như nhiệt độ, độ ẩm, và ánh sáng trong thời gian thực.

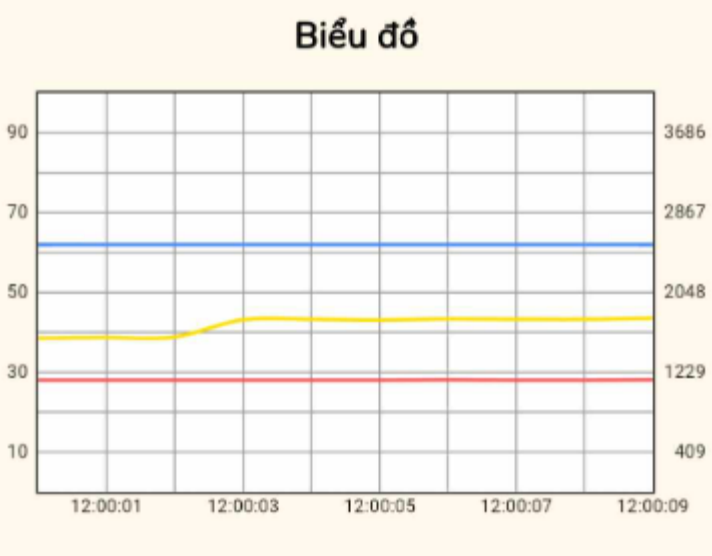
- **Với nhiệt độ, độ ẩm:** Dùng cảm biến DHT11 để đo
- **Với ánh sáng:** Dùng cảm biến LDR – TH078 để đo


***Ảnh minh hoạ hiển thị các thông số***

Các chỉ số này đã được hiển thị trực tiếp trên giao diện mobile, cho phép người dùng theo dõi các thay đổi tức thời.

Đồng thời, nền hiển thị trên mobile sẽ thay đổi màu sắc dựa trên các chỉ số cao hoặc thấp, giúp người dùng dễ dàng nhận biết trạng thái của các thông số môi trường. Hệ thống cũng cung cấp đồ thị thời gian thực, phản ánh sự biến đổi của các giá trị này theo thời gian.

Ngoài ra, hệ thống cũng cung cấp đồ thị thời gian thực, phản ánh sự biến đổi của các giá trị này theo từng thời điểm, giúp người dùng dễ dàng quan sát và phân tích dữ liệu.




***Ảnh đồ thị phản ánh sự thay đổi các chỉ số***

## 5.3. Điều khiển các thiết bị điện

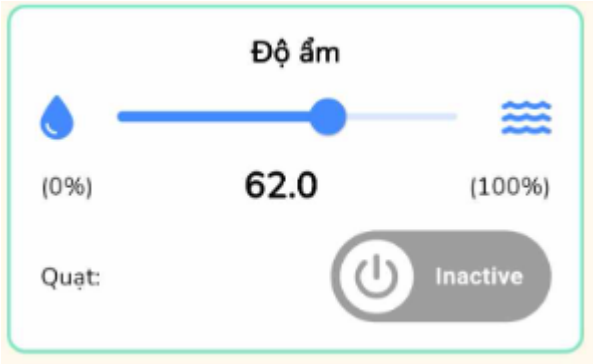
Em đã thiết kế và triển khai thành công phần điều khiển các thiết bị điện từ xa thông qua dashboard trên giao diện mobile. Người dùng có thể bật và tắt các thiết bị như quạt, điều hòa, và đèn từ bất kỳ đâu.

Hệ thống chỉ hiển thị thông báo thành công trên giao diện khi thiết bị thực sự đã bật hoặc tắt, đảm bảo rằng người dùng nhận được phản hồi chính xác về trạng thái của thiết bị. Điều này giúp nâng cao trải nghiệm sử dụng và đảm bảo tính hiệu quả trong việc điều khiển các thiết bị điện.

Khi bật các thiết bị:


*Ảnh khi đang bật thiết bị điện*

Khi tắt các thiết bị điện:



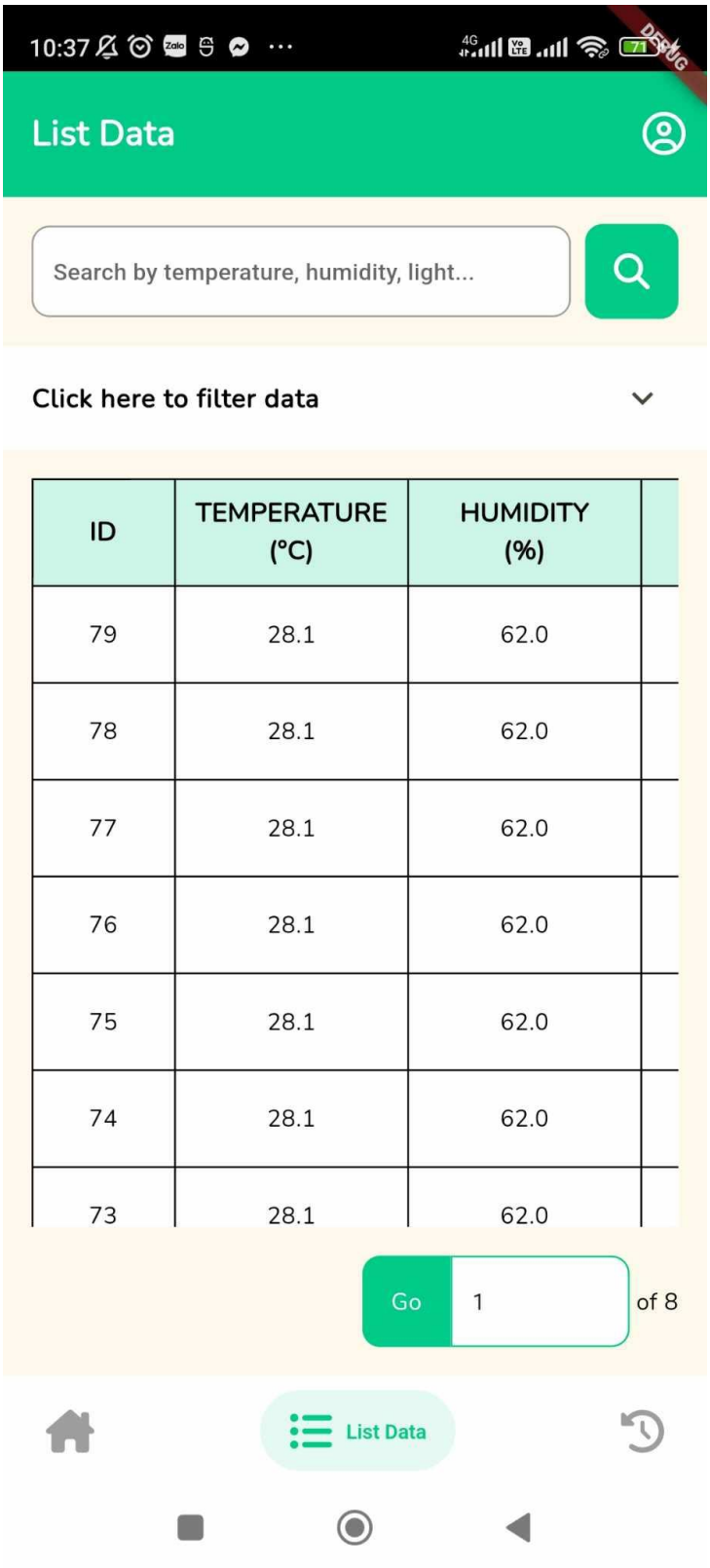

*Ảnh khi đang tắt thiết bị điện*

## 5.4. Lưu trữ và đưa ra dữ liệu cho người dùng

### 5.4.1. Data Sensor

Trang này chứa thông tin về các thông số như nhiệt độ, độ ẩm, và ánh sáng đã được lưu trữ từ trước đến nay.

Người dùng có thể xem lại lịch sử các chỉ số này, giúp theo dõi sự thay đổi và đánh giá tình trạng môi trường qua thời gian.

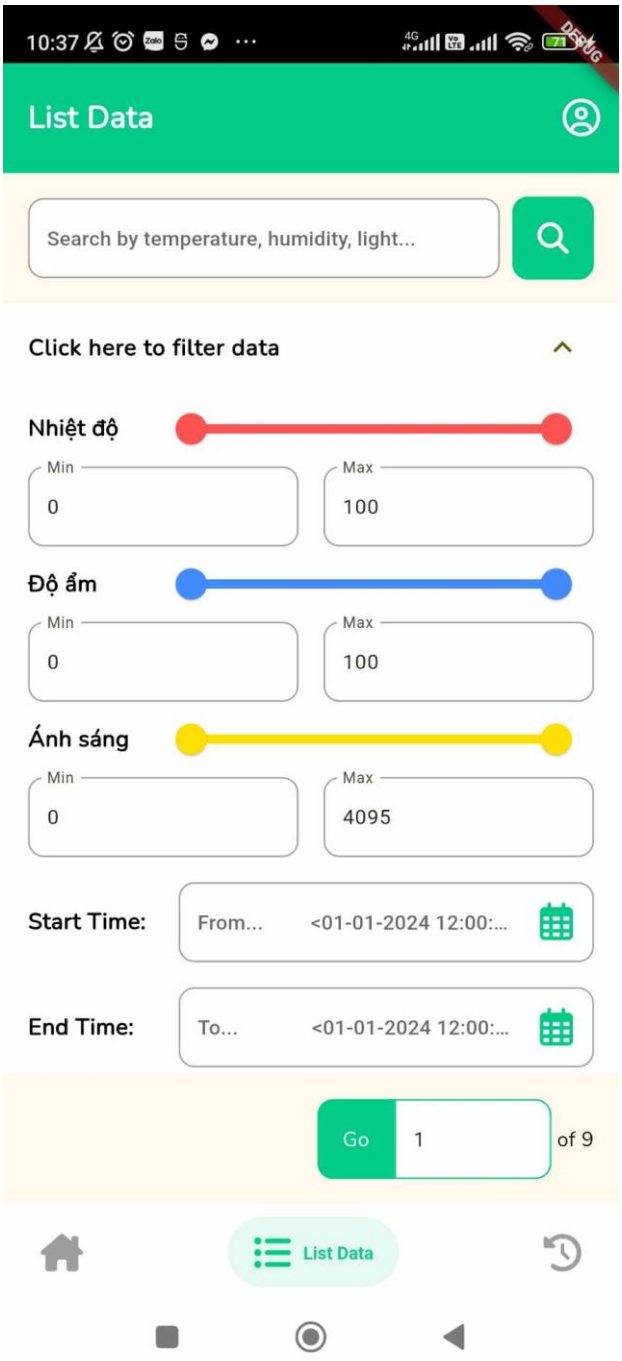



***Ảnh trang Data Sensor***

Ngoài ra còn có các chức năng khác như:

- Phân trang
- Tìm kiếm thông số theo loại chỉ số
- Tìm kiếm trong khoảng thời gian

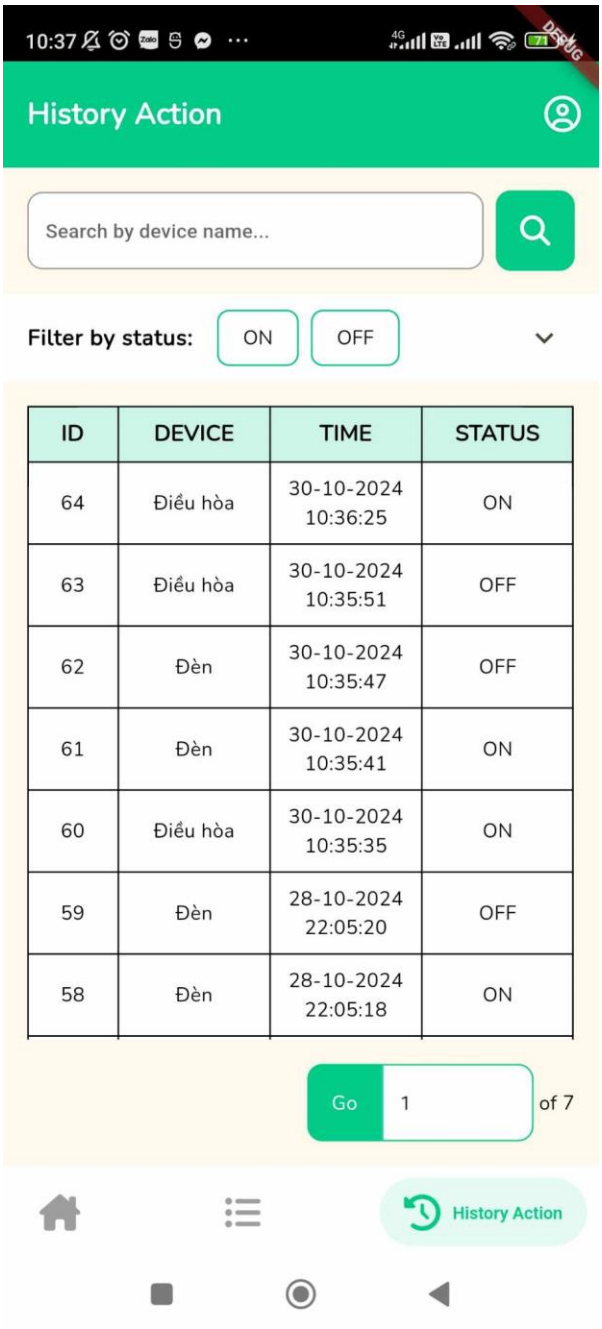
Dưới đây là ví dụ minh họa cho tìm kiếm và lọc




*Ảnh minh họa ví dụ*

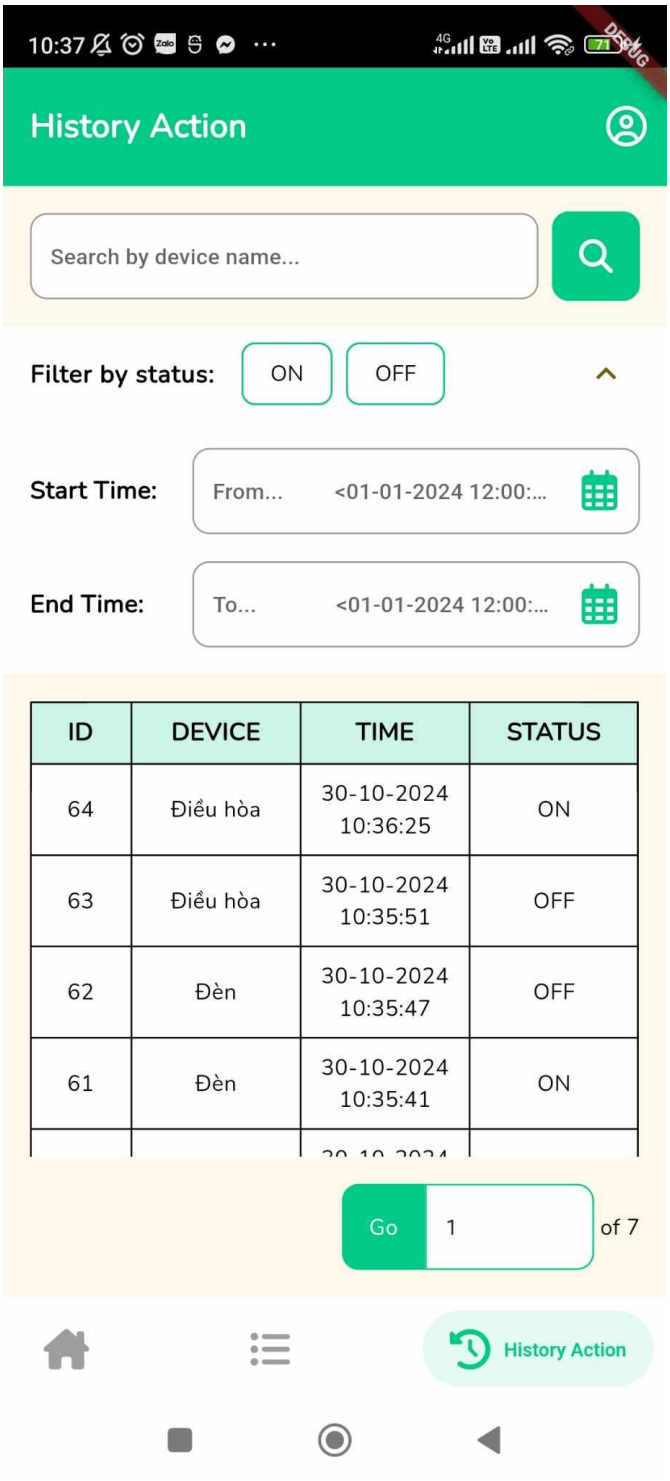
### 5.4.2. Action History

Trang này lưu trữ và hiển thị thông tin về lịch sử các hành động bật/tắt thiết bị từ trước đến nay. Người dùng có thể kiểm tra các thao tác điều khiển thiết bị đã thực hiện, đảm bảo rằng mọi thay đổi trạng thái của các thiết bị đều được ghi lại và có thể kiểm tra lại khi cần thiết.




***Ảnh về trang Action History***

Ngoài ra, còn có chức năng tìm kiếm theo khoảng thời gian và lọc theo tên thiết bị và phân trang. Dưới đây là ví dụ:




*Ảnh về ví dụ tìm kiếm và lọc của* **Action History**