HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN 1

BÁO CÁO HỌC PHẦN

ĐỀ TÀI: Xây dựng hệ thống quản lí, giám sát thiết bị Iot

Học phần	**: Iot và ứng dụng**
Giảng viên hướng dẫn	**: Nguyễn Quốc Uy**
Sinh viên thực hiện	**: Nguyễn Tiến Minh**
Mã sinh viên	**: B22DCCN539**

Hà Nội – 2025

LỜI CẢM ƠN

Lời đầu tiên, em xin chân thành cảm ơn các thầy cô tại Học viện Công nghệ Bưu chính Viễn thông nói chung và các thầy cô trong khoa Công nghệ Thông tin nói riêng, những người đã luôn tận tâm giảng dạy và truyền đạt cho em những kiến thức quý báu trong suốt quá trình học tập, đặc biệt là trong môn học IoT và Ứng dụng. Sự tận tụy và nhiệt huyết của các thầy cô là nền tảng quan trọng giúp em hoàn thành tốt bài tập lớn này.

Em xin gửi lời cảm ơn sâu sắc đến Thầy Nguyễn Quốc Uy, giảng viên phụ trách môn IoT và Ứng dụng, người đã nhiệt tình hướng dẫn và hỗ trợ em trong suốt quá trình thực hiện bài tập lớn. Những lời chỉ dẫn và góp ý từ Thầy đã giúp em hiểu rõ hơn về kiến thức chuyên môn, phát hiện và khắc phục những thiếu sót trong sản phẩm, từ đó hoàn thiện bài làm một cách tốt nhất. Em cũng rất trân trọng việc Thầy đã tạo điều kiện để sản phẩm được thử nghiệm thực tế, giúp em có thêm nhiều kinh nghiệm quý báu.

Do kiến thức và kinh nghiệm còn hạn chế, bài báo cáo không thể tránh khỏi những thiếu sót. Em rất mong nhận được những ý kiến đóng góp từ Thầy và các bạn để có thể tiếp tục cải thiện và nâng cao kỹ năng của mình trong tương lai.

Hà Nội, ngày … tháng … năm 2025
Sinh viên

Nguyễn Tiến Minh

MỤC LỤC

HÌNH ẢNH

Chương I. MÔ TẢ HỆ THỐNG

1.1.Mục đích

Hệ thống được xây dựng nhằm mục tiêu quản lý và giám sát các thông số môi trường trong thời gian thực, đồng thời hỗ trợ điều khiển thiết bị IoT thông qua giao diện web. Cụ thể, hệ thống hướng đến các mục đích chính:

❖ **Giám sát môi trường**:

- Hiển thị các thông số cảm biến nhiệt độ, độ ẩm và độ sáng theo thời gian thực.
- Trực quan hóa dữ liệu bằng biểu đồ giúp người dùng dễ dàng theo dõi và phân tích sự thay đổi của các thông số môi trường.

❖ **Điều khiển thiết bị IoT**:

- Cung cấp bảng điều khiển trên giao diện web cho phép bật/tắt quạt, điều hòa và đèn một cách nhanh chóng, thuận tiện.
- Ghi nhận và lưu trữ lịch sử các thao tác điều khiển để phục vụ việc theo dõi và đánh giá.

❖ **Quản lý thông tin người dùng và tài liệu liên quan**:

- Trang hồ sơ hiển thị thông tin cá nhân của sinh viên (ảnh, tên, mã sinh viên) và liên kết đến GitHub cũng như báo cáo dạng PDF.

❖ **Lưu trữ và tra cứu dữ liệu**:

- Ghi lại dữ liệu cảm biến (nhiệt độ, độ ẩm, độ sáng) theo thời gian để hỗ trợ người dùng tra cứu và phân tích sau này.
- Hiển thị lịch sử các hành động bật/tắt thiết bị, giúp minh bạch và dễ kiểm soát quá trình vận hành.

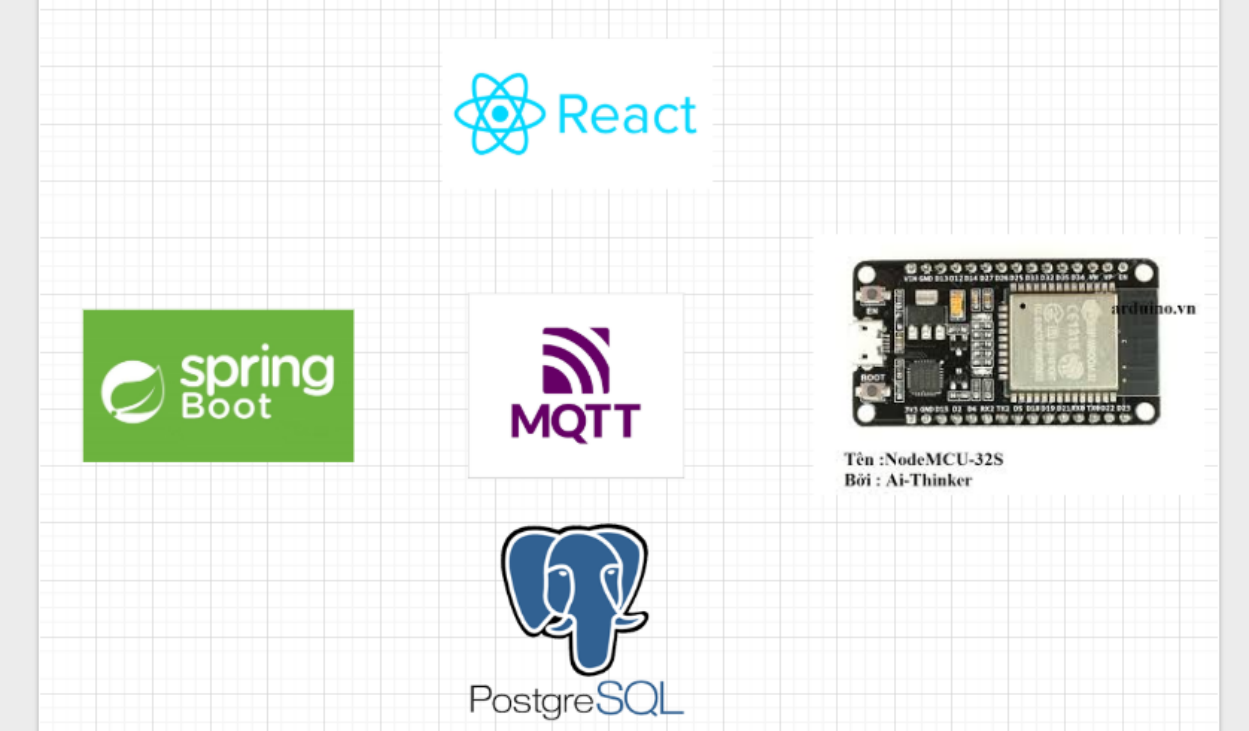
1.2.Các thiết bị sử dụng

a) Kit ESP32 Wifi
b) Cảm biến nhiệt độ, độ ẩm DHT11
c) Đèn led
d) Dây nối
e) Cảm biến ánh sáng BH1750
f) Breadboard MB, Điện trở

Hình 1: Các thiết bị sử dụng

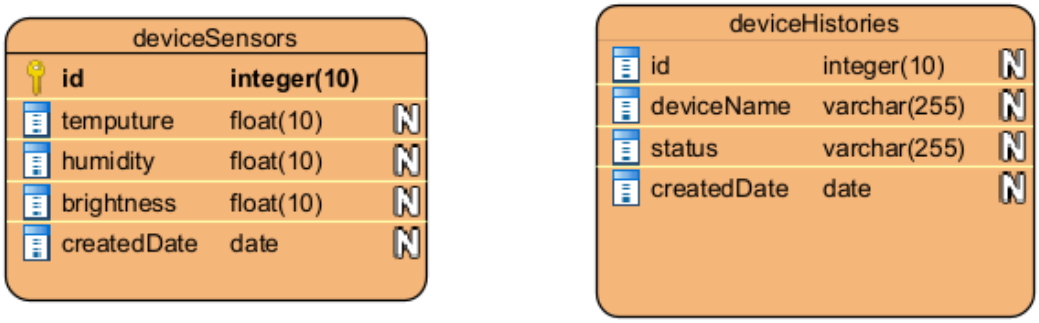
CHƯƠNG 2: Sơ đồ thiết kế

2.1.Kiến trúc tổng thể



Hình 2: Cơ sở dữ liệu

2.2.Cơ sở dữ liệu



Hình 3: Cơ sở dữ liệu

2.1.Bảng DeviceSensor

- Mục đích: Lưu dữ liệu thu thập được từ các cảm biến IoT.
- id (integer(10)): Khóa chính, dùng để định danh duy nhất cho mỗi bản ghi dữ liệu cảm biến.
- humidity (float): Lưu giá trị độ ẩm (%) đo được từ cảm biến DHT11 hoặc DHT22.
- light (float): Lưu cường độ ánh sáng đo được từ cảm biến BH1750, đơn vị là lux.
- temperature (float): Lưu giá trị nhiệt độ đo được từ cảm biến DHT11/DHT22, đơn vị °C.
- createdDate (datetime): Thời điểm ghi nhận dữ liệu từ cảm biến.

2.2.Bảng deviceHistories

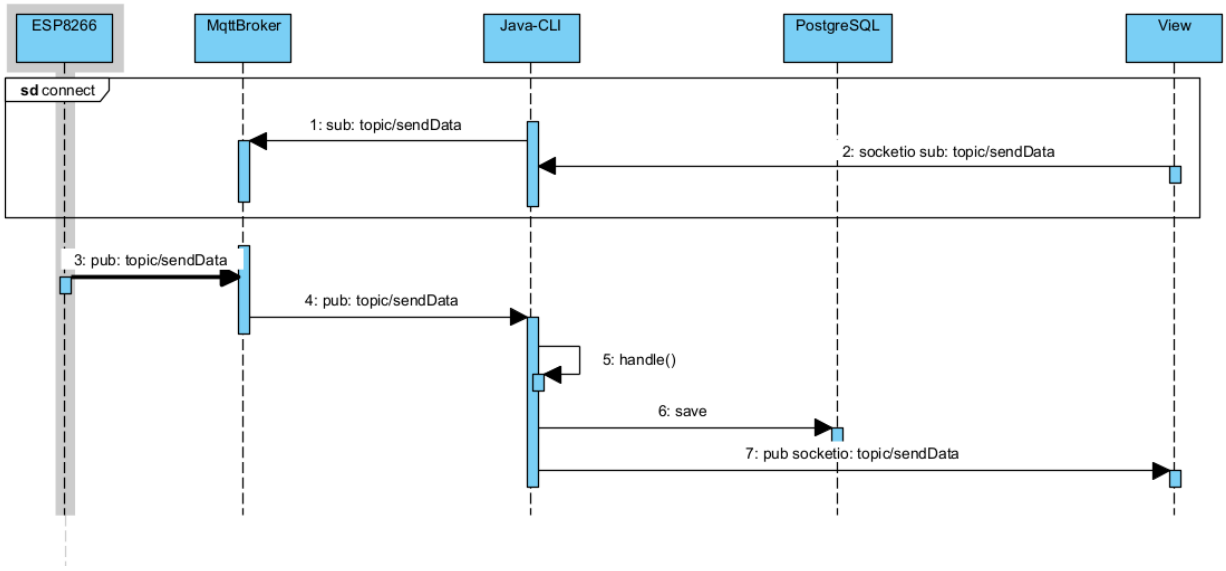
- Mục đích: Lưu các hành động điều khiển thiết bị (ví dụ: bật/tắt đèn, quạt...).
- id (varchar(255)): Khóa chính, định danh duy nhất cho mỗi hành động điều khiển.
- deviceName (varchar(255)): Tên thiết bị được điều khiển, ví dụ "LED" hoặc "FAN".
- status (varchar(255)): Hành động được thực hiện trên thiết bị, ví dụ "ON" hoặc "OFF".
- createdDate (datetime): Thời điểm hành động được thực hiện.

2.3.Sequence

3.3.1. Sequence hiển thị thông số cảm biến

- **Scenario chuẩn:**

1. Lớp Java-CLI subcrice vào topic "topic/sendData" của Lớp MqttBroker
2. Lớp View subcrice vào topic "topic/sendData" của lớp Java-CLI.
3. Thiết bị đo các thông số publish dữ liệu đo được vào topic "topic/sendData" lớp MqttBroker.
4. Lớp MqttBroker deliver dữ liệu vào lớp Java-Client dữ liệu nhận được.
5. Lớp Java-Client thực hiện phương thức handle() để lưu và hiển thị.
6. Phương thức handle() gọi đến lớp PostgreSQL thực hiện save vào DB.
7. Phương thức handle() public dữ liệu thời gian thực Websocket đến lớp View hiển thị đến người dùng.

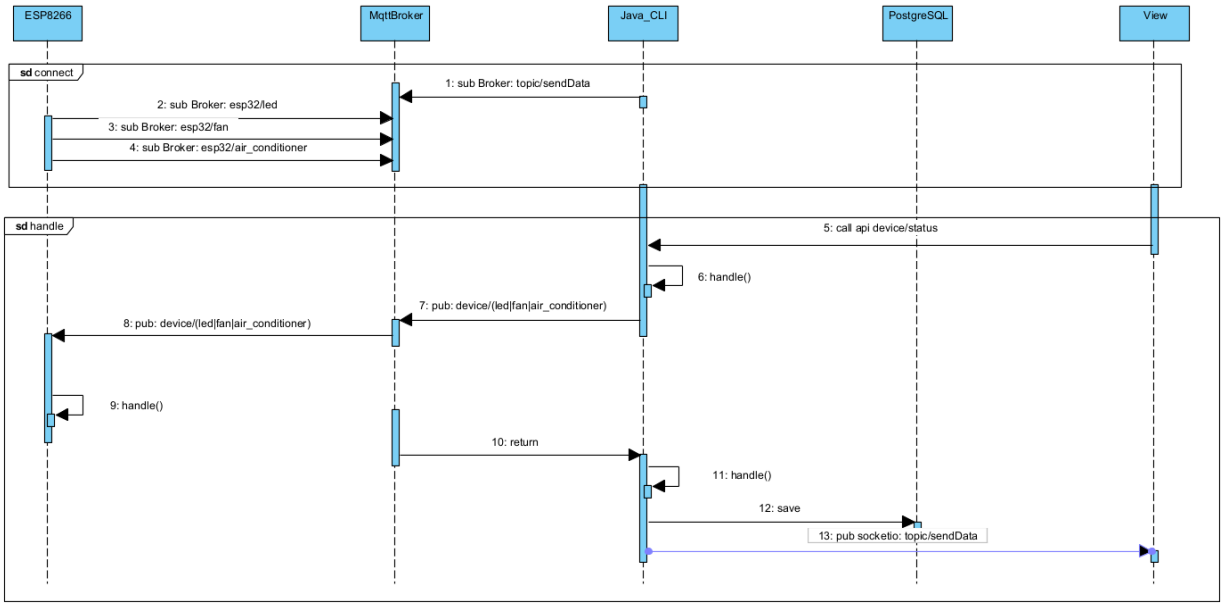


Hình 4: Sequence hiển thị thông số cảm biến

3.3.2. Sequence điều khiển thiết bị

- **Scenario chuẩn:**

1. Lớp ESP8266 subcribe vào các topic esp32/led, esp32/fan, esp32/air_conditioner của lớp MqttBroker để nhận yêu cầu điều khiển thiết bị.
2. Lớp Java-CLI subcribe vào topic topic/sendData của lớp MqttBroker để nhận dữ liệu phản hồi và trạng thái thiết bị.
3. Lớp DeviceView gọi API device/status tới lớp Java-CLI để lấy thông tin trạng thái thiết bị.
4. Lớp Java-CLI thực hiện phương thức handle().
5. Phương thức handle() publish yêu cầu điều khiển vào topic device/(led|fan|air_conditioner) của lớp MqttBroker.
6. Lớp MqttBroker chuyển yêu cầu này đến lớp ESP8266.
7. Lớp ESP8266 thực hiện phương thức handle() để xử lý yêu cầu điều khiển thiết bị.
8. Lớp ESP8266 publish phản hồi vào topic device/(led|fan|air_conditioner) trở lại lớp MqttBroker.
9. Lớp MqttBroker chuyển phản hồi đến lớp Java-CLI thông qua phương thức handle().
10. Lớp Java-CLI trả kết quả xử lý về sau khi thực hiện xong.
11. Lớp Java-CLI tiếp tục thực hiện phương thức handle() để lưu dữ liệu xuống lớp PostgreSQL.
12. Lớp PostgreSQL lưu thông tin phản hồi thành công.
13. Lớp Java-CLI publish dữ liệu tới View thông qua socket với topic sendData.



Hình 5: Sequence điều khiển thiết bị

2.4.Activity

3.4.1. Activity hiển thị thông số cảm biến

a) Người dùng click bật/tắt thiết bị trên giao diện (Client)

- Người dùng thao tác trực tiếp trên ứng dụng (web/mobile), ví dụ nhấn nút "Bật quạt".
- Ứng dụng client thực hiện đồng thời 2 hành động:

- Gửi API request đến Java Server theo endpoint /api/v1/fan/on để ghi nhận lệnh.
- Gửi một message MQTT đến topic esp32/fan với payload chứa thông tin thiết bị và trạng thái (device=fan, status=on).

b) MQTT Broker nhận và chuyển tiếp yêu cầu

- MQTT Broker tiếp nhận message từ Client.
- Broker sẽ xác định topic đích và chuyển tiếp thông điệp điều khiển đến ESP32 (thiết bị đã subscribe topic esp32/fan).

c) ESP32 nhận lệnh và thực thi

- ESP32 nhận thông điệp từ MQTT Broker.
- Giải mã thông tin trong payload để biết cần điều khiển thiết bị nào và trạng thái mong muốn (ví dụ: fan=on).
- Thực hiện điều khiển phần cứng, trong trường hợp này là bật quạt.
- Sau khi hoàn thành, ESP32 gửi một thông điệp phản hồi lên topic status/device/fan nhằm xác nhận trạng thái thiết bị.

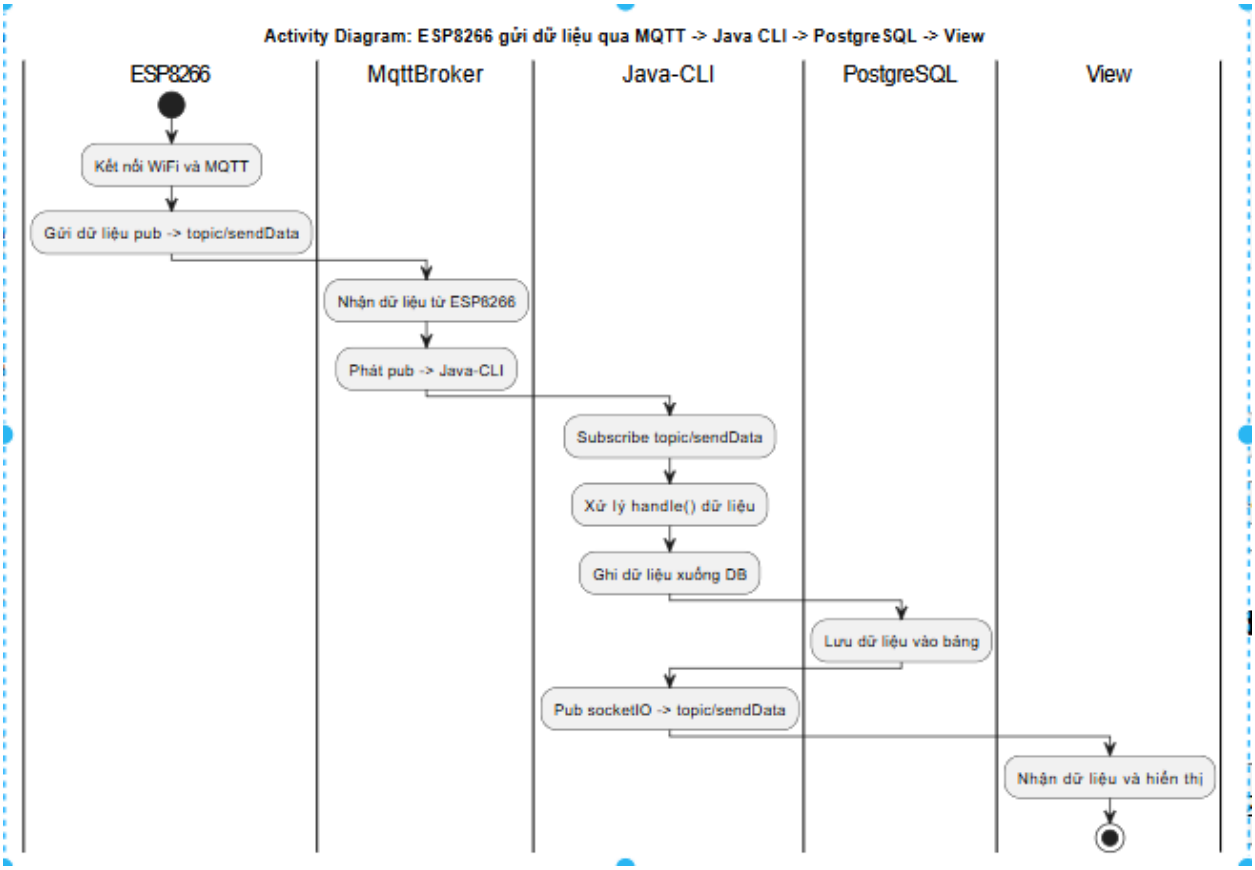
d) MQTT Broker nhận phản hồi và gửi đến Java Server

- MQTT Broker nhận thông điệp phản hồi từ ESP32.
- Thông điệp này được phân phối đến Java Server (đã subscribe topic status/device/fan).

- Java Server xử lý thông tin phản hồi: lưu xuống cơ sở dữ liệu, ghi log, hoặc thực hiện các nghiệp vụ khác.

e) Java Server cập nhật giao diện (View)

- Sau khi xử lý, Java Server gửi dữ liệu trạng thái mới về Client.
- Giao diện (View) được cập nhật để người dùng thấy kết quả theo thời gian thực, ví dụ hiển thị trạng thái "Quạt đang bật".



Hình 6: Activity hiển thị thông số cảm biến

3.4.2. Activity điều khiển thiết bị

- Người dùng click bật/tắt thiết bị trên giao diện (Client)

• Người dùng nhấn nút điều khiển, ví dụ bật/tắt quạt.

• Ứng dụng client gửi API request /api/v1/fan/on và đồng thời gửi message MQTT đến topic esp32/fan.

a) MQTT Broker nhận và chuyển tiếp yêu cầu

• MQTT Broker tiếp nhận message từ Client.

• Broker chuyển tiếp message này đến ESP32 (thiết bị đã subscribe topic esp32/fan).

b) ESP32 nhận lệnh và thực thi

• ESP32 nhận thông điệp điều khiển từ Broker.

• Đọc param: device = fan, status = on.
• Thực thi lệnh điều khiển phần cứng (“Bật quạt”).
• Sau khi thực thi, ESP32 gửi thông điệp xác nhận trạng thái lên topic status/device/fan.

c) MQTT Broker nhận phản hồi và gửi đến Java Server

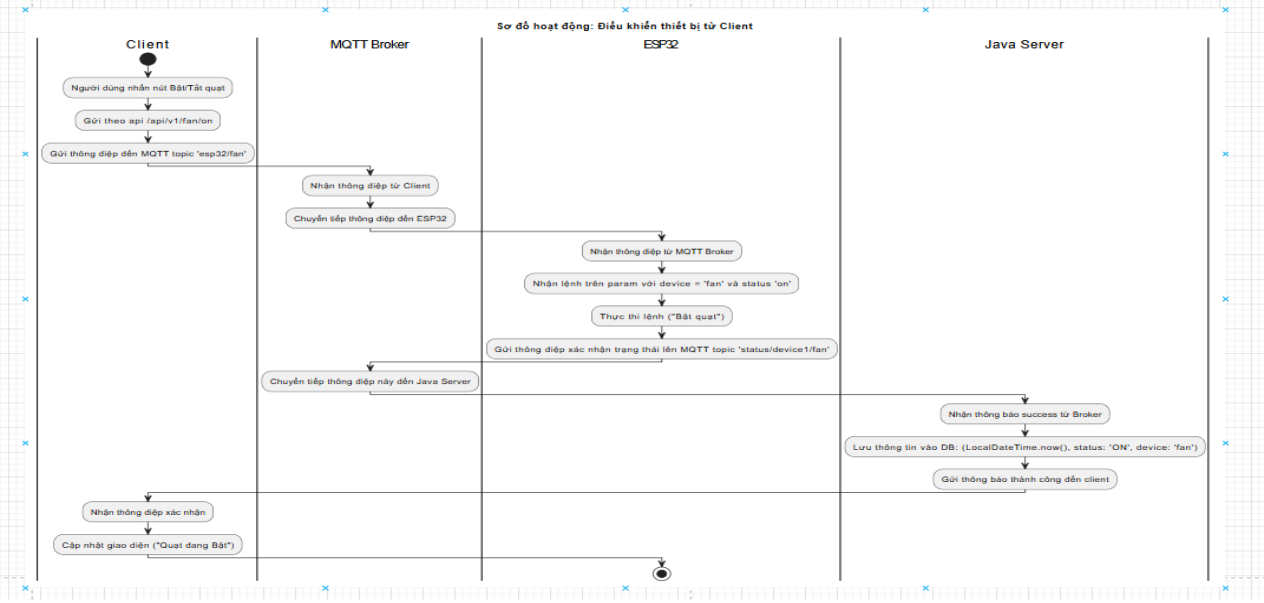
• Broker nhận thông điệp phản hồi từ ESP32.
• Chuyển tiếp thông điệp này đến Java Server (subscriber).

d) Java Server xử lý phản hồi

• Nhận thông báo thành công từ Broker.
• Lưu kết quả vào cơ sở dữ liệu: (LocalDateTime.now, status = ON, device = fan).
• Gửi thông báo “Success” về cho Client.

e) Client cập nhật giao diện

• Nhận thông báo xác nhận từ Server.
• Giao diện hiển thị trạng thái mới của thiết bị (ví dụ: “Quạt đang Bật”).



Hình 7: Activity điều khiển thiết bị

2.5 Giải thích code

```
#include "DHT.h"
#include <WiFi.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>
#include <BH1750.h>
#include <Wire.h>

#define DHTPIN 4  // cổng trên esp kết nối với sensor
#define DHTTYPE DHT11
#define LEDPIN 5
#define FANPIN 16
#define AIR_CONDITIONERPIN 17
```

Hình 8: Khai báo thư viện và các cổng

```
// WiFi Config
const char* WIFI_SSID = "Minh";
const char* WIFI_PASS = "123456789";

// MQTT Config
const char* MQTT_SERVER = "10.44.152.251";
const int MQTT_PORT = 1883;
const char* username = "user1";
const char* password = "123";
const int qos = 1;
const String topicLed = "esp32/led";
const String topicFan = "esp32/fan";
const String topicAirConditioner = "esp32/air_conditioner";
const String topicInitDevice = "initDevice";
const String topicSetupDevice = "setupDevice";

// MQTT client sử dụng WiFi
WiFiClient wifiClient;
PubSubClient mqttClient(wifiClient);

// dht
DHT dht(DHTPIN, DHTTYPE);

// bh1750
BH1750 lightMeter;
```

Hình 9: Khai báo hằng số

```
void mqttCallback(char* topic, byte* payload, unsigned int length) {
  String topicStr = String(topic);
  String message;

  for (unsigned int i = 0; i < length; i++) {
    message += (char)payload[i];
  }
  JsonDocument dataReceive;
  deserializeJson(dataReceive, message);
  if (topicStr != topicInitDevice) {
    String status = dataReceive["status"];
    if (topicStr == topicLed) {
      digitalWrite(LEDPIN, status == "off" ? LOW : HIGH);
    } else if (topicStr == topicFan) {
      digitalWrite(FANPIN, status == "off" ? LOW : HIGH);
    } else if (topicStr == topicAirConditioner) {
      digitalWrite(AIR_CONDITIONERPIN, status == "off" ? LOW : HIGH);
    }
    String requestId = dataReceive["requestId"];
    if (requestId != NULL) {
      JsonDocument response;
      response["id"] = requestId;
      size_t len = measureJson(response) + 1;
      char output[len];
      serializeJson(response, output, len);
      mqttClient.publish("control-device", output);
    }
  } else if (topicStr == topicInitDevice) {
    digitalWrite(LEDPIN, dataReceive["led"] == "off" ? LOW : HIGH);
    digitalWrite(FANPIN, dataReceive["fan"] == "off" ? LOW : HIGH);
    digitalWrite(AIR_CONDITIONERPIN, dataReceive["air_conditioner"] == "off" ? LOW : HIGH);
  }
}
```

Hình 10: Callback khi nhận message

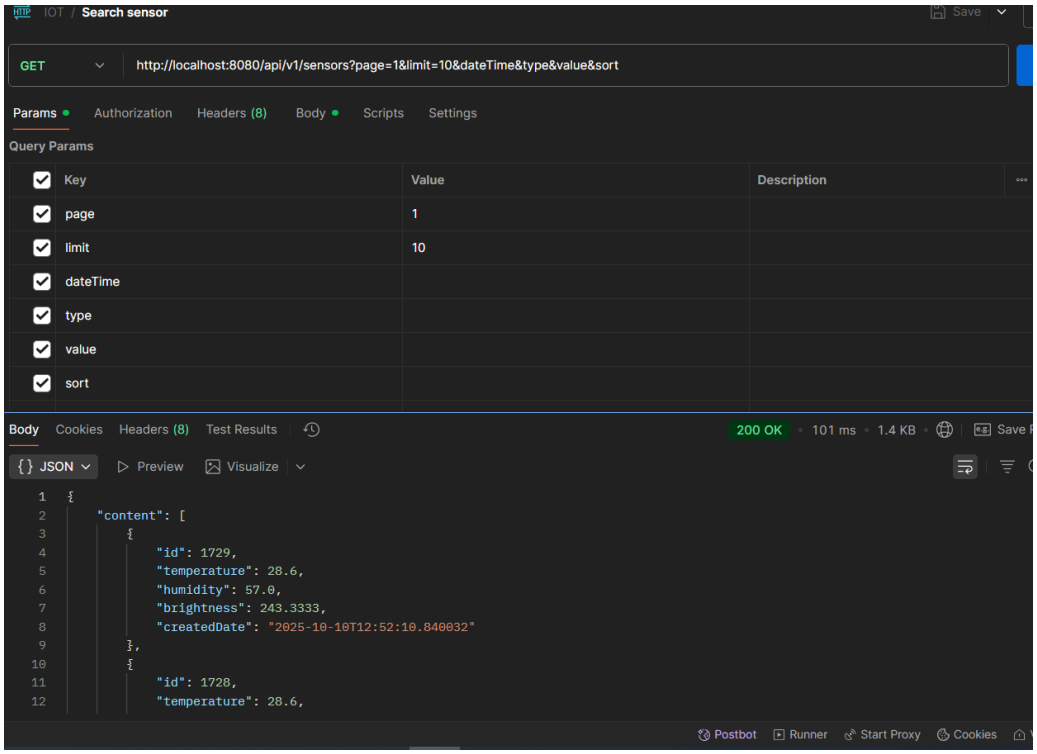
```
void setup() {
  Serial.begin(9600);
  // kết nối wifi
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.println("Lỗi kết nối wifi");
  }
  Serial.print("\n WiFi Connected. IP: ");
  Serial.println(WiFi.localIP());

  // Cấu hình MQTT
  mqttClient.setServer(MQTT_SERVER, MQTT_PORT);
  mqttClient.setCallback(mqttCallback);
  mqttClient.setKeepAlive(60);
  mqttClient.setSocketTimeout(30);
  String clientId = "ESP32Client";
  while (!mqttClient.connected()) {
    Serial.print("Kết nối MQTT...");
    if (mqttClient.connect(clientId.c_str(), username, password)) {
      Serial.println("Thành công!");
      mqttClient.subscribe(topicLed.c_str(), qos);
      mqttClient.subscribe(topicFan.c_str(), qos);
      mqttClient.subscribe(topicAirConditioner.c_str(), qos);
      mqttClient.subscribe(topicInitDevice.c_str(), qos);
    } else {
      Serial.print("Thất bại, lỗi: ");
      Serial.println(mqttClient.state());
      delay(2000);
    }
  }
  dht.begin();
  pinMode(LEDPIN, OUTPUT);
  pinMode(FANPIN, OUTPUT);
  pinMode(AIR_CONDITIONERPIN, OUTPUT);

  mqttClient.publish(topicSetupDevice.c_str(), NULL);
  // cấu hình bh1750
  Wire.begin(21, 22);
  lightMeter.begin();
}
```

Hình 11: Setup giá trị

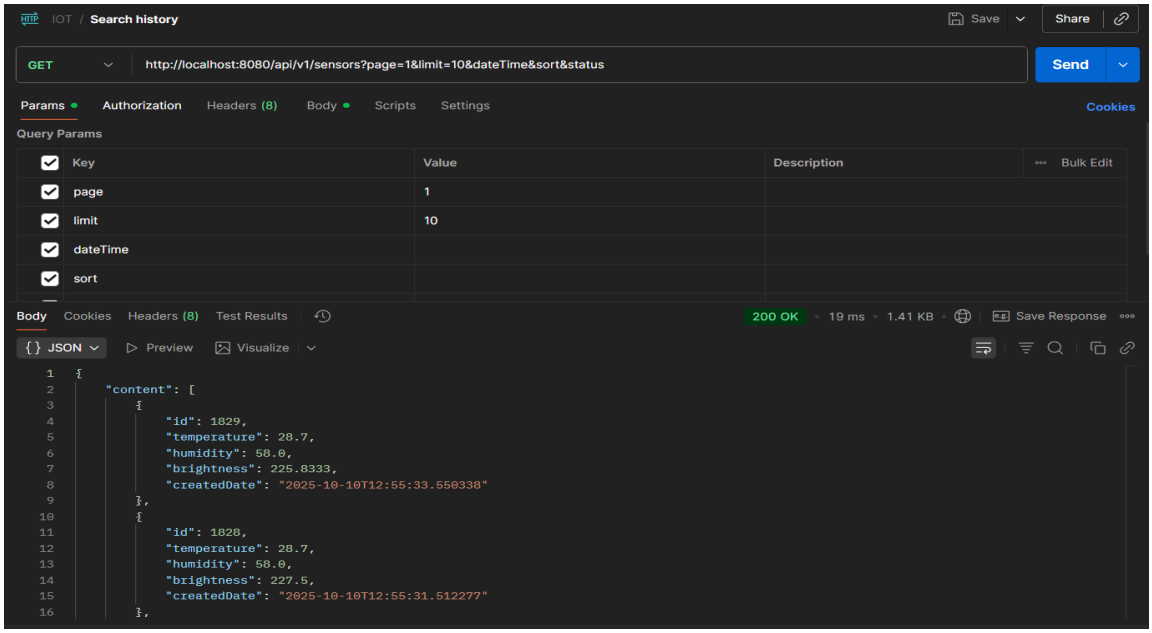
```
13
14  void loop() {
15    delay(2000);
16    if (WiFi.status() != WL_CONNECTED) {
17      WiFi.disconnect();
18      WiFi.begin(WIFI_SSID, WIFI_PASS);
19      unsigned long startAttemptTime = millis();
20      while (WiFi.status() != WL_CONNECTED && millis() - startAttemptTime < 5000) {
21        delay(500);
22        Serial.print(".");
23      }
24    }
25    if (!mqttClient.connected()) {
26      mqttClient.setServer(MQTT_SERVER, MQTT_PORT);
27      mqttClient.setCallback(mqttCallback);
28      mqttClient.setKeepAlive(60);
29      mqttClient.setSocketTimeout(30);
30      String clientId = "ESP32Client";
31      while (!mqttClient.connected()) {
32        Serial.print("Kết nối MQTT...");
33        if (mqttClient.connect(clientId.c_str(), username, password)) {
34          Serial.println("Thành công!");
35          mqttClient.subscribe(topicLed.c_str(), qos);
36          mqttClient.subscribe(topicFan.c_str(), qos);
37          mqttClient.subscribe(topicAirConditioner.c_str(), qos);
38          mqttClient.subscribe(topicInitDevice.c_str(), qos);
39          mqttClient.publish(topicSetupDevice.c_str(), NULL);
40        } else {
41          Serial.print("Thất bại, lỗi: ");
42          Serial.println(mqttClient.state());
43          delay(2000);
44        }
45      }
46    }
47    JsonDocument data;
48    // Độ ẩm
49    float h = dht.readHumidity();
50    // Nhiệt độ
51    float t = dht.readTemperature();
52    // có ánh sáng không
53    float lux = lightMeter.readLightLevel();
54    data["humidity"] = h;
```

Hình 12: Gửi data sensor và xử lý khi mất kết nối

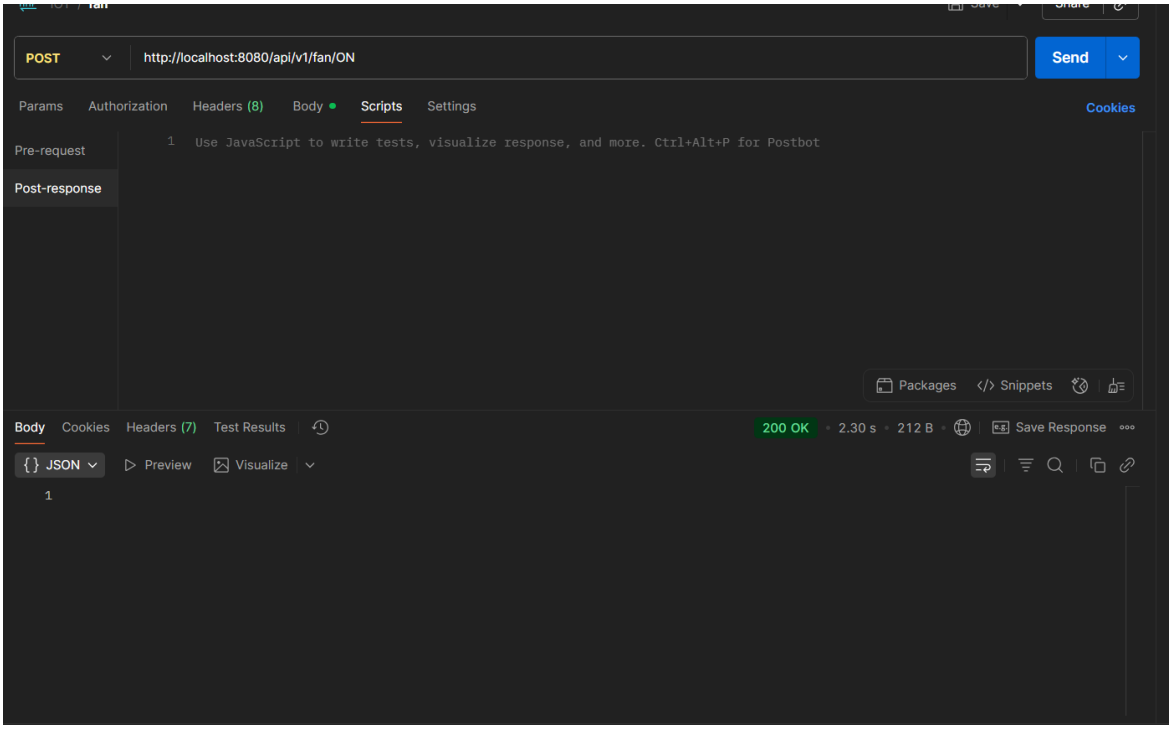
2.6. Api Doc



Hình 13: Api lấy data sensor



Hình 14: Api lấy thông tin lịch sử bật tắt thiết bị

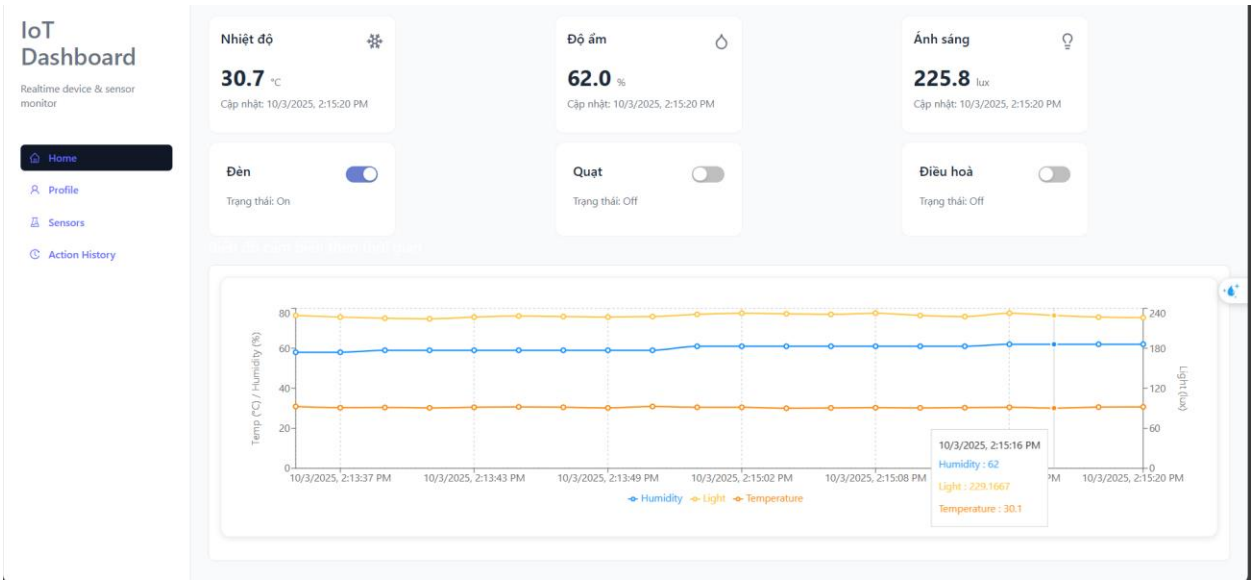


Hình 15: Api bật tắt thiết bị

Chương III. Giao diện hệ thống

3.1.Home Page

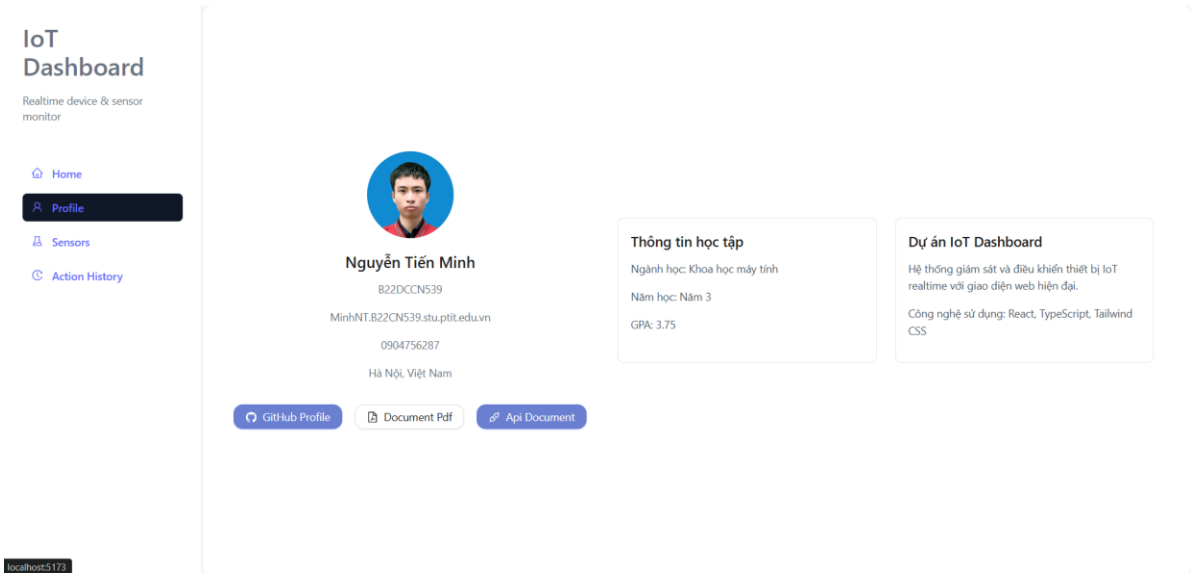
- Hiển thị các thông số nhiệt độ, độ ẩm, ánh sáng thời gian thức đến cho người dùng từ thiết bị điều khiển và cảm biến.
- Hiển thị biểu đồ tổng quan của các thông số đến người dùng để dễ dàng xem sự biến đổi.
- Giao diện điều khiển ON/OFF các thiết bị.



Hình 16: Giao diện HomePage

3.2.Profile Page

Hiển thị thông tin cá nhân và thông tin học tập của người điều khiển cùng với các thông tin về dự án như: Github, Swapgger, Pdf.

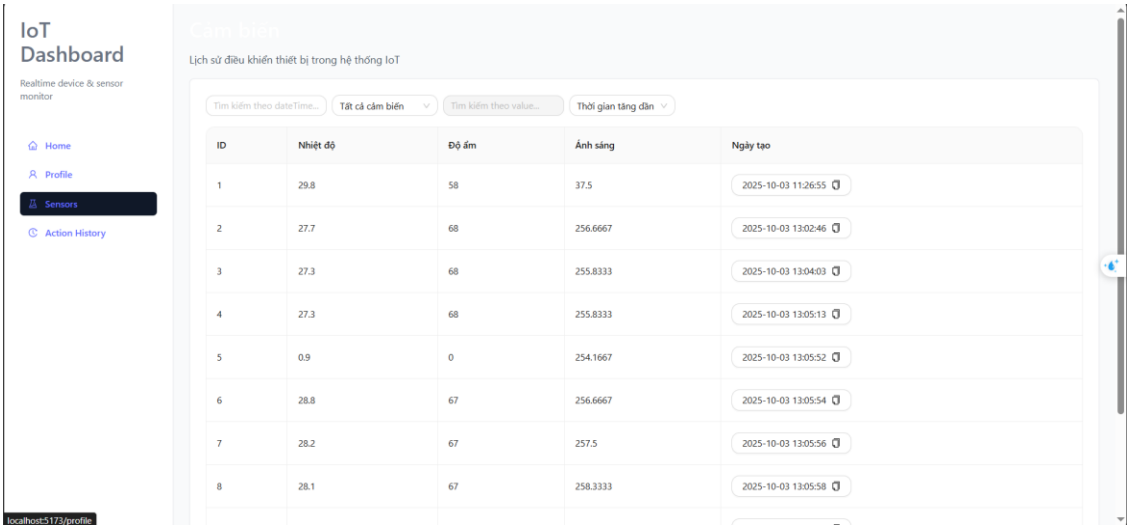


Hình 17: Giao diện Profile Page

3.3.Data Sensor Page

Giao diện hiển thị lịch sử biến đổi các thông số đo từ thiết bị.

Thanh thực hiện tìm kiếm theo yêu cầu như: theo thời gian, số liệu chính xác từ người dùng.

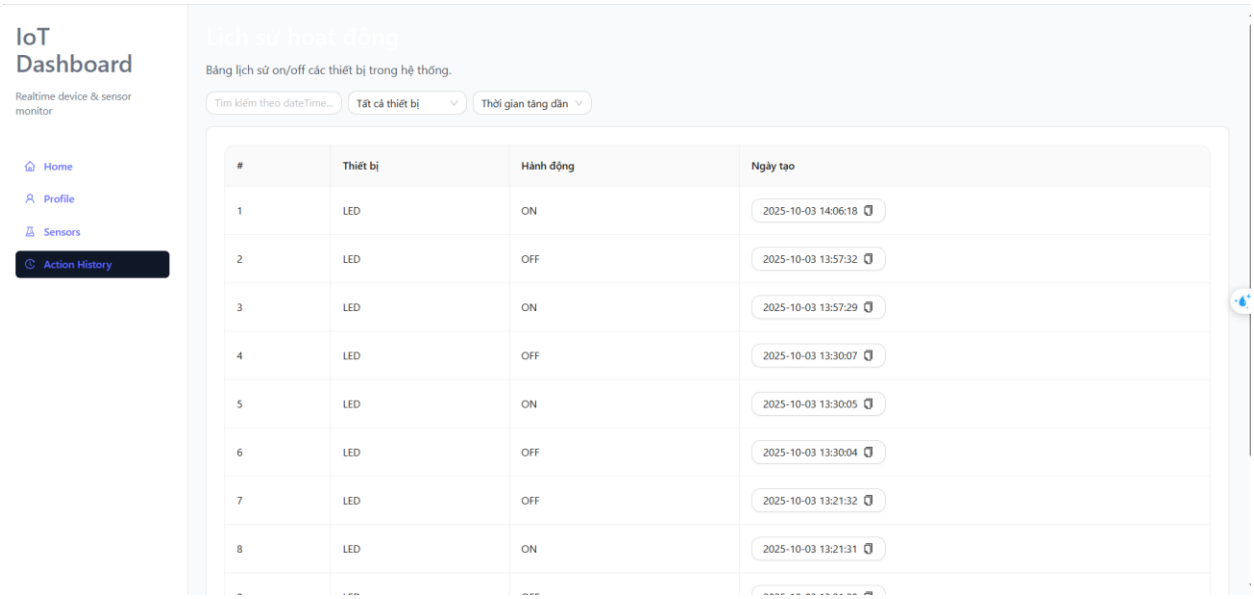


Hình 18: Giao diện Data Sensor

3.4.Action Device History Page

Giao diện hiển thị lịch sử điều khiển các thiết bị.

Thanh thực hiện tìm kiếm theo yêu cầu như: theo thời gian, thiết bị chính xác từ người dùng.



Hình 19: Giao diện Action Device History

CHƯƠNG 4: Kết quả thực hiện

4.1.**Về chức năng**

- Hệ thống đã thực hiện đúng các chức năng mong đợi:
- Người dùng có thể bật/tắt thiết bị trực tiếp từ giao diện.
- Lệnh điều khiển được gửi đến ESP32 thông qua MQTT Broker.
- ESP32 thực thi lệnh và phản hồi lại trạng thái cho hệ thống.
- Server xử lý phản hồi, lưu vào cơ sở dữ liệu
- Kết quả được trả lại giao diện, giúp người dùng quan sát trạng thái thiết bị theo thời gian thực.

4.2.**Về tính chính xác**

- Các dữ liệu gửi/nhận qua MQTT đảm bảo đúng định dạng JSON.
- Trạng thái thiết bị được cập nhật chính xác và đồng bộ giữa ESP32 – Server – Client.

4.3.**Về hiệu năng**

- Quá trình điều khiển và phản hồi diễn ra nhanh (gần như thời gian thực).
- MQTT giúp giảm tải băng thông, tối ưu cho IoT.

4.4.**Về độ tin cậy**

- Hệ thống có thể xử lý cả tình huống thành công và lỗi, không gây gián đoạn cho người dùng.
- Các bản ghi được lưu vào DB giúp dễ dàng kiểm tra lại lịch sử hoạt động.

4.5.**Hạn chế**

- Giao diện chỉ dừng ở mức cơ bản, cần cải thiện để trực quan hơn.

4.6.**Kết luận**

- Hệ thống đã đáp ứng tốt yêu cầu cơ bản về điều khiển và giám sát thiết bị IoT qua ESP32 MQTT.
- Có thể mở rộng thêm bảo mật, UI/UX và tích hợp thông báo để hệ thống hoàn thiện hơn.