HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN

--------oOo--------

Môn IOT và ứng dụng

Báo Cáo Bài Tập Lớn

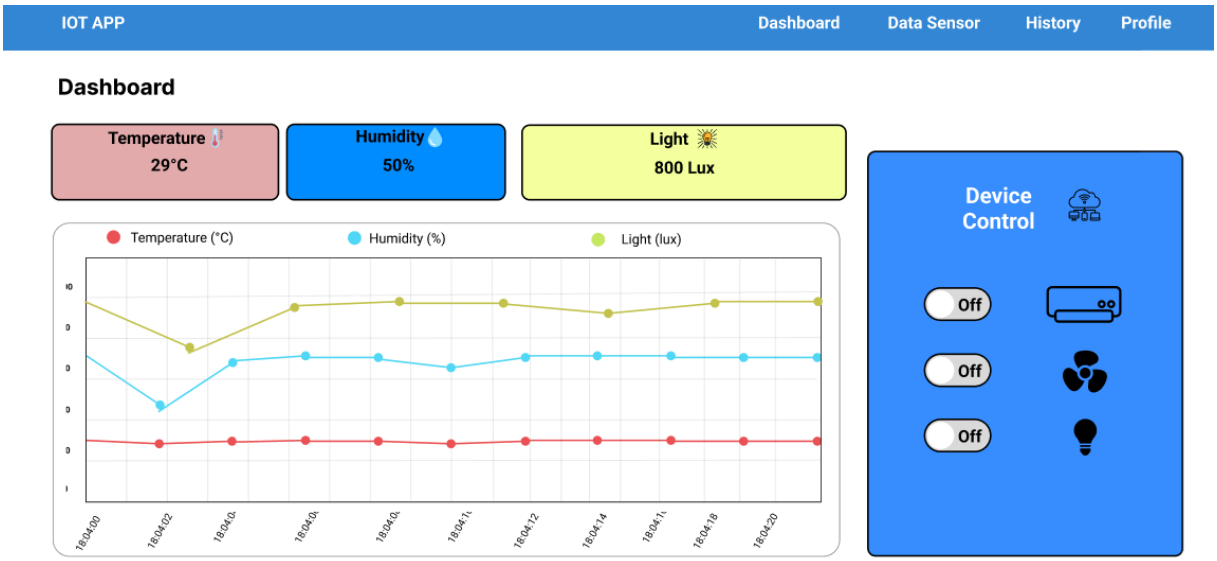
Giảng viên	**: TS. Nguyễn Quốc Uy**
Nhóm môn học	**: 15**
Họ và tên	**: Nguyễn Bá Cường**
Mã sinh viên	**: B22DCCN096**
Lớp	**: D22HTTT-06**

Hà Nội - 2025

I. Tổng quan hệ thống

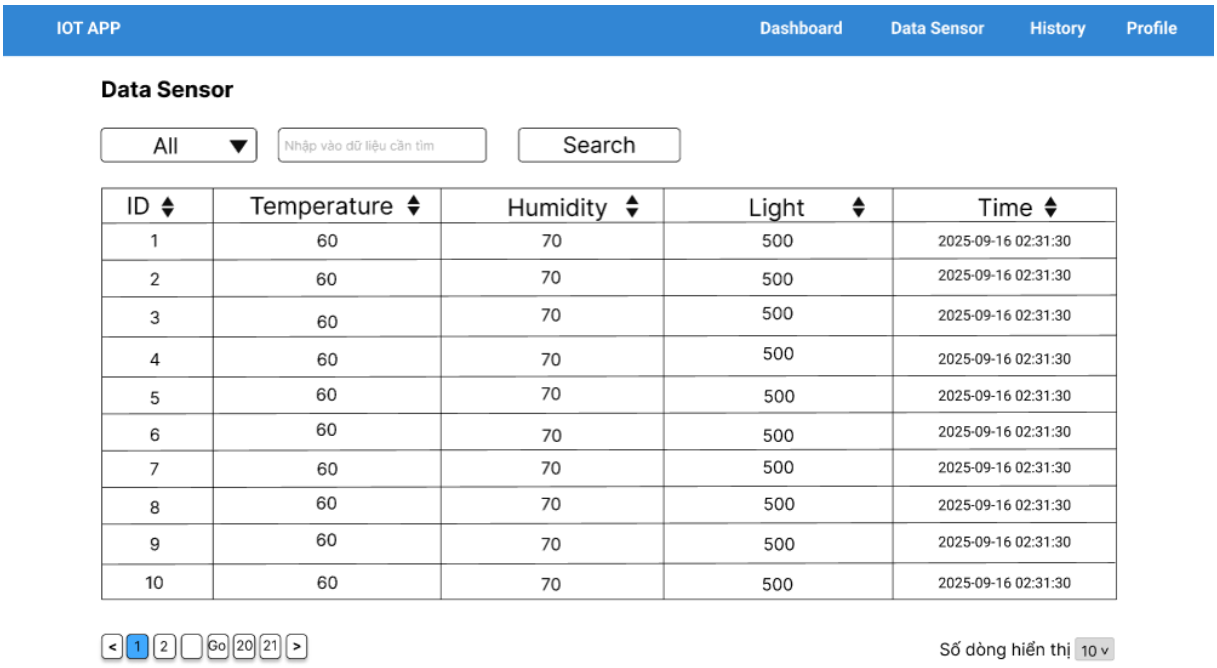
1. Giao diện figma

Trang Dashboard: hiển thị thông tin tổng quan, nhiệt độ, độ ẩm, ánh sáng, các đồ thị biểu diễn các chỉ số, các nút on off thiết bị



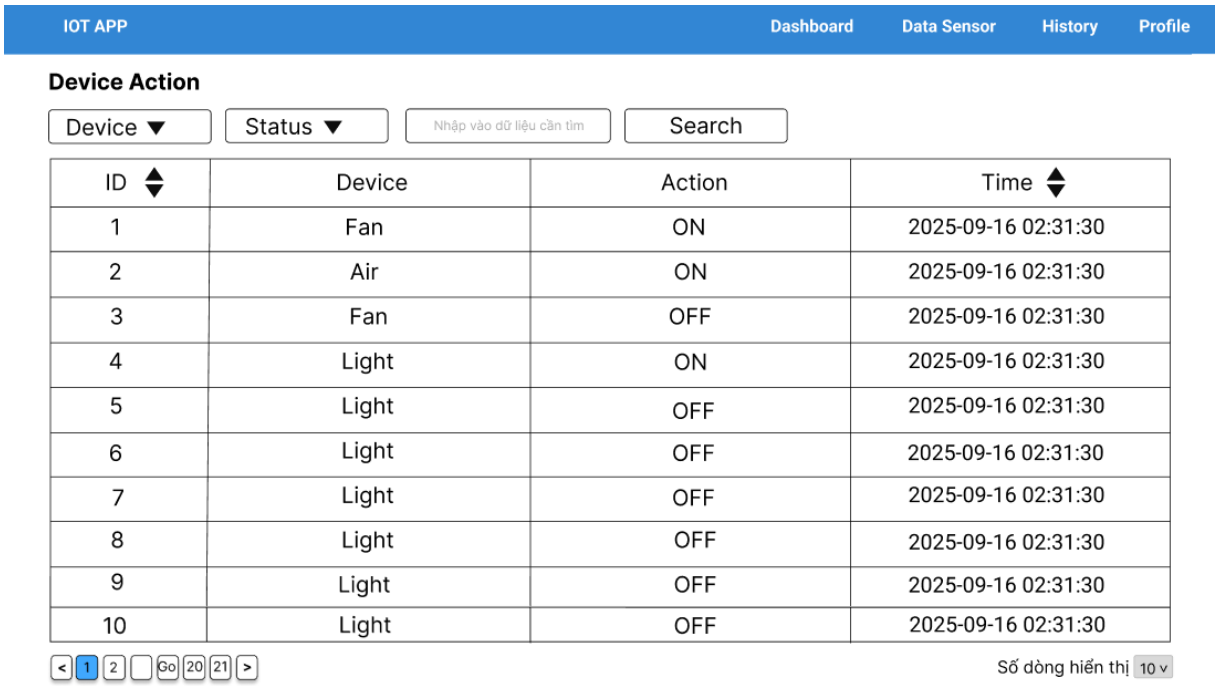
Hình 1: Trang Dashboard figma

Trang DataSensor: hiển thị thông tin cảm biến thu thập theo thời gian thực, mỗi 2 giây cập nhật 1 lần. Có các trường để lọc thông tin theo yêu cầu của người dùng.



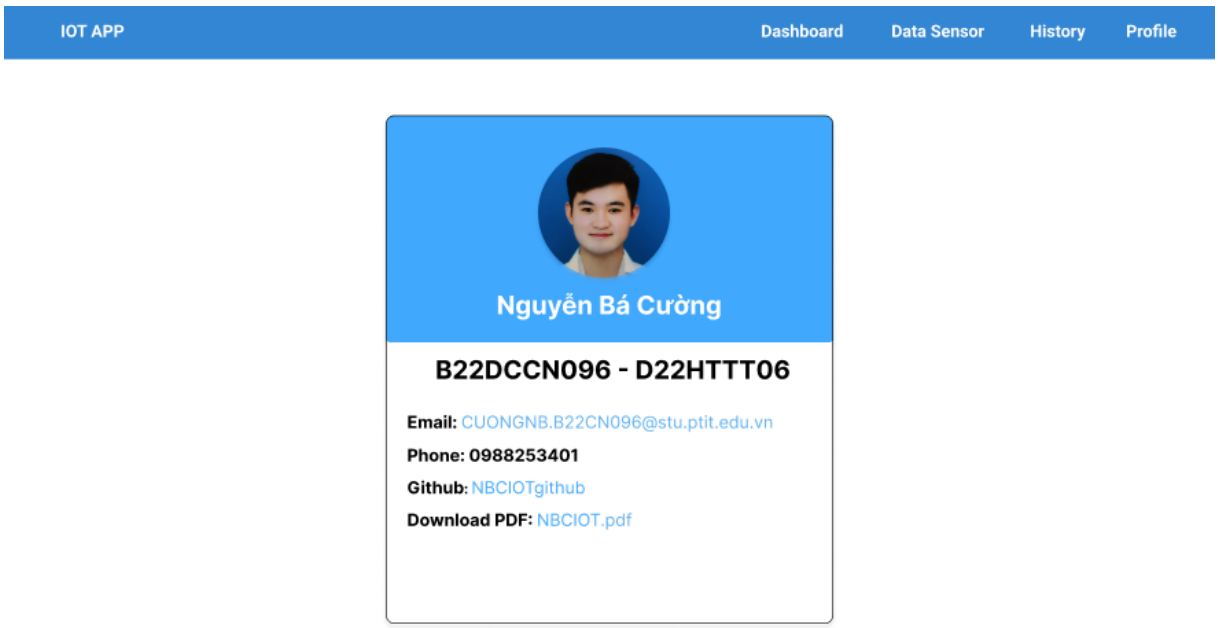
Hình 2: Trang Data Sensor figma

Trang Action History: hiển thị lịch sử hoạt động của các thiết bị, có các trường để lọc thông tin theo yêu cầu.



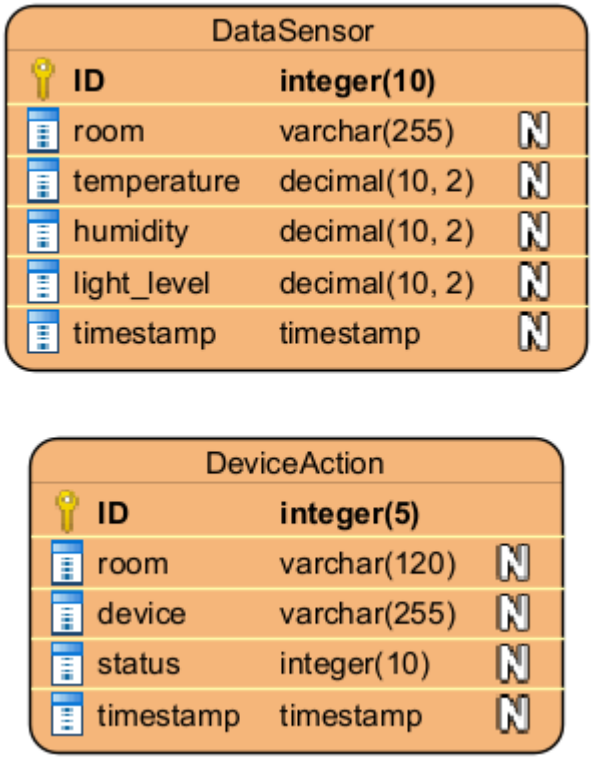
Hình 3: Trang Device Action figma

Trang profile: hiển thị thông tin sinh viên



Hình 4: Trang Profile figma

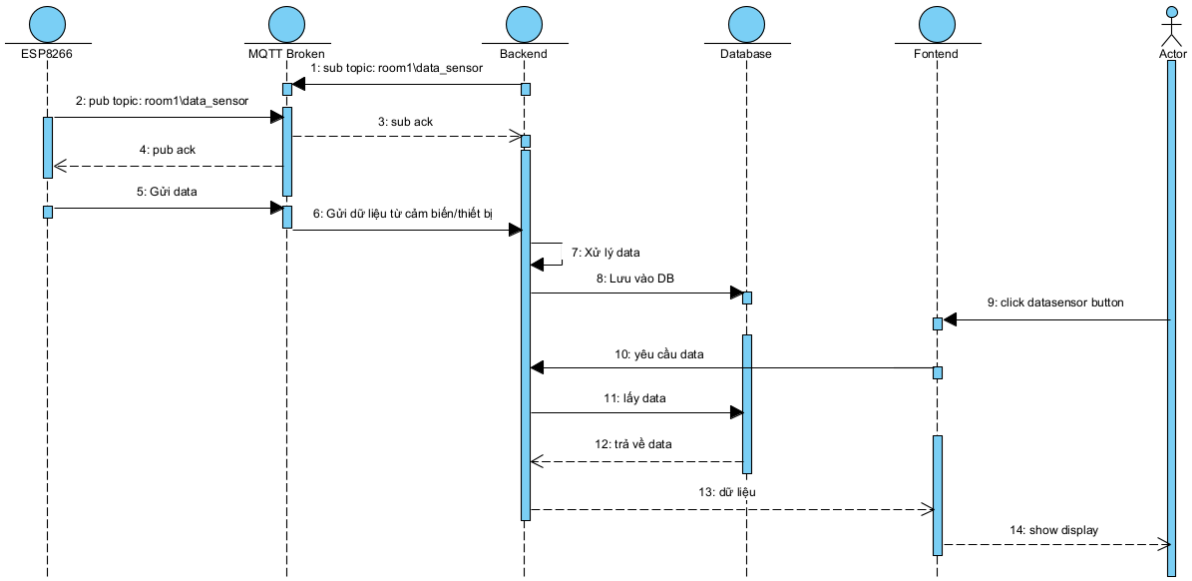
2. DataBase - ERD



Hình 5: Các Bảng trong database

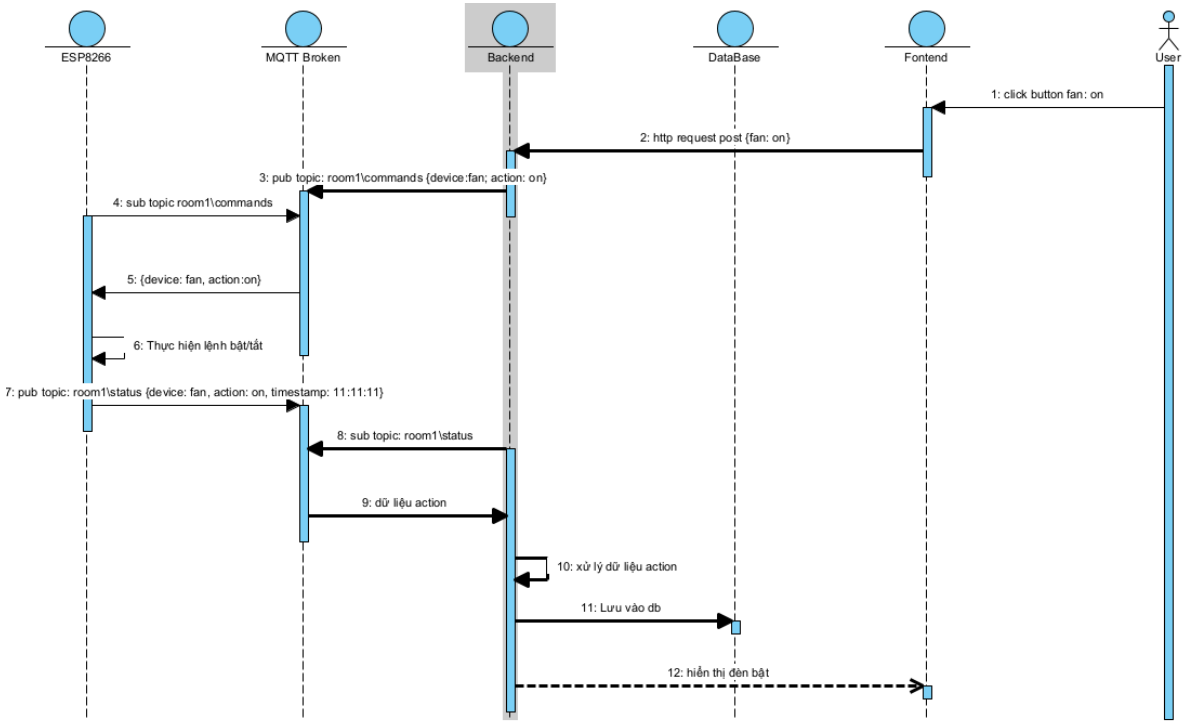
3. Sequence Diagram

Sequence Diagram cho chức năng xem thông tin trên trang Datasensor



Hình 6: Sequence diagram cho chức năng xem thông tin

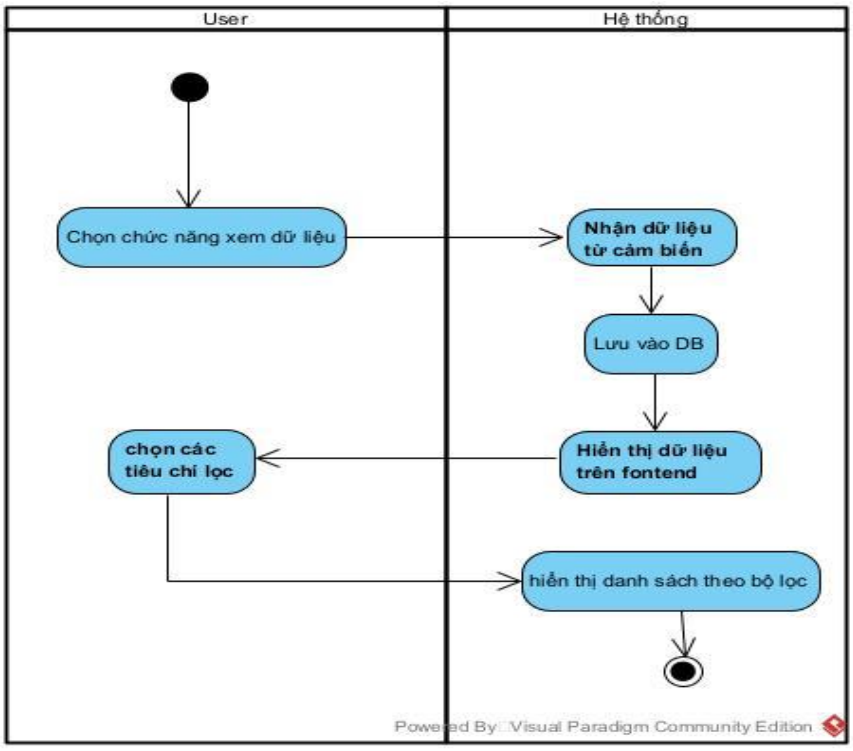
Sequence Diagram cho chức năng bật tắt thiết bị



Hình 7: Sequence diagram cho chức năng điều khiển thiết bị

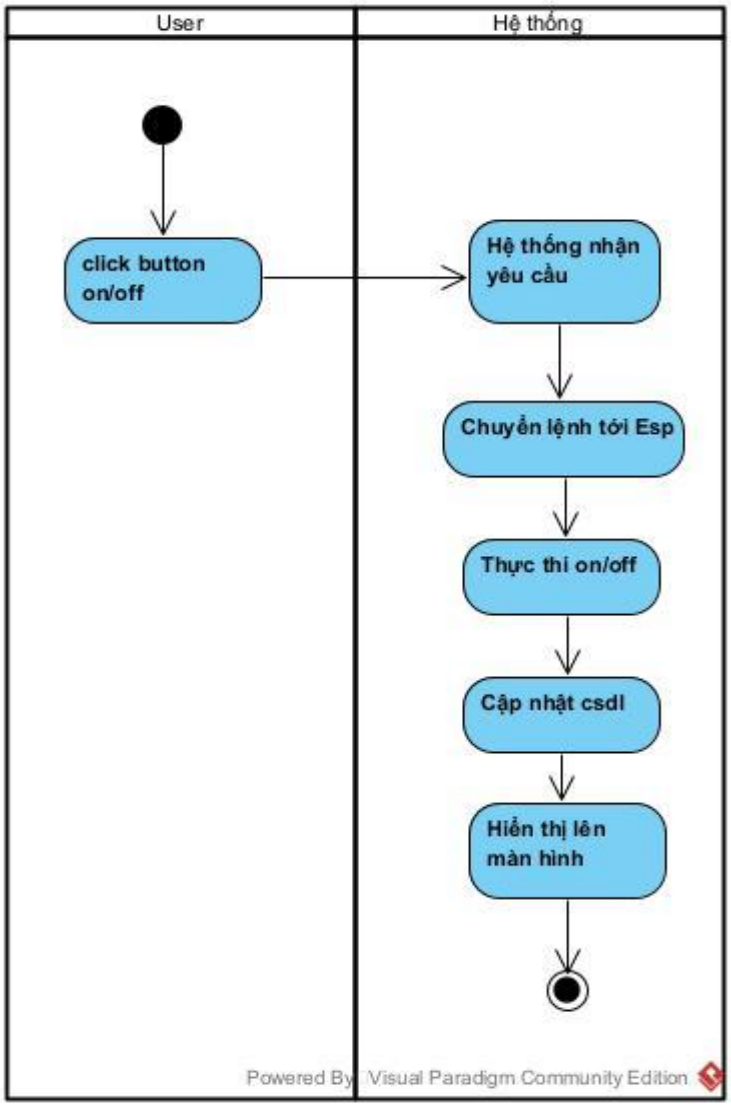
4. Activity Diagram

Activity diagram cho chức năng xem thông tin



Hình 8: Activity diagram cho chức năng hiển thị dữ liệu

Activity diagram cho chức năng bật tắt thiệt bị.

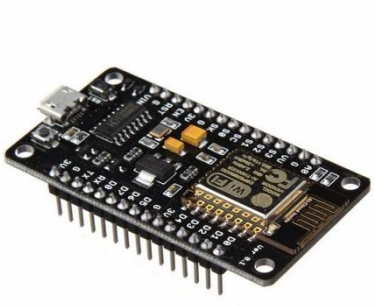


Hình 9: Activity diagram cho chức năng điều khiển thiết bị

II. Thiết kế Phần cứng và Lập trình phần cứng

1. Danh sách thiết bị và linh kiện

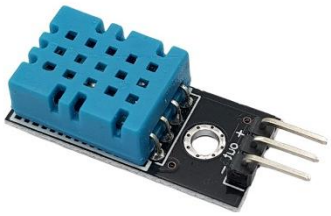
1.1 Vi điều khiển ESP8266



Hình 10: Vi điều khiển Esp8266

- Vi xử lý: 32-bit Tensilica L106, xung nhịp 80 MHz (có thể nâng lên 160 MHz)
- Bộ nhớ RAM: ~64 KB lệnh, 96 KB dữ liệu
- Bộ nhớ Flash: 512 KB – 4 MB (tùy phiên bản)
- Điện áp hoạt động: 3.0 – 3.6 V
- Dòng tiêu thụ: 70 mA (trung bình), 200 mA (cực đại khi phát Wi-Fi)
- Giao tiếp: UART, SPI, I2C, PWM, ADC (1 kênh, 10 bit)
- Tích hợp Wi-Fi 802.11 b/g/n, hỗ trợ chế độ AP/STA
- Số chân GPIO: tối đa 11 (tùy module)

1.2 Cảm biến nhiệt độ, độ ẩm DHT11



Hình 11: Cảm biến nhiệt độ, độ ẩm DHT11

- Nhiệt độ đo: 0 °C – 50 °C
- Sai số nhiệt độ: ±2 °C
- Độ ẩm đo: 20 % – 90 % RH
- Sai số độ ẩm: ±5 % RH
- Điện áp hoạt động: 3.3 V – 5 V
- Dòng tiêu thụ: 0.3 mA (đo), 60 µA (chờ)
- Chu kỳ đo tối thiểu: 1 giây/lần
- Giao tiếp: tín hiệu số 1 dây (single-wire)

1.3 Cảm biến ánh sáng BH1750

Hình 12: Cảm biến ánh sáng BH1750

- Khoảng đo cường độ sáng: 1 – 65.535 Lux
- Điện áp hoạt động: 3.0 V – 5.0 V
- Dòng tiêu thụ: khoảng 0.12 mA (chế độ đo), 0.01 µA (standby)
- Giao tiếp: I2C (địa chỉ mặc định 0x23 hoặc 0x5C)
- Thời gian chuyển đổi: ~120 ms
- Góc đo: khoảng 120°

1.4 Bóng đèn led 5mm

Hình 13: Đèn led 5mm

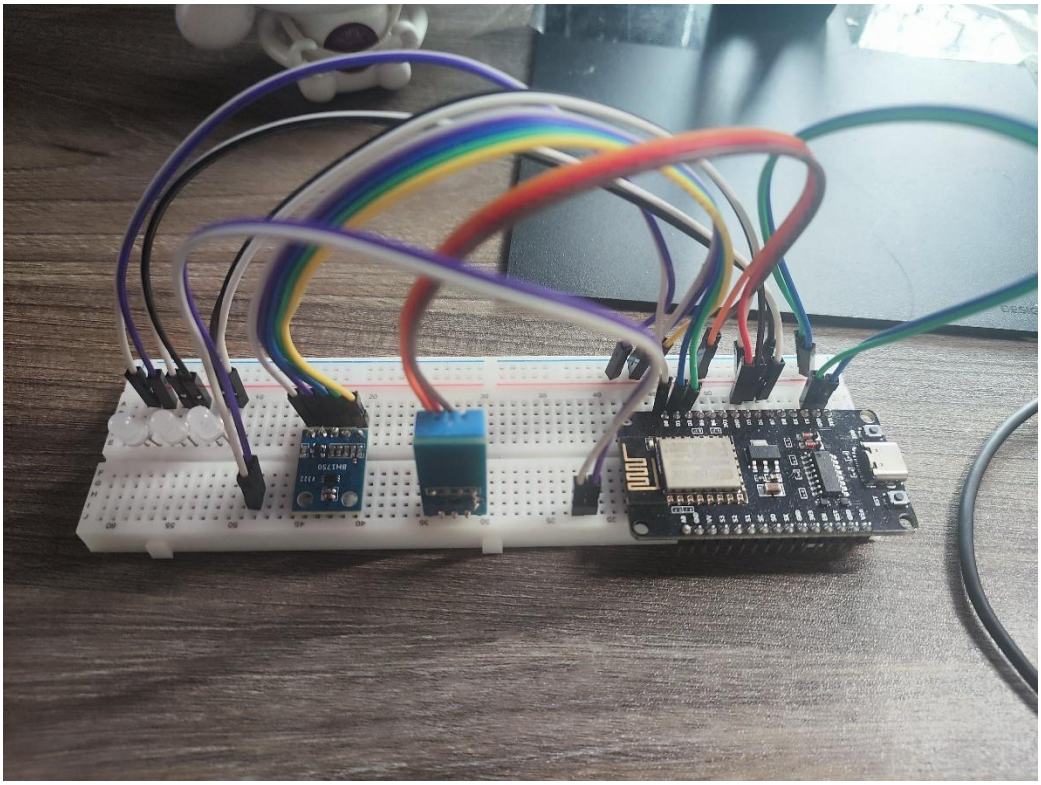
- Điện áp hoạt động: 3.3 V
- Dòng điện hoạt động: 10 – 20 mA
- Công suất tiêu thụ: ~60 mW
- Góc chiếu sáng: 20° – 40°
- Tuổi thọ trung bình: > 50.000 giờ

1.5 Testboard và dây dẫn

Hình 14: Testboard

Hình 15: Dây dẫn Jumper wires đực đực

2. Kết nối linh kiện



Hình 16: Các linh kiện sau khi được ghép nối

2.1. Kết nối DHT11 với Esp8266

DHT11	Esp8266	Ghi chú
VCC	3V3	Cấp nguồn 3.3v
GND	GND	Dây nối đất
DATA	D5(GPIO14)	Chân tín hiệu đọc dữ liệu

2.2. Kết nối BH1750 với Esp8266

BH1750	ESP8266	Ghi chú
VCC	3V3	Cấp nguồn 3.3v
GND	GND	Chân nối đất
SDA	D2(GPIO04)	Dây dữ liệu I2C
SCL	D1(GPIO05)	Dây xung nhịp I2C
ADDR	GND	Đặt địa chỉ I2C là 0x23 (mặc định)

2.3. Kết nối đèn Led 5mm với Esp8266

Led	Esp8266	Ghi chú
Chân dài	3V3	Chân dương nối nguồn 3.3v
Chân ngắn	GND	Chân âm nối đất

3. Lập trình phần cứng với Arduino IDE

3.1 Cấu hình board và thư viện

a. Board Esp8266 by Esp8266 Community

b. Thư viện Adafruit_Unified_Sensor

c. Thư viện BH1750

d. Thư viện DHT_sensor_library

e. Thư viện NTPClient

3.2 Thiết lập giao thức và kết nối

3.2.1 Câu hình MQTT

```
// ===== MQTT Broker =====
#define BROKER_IP   "192.168.0.104"
#define BROKER_PORT 1883
#define ROOM        "room1"
#define MQTT_USER   "cuong123"
#define MQTT_PASS   "cuong123"
```

Hình 17: Cấu hình kết nối MQTT

3.2.2 Thiết lập wifi

```
#define WIFI_SSID
#define WIFI_PASS
```

Hình 18: Thiết lập wifi

3.2.3 Thiết lập chân cắm

```
// ===== DHT22 =====
#define DHTPIN   D5
#define DHTTYPE  DHT22
DHT dht(DHTPIN, DHTTYPE);

// ===== BH1750 (I2C on D2/D1) =====
#define SDA_PIN  D2
#define SCL_PIN  D1

BH1750 lightMeter(0x23);
bool bhReady = false;

// ===== GPIO điều khiển thiết bị =====
// D6 (GPIO12), D7 (GPIO13), D0 (GPIO16)
#define PIN_FAN          D6
#define PIN_AIRCOND      D7
#define PIN_LIGHT        D0


#define RELAY_ACTIVE_LOW 1

inline void setDeviceState(int pin, bool turnOn){
  if (RELAY_ACTIVE_LOW) digitalWrite(pin, turnOn ? LOW : HIGH);
  else                  digitalWrite(pin, turnOn ? HIGH : LOW);
}
inline bool readDeviceIsOn(int pin){
  int level = digitalRead(pin);
  return RELAY_ACTIVE_LOW ? (level == LOW) : (level == HIGH);
}
```

Hình 19: Thiết lập các chân điều khiển cho các thiết bị

3.3 Pub Topic

Topic được xây dựng trong setup() bằng snprintf với tiền tố là ROOM:

- Data sensor topic: "<ROOM>/data_sensor"

```
String luxStr = (!isnan(lux) && lux >= 0.0f && lux < 200000.0f) ? String(lux, 0) : "null";
String payload = String("{\"ts\":") + String(ms) +
                 ",\"temp\":" + String(t, 1) +
                 ",\"hum\":"  + String(h, 0) +
                 ",\"lux\":"  + luxStr + "}";

bool ok = mqtt.publish(TOPIC_DATASENSOR, payload.c_str(), false);
Serial.printf("PUB [%s] %s -> %s\n",
              TOPIC_DATASENSOR, payload.c_str(), ok ? "OK" : "FAIL");
```

Hình 20: mqtt publish topic datasensor

- Biến: TOPIC_DATASENSOR

- Nội dung publish định kỳ (mỗi 2s): JSON chứa ts, temp, hum, lux.
- Ví dụ payload: {"ts":<ms>,"temp":25.3,"hum":60,"lux":123}

- Status topic: "<ROOM>/status"

```
String payload = String("{\"device\":\"") + deviceName + "\",\"action\":\"" + (isOn ? "on" : "off") + "\",\"ts\":" + String(tsMs) + "}";

bool ok = mqtt.publish(TOPIC_STATUS, payload.c_str(), true );
Serial.printf("PUB [%s] %s -> %s\n", TOPIC_STATUS, payload.c_str(), ok ? "OK" : "FAIL");
```

Hình 21: mqtt publish topic gửi trạng thái sau khi điều khiển thiết bị

- Biến: TOPIC_STATUS
- Dùng để báo trạng thái on/off của thiết bị, cũng dùng làm Last Will (retained) khi kết nối MQTT.
- Khi ESP kết nối thành công nó publish một payload online retained: {"online":true,"ts":...}.
- Khi thiết bị bật/tắt (qua lệnh), hàm publishStatusEvent(deviceName, isOn) gửi payload như {"device":"fan","action":"on","ts":...} và publish với retained = true (hàm gọi mqtt.publish(TOPIC_STATUS, payload.c_str(), true)).
- Last Will được cấu hình trong mqtt.connect(..., TOPIC_STATUS, 1, true, lastWill); lastWill là {"online":false}.

3.4 Sub Topic

Command topic: "<ROOM>/commands"

- Biến: TOPIC_COMMANDS
- ESP subscribe chỉ topic này: mqtt.subscribe(TOPIC_COMMANDS, 1);

Nội dung lệnh mong đợi: JSON có hai key "device" và "action" (không phân biệt hoa/thường vì code lower-case hoá message). Ví dụ:

- {"device":"fan","action":"on"}
- {"device":"air","action":"off"}
- {"device":"light","action":"on"}

Xử lý trong onMqttMessage:

- Chỉ xử lý nếu topic trùng TOPIC_COMMANDS.
- Chuyển payload về lowercase, tìm \"device\" và \"action\", lấy giá trị (hàm helper extractJsonStringValue dùng dấu ngoặc kép).
- Hỗ trợ device values:
- "fan" => PIN_FAN

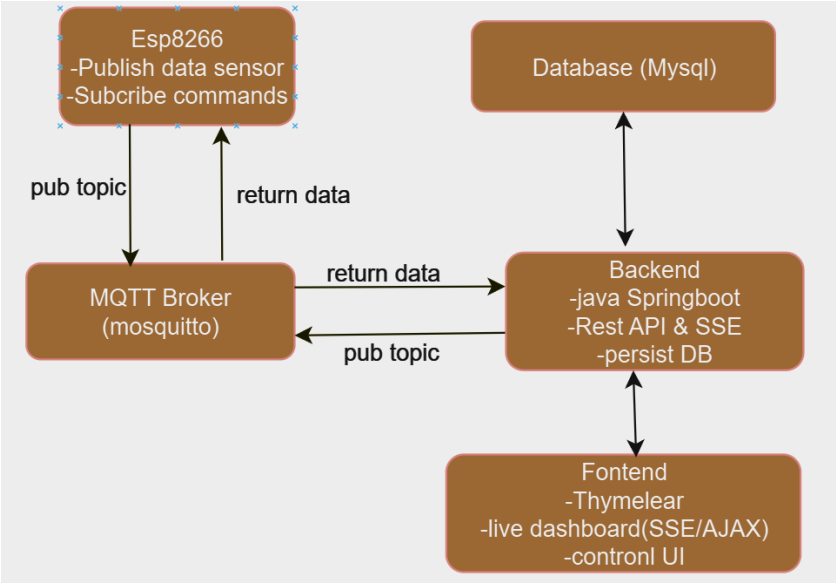
- "air" | "airconditioner" | "ac" => PIN_AIRCOND
- "light" | "lamp" => PIN_LIGHT

- Action supported: "on" hoặc "off". Các action khác bị từ chối bằng thông báo serial "Action chỉ hỗ trợ on/off".
- Sau khi kích hoạt relay, code gọi publishStatusEvent(devName, readDeviceIsOn(pin)) để gửi trạng thái (với retained).

III. Thiết kế hệ thống Web

1. Kiến trúc tổng quan

1.1 Luồng hoạt động



Hình 22: Luồng hoạt động của hệ thống

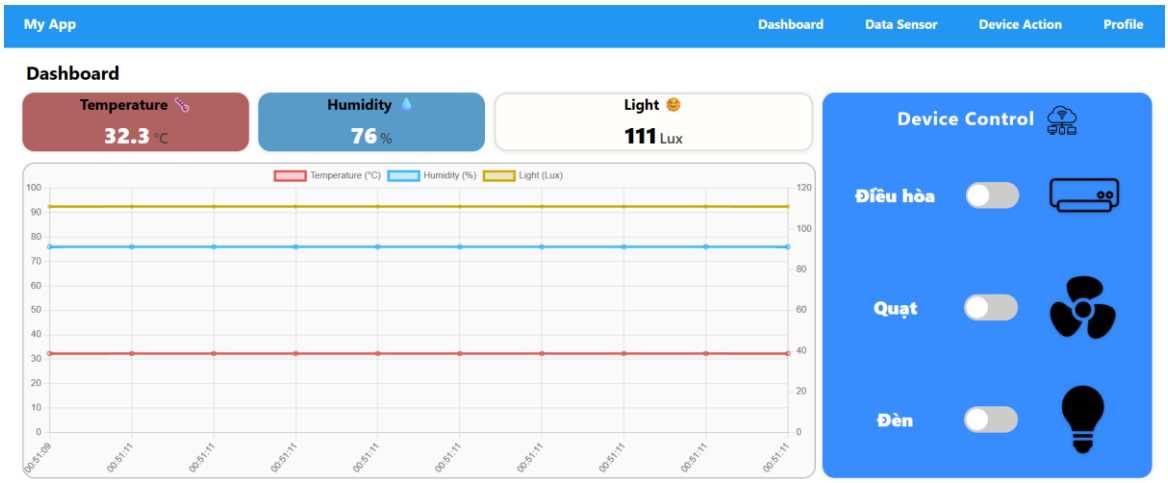
1.2 Fontend

1.2.1 Giới thiệu công nghệ dùng cho Fontend

Thymeleaf là một **Java template engine** được sử dụng phổ biến để xây dựng giao diện người dùng (front-end) trong các ứng dụng web viết bằng **Spring Boot**. Thay vì phải dùng JavaScript framework riêng, Thymeleaf cho phép **nhúng dữ liệu trực tiếp từ backend vào các trang HTML**, giúp việc phát triển trở nên đơn giản và đồng bộ hơn.

Thymeleaf có thể chạy **cả trên server (server-side rendering)** lẫn trên trình duyệt (trong chế độ template), giúp lập trình viên dễ dàng kiểm tra và thiết kế giao diện mà không cần khởi động server.

1.2.2 Trang Dashboard



Hình 23: Trang Dashboard

Trang Dashboard cung cấp cái nhìn tổng quan về hệ thống IOT. Trang gồm các thành phần:

a. Cụm hiển thị dữ liệu từ cảm biến

Dashboard nhận dữ liệu từ Backend mỗi 2 giây và hiển thị lên các card cho từng cảm biến.

- Card nhiệt độ (temperature): Gồm Icon nhiệt kế, giá trị hiện tại của nhiệt độ, màu sắc thay đổi độ đậm nhạt theo nhiệt độ.
- Card độ ẩm (humidity): gồm Icon giọt nước, giá trị hiện tại cả độ ẩm, màu sắc thay đổi độ đậm nhạt theo độ ẩm
- Card ánh sáng (light): gồm Icon bóng đèn, giá trị hiện tại của ánh sáng, màu sắc thay đổi độ đậm nhạt theo độ sáng.

b. Biểu đồ theo thời gian thực

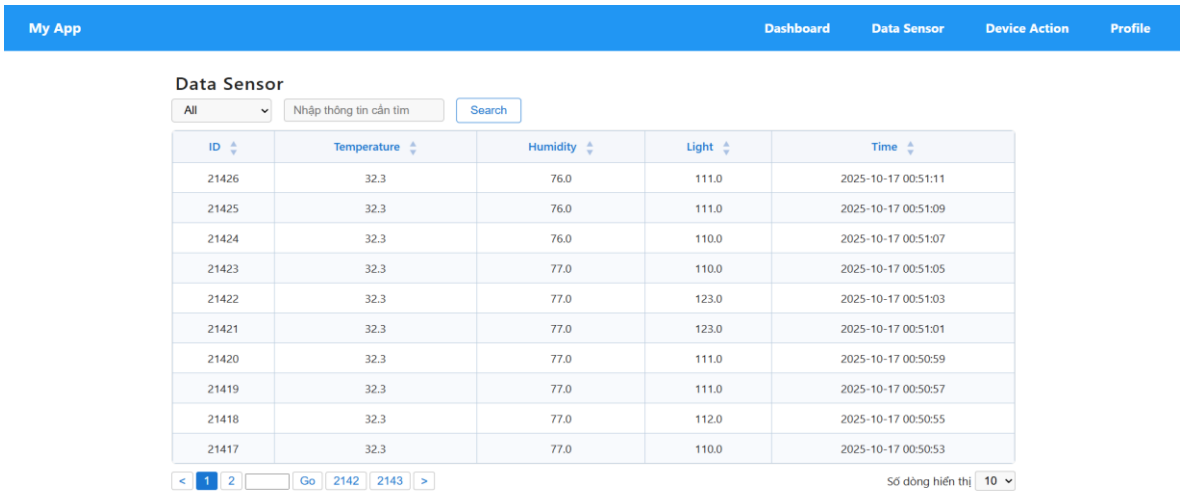
Dashboard nhận dữ liệu từ Backend mỗi 2 giây và đẩy dữ liệu cảm biến theo thời gian thực cho Fontend hiển thị biểu đồ đường lên màn hình.

- Trục X: hiển thị thời gian
- Trục Y bên trái: hiển thị khoảng nhiệt độ và độ ẩm
- Trục Y bên phải: hiển thị khoảng ánh sáng
- Biểu đồ gồm 3 đường: màu đỏ (temperature), màu vàng (ánh sáng), màu xanh (độ ẩm)

c. Bảng điều khiển thiết bị

Bảng điều khiển giúp người dùng có thể bật tắt, điều khiển các thiết bị ngay trên trang web. Bảng gồm tên thiết bị, icon thiết bị (thay đổi khi bật và tắt) và nút bật tắt thiết bị.

1.2.3 Trang Data Sensor

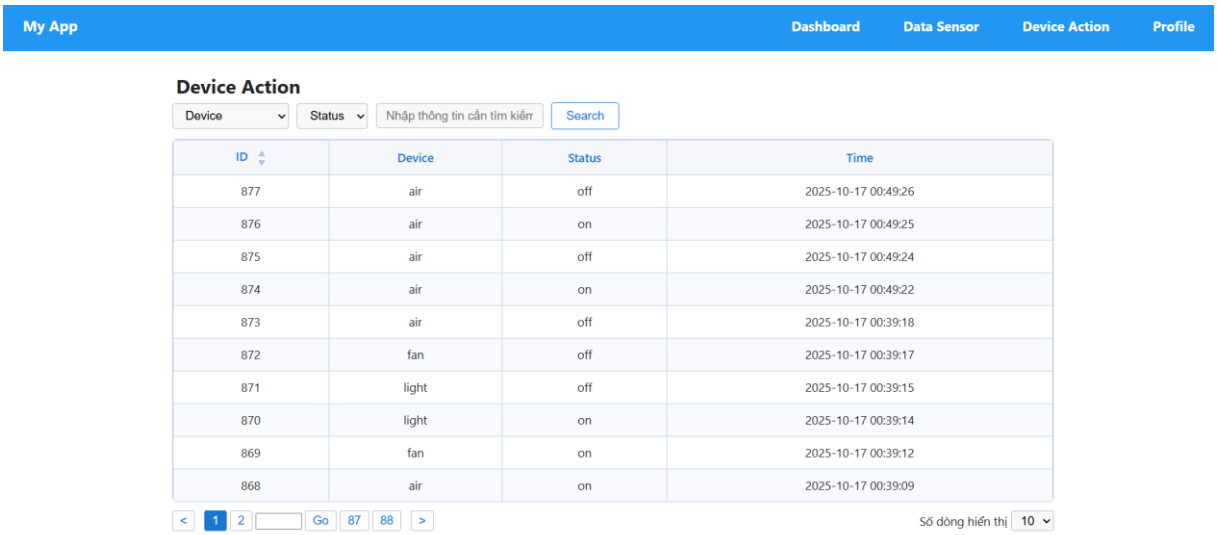


Hình 24: Trang Data Sensor

Trang Data Sensor gồm các tính năng sau:

- Hiển thị dữ liệu: Trang hiển thị dữ liệu thu được từ các cảm biến gồm: id, nhiệt độ (temperature), độ ẩm (humidity), ánh sáng(light), thời gian nhận dữ liệu (time).
- Thanh tim kiếm: người dùng có thế tìm kiếm theo loại cảm biến và nhập thông tin cho cảm biến đó rồi tìm kiếm.
- Thanh phân trang và chọn số lượng dòng: hiển thị trang hiện tại, tổng số trang, cho phép người dùng nhập trang cần đến, chọn số lượng dòng cho mỗi trang

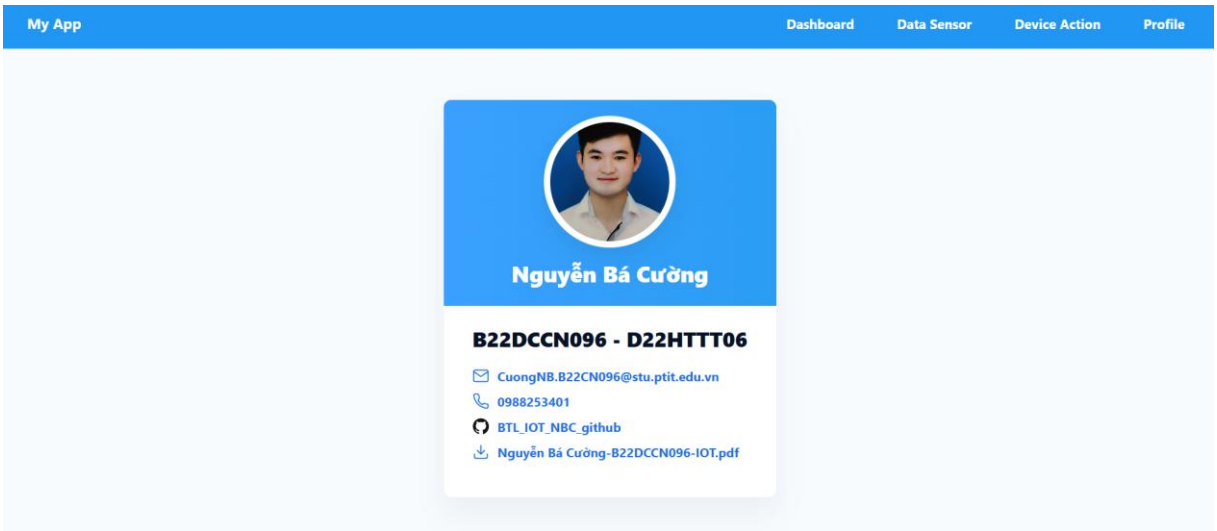
1.2.4 Trang Device Action



Hình 25: Trang Device Action

- Trang Device Action cho phép người dùng xem được lịch sử hoạt động của các thiết bị.
- Cho phép tìm kiếm theo bộ lọc: lọc theo thiết bị, lọc theo trạng thái hoạt động, tìm kiếm dựa vào thanh tìm kiếm.
- Thanh phân trang và chọn số lượng dòng: hiển thị trang hiện tại, tổng số trang, cho phép người dùng nhập trang cần đến, chọn số lượng dòng cho mỗi trang

1.2.5 Trang Profile



Hình 26: Trang Profile

Trang profile cung cấp các thông tin sau:

- Thông tin chung: ảnh, họ tên, mã sinh viên, lớp
- Email
- Số điện thoại
- Link github bài tập lớn
- Link PDF báo cáo

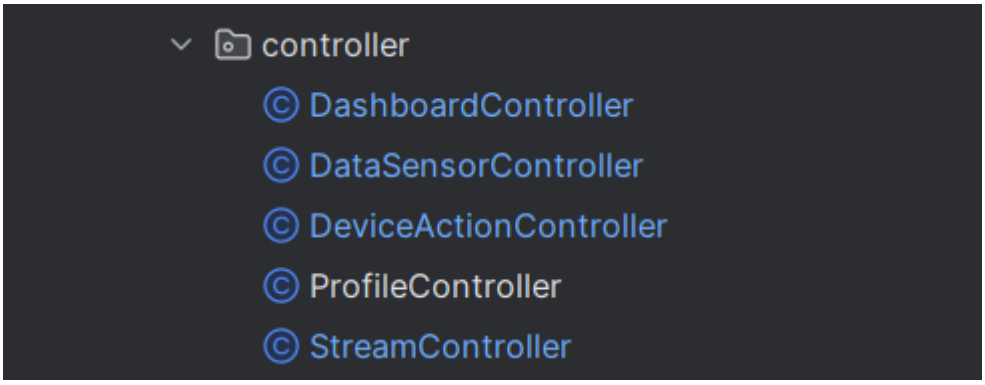
1.3 Backend

1.3.1 Giới thiệu công nghệ dùng cho Backend

- Spring Boot là một framework phát triển ứng dụng web và dịch vụ backend trên nền tảng Java. Nó được xây dựng dựa trên Spring Framework, nhưng đơn giản hóa quá trình cấu hình và khởi tạo, giúp lập trình viên tạo ứng dụng web nhanh chóng mà không cần viết nhiều cấu hình phức tạp

- Luồng hoạt động: Controller -> Service -> Repository -> Database (Controller gọi Service; Service chứa @Transactional, gọi Repository; Repository thao tác DB trả Entity; Service trả DTO/Entity cho Controller)

1.3.2 Controller



Hình 27: package controller

Controller: Controller chịu trách nhiệm xử lý các yêu cầu (request) từ phía người dùng hoặc thiết bị, sau đó trả về phản hồi (response) tương ứng.

- Dashboard Controller: Thực hiện nhiệm vụ chính

o Điều khiển luồng lấy dữ liệu cảm biến từ database và trả cho Fontend đối tượng cần thiết

o Điều khiển luồng điều khiển thiết bị

- Data Sensor Controller

o Điều khiển luồng lấy dữ liệu cảm biến từ Database và trả cho View để hiển thị cho người dùng.

o Thực hiện nhận dữ liệu request từ Fontend và điều khiển luồng tìm kiếm, lọc, sắp xếp, sau đó trả về dữ liệu cần thiết.

- Device Action Controller

o Điều khiển luồng lấy dữ liệu cảm biến từ Device Action và trả cho View để hiển thị cho người dùng.

o Thực hiện nhận dữ liệu request từ Fontend và điều khiển luồng tìm kiếm, lọc, sắp xếp, sau đó trả về dữ liệu cần thiết.

- **Profile Controller :** Trả cho View thông tin cần thiết cho trang Profile
- **Stream Controller**

- Tạo và quản lý một luồng (Stream) dữ liệu chủ động: từ StreamController bạn có thể phát (emit) sự kiện dữ liệu, lỗi hoặc sự kiện đóng, còn các bên khác lắng nghe (listen) các sự kiện đó qua stream.
- Cung cấp cả hai đầu: một sink (để gửi dữ liệu vào) và một stream (để người khác nghe). Điều này cho phép tách phần phát dữ liệu và phần tiêu thụ dữ liệu.
- Chuyển đổi API callback hoặc sự kiện (ví dụ: callback từ native, socket, timer…) thành một Stream có thể quản lý được (subscribe/unsubscribe, pause/resume).
- Quyết định chế độ phát: single-subscription (mỗi stream chỉ có 1 listener) hoặc broadcast (nhiều listener cùng nhận sự kiện). StreamController.broadcast() cho nhiều listener

1.3.3 Entity

Entity — mô tả cấu trúc dữ liệu (bảng DB) dưới dạng đối tượng Java (POJO) và gắn ánh xạ JPA (@Entity).

- Input: dữ liệu từ DB / từ client (sau mapping).
- Output: đối tượng domain (đại diện một hàng trong DB).

Ví dụ: Bảng datasensor sẽ được mô tả dưới dạng đối tượng java như sau:

```
@Table(name = "datasensor", indexes = {
        @Index(name="idx_room_ts", columnList="room,timestamp")
})
public class DataSensorEntity {
    @Id
    @GeneratedValue (strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "temperature")
    private double temperature;

    @Column(name = "humidity")
    private double humidity;

    @Column(name = "light_level")
    private double lightLevel;

    @Column(name = "room")
    private String room;

    @Column(name = "timestamp")
```

Hình 28: dataSensorEntity

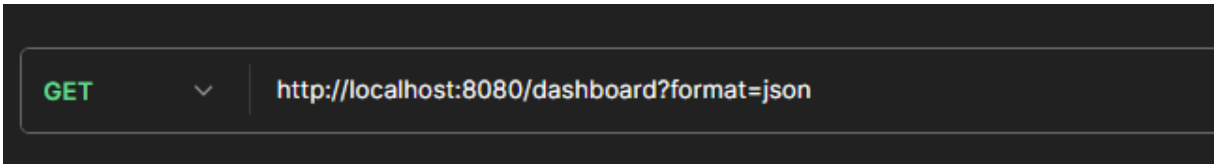
1.3.4 Service và Repository

- Repository — lớp/trình bày dữ liệu (DAO). Ở Spring Data JPA thường là interface kế thừa JpaRepository/CrudRepository.Trách nhiệm: truy vấn/luu/xóa/ tìm (CRUD), các query tuỳ chỉnh.
- Service — tầng business logic / orchestration. Trách nhiệm: thực thi logic nghiệp vụ, quản lý giao dịch (transaction), validate, gọi repository, chuyển/mapping DTO ↔ Entity, gọi các service khác.

2. API điều khiển hoạt động

2.1 API lấy dữ liệu hiển thị trang Dashboard

http://localhost:8080/dashboard



Hình 29: API lấy dữ liệu trang Dashboard

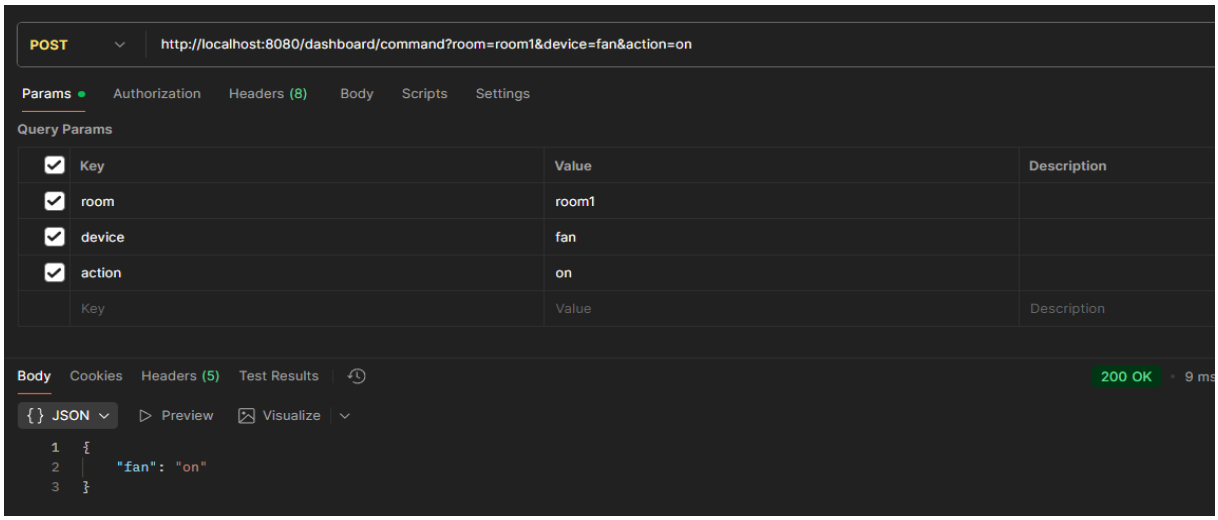
Hình 30: kết quả trả về sau khi chạy API

Dữ liêu trả về dạng JSON gồm:

- Sensor: chứa thông số cảm biến mới nhất
- State: trạng thái gần nhất của các thiết bị để hiển thị bật tắt cho phần điều khiển
- latestTenForDashboard: Danh sách 10 thông số cảm biến mới nhất để hiển thị lên đồ thị

2.2 API điều khiển thiết bị

http://localhost:8080/dashboard/command

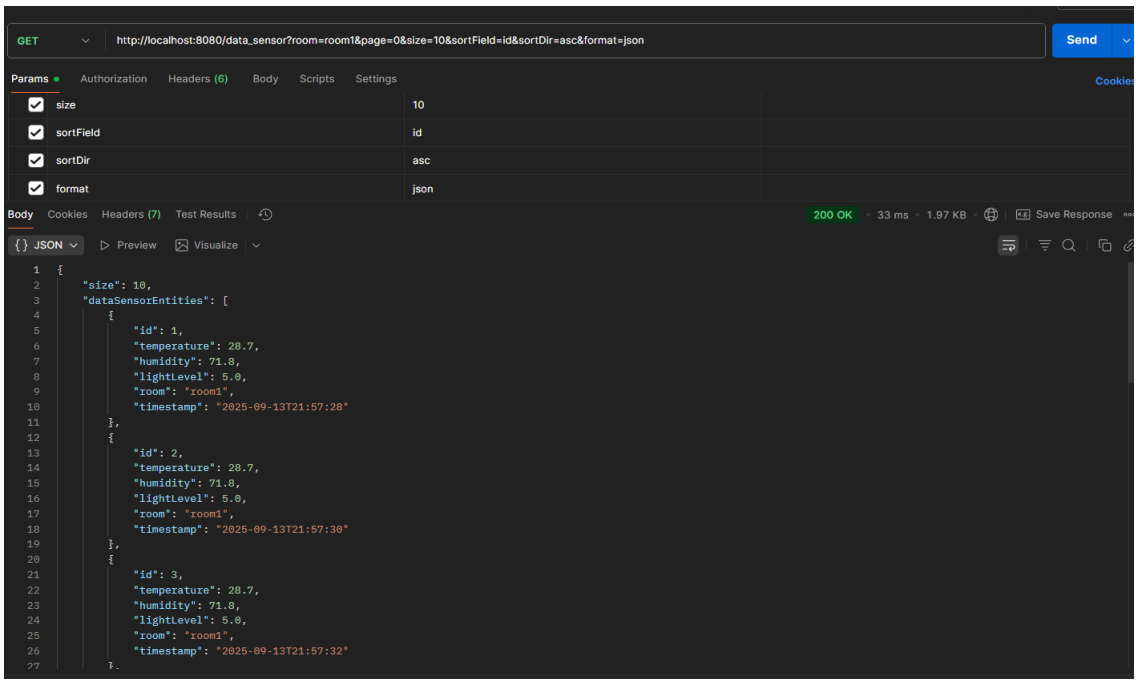


Hình 32: API Điều khiển thiết bị

2.3 API lấy dữ liệu hiển thị trang Data Sensor

- API lấy dữ liệu cảm biến

http://localhost:8080/data_sensor

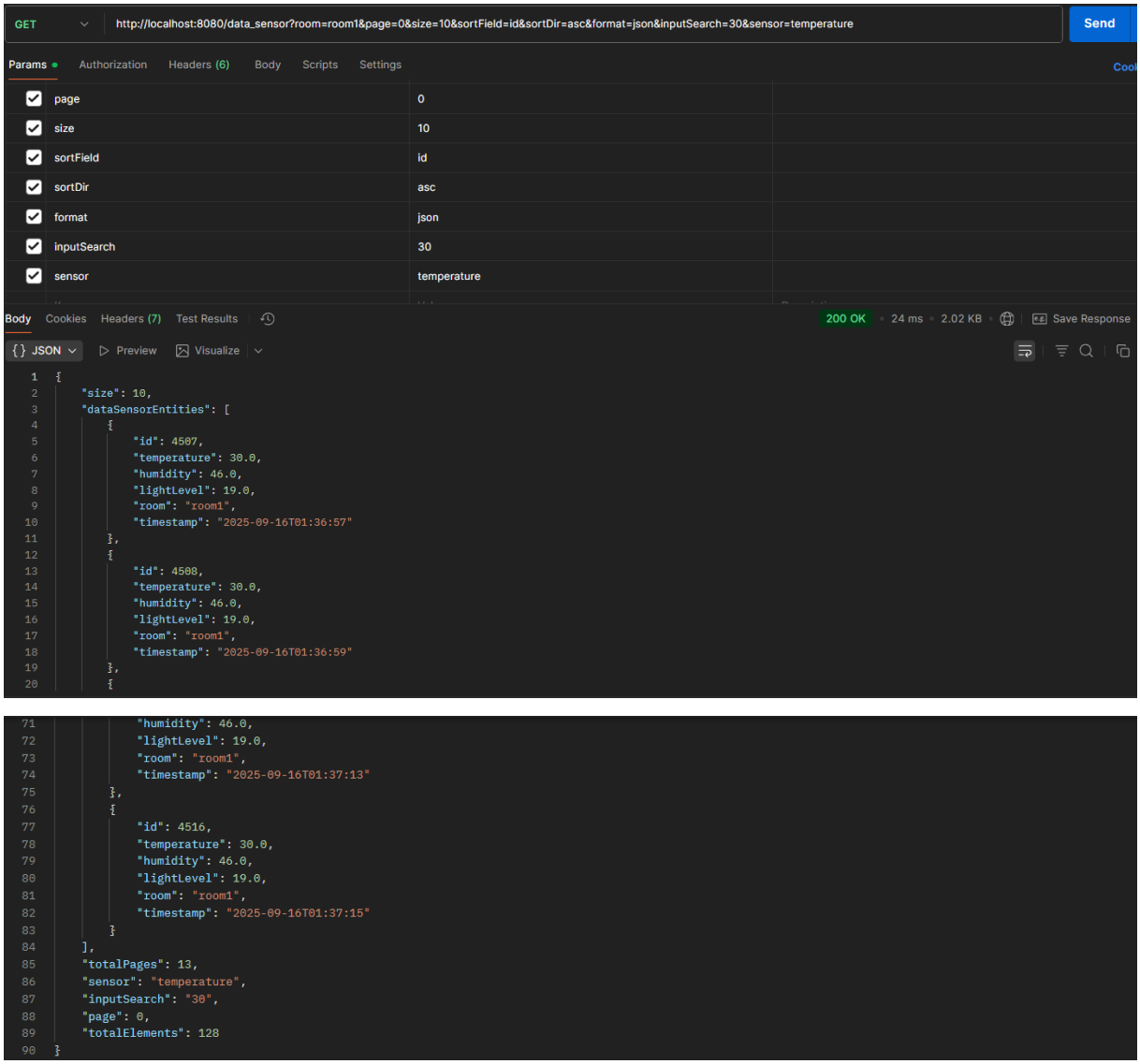


Hình 33: API hiển thị dữ liệu Data Sensor

- API tiềm kiếm

Ví dụ tìm theo temperature = 30:

http://localhost:8080/data_sensor?room=room1&page=0&size=10&sortField=id&sortDir=asc&format=json&inputSearch=30&sensor=temperature



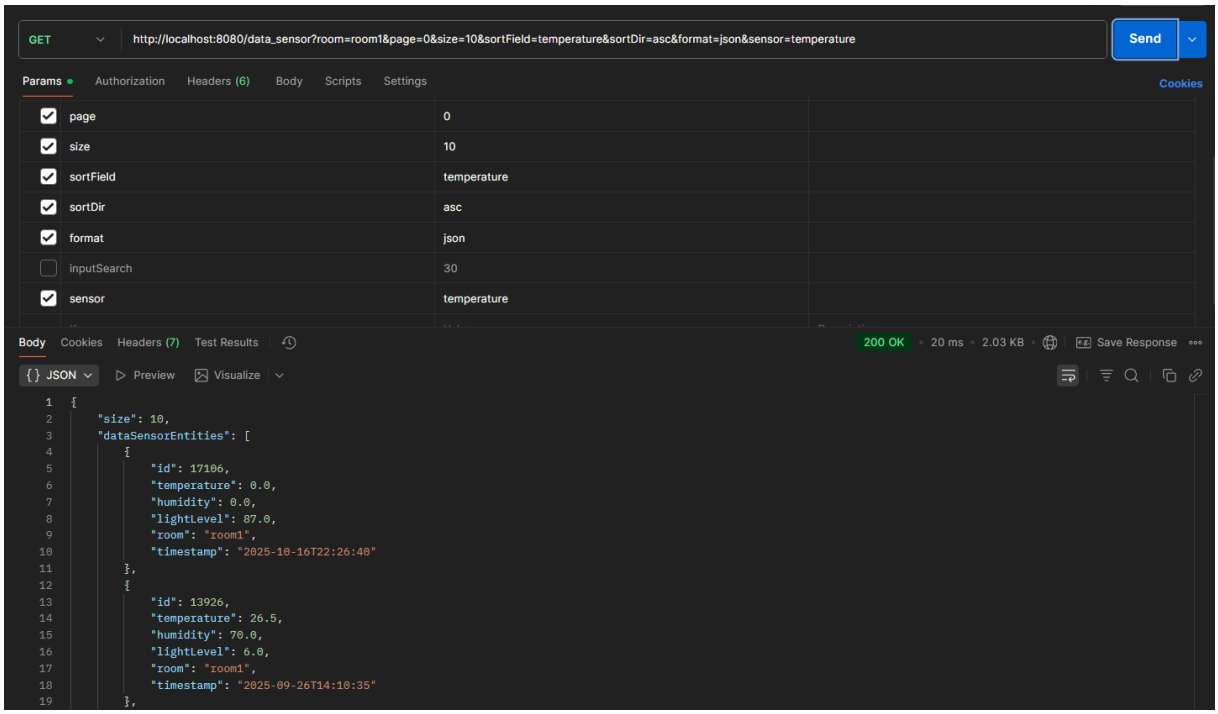
Hình 34: API tìm kiếm trên trang datasensor

- API sắp xếp

Khi người dùng ấn vào icon sắp xếp tại mỗi column thì hệ fontend gửi thêm 2 trường là sortField (trường sắp xếp) và sortDir(kiểu sắp xếp)

Ví dụ: sắp xếp theo nhiệt độ asc

http://localhost:8080/data_sensor?room=room1&page=0&size=10&sortField=temperature&sortDir=asc&format=json



Hình 35: Sắp xếp theo temperature asc (tăng dần)

2.4 API lấy dữ liệu hiển thị trang Device Action

- API hiển thị lịch sử thiết bị

http://localhost:8080/device_action

Kết quả trả về Json với các trường thông tin: filter(gồm các thông tin người dùng nhập vào), dữ liệu lịch sử hoạt động.

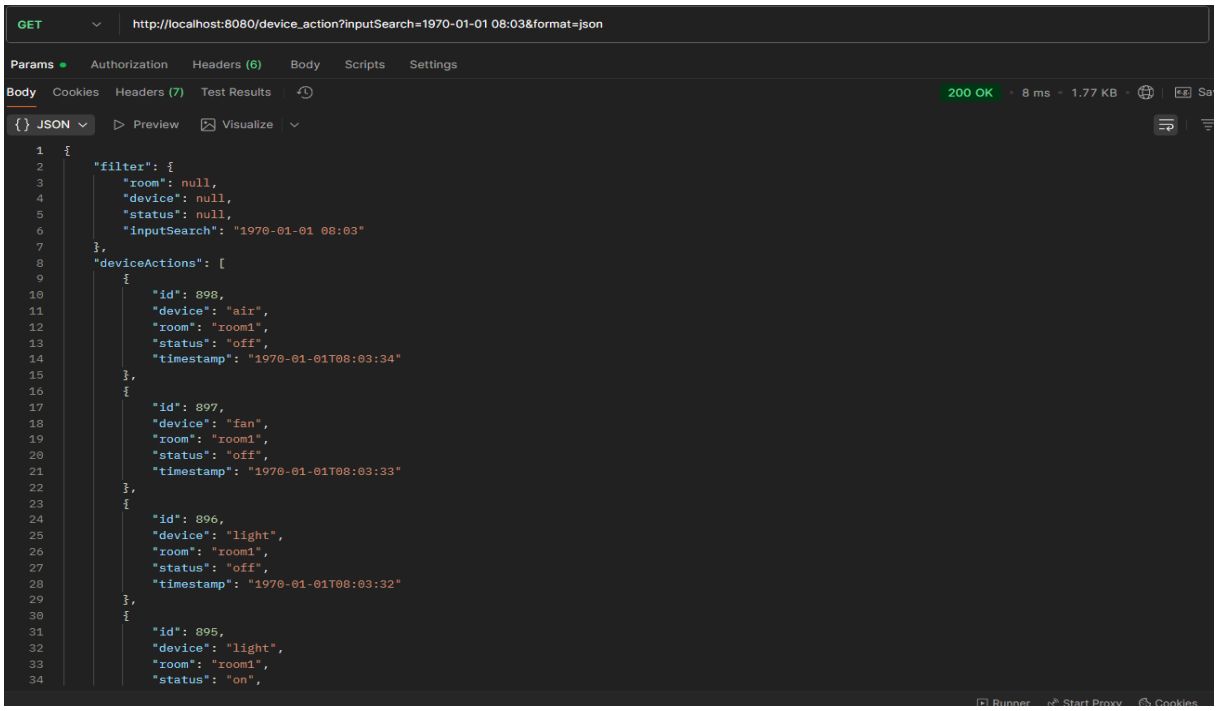

```
{
    "filter": {
        "room": null,
        "device": null,
        "status": null,
        "inputSearch": null
    },
    "deviceActions": [
        {
            "id": 898,
            "device": "air",
            "room": "room1",
            "status": "off",
            "timestamp": "1970-01-01T08:03:34"
        },
        {
            "id": 897,
            "device": "fan",
            "room": "room1",
            "status": "off",
            "timestamp": "1970-01-01T08:03:33"
        },
        {
            "id": 896,
            "device": "light",
            "room": "room1",
            "status": "off",
            "timestamp": "1970-01-01T08:03:32"
        },
        {
            "id": 895,
            "device": "light",
            "room": "room1",
            "status": "on",
```


Hình 36: API hiển thị lịch sử thiết bị

- API tìm kiếm theo bộ lọc

Ví dụ tìm kiếm theo thời gian tính bằng phút: 1970-01-01 08:03

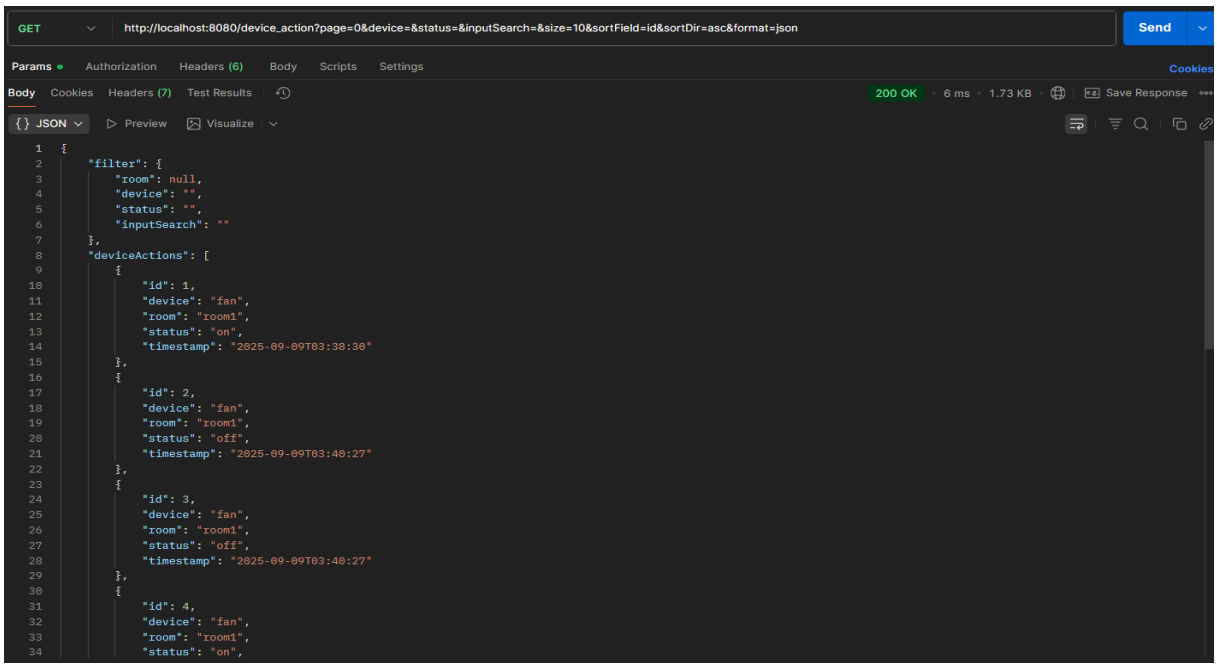
http://localhost:8080/device_action?inputSearch=1970-01-01 08:03&format=json



Hình 36: API hiển thị lịch sử thiết bị theo bộ lọc

- API Sắp xếp theo Id

http://localhost:8080/device_action?page=0&device=&status=&inputSearch=&size=10&sortField=id&sortDir=asc



Hình 36: API hiển thị lịch sử thiết bị theo sắp xếp