HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA AN TOÀN THÔNG TIN

BÁO CÁO BÀI TẬP LỚN

Theo dõi thông số trong nhà: ánh sáng, nhiệt độ, độ ẩm và điều khiển các thiết bị

Sinh viên: Nguyễn Khắc Tuyên – B21DCAT217

Môn học: IOT và Ứng dụng

Nhóm môn học: 06

Giảng viên hướng dẫn: Nguyễn Quốc Uy

~ Hà Nội, tháng 11/2024 ~

Mục Lục

Mục Lục Hình ảnh

1. Đặt vấn đề

Một hệ thống thời gian thực cho phép truyền dữ liệu từ cảm biến để hiển thị cho người dùng. Người dùng có thể tương tác ngược lại cho thiết bị. Một hệ thống IoT đơn giản.

2. Mục tiêu

Hệ thống sử dụng hai cảm biến để thông báo nhiệt độ và độ ẩm, ánh sáng cho người dùng thông qua trang web có thể truy cập với mạng LAN. Trang web sẽ cung cấp nhiệt độ và độ ẩm, ánh sáng với dashboard hiển thị dễ nhìn, cho phép người dùng có thể xem thông tin nhiệt độ và độ ẩm trực tiếp ở thời gian thực và có thể điều khiển led.

3. Các thiết bị được sử dụng

3.1. CHIP WIFI ESP WROOM32

a. Mô tả CHIP WIFI ESP WROOM32:

Chip WiFi ESP WROOM32 là một module phát triển mạnh mẽ dựa trên vi xử lý ESP32, thiết kế nhằm cung cấp giải pháp kết nối và giao tiếp không dây qua mạng WiFi hoặc Bluetooth cho các ứng dụng IoT (Internet of Things) và các ứng dụng không dây khác. Với sự tích hợp cao và tính năng đa dạng, ESP WROOM32 là lựa chọn lý tưởng cho những dự án cần khả năng kết nối mạnh mẽ và hiệu quả. Chip ESP WROOM32 được tích hợp sẵn khả năng kết nối WiFi theo chuẩn 802.11 b/g/n và giao tiếp qua giao thức TCP/IP, hỗ trợ cả Bluetooth và Bluetooth Low Energy (BLE). Ngoài ra, nó cũng hỗ trợ giao thức MQTT (Message Queuing Telemetry Transport), giúp dễ dàng trao đổi dữ liệu trong các ứng dụng IoT. Được thiết kế với nhiều giao tiếp ngoại vi và bộ nhớ lớn, ESP WROOM32 đáp ứng nhiều nhu cầu đa dạng trong các ứng dụng phát triển.

Hình 1: CHIP WIFI ESP WROOM32

b. Tính năng và thông số kỹ thuật của ESP WROOM32:

● Vi xử lý: ESP32, dựa trên kiến trúc Xtensa dual-core 32-bit LX6.

● Tần số hoạt động: 2.4 GHz cho WiFi, hỗ trợ Bluetooth vàBLE.

● Giao tiếp: WiFi 802.11 b/g/n, Bluetooth 4.2, BLE, hỗ trợgiao thức TCP/IP.

● Giao tiếp ngoại vi: GPIO (General Purpose Input/Output),UART, I2C, SPI, DAC (Digital-to-Analog Converter),ADC (Analog-to-Digital Converter), PWM (Pulse-WidthModulation).

● Bộ nhớ: Flash tích hợp lên đến 4MB, RAM 520KB.

● Điện áp hoạt động: 3.3V.

● Tiêu thụ năng lượng: Tiết kiệm điện năng với các chế độsleep.

● Kích thước: Thông thường là 18 mm x 25.5 mm.

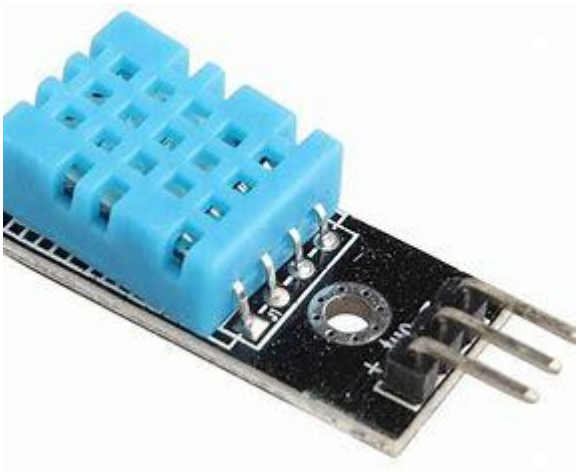
3.2 Cảm biến nhiệt độ độ ẩm DHT11

a, Mô tả

Cảm biến nhiệt độ và độ ẩm DHT11 là một cảm biến giá rẻ và phổ biến trong các dự án liên quan đến đo nhiệt độ và độ ẩm môi trường. Nó được sử dụng để đo và cung cấp thông tin về nhiệt độ và độ ẩm hiện tại trong một môi trường cụ thể

Cảm biến DHT11 sử dụng giao thức 1-wire đơn giản để giao tiếp với vi điều khiển. Nó truyền dữ liệu nhiệt độ và độ ẩm dưới dạng tín hiệu kỹ thuật số qua chân Data.

Cảm biến DHT11 thường được sử dụng trong các ứng dụng như đo nhiệt độ và độ ẩm trong phòng, điều khiển tự động, hệ thống quản lý môi trường và các dự án IoT. Để sử dụng cảm biến, bạn cần cài đặt thư viện DHT11 cho vi điều khiển hoặc vi xử lý mà bạn đang sử dụng và thực hiện các lệnh đọc dữ liệu từ cảm biến thông qua chân Data



Hình 2: Cảm biến DHT11

b,DHT11 có một thiết kế đơn giản với ba chấn cắm

● VCC: Chân nguồn dương, được kết nối với nguồn cung cấp 3.3V - 5V.

● Data: Chân dữ liệu, được sử dụng để giao tiếp với vi điều khiển hoặc vi xử lý để truyền dữ liệu nhiệt độ và độ ẩm.

● GND: Chân nguồn âm, được kết nối với chân GND của nguồn cung cấp và vi điều khiển.

c, Đặc điểm kỹ thuật chính của cảm biến nhiệt độ và độ ẩm

DHT11 bao gồm:

● Phạm vi đo nhiệt độ: 0°C đến 50°C với độ chính xác ±2°C.

● Phạm vi đo độ ẩm: 20% đến 90% RH với độ chính xác ±5% RH.

● Điện áp hoạt động: 3.3V - 5V.

● Điện áp logic: 3.3V - 5V (hỗ trợ giao tiếp với các vi điều khiển 3.3V và 5V).

● Tốc độ truyền dữ liệu: Tối đa 1Hz (1 lần đo dữ liệu mỗi giây).

● Kích thước: Thường có kích thước nhỏ gọn và hình dạng hình hộp chữ nhật.

3.3. Cảm biến quang trở

Cảm biến quang trở (LDR - Light Dependent Resistor) là mộtloại cảm biến ánh sáng thụ động, hoạt động dựa trên nguyên lý hay đổi điện trở khi cường độ ánh sáng chiếu vào nó thay đổi.

Cụ thể:

Nguyên lý hoạt động: Khi ánh sáng mạnh chiếu vào, điện trở của quang trở giảm xuống. Ngược lại, khi ánh sáng yếu hoặc không có ánh sáng, điện trở của quang trở tăng lên. Điều này cho phép quang trở đo được mức độ cường độ ánh sáng môi trường.

Ứng dụng: Cảm biến quang trở thường được sử dụng trong các thiết bị như đèn tự động bật/tắt theo ánh sáng môi trường, hệ thống đo cường độ ánh sáng, hoặc các ứng dụng điều khiển ánh sáng dựa vào điều kiện ánh sáng hiện tại.

Cấu tạo: Quang trở thường được làm từ các chất bán dẫn như cadmium sulfide (CdS). Khi ánh sáng chiếu vào, các electron trong vật liệu bán dẫn được kích thích, làm giảm điện trở của cảm biến.

Tính chất: Quang trở có đặc tính phi tuyến tính, có nghĩa là sự thay đổi điện trở không tỉ lệ thuận với sự thay đổi của cường độ ánh sáng.

Hình 3: Cảm biến quang trở

3.4. Led

LED 2 chân (Light Emitting Diode) là một loại diode phát sáng khi có dòng điện đi qua theo chiều thuận. Nó là một linh kiện điện tử phổ biến, được sử dụng rộng rãi trong các thiết bị điện tử để chiếu sáng hoặc làm đèn báo hiệu.

Cấu tạo:

2 chân:

● Chân dương (Anode): Đây là chân dài hơn, được kết nối với cực dương (+) của nguồn điện.

● Chân âm (Cathode): Chân này ngắn hơn và được kết nối với cực âm (-) của nguồn điện.

Nguyên lý hoạt động:

Khi có dòng điện đi từ Anode sang Cathode (theo chiều thuậncủa diode), các electron sẽ chuyển động và tái hợp với các lỗ trống trong cấu trúc vật liệu bán dẫn của LED. Quá trình này giải phóng năng lượng dưới dạng ánh sáng. LED là một diode nên chỉ phát sáng khi dòng điện đi theo một hướng nhất định (từ chân dương sang chân âm). Nếu kết nối ngược chiều, LED sẽ không sáng.

Đặc điểm:

Tiêu thụ năng lượng thấp: LED sử dụng ít năng lượng hơn so với các loại đèn khác.

Tuổi thọ cao: LED có tuổi thọ dài, có thể hoạt động hàng nghìn giờ.

Kích thước nhỏ gọn: Thường rất nhỏ và dễ dàng tích hợp vào

các mạch điện tử.

Ánh sáng đa dạng: LED có thể phát ra nhiều màu sắc khác nhau (đỏ, xanh lá, xanh dương, trắng, vàng, v.v.) tùy thuộc vào loại vật liệu bán dẫn được sử dụng.

Ứng dụng: Được sử dụng trong đèn chỉ báo, màn hình LED, đèn nền cho màn hình hiển thị, thiết bị điện tử gia dụng, hệ thống chiếu sáng và nhiều ứng dụng khác.

Hình 4: Đèn led

4. Công nghệ sử dụng

Node.js

Node.js là một môi trường chạy JavaScript phía server dựa trên công cụ V8 JavaScript của Google. Với Node.js, bạn có thể xây dựng các ứng dụng web, API, công cụ CLI, và nhiều loại ứng dụng khác mà không cần phải dựa vào trình duyệt.

Các đặc điểm nổi bật:

- Asynchronous & Non-blocking I/O: Node.js xử lý nhiều yêu cầu cùng lúc mà không chặn luồng chính.
- Event-driven: Dựa trên cơ chế vòng lặp sự kiện (event loop), rất hiệu quả với các ứng dụng thời gian thực.
- Single-threaded: Mặc dù chỉ dùng một luồng chính, Node.js có thể xử lý rất nhiều kết nối đồng thời nhờ kiến trúc non-blocking.
- Rich Ecosystem: Có thư viện npm (Node Package Manager) khổng lồ.

HTML, CSS

HTML (HyperText Markup Language) và CSS (Cascading Style Sheets) là hai công nghệ cốt lõi để xây dựng giao diện website.

- HTML: Dùng để tạo cấu trúc và nội dung cho trang web, như văn bản, hình ảnh, và liên kết.
- CSS: Dùng để thiết kế và định kiểu giao diện, như màu sắc, bố cục, và hiệu ứng.

Mosquitto

Mosquitto là một phần mềm mã nguồn mở cung cấp MQTT (Message Queuing Telemetry Transport) – một giao thức truyền tải tin nhắn nhẹ, được thiết kế cho các thiết bị IoT và ứng dụng cần truyền dữ liệu hiệu quả qua mạng. Mosquitto có vai trò như một MQTT broker (máy chủ trung gian), cho phép các client (thiết bị hoặc ứng dụng) xuất bản (publish) và đăng ký (subscribe) tin nhắn.

Tính năng nổi bật của Mosquitto:

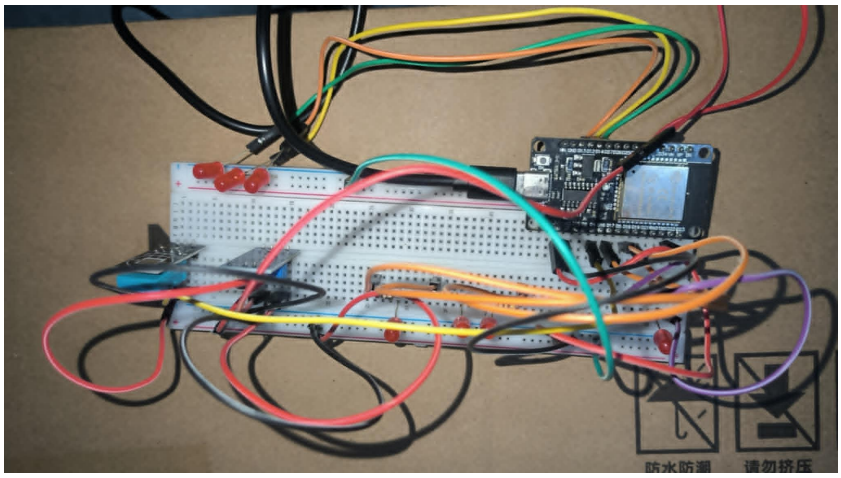
1. Nhẹ và hiệu quả: Phù hợp với các thiết bị IoT hạn chế tài nguyên.
2. Hỗ trợ chuẩn MQTT 3.1 và 3.1.1: Tương thích với nhiều ứng dụng IoT.
3. Mã nguồn mở: Có thể tuỳ chỉnh để phù hợp với nhu cầu.
4. Hỗ trợ bảo mật:
    1. Kết nối an toàn bằng TLS.
    2. Xác thực qua username/password hoặc chứng chỉ.
5. Đa nền tảng: Có thể chạy trên Windows, Linux, macOS và các thiết bị nhúng.

Ứng dụng của Mosquitto trong IoT:

1. Giám sát và điều khiển thiết bị từ xa.
2. Truyền tải dữ liệu cảm biến đến server hoặc các ứng dụng.
3. Kết nối nhiều thiết bị IoT với chi phí thấp.

5.Giao diện

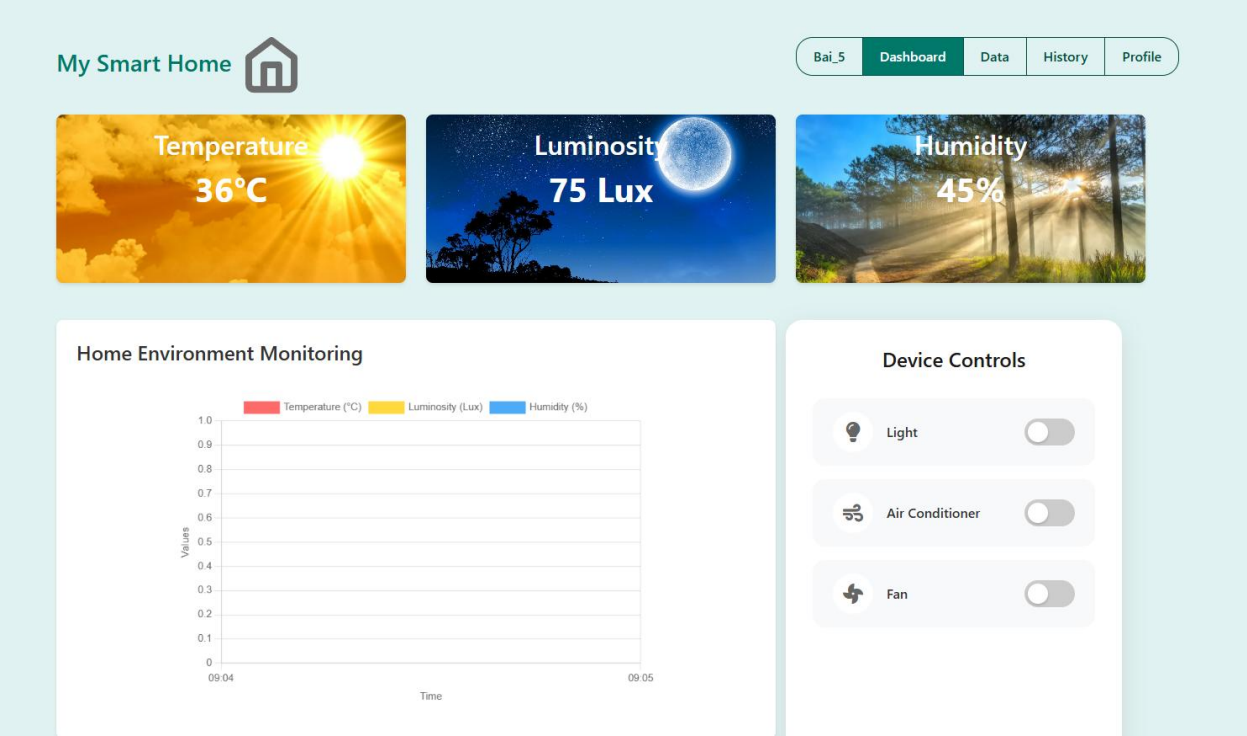
1. Thiết kế tổng thể



Hình 5: Lắp mạch

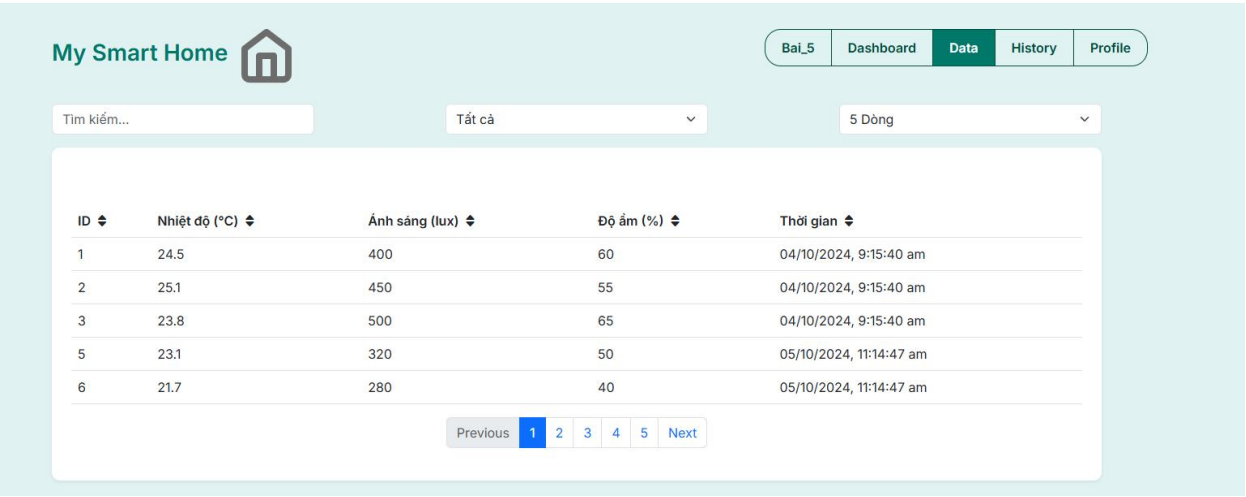
Giao diện web sẽ gồm 4 trang:

+ Dashboard: Sẽ chứa biểu đồ, các nút điều khiển thiết bị và 3 ô giá trị thời gian thực nhận được từ thiết bị



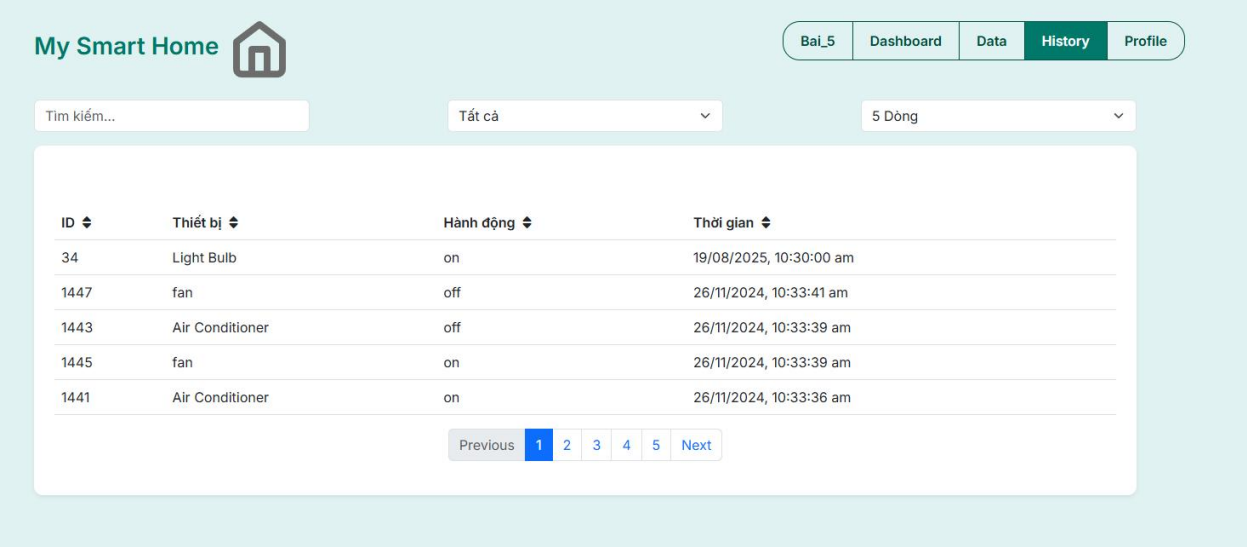
Hình 6 Trang Dashboard

+ Data: Một bảng gồm các cột thông tin về sensor và có thể tìm kiếm và sắp xếp



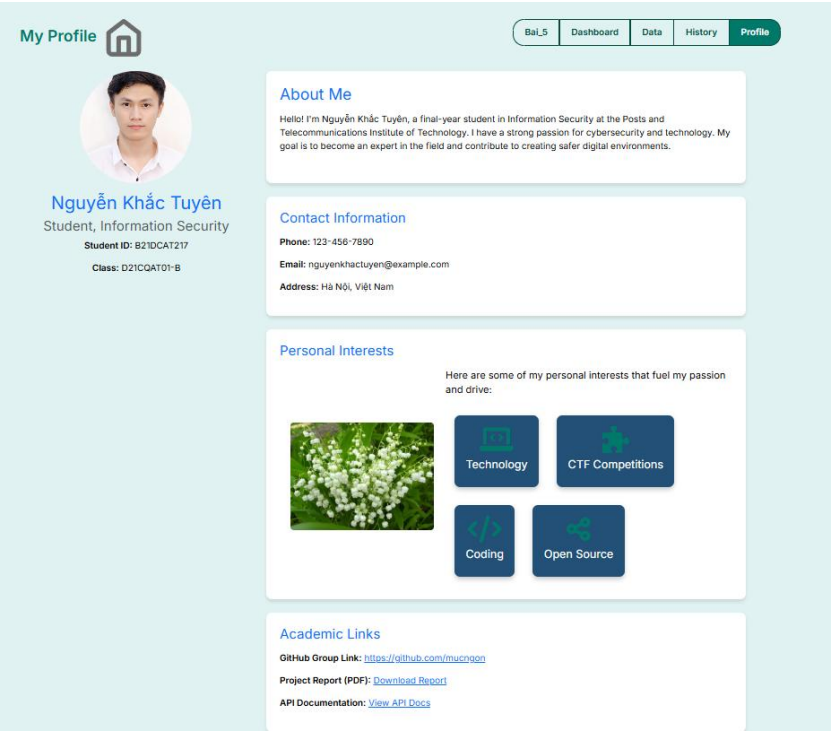
Hình 7: Trang Data

+ ActionHistory: Một bảng gồm các cột thông tin về lịch sử bật tắt thiết bị và có thể tìm kiếm.



Hình 8: Trang History

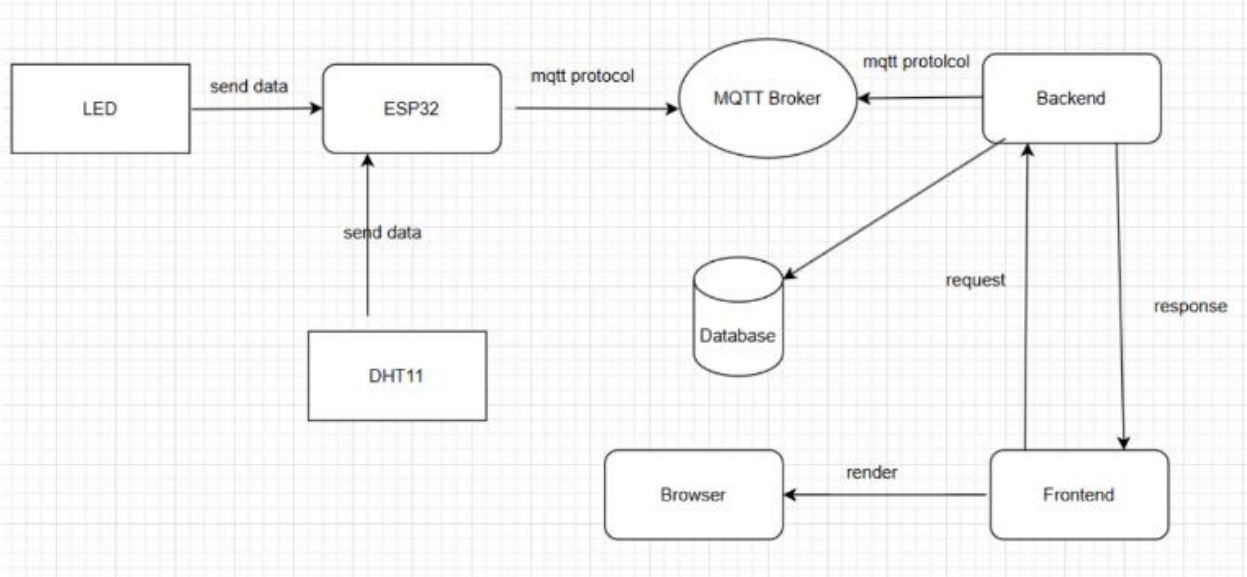
+ Profile: Chứa các thông tin các nhân và link src, api, báo cáo



Hình 9: Trang Profile

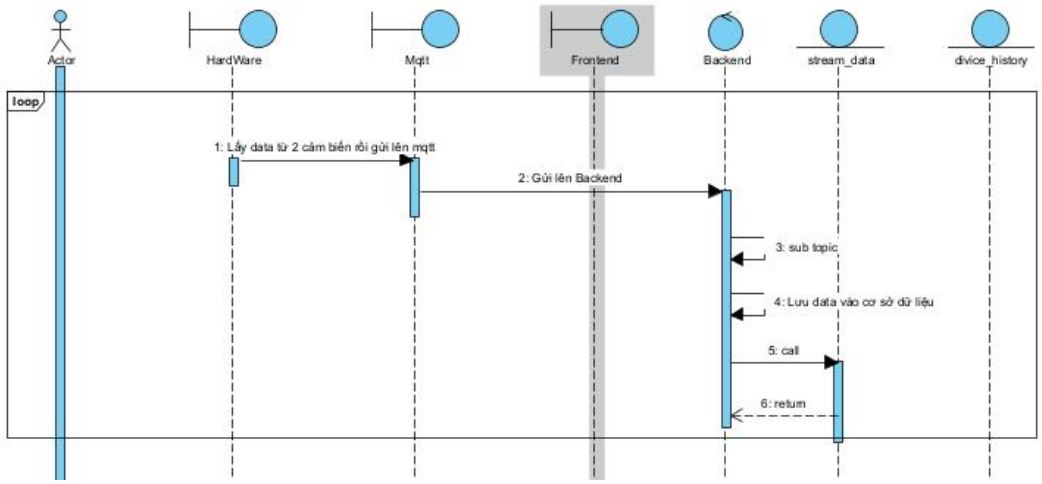
2. Thiết kế chi tiết

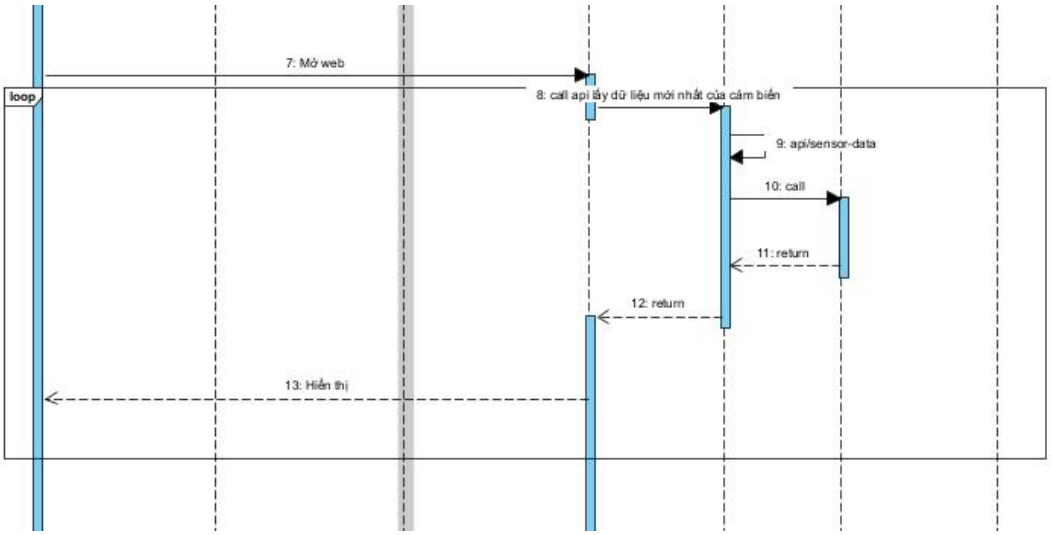
2.1. Luồng dữ liệu

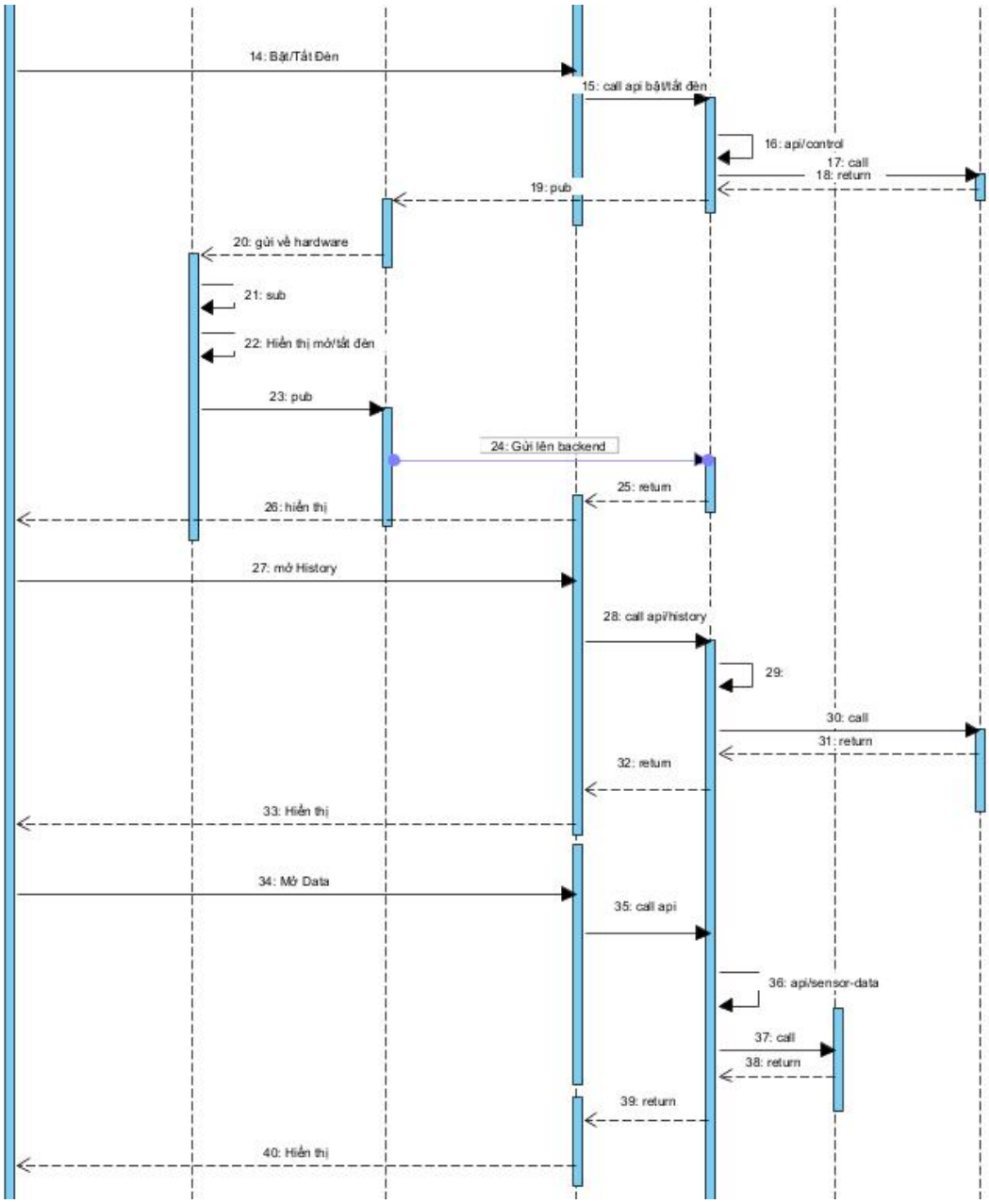


Hình 10: Luồng dữ liệu

2.2. Sơ đồ tuần tự

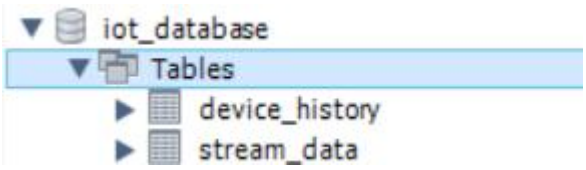



loop
7: Mở web
8: call api lấy dữ liệu mới nhất của cảm biến
9: api/sensor-data
10: call
11: return
12: return
13: Hiển thị



Hình 11: Sơ đồ tuần tự

2.3. Database



Hình 12: database

device_history: chứa data lịch sử bật tắt của thiết bị

Stream_data : chứa dữ liệu của 3 cảm biến độ ẩm nhiệt độ ánh sáng

id	device_name	action	timestamp	description
1	Light Bulb	on	2024-08-19 10:30:00	Light Bulb turned on at living room
2	Fan	off	2024-08-19 10:35:00	Fan turned off in bedroom
3	Air Conditioner	on	2024-08-19 10:39:00	Air Conditioner turned on at 24°C
34	Light Bulb	on	2025-08-19 10:30:00	Light Bulb turned on at living room
35	Fan	off	2023-08-19 10:35:00	Fan turned off in bedroom
36	Air Conditioner	on	2014-08-19 10:39:00	Air Conditioner turned on at 24°C
37	Air Conditioner	off	2024-08-19 10:40:00	Air Conditioner turned off
38	Light Bulb	off	2024-08-19 10:45:00	Light Bulb turned off in living room
39	Fan	on	2024-08-19 10:50:00	Fan turned on in bedroom
40	Light Bulb	on	2024-08-19 10:55:00	Light Bulb turned on in living room
41	Air Conditioner	on	2024-08-19 11:00:00	Air Conditioner turned on at 22°C
42	Fan	off	2024-08-19 11:05:00	Fan turned off in bedroom

Hình 13: Bảng History

id	temperature	light	humidity	time
1	24.5	400	60	2024-10-04 09:15:40
2	25.1	450	55	2024-10-04 09:15:40
3	23.8	500	65	2024-10-04 09:15:40
4	22.5	300	45	2024-10-05 11:14:47
5	23.1	320	50	2024-10-05 11:14:47
6	21.7	280	40	2024-10-05 11:14:47
7	20.3	310	55	2024-10-05 11:14:47
8	22	290	48	2024-10-05 11:14:47
9	24.5	350	60	2024-10-05 11:14:47
10	23.5	330	52	2024-10-05 11:14:47
11	19.9	270	35	2024-10-05 11:14:47
12	25.1	400	65	2024-10-05 11:14:47

Hình 14: Bảng data

6.Code

```
app.get('/api/sensor-data', (req, res) => {
    const query = 'SELECT temperature, humidity, light FROM stream_data ORDER BY time DESC LIMIT 1';

    connection.query(query, (err, results) => {
        if (err) {
            return res.status(500).send('Error fetching data from database');
        }

        res.json(results[0]);  // Trả về hàng đầu tiên (mới nhất)
    });
});
```

Hình 15: API lấy dữ liệu cảm biến mới nhất

```
app.post('/api/control', (req, res) => {
    console.log('Request body:', req.body);  // Ghi log req.body để kiểm tra nội dung

    const { deviceName, action } = req.body;
    console.log(`Device: ${deviceName}, Action: ${action}`);

    if (!deviceName || !action) {
        return res.status(400).json({ error: 'deviceName and action are required' });
    }

    const query = `INSERT INTO device_history (device_name, action, timestamp, description) VALUES (?, ?, NOW(), ?)`;
    const description = `Turned ${action.toLowerCase()} the ${deviceName}`;

    connection.query(query, [deviceName, action, description], (err) => {
        if (err) {
            console.error('Error inserting action into database:', err);
            return res.status(500).json({ error: 'Internal Server Error' });
        }
        res.json({ success: true, message: `Action ${action} sent to ${deviceName}` });
    });
});
```

Hình 16: API điều khiển đèn và lưu lịch sử

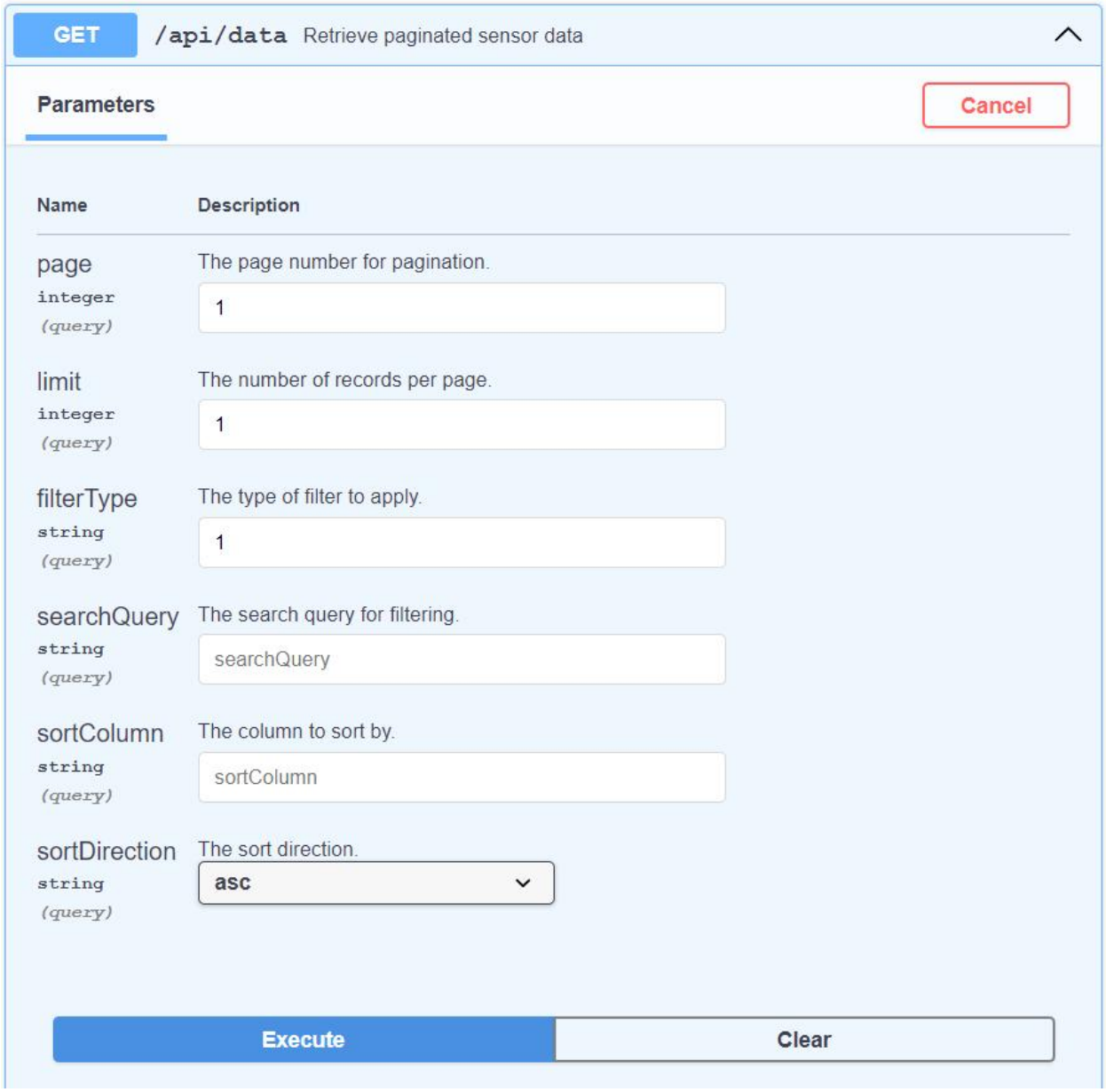
```
app.get('/api/history', (req, res) => {
    const page = parseInt(req.query.page) || 1;
    const limit = parseInt(req.query.limit) || 5;
    const offset = (page - 1) * limit;
    const filterType = req.query.filterType;
    const searchQuery = req.query.searchQuery;
    const sortColumn = req.query.sortColumn || 'timestamp';
    const sortDirection = req.query.sortDirection || 'desc';

    let baseQuery = 'SELECT id, device_name, action, timestamp FROM device_history';
    let countQuery = 'SELECT COUNT(*) as total FROM device_history';
    let whereClause = '';
    let params = [];

    // Add search filtering if provided
    if (searchQuery && filterType && filterType !== 'all') {
        if (filterType === 'timestamp') {
            whereClause = ` WHERE DATE_FORMAT(timestamp, '%Y-%m-%d %H:%i:%s') LIKE ?`;
        } else {
            whereClause = ` WHERE ${filterType} LIKE ?`;
        }
        params.push(`%${searchQuery}%`);
    }

    // Add sorting
    baseQuery += whereClause + ` ORDER BY ${sortColumn} ${sortDirection} LIMIT ? OFFSET ?`;
    countQuery += whereClause;

    // Add pagination parameters
    params.push(limit, offset);

    // Execute count query first
    connection.query(countQuery, params.slice(0, -2), (err, countResults) => {
        if (err) {
            console.error('Error executing count query:', err);
            return res.status(500).json({ error: 'Internal Server Error' });
        }

        // Then execute data query
        connection.query(baseQuery, params, (err, results) => {
            if (err) {
                console.error('Error executing data query:', err);
                return res.status(500).json({ error: 'Internal Server Error' });
            }

            res.json({
                data: results,
                pagination: {
                    total: countResults[0].total,
                    currentPage: page,
                    totalPages: Math.ceil(countResults[0].total / limit),
                    limit
                }
            });
        });
    });
});
// Endpoint để điều khiển đèn
```

Hình 17: API Tìm kiếm và phần trang History

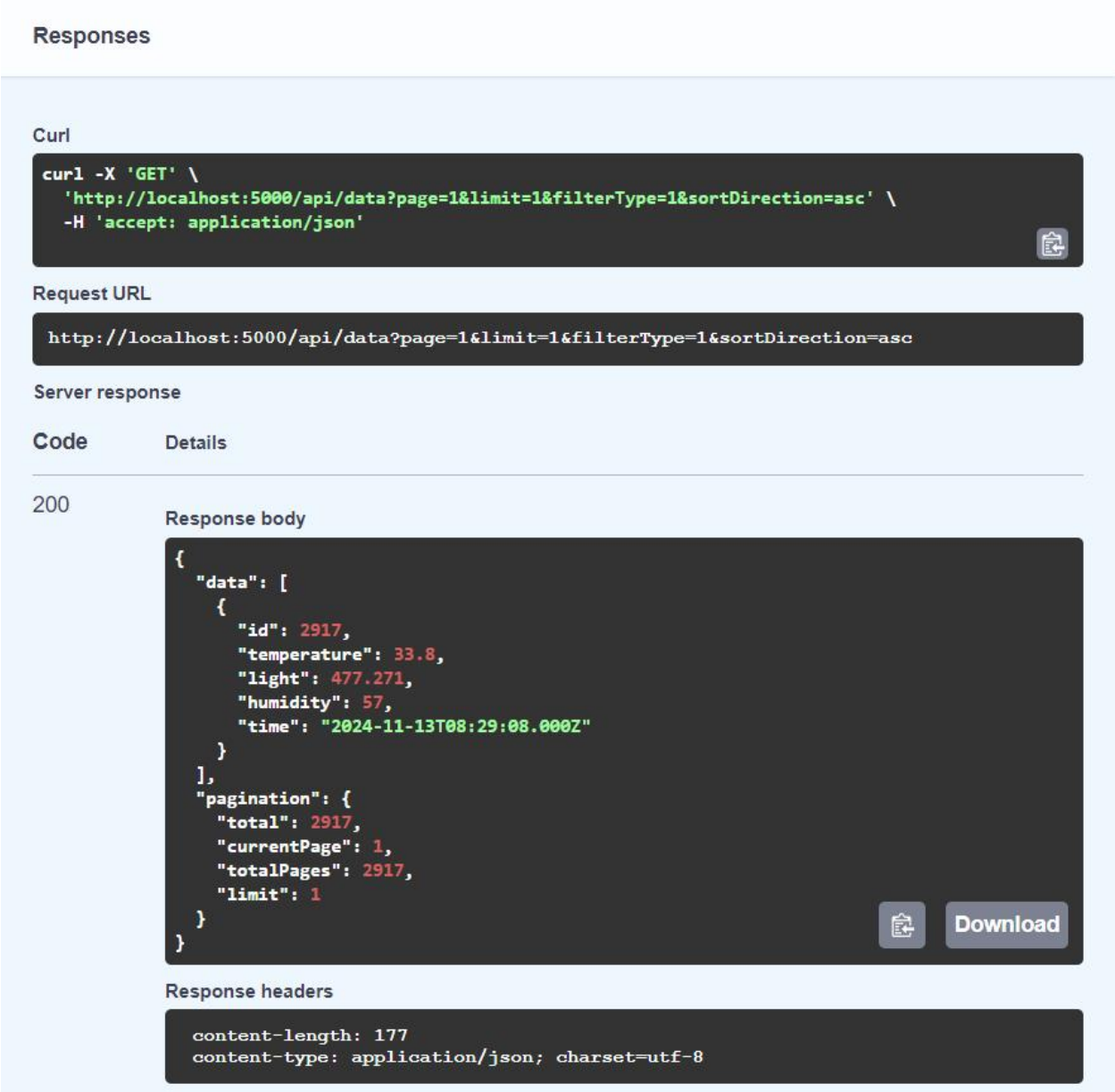
```
// API endpoint để lấy dữ liệu từ MySQL
app.get('/api/data', (req, res) => {
    const page = parseInt(req.query.page) || 1;
    const limit = parseInt(req.query.limit) || 5;
    const offset = (page - 1) * limit;
    const filterType = req.query.filterType;
    const searchQuery = req.query.searchQuery;
    const sortColumn = req.query.sortColumn;
    const sortDirection = req.query.sortDirection || 'asc';

    let baseQuery = 'SELECT id, temperature, light, humidity, time FROM stream_data';
    let countQuery = 'SELECT COUNT(*) as total FROM stream_data';
    let whereClause = '';
    let params = [];

    // Add search filtering if provided
    if (searchQuery && filterType && filterType !== 'all') {
        whereClause = ` WHERE ${filterType} LIKE ?`;
        params.push(`%${searchQuery}%`);
    }

    // Add sorting
    let orderClause = ' ORDER BY time DESC';
    if (sortColumn) {
        orderClause = ` ORDER BY ${sortColumn} ${sortDirection}`;
    }

    baseQuery += whereClause + orderClause + ' LIMIT ? OFFSET ?';
    countQuery += whereClause;

    // Add pagination parameters
    params.push(limit, offset);

    connection.query(countQuery, params.slice(0, -2), (err, countResults) => {
        if (err) {
            console.error('Error executing count query:', err);
            return res.status(500).json({ error: 'Internal Server Error' });
        }

        connection.query(baseQuery, params, (err, results) => {
            if (err) {
                console.error('Error executing data query:', err);
                return res.status(500).json({ error: 'Internal Server Error' });
            }

            res.json({
                data: results,
                pagination: {
                    total: countResults[0].total,
                    currentPage: page,
                    totalPages: Math.ceil(countResults[0].total / limit),
                    limit
                }
            });
        });
    });
});
```

Hình 18 API tìm kiếm phân trang Data

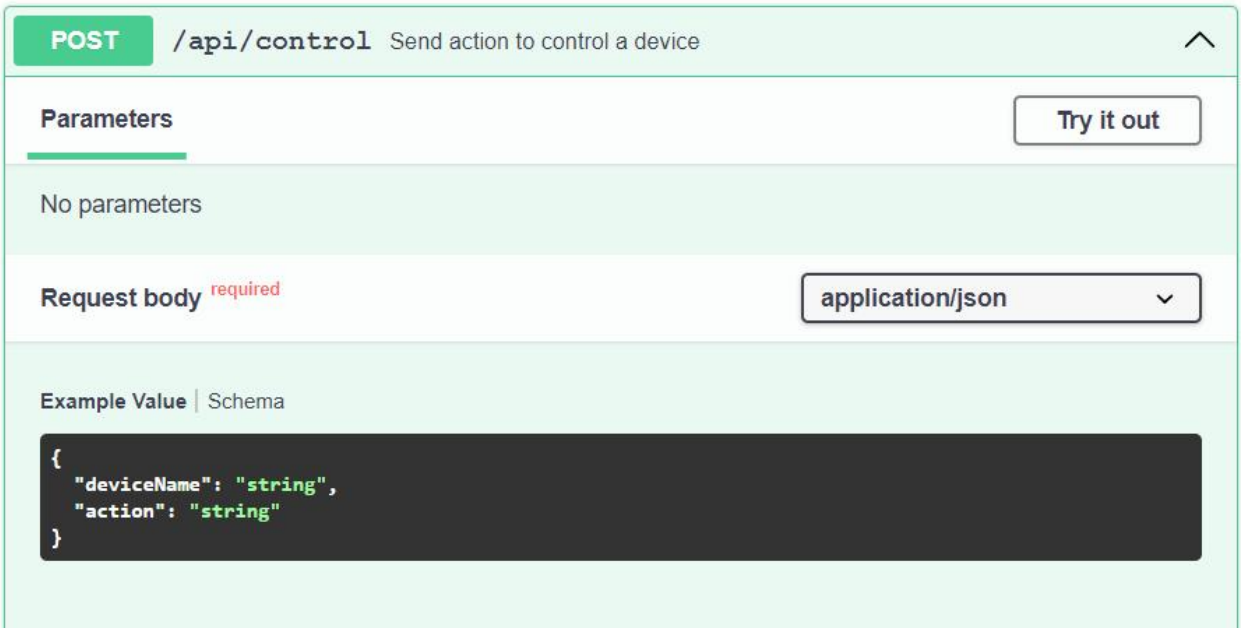
7.Swagger



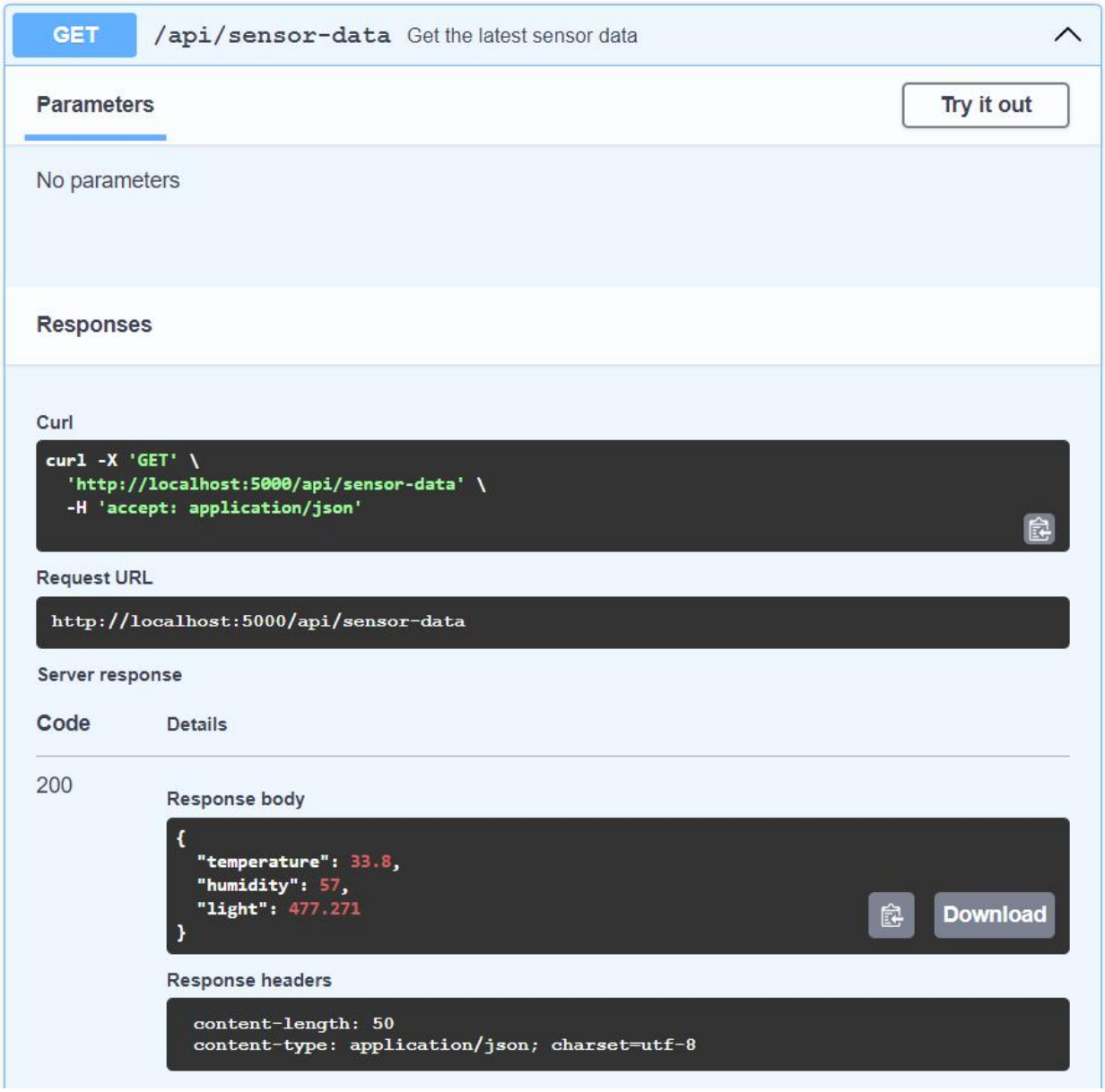
Hình 19: /api/data



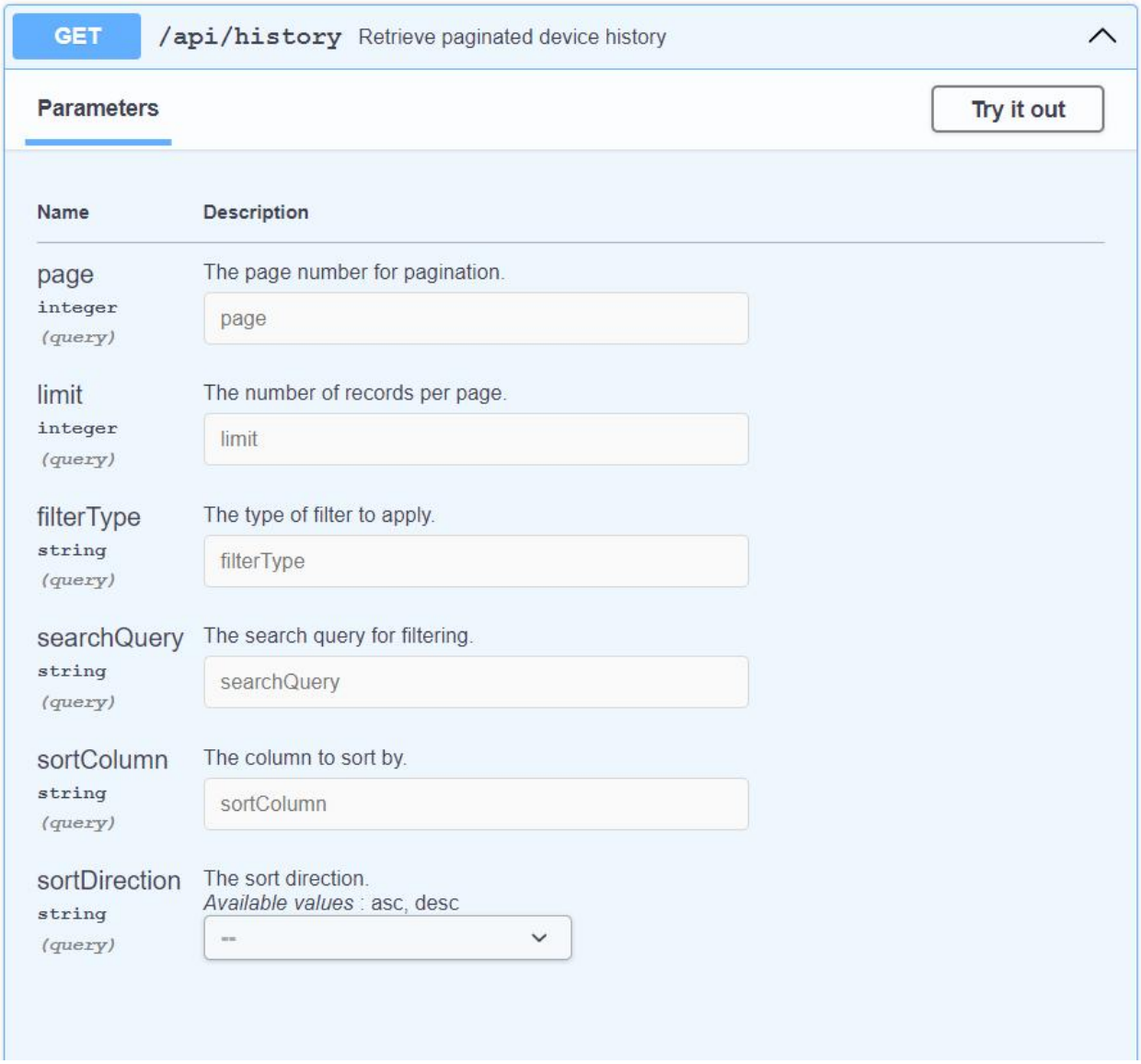
Hình 20 : Response api/data



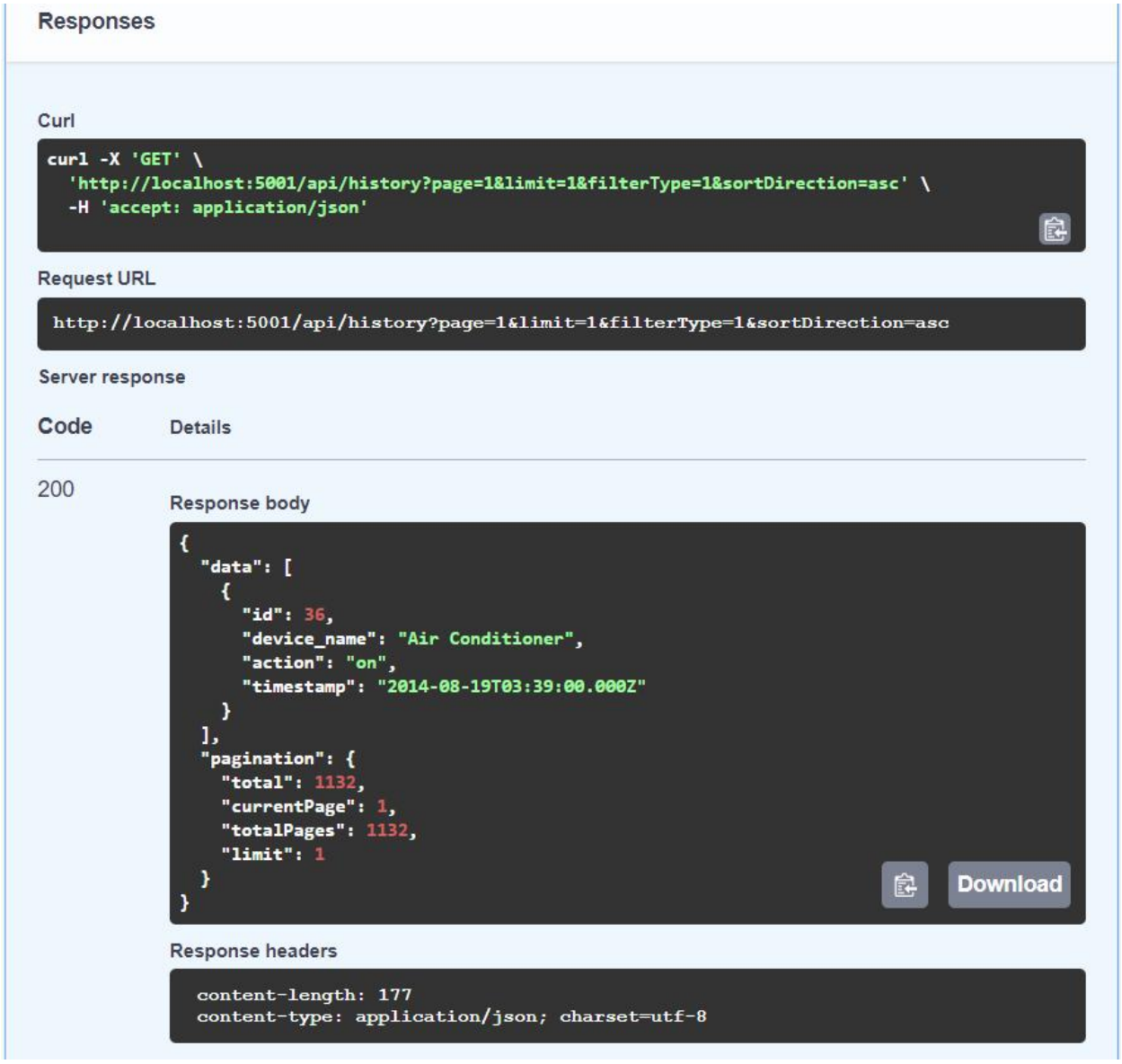
Hình 21 api/control



Hình 22. api/sensor-data



Hình 23: api/history



Hình 24 : Response api/history

8.Kết quả