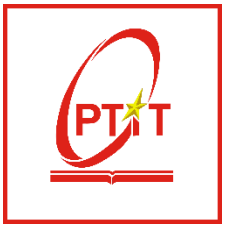
---------o0o---------



BÁO CÁO

MÔN HỌC: IOT VÀ ỨNG DỤNG

CHỦ ĐỀ: Xây dựng hệ thống cảm biến IOT

Giảng viên hướng dẫn: Nguyễn Quốc Uy

Sinh viên thực hiện: Mai Quốc Bình-B22DCCN082

Hà Nội

---------o0o---------

MỤC LỤC

Chương 1: Giới thiệu

I. Tổng quan dự án

Dự án IoT Dashboard được phát triển với mục đích xây dựng một hệ thống giám sát và điều khiển thông minh cho môi trường trong nhà. Hệ thống này kết hợp công nghệ Internet of Things (IoT) với giao diện người dùng trực quan, cho phép người dùng dễ dàng theo dõi các thông số môi trường và điều khiển thiết bị từ xa một cách hiệu quả.

Các mục tiêu chính của dự án bao gồm: giám sát môi trường thông qua việc thu thập và hiển thị dữ liệu nhiệt độ, độ ẩm và ánh sáng theo thời gian thực; hỗ trợ điều khiển từ xa các thiết bị như đèn LED, quạt và điều hòa không khí; lưu trữ và phân tích dữ liệu lịch sử để phát hiện xu hướng cũng như cung cấp thông tin chi tiết; tự động hóa bằng cách thiết lập các quy tắc vận hành dựa trên điều kiện môi trường; và cuối cùng là mang đến một giao diện thân thiện, trực quan, dễ sử dụng cho mọi đối tượng.

Hệ thống IoT Dashboard được thiết kế với nhiều tính năng nổi bật. Trước hết, nó có khả năng hiển thị dữ liệu thời gian thực thông qua các biểu đồ động trực quan về nhiệt độ, độ ẩm và cường độ ánh sáng, với dữ liệu được cập nhật liên tục từ các cảm biến. Tiếp theo là chức năng điều khiển thiết bị từ xa, cho phép người dùng dễ dàng bật hoặc tắt các thiết bị điện ngay trên dashboard. Bảng điều khiển tương tác cung cấp cái nhìn tổng quan về trạng thái hiện tại của tất cả thiết bị, giúp việc quản lý trở nên đơn giản hơn. Hệ thống còn hỗ trợ lưu trữ và truy xuất dữ liệu lịch sử, giúp người dùng phân tích và theo dõi xu hướng lâu dài. Bên cạnh đó, tính bảo mật được đặc biệt chú trọng thông qua cơ chế xác thực người dùng và mã hóa dữ liệu truyền tải giữa các thành phần. Ngoài ra, IoT Dashboard cung cấp API RESTful cho phép tích hợp với các hệ thống khác và hỗ trợ truy vấn dữ liệu cũng như điều khiển thiết bị thông qua các yêu cầu HTTP. Đặc biệt, hệ thống được thiết kế theo hướng module hóa, dễ dàng mở rộng bằng cách thêm cảm biến, thiết bị mới, hoặc áp dụng cho nhiều phòng và khu vực khác nhau. Nhờ vậy, dự án không chỉ đáp ứng nhu cầu hiện tại mà còn mở ra tiềm năng phát triển thành một hệ sinh thái nhà thông minh trong tương lai.

II. Công nghệ sử dụng

Để hiện thực hóa các tính năng trên, dự án IoT Dashboard đã ứng dụng một loạt công nghệ hiện đại ở cả phần mềm và phần cứng. Về phía backend**,** hệ thống sử dụng Node.js – nền tảng chạy JavaScript phía máy chủ, kết hợp với Express.js – framework mạnh mẽ giúp xây dựng API RESTful một cách nhanh chóng và hiệu quả. Đồng thời, Prisma ORM được sử dụng để hỗ trợ quản lý cơ sở dữ liệu theo cách trực quan, dễ bảo trì và an toàn.

Ở phía frontend**,** dự án lựa chọn React.js để xây dựng giao diện người dùng năng động, hiện đại và hiệu quả. React Router được sử dụng để định tuyến, hỗ trợ phát triển ứng dụng một trang (SPA) có nhiều chức năng. Để hiển thị dữ liệu cảm biến trực quan, hệ

thống dùng Chart.js, kết hợp với Axios để gửi yêu cầu API, và WebSocket để đồng bộ dữ liệu theo thời gian thực giữa máy chủ và người dùng.

Về cơ sở dữ liệu, hệ thống sử dụng MySQL – một hệ quản trị cơ sở dữ liệu quan hệ mã nguồn mở, được tích hợp với Prisma ORM. Cơ sở dữ liệu này lưu trữ dữ liệu cảm biến như nhiệt độ, độ ẩm, ánh sáng, đồng thời ghi lại lịch sử điều khiển thiết bị và quản lý thông tin người dùng.

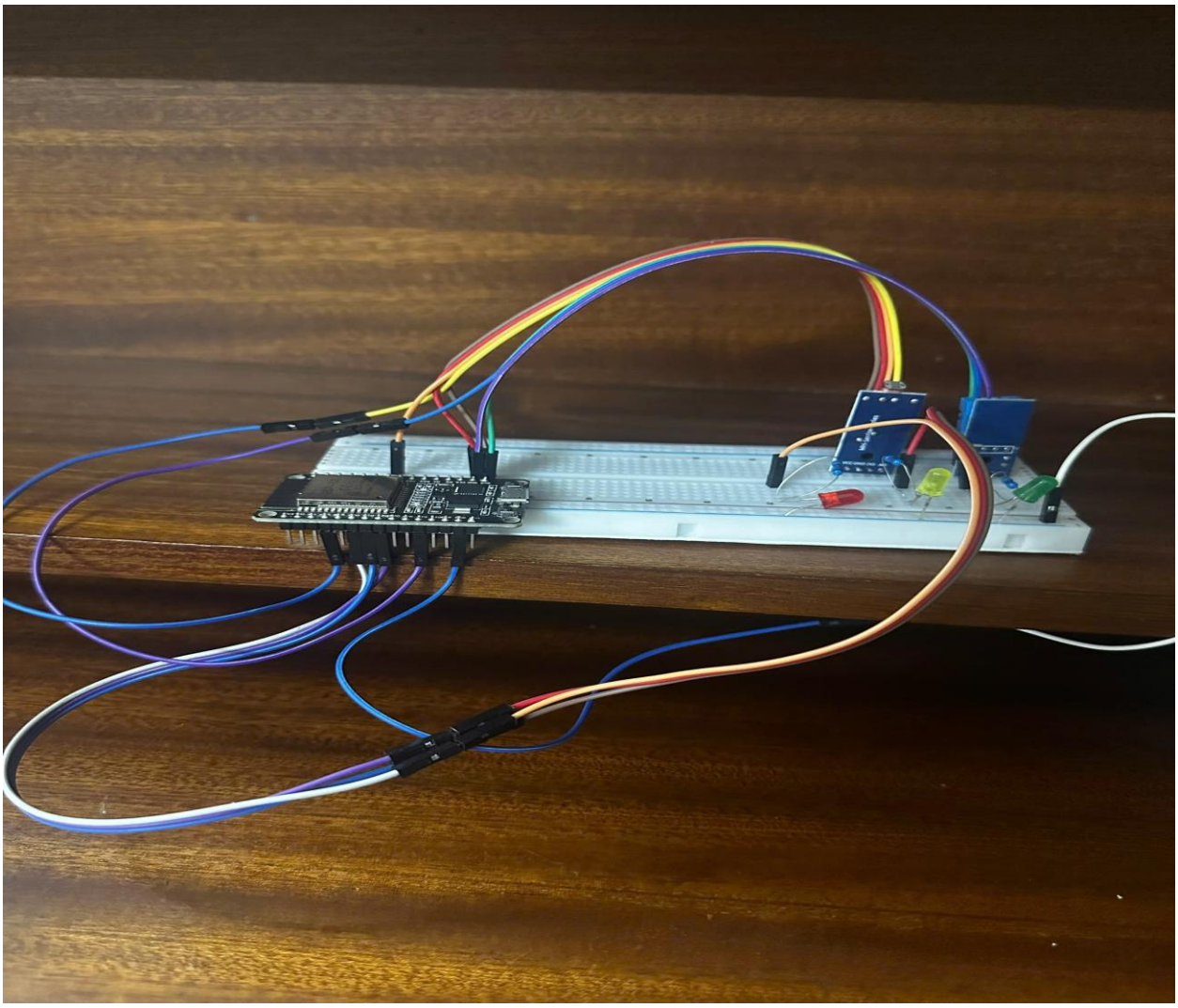
Trong phần giao thức truyền thông, dự án triển khai MQTT (Message Queuing Telemetry Transport) – một giao thức truyền thông nhẹ, phù hợp với thiết bị IoT, để truyền tải dữ liệu giữa ESP32 và server. Song song với đó, HTTP/HTTPS được sử dụng để giao tiếp trong API RESTful giữa frontend và backend.

Về phần cứng, dự án ứng dụng ESP32 làm vi điều khiển trung tâm với khả năng kết nối WiFi và Bluetooth. Thiết bị này được lập trình bằng Arduino IDE và kết nối với các cảm biến như DHT11 (dùng để đo nhiệt độ và độ ẩm) và LDR (Light Dependent Resistor) (dùng để đo cường độ ánh sáng). Nhờ sự kết hợp này, hệ thống có thể thu thập đầy đủ dữ liệu môi trường cần thiết.

Cuối cùng, để quản lý mã nguồn và phát triển hiệu quả, dự án sử dụng Git như một hệ thống quản lý phiên bản phân tán, kết hợp với GitHub để lưu trữ và chia sẻ mã nguồn trực tuyến. Ngoài ra, Postman được tận dụng để kiểm thử và tài liệu hóa API, giúp đảm bảo hệ thống hoạt động ổn định và đúng yêu cầu.

→Việc ứng dụng đồng bộ các công nghệ hiện đại trên đã giúp IoT Dashboard trở thành một hệ thống mạnh mẽ, dễ bảo trì, dễ mở rộng, và có khả năng đáp ứng nhu cầu của cả hiện tại lẫn tương lai. Sự kết hợp hài hòa giữa công nghệ phần mềm và phần cứng chính là yếu tố tạo nên tính toàn diện và bền vững cho dự án.

*) Thiết kế tổng thể

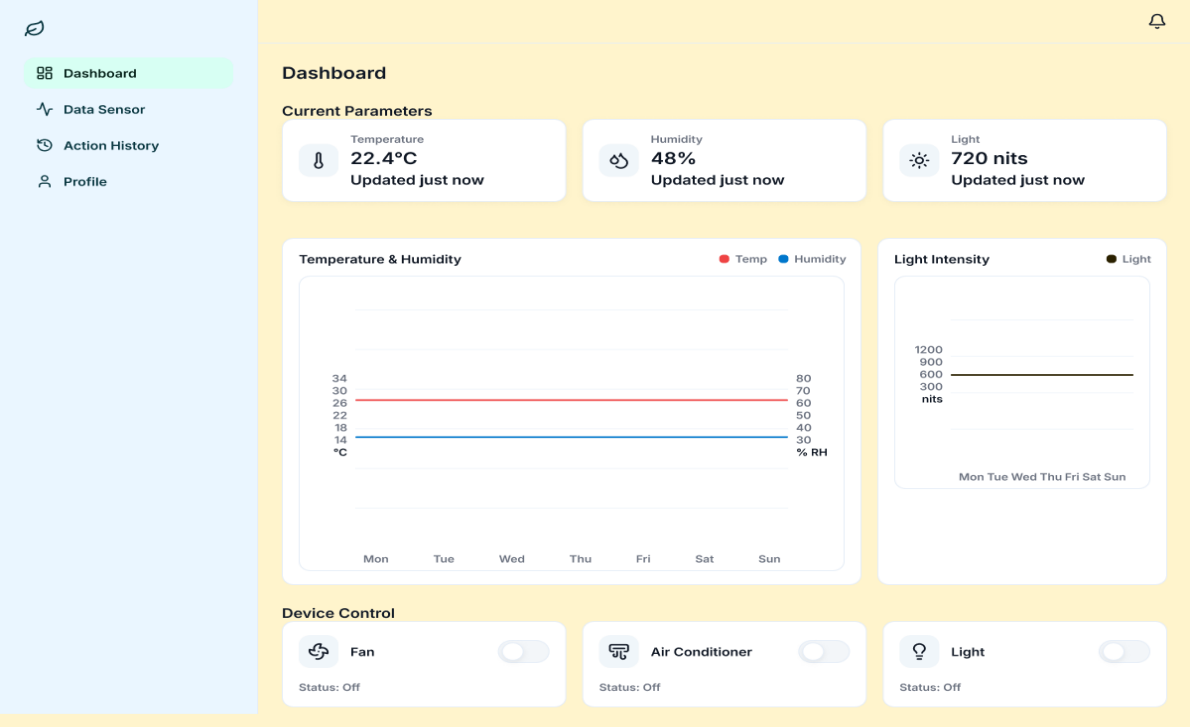


Hình 1 :Thiết kế hệ thống tổng thể

Hệ thống là mô hình IoT cơ bản sử dụng ESP32 làm bộ điều khiển trung tâm. ESP32 được nối với cảm biến DHT11 để đo nhiệt độ, độ ẩm, cảm biến ánh sáng để ghi nhận cường độ sáng, và ba đèn LED mô phỏng thiết bị điều khiển (bật/tắt). Dữ liệu từ các cảm biến được ESP32 đọc và gửi qua giao thức MQTT đến server Node.js, nơi dữ liệu được lưu vào MySQL và hiển thị lên giao diện web theo thời gian thực. Khi người dùng gửi lệnh điều khiển trên web, server gửi lại lệnh MQTT đến ESP32 để thay đổi trạng thái LED, tạo thành vòng khép kín thu thập – truyền – điều khiển – phản hồi trong hệ thống IoT.

CHƯƠNG 2:GIAO DIỆN

1.Trang Dashboard



Hình 2: dashboard

Giao diện Dashboard đóng vai trò trung tâm của hệ thống, được thiết kế trực quan với các khối thông tin và công cụ điều khiển bố trí hợp lý. Phía bên trái màn hình là thanh menu điều hướng màu xanh nhạt gồm bốn mục: *Dashboard*, *Data Sensor*, *Action History* và *Profile*, giúp người dùng dễ dàng chuyển đổi giữa các chức năng.

Phần chính của giao diện được chia thành ba khu vực rõ ràng:

a. Current Parameters (Thông số hiện tại):
Ba khối thông tin nằm ngang ở phía trên cùng hiển thị dữ liệu môi trường theo thời gian thực.
   - Temperature (Nhiệt độ)**:** 22.4°C, minh họa bằng biểu tượng nhiệt kế.
   - Humidity (Độ ẩm): 48%, minh họa bằng biểu tượng giọt nước.
   - Light (Ánh sáng): 720 nits, minh họa bằng biểu tượng mặt trời.
   Các thông số này đều có trạng thái *Updated just now*, thể hiện rằng dữ liệu được cập nhật liên tục và chính xác.

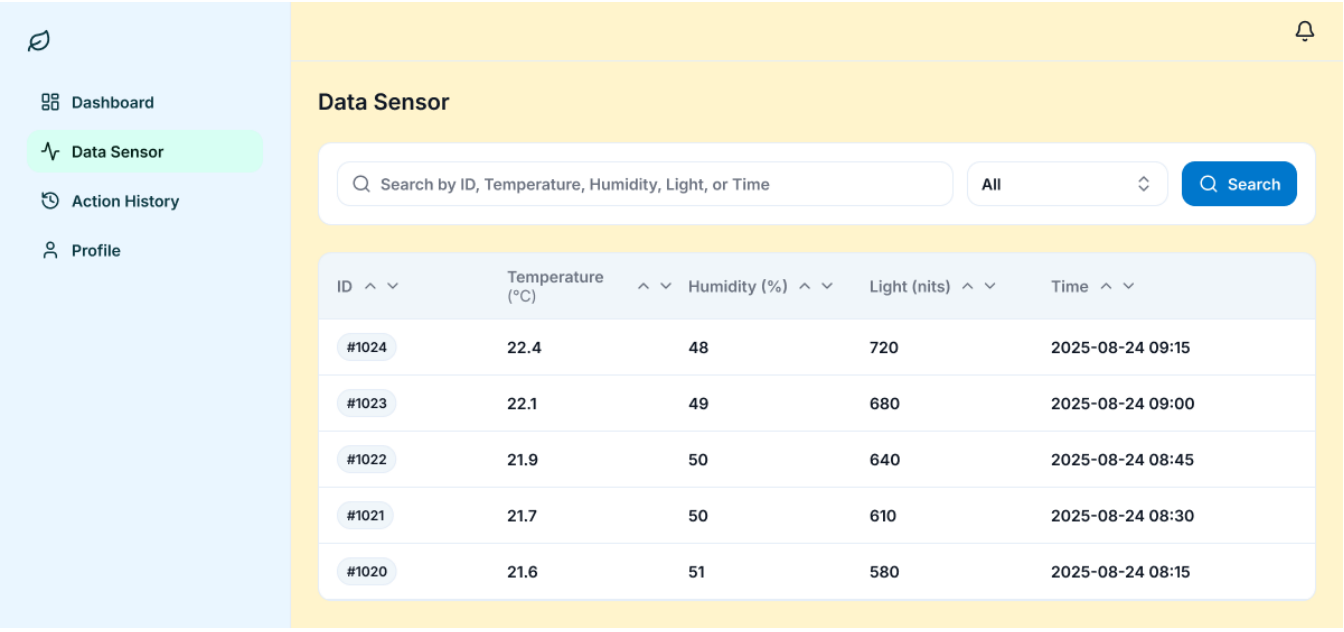
b. Monitoring Charts (Biểu đồ theo dõi):
Nằm ở phần giữa, gồm hai biểu đồ:

- Temperature & Humidity**:** Biểu đồ đường kết hợp, trong đó đường màu đỏ biểu diễn nhiệt độ và đường màu xanh dương biểu diễn độ ẩm. Trục tung bên trái thể hiện nhiệt độ (°C), trục tung bên phải thể hiện độ ẩm (%RH), còn trục hoành hiển thị theo ngày trong tuần (Mon – Sun).
- Light Intensity**:** Biểu đồ đường riêng cho cường độ ánh sáng, biểu diễn bằng đường màu đen, đơn vị tính là *nits* và cũng được sắp xếp theo các ngày trong tuần.

c. Device Control (Điều khiển thiết bị):
Nằm ở phía dưới cùng, khu vực này cho phép quản lý ba thiết bị điện tử chính:
- Fan (Quạt): biểu tượng cánh quạt với công tắc bật/tắt.
- Air Conditioner (Điều hòa**):** biểu tượng máy điều hòa với công tắc bật/tắt.
- Light (Đèn): biểu tượng bóng đèn với công tắc bật/tắt.
  Trạng thái hiện tại của cả ba thiết bị đều là *Off*, được hiển thị rõ ràng ngay dưới tên thiết bị.

Nhìn chung, giao diện này cung cấp cho người dùng cái nhìn tổng quan về các thông số môi trường (nhiệt độ, độ ẩm, ánh sáng) theo thời gian thực, đồng thời hỗ trợ theo dõi xu hướng biến đổi thông qua biểu đồ trực quan. Ngoài ra, hệ thống còn tích hợp chức năng điều khiển thiết bị ngay trên cùng một màn hình, giúp người dùng quản lý và thao tác thuận tiện, nhanh chóng.

2.Trang Data Sensor



Hình 3: datasensor

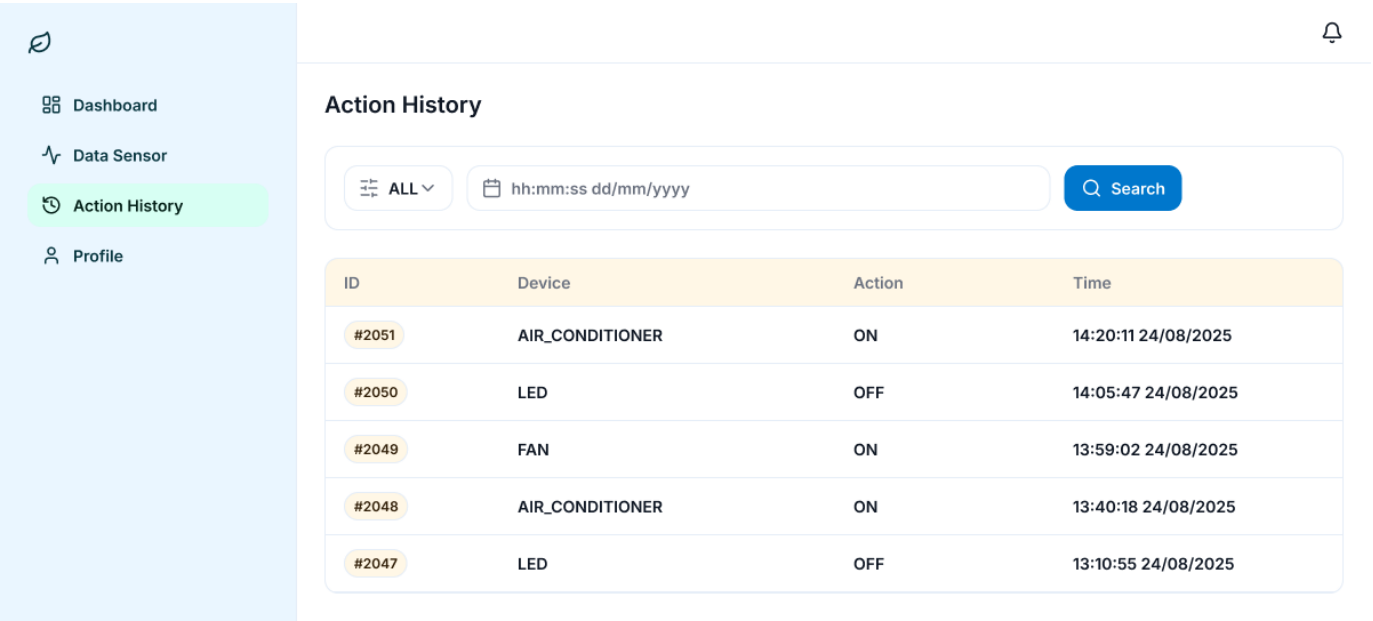
Giao diện Data Sensor được thiết kế gọn gàng, bố cục khoa học, gồm hai phần chính: thanh tìm kiếm và bảng dữ liệu. Phía trên cùng là thanh tìm kiếm, kéo dài toàn chiều ngang màn hình, cho phép người dùng nhập từ khóa theo các tiêu chí như ID, Temperature, Humidity, Light hoặc Time. Bên cạnh đó còn có một menu thả xuống để chọn bộ lọc hiển thị và nút Search màu xanh giúp thao tác nhanh, rõ ràng và nổi bật trên nền vàng nhạt.

Phía dưới là bảng dữ liệu cảm biến, được chia thành năm cột:

- ID: hiển thị mã định danh của từng bản ghi, ví dụ #1024, #1023…
- Temperature (°C): thể hiện giá trị nhiệt độ đo được, dao động từ 21.6°C đến 22.4°C.
- Humidity (%): biểu thị độ ẩm theo phần trăm, có sự thay đổi nhẹ từ 48% đến 51%.
- Light (nits): ghi lại cường độ ánh sáng, giảm dần từ 720 xuống 580 nits qua các mốc thời gian.
- Time: thể hiện chính xác thời điểm ghi nhận dữ liệu, cách nhau 15 phút (từ 08:15 đến 09:15 ngày 24/08/2025).

Mỗi hàng dữ liệu trong bảng là một bản ghi đầy đủ của cảm biến tại thời điểm cụ thể, qua đó người dùng có thể dễ dàng so sánh sự thay đổi của nhiệt độ, độ ẩm và ánh sáng theo từng mốc giờ. Nhờ bố cục trực quan này, giao diện Data Sensor không chỉ cung cấp dữ liệu chi tiết mà còn hỗ trợ hiệu quả cho việc giám sát và phân tích xu hướng môi trường theo thời gian.

3.Trang Action History



Hình 4: action History

Giao diện Action History được thiết kế nhằm ghi lại và quản lý toàn bộ lịch sử thao tác điều khiển thiết bị trong hệ thống.

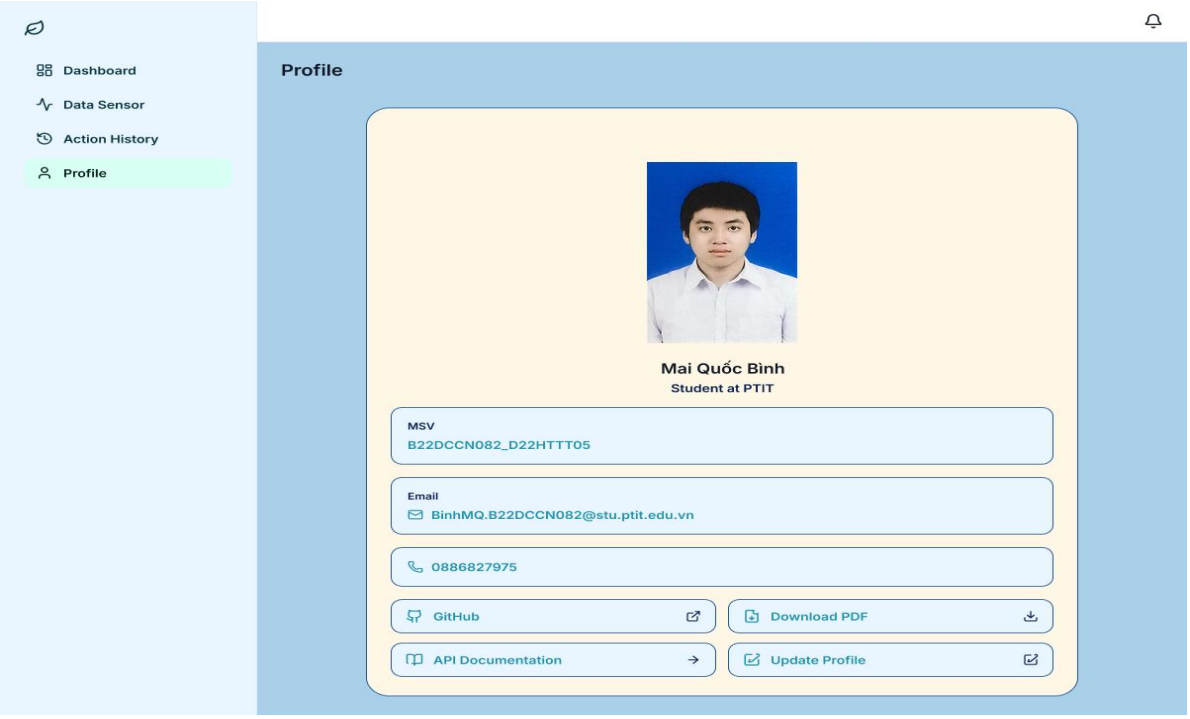
Phần chính của giao diện gồm hai khu vực chính:

1. Thanh công cụ tìm kiếm:
   Nằm ở phía trên cùng, thanh tìm kiếm cho phép người dùng lọc dữ liệu theo hai tiêu chí. Thứ nhất là menu thả xuống (*ALL*) để lựa chọn hiển thị toàn bộ hoặc chỉ riêng một loại thiết bị nhất định. Thứ hai là ô nhập thời gian theo định dạng *hh:mm:ss dd/mm/yyyy* nhằm tìm kiếm chính xác theo mốc thời gian mong muốn. Nút Search màu xanh nằm bên phải, tạo sự nổi bật và dễ thao tác.
2. Bảng lịch sử thao tác:
   Được bố trí ngay bên dưới, bảng hiển thị dữ liệu lịch sử theo bốn cột:
   - ID: mã định danh cho từng thao tác, ví dụ #2051, #2050…
   - Device: tên thiết bị được điều khiển, bao gồm *AIR_CONDITIONER, LED* và *FAN*.
   - Action: hành động đã thực hiện, hiển thị rõ là *ON* (bật) hoặc *OFF* (tắt).
   - Time: thời điểm chính xác ghi nhận hành động, bao gồm giờ, phút, giây và ngày tháng năm.

Các bản ghi được sắp xếp theo thứ tự thời gian giảm dần, ví dụ bản ghi #2051 cho biết thiết bị *AIR_CONDITIONER* được bật lúc 14:20:11 ngày 24/08/2025, trong khi bản ghi #2050 ghi nhận việc tắt *LED* lúc 14:05:47 cùng ngày.

Nhờ bố cục trực quan và thông tin chi tiết, giao diện này không chỉ giúp người dùng dễ dàng theo dõi và kiểm soát các thao tác đã thực hiện mà còn đảm bảo tính minh bạch và thuận tiện trong quản lý hệ thống.

4.Trang profile



Hình 5: trang profile

Giao diện Profile được thiết kế nhằm hiển thị và quản lý thông tin cá nhân của sinh viên trong hệ thống.
Phần chính của giao diện gồm hai khu vực chính:

1. Khu vực thông tin cá nhân:
    - Nằm ở vị trí trung tâm, nổi bật với ảnh chân dung của sinh viên.
    - Bên dưới ảnh là tên đầy đủ **(**Mai Quốc Bình**)** và chức danh **(**Student at PTIT**),** giúp xác định rõ chủ thể sử dụng hệ thống.
    - Các thông tin được trình bày dưới dạng thẻ, bao gồm:
        - MSV (Mã sinh viên): B22DCCN082_D22HTTT05
        - Email**:** BinhMQ.B22DCCN082@stu.ptit.edu.vn
        - Số điện thoại**:** 0886827975
2. Thanh chức năng hỗ trợ:
    - Được bố trí ngay dưới phần thông tin cá nhân, hiển thị dưới dạng các nút bấm có biểu tượng trực quan.
    - Các chức năng chính bao gồm:
        - GitHub**:** liên kết đến kho lưu trữ mã nguồn.
        - API Documentation: truy cập tài liệu API.
        - Download PDF: tải hồ sơ cá nhân dưới định dạng PDF.
        - Update Profile: cập nhật thông tin hồ sơ.

Nhờ cách bố trí gọn gàng, màu sắc hài hòa và chức năng rõ ràng, giao diện Profile giúp người dùng dễ dàng nắm bắt, tra cứu cũng như chỉnh sửa thông tin của mình. Đồng thời, các nút chức năng hỗ trợ tạo sự thuận tiện trong việc kết nối và quản lý dữ liệu cá nhân.

Chương 3. Thiết kế tổng thể và chi tiết

I. Kiến trúc hệ thống

1. Tổng quan kiến trúc

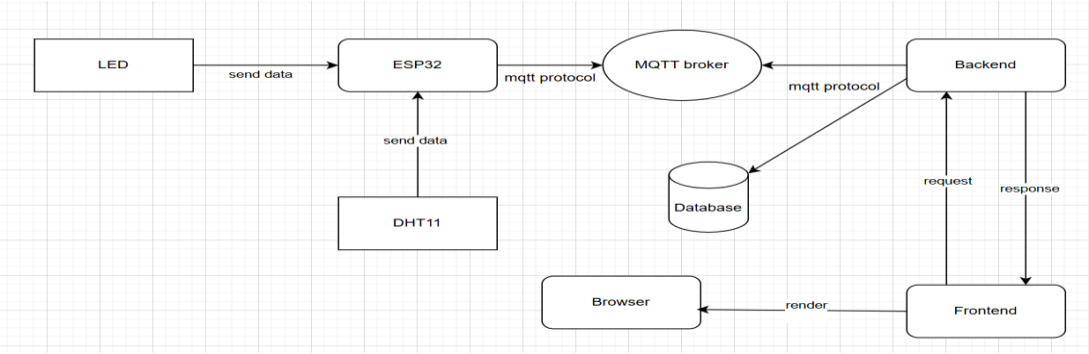
Hệ thống IoT được tổ chức theo kiến trúc nhiều lớp, gồm các thành phần chính:

- ESP32: phần cứng đọc dữ liệu cảm biến và điều khiển thiết bị (quạt, đèn LED, điều hòa).
- MQTT Broker: trung gian truyền thông điệp giữa ESP32 và Backend theo cơ chế publish/subscribe.
- Backend (Node.js)**:** xử lý nghiệp vụ, cung cấp API cho Frontend và ghi/đọc dữ liệu.
- Database (MySQL)**:** lưu trữ dữ liệu cảm biến và lịch sử thao tác điều khiển.
- Frontend (React)**:** giao diện người dùng để hiển thị số liệu và gửi lệnh điều khiển.

2. Luồng dữ liệu và tương tác

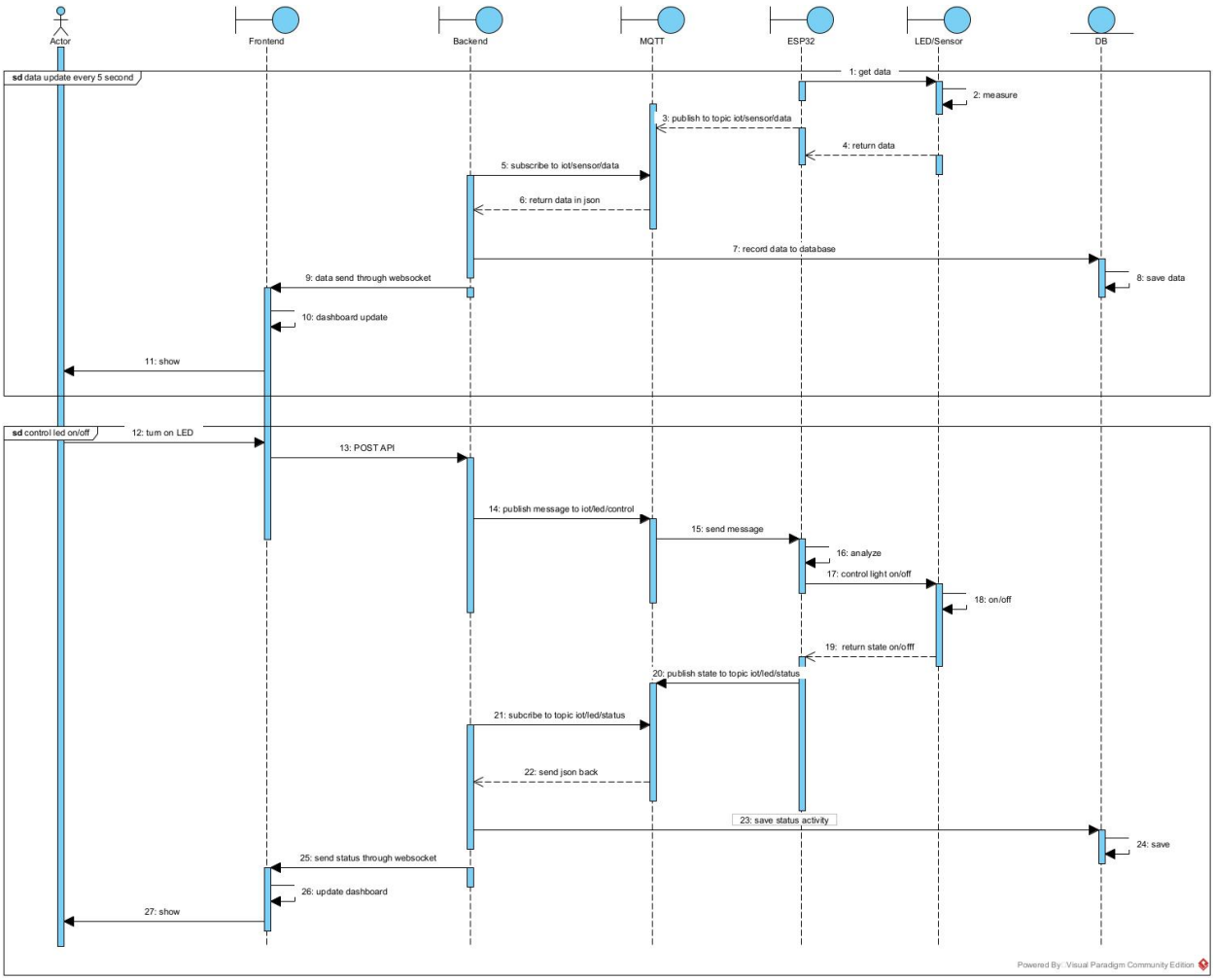
a) Luồng dữ liệu cảm biến
ESP32 thu thập các giá trị (nhiệt độ, độ ẩm, ánh sáng) → phát hành bản tin qua MQTT đến MQTT Broker → Backend nhận bản tin, xử lý và lưu vào MySQL → Frontend định kỳ gọi API của Backend để lấy dữ liệu mới nhất và hiển thị cho người dùng.

b) Luồng điều khiển thiết bị
Người dùng thao tác trên Frontend → Frontend gửi yêu cầu điều khiển đến Backend qua API → Backend xử lý và phát lệnh điều khiển qua MQTT đến ESP32 → ESP32 nhận lệnh và thực thi trên thiết bị tương ứng.



Hình 6: luồng điều khiển thiết bị

*Sequence diagram



Hình 7: biểu đồ tuần tự

1. Hardware nhận và đọc dữ liệu cảm biến.
2. Dữ liệu được gửi lên MQTT.
3. Backend subscribe vào topic để nhận dữ liệu.
4. Backend lưu dữ liệu vào DataSensor.
5. Backend xử lý logic nội bộ và trả kết quả (ack).

Người dùng mở web và lấy dữ liệu mới nhất:

6. Người dùng mở website (Frontend).
7. Frontend gọi API "getLatest".
8. Backend định tuyến đến /api/sensor-data.
9. Backend truy vấn DataSensor.
10. DB trả kết quả cho Backend.
11. Backend trả dữ liệu cho Frontend.
12. Frontend hiển thị cho Actor.

Điều khiển thiết bị (bật/tắt đèn):
13. Actor bấm “turn on light”.
14. Frontend gọi API "light on/off".
15. Backend xử lý tại /api/control.
16–18. Backend xác nhận lệnh đã được xử lý.
19. Backend gửi lệnh xuống MQTT.
20. MQTT chuyển lệnh tới Hardware.
21. Hardware nhận lệnh và thực thi.
22. Frontend cập nhật trạng thái "light on/off".
23. Hardware/MQTT pub lại trạng thái thực tế.
24. Backend ghi lại ActionHistory (deviceID, userID, action, payload, executedAt).
25. Backend trả kết quả về cho Frontend.
26. Frontend hiển thị kết quả cuối.

Xem lịch sử thao tác:
27. Actor mở "History".
28. Frontend gọi /api/history.
29. Backend xử lý yêu cầu.
30–31. Backend truy vấn và nhận dữ liệu từ ActionHistory.
32. Backend trả danh sách lịch sử thao tác.
33. Frontend hiển thị.

Xem dữ liệu cảm biến:
34. Actor mở "Data".
35. Frontend gọi /api (lấy dữ liệu).
36. Backend gọi /api/sensor-data (hoặc truy vấn DataSensor).
37–38. Dữ liệu được lấy từ DataSensor và trả về Backend.
39. Backend trả dữ liệu cho Frontend.
40. Frontend hiển thị dữ liệu.

*DataBase

Hình 8: database

Result Grid | Filter Rows: | Edit: | Export/Im

id	device_name	action	timestamp	description
1	Fan	on	2025-10-10 10:45:47	Fan control command: on
2	Fan	on	2025-10-10 10:46:05	Fan control command: on
3	Fan	on	2025-10-10 10:46:42	Fan control command: on
4	Fan	on	2025-10-10 10:47:06	Fan control command: on
5	Fan	on	2025-10-10 10:48:38	Fan control command: on
6	Fan	off	2025-10-10 10:48:40	Fan control command: off
7	Light	on	2025-10-10 10:48:42	Light control command: on
8	Light	on	2025-10-10 10:48:42	Light control command: on
9	Light	on	2025-10-10 10:48:43	Light control command: on
10	Fan	on	2025-10-10 10:48:43	Fan control command: on
11	Air Conditioner	on	2025-10-10 10:48:46	Air Conditioner control co...
12	Air Conditioner	off	2025-10-10 10:48:47	Air Conditioner control co...
13	Light	on	2025-10-10 10:48:48	Light control command: on
14	Light	off	2025-10-10 10:48:49	Light control command: off
15	Fan	on	2025-10-10 10:50:14	Fan control command: on
16	Fan	on	2025-10-10 10:50:16	Fan control command: on
17	Fan	off	2025-10-10 10:50:17	Fan control command: off
18	Air Conditioner	on	2025-10-10 10:50:18	Air Conditioner control co...

Hình 9 : bảng action_history

Result Grid | Filter Rows: | Edit:

id	temperature	light	humidity	time
1	33.70	393	61	2025-10-10 10:39:02
2	33.70	393	61	2025-10-10 10:39:07
3	33.70	383	61	2025-10-10 10:39:12
4	33.60	365	61	2025-10-10 10:39:17
5	33.60	355	61	2025-10-10 10:39:22
6	33.70	350	61	2025-10-10 10:39:27
7	33.70	368	61	2025-10-10 10:39:32
8	33.70	361	61	2025-10-10 10:39:37
9	33.70	361	61	2025-10-10 10:39:42
10	33.70	395	61	2025-10-10 10:39:47
11	33.70	356	61	2025-10-10 10:39:52
12	33.70	360	61	2025-10-10 10:39:57
13	33.70	369	61	2025-10-10 10:40:02
14	33.70	360	61	2025-10-10 10:40:07
15	33.70	332	61	2025-10-10 10:40:12
16	33.70	302	61	2025-10-10 10:40:17
17	33.70	288	61	2025-10-10 10:40:22
18	33.70	278	61	2025-10-10 10:40:27

Hình 10 : bảng sensor_data

CHƯƠNG 4:CODE

1.API

*)API lấy dữ liệu cảm biến

```
//api data
app.get('/api/data', async (req, res) => {
  try {
    const [rows] = await poolPromise.query(`
      SELECT
        id,
        temperature,
        humidity,
        light,
        DATE_FORMAT(\`time\`, '%Y-%m-%d %H:%i:%s') AS created_at,
        DATE_FORMAT(\`time\`, '%Y-%m-%d %H:%i:%s') AS \`time\`
      FROM sensor_data
      ORDER BY \`time\` DESC
      LIMIT 1
    `);

    res.json({
      sensors: rows[0] || {},
      leds: currentLedStatus
    });
  } catch (error) {
    console.error('Error fetching real-time data:', error.message);
    res.status(500).json({ error: 'Failed to fetch real-time data' });
  }
});
```

Hình 11: api lấy dữ liệu cảm biến mới nhất

*)API điều khiển đèn

```
app.post('/api/control', (req, res) => {
  console.log('Request body:', req.body);
  const { deviceName, action } = req.body;

  if (!deviceName || !action) {
    return res.status(400).json({ error: 'Thiếu deviceName hoặc action' });
  }

  const normalizedDevice = normalizeDeviceName(deviceName);
  const normalizedAction = normalizeAction(action);

  if (!/^led[123]$/.test(normalizedDevice) || !/^(on|off)$/.test(normalizedAction)) {
    return res.status(400).json({ error: 'Giá trị deviceName/action không hợp lệ' });
  }

  const description = `Turned ${normalizedAction} the ${normalizedDevice}`;
  const sql = `INSERT INTO action_history (device_name, action, timestamp, description) VALUES (?, ?, NOW(), ?)`;
  const params = [normalizedDevice, normalizedAction, description];

  pool.query(sql, params, (err) => {
    if (err) {
      console.error('Lỗi ghi lịch sử điều khiển:', err);
      return res.status(500).json({ error: 'Lỗi hệ thống' });
    }
```

Hình 12: Api điều khiển đèn và lưu lịch sử

*)API tìm kiếm và phân trang data

```
//api tìm kiếm và phân trang data
app.get('/api/data/history', async (req, res) => {
  try {
    const page = parseInt(req.query.page, 10) || 1;
    const limitParam = parseInt(req.query.limit, 10) || 13;
    const limit = Number.isFinite(limitParam) && limitParam > 0 ? Math.min(limitParam, 100) : 13;
  const offset = (page - 1) * limit;
    const rawSearch = (req.query.search || req.query.searchQuery || '').trim();
    const sortKeyRaw = (req.query.sortKey || req.query.sortColumn || 'created_at').toLowerCase();
    const sortDirectionRaw = (req.query.sortDirection || req.query.order || 'descending').toLowerCase();
    const searchField = (req.query.searchField || '').trim().toLowerCase();

    const allowedSortKeys = {
      id: ``id``,
      temperature: ``temperature``,
      humidity: ``humidity``,
      light: ``light``,
      created_at: ``time``
    };

    const sortColumn = allowedSortKeys[sortKeyRaw] || ``time``;
    const sortDirection = sortDirectionRaw === 'ascending' || sortDirectionRaw === 'asc' ? 'ASC' : 'DESC';

  let whereClause = '';
    const params = [];

    if (rawSearch) {
      const searchTerm = `%${rawSearch}%`;

      const dateTimePattern = /^\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}$/;
      const isDateTime = dateTimePattern.test(rawSearch);

      if (isDateTime) {

        whereClause = `
          WHERE DATE_FORMAT(\`time\`, '%Y-%m-%d %H:%i:%s') = ?
        `;
        params.push(rawSearch);
      } else if (searchField) {

        const fieldMap = {
          'temperature': 'CAST(temperature AS CHAR)',
          'humidity': 'CAST(humidity AS CHAR)',
          'light': 'CAST(light AS CHAR)',
          'id': 'CAST(id AS CHAR)'
        };
        const searchColumn = fieldMap[searchField];

        if (searchColumn) {
          whereClause = `WHERE ${searchColumn} LIKE ?`;
          params.push(searchTerm);
        }
      } else {

    whereClause = `
      WHERE CAST(id AS CHAR) LIKE ?
        OR CAST(temperature AS CHAR) LIKE ?
        OR CAST(humidity AS CHAR) LIKE ?
        OR CAST(light AS CHAR) LIKE ?
           OR DATE_FORMAT(\`time\`, '%Y-%m-%d %H:%i:%s') LIKE ?
    `;
    params.push(searchTerm, searchTerm, searchTerm, searchTerm, searchTerm);
  }
    }

    const countQuery = `SELECT COUNT(*) AS totalItems FROM sensor_data ${whereClause}`;
    const [[countRow]] = await poolPromise.query(countQuery, params);
    const totalItems = countRow?.totalItems || 0;
    const totalPages = totalItems > 0 ? Math.ceil(totalItems / limit) : 0;
```

```
404
405     const countQuery = `SELECT COUNT(*) AS totalItems FROM sensor_data ${whereClause}`;
406     const [[countRow]] = await poolPromise.query(countQuery, params);
407     const totalItems = countRow?.totalItems || 0;
408     const totalPages = totalItems > 0 ? Math.ceil(totalItems / limit) : 0;
409
410     const dataQuery = `
411       SELECT
412         id,
413         temperature,
414         humidity,
415         light,
416         DATE_FORMAT(\`time\`, '%Y-%m-%d %H:%i:%s') AS created_at,
417         DATE_FORMAT(\`time\`, '%Y-%m-%d %H:%i:%s') AS \`time\`
418       FROM sensor_data
419       ${whereClause}
420       ORDER BY ${sortColumn} ${sortDirection}
421       LIMIT ? OFFSET ?
422     `;
423
424     const [data] = await poolPromise.query(dataQuery, [...params, limit, offset]);
425
426       res.json({
427       data,
428         pagination: {
429         total: totalItems,
430         totalPages,
431           currentPage: page,
432           limit
433         }
434     });
435   } catch (error) {
436     console.error('Error fetching sensor history:', error.message);
437     res.status(500).json({ error: 'Failed to fetch sensor history' });
438   }
439 });
```

Hình 13 : api tìm kiếm và phân trang data

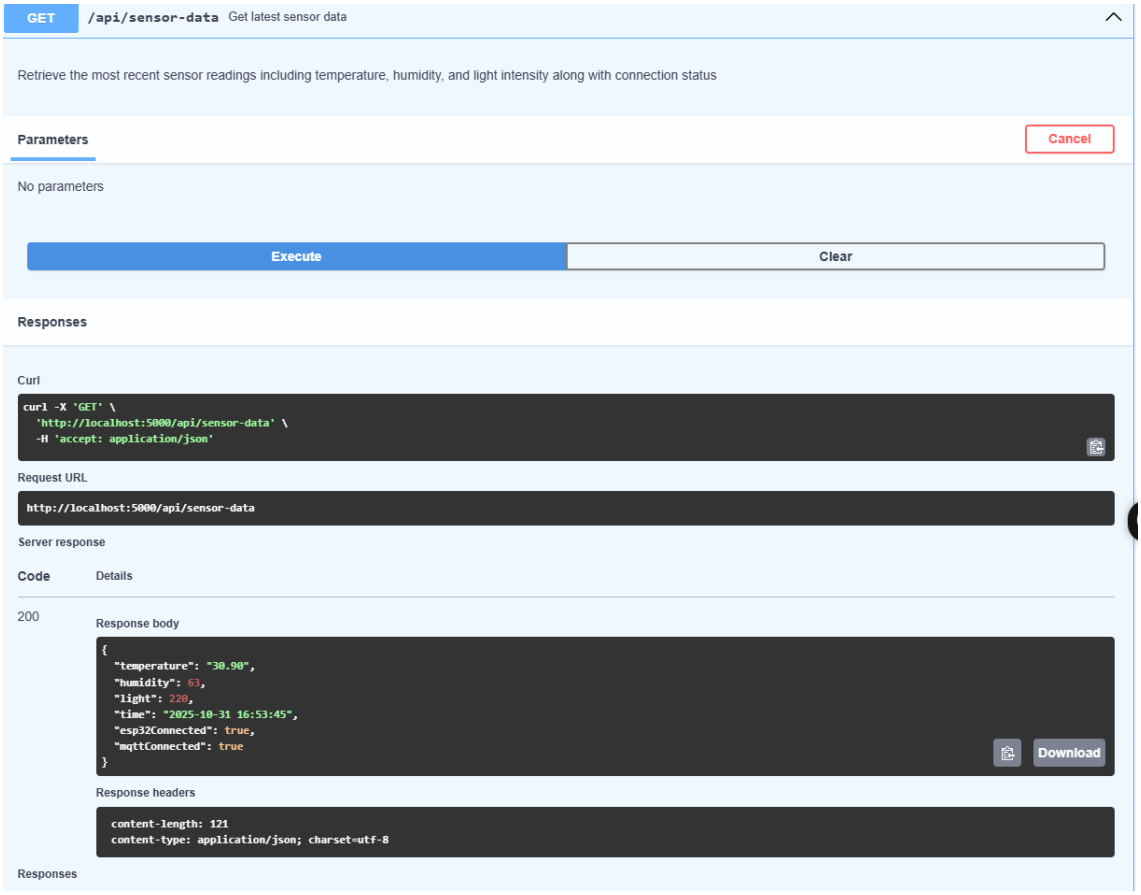
*)API tìm kiếm và phân trang history

```
//api tìm kiếm và phân trang history

app.get('/api/actions/history', async (req, res) => {
  try {
  const page = parseInt(req.query.page) || 1;
    const limitParam = parseInt(req.query.limit) || 10;
    const limit = Number.isFinite(limitParam) && limitParam > 0 ? Math.min(limitParam, 100) : 10;
  const offset = (page - 1) * limit;
    const searchQuery = (req.query.searchQuery || '').trim();
    const filterType = (req.query.filterType || '').trim();
    const deviceName = (req.query.device_name || '').trim();
    const action = (req.query.action || '').trim();
  const sortColumn = req.query.sortColumn || 'timestamp';
  const sortDirection = (req.query.sortDirection || 'desc').toLowerCase();

  const allowedCols = ['device_name', 'action', 'timestamp'];
    const sortColSafe = allowedCols.includes(sortColumn) ? `\`${sortColumn}\`` : ``timestamp``;
  const sortDirSafe = sortDirection === 'asc' ? 'ASC' : 'DESC';

  let whereClause = '';
  const params = [];
    const conditions = [];

    // Handle device_name filter
    if (deviceName) {
      conditions.push(``device_name` = ?');
      params.push(deviceName);
    }

    // Handle action filter
    if (action) {
      conditions.push(``action` = ?');
      params.push(action.toLowerCase());
    }

    // Handle datetime search
    if (searchQuery) {
      const searchTerm = `%${searchQuery}%`;
      const dateTimePattern = /^\d{2}\/\d{2}\/\d{4}, \d{2}:\d{2}:\d{2}$/;
      const isDateTime = dateTimePattern.test(searchQuery);

      if (isDateTime) {
        conditions.push('DATE_FORMAT(`timestamp`, "%d/%m/%Y, %H:%i:%s") = ?');
        params.push(searchQuery);
      } else {
        conditions.push(`(CAST(id AS CHAR) LIKE ? OR device_name LIKE ? OR action LIKE ? OR DATE_FORMAT(\`timestamp\`, "%d/%m/%Y, %H:%i:%s") LIKE ?)`);
        params.push(searchTerm, searchTerm, searchTerm, searchTerm);
      }
    }

    // Combine all conditions with AND
    if (conditions.length > 0) {
      whereClause = ' WHERE ' + conditions.join(' AND ');
    }

    const countQuery = `SELECT COUNT(*) AS totalItems FROM action_history ${whereClause}`;
    const [[countRow]] = await poolPromise.query(countQuery, params);
    const totalItems = countRow?.totalItems || 0;
    const totalPages = totalItems > 0 ? Math.ceil(totalItems / limit) : 0;

    const dataQuery = `
      SELECT
        id,
        device_name,
        action,
        DATE_FORMAT(\`timestamp\`, '%d/%m/%Y, %H:%i:%s') AS timestamp
      FROM action_history
      ${whereClause}
      ORDER BY ${sortColSafe} ${sortDirSafe}
      LIMIT ? OFFSET ?
    `;

    const [data] = await poolPromise.query(dataQuery, [...params, limit, offset]);

      res.json({
      data,
        pagination: {
        total: totalItems,
        totalPages,
          currentPage: page,
          limit
        }
    });
  } catch (error) {
    console.error('Error fetching action history:', error.message);
    res.status(500).json({ error: 'Failed to fetch action history' });
  }
});
```

Hình 14 : Api tìm kiếm và phân trang history

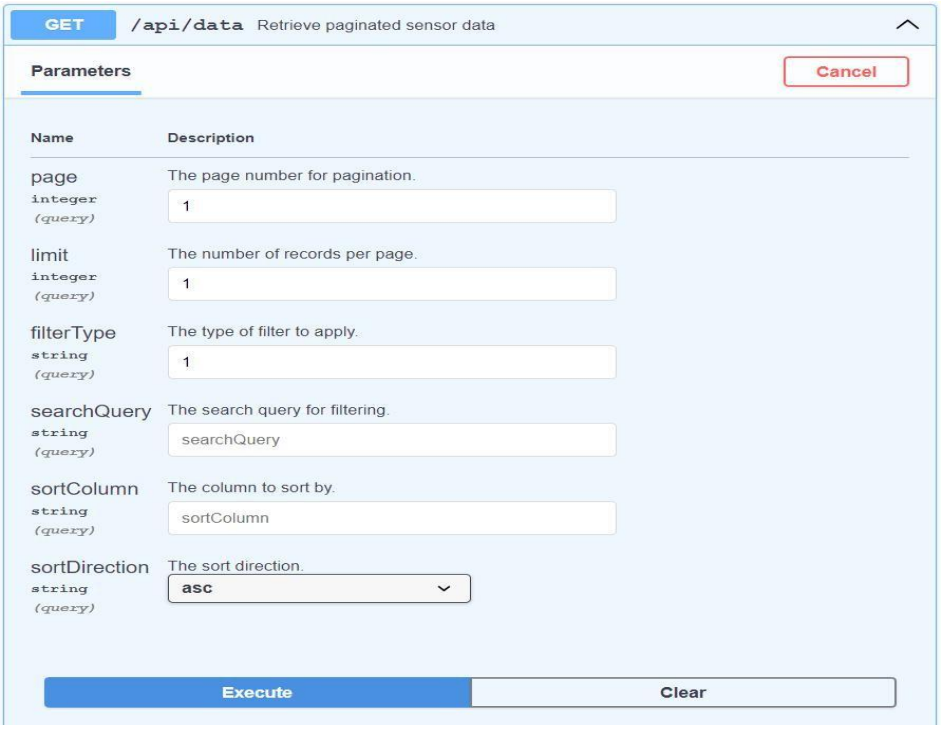
2.SWAGGER

*)api/sensor-data

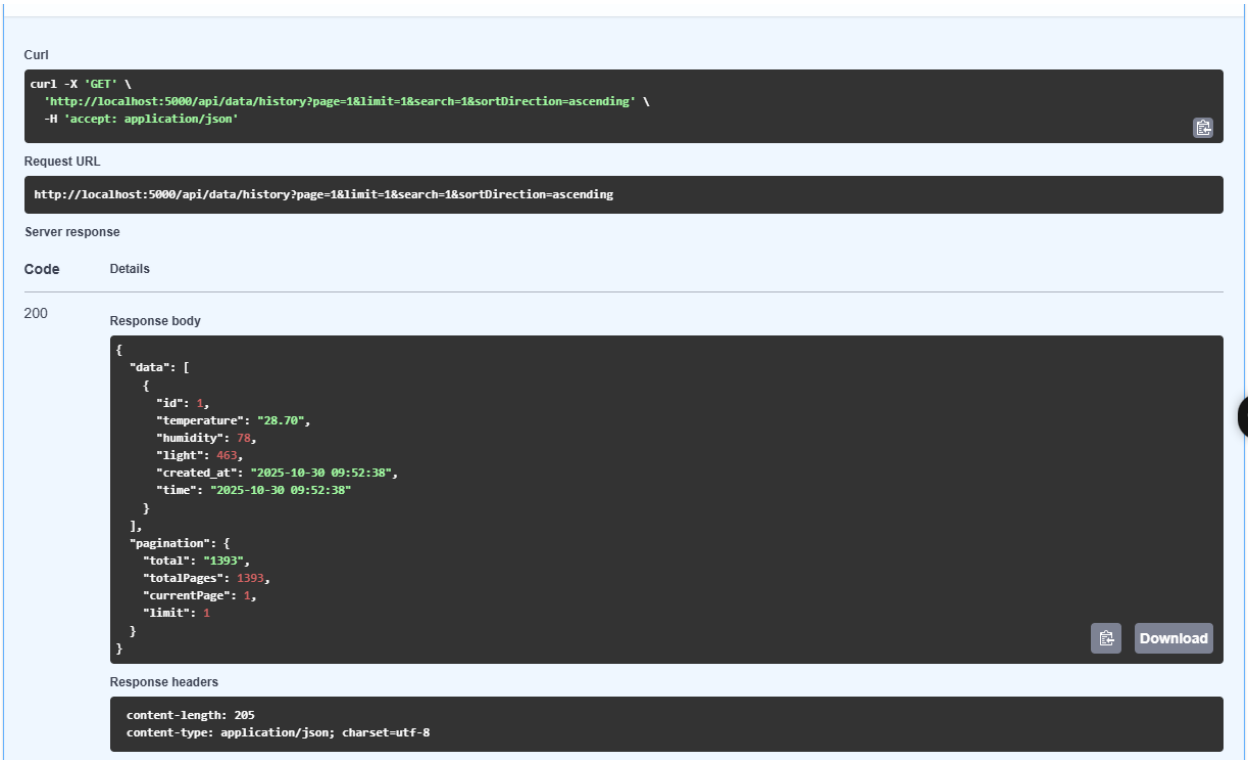


Hình 15:api/sensor-data

*)api/data

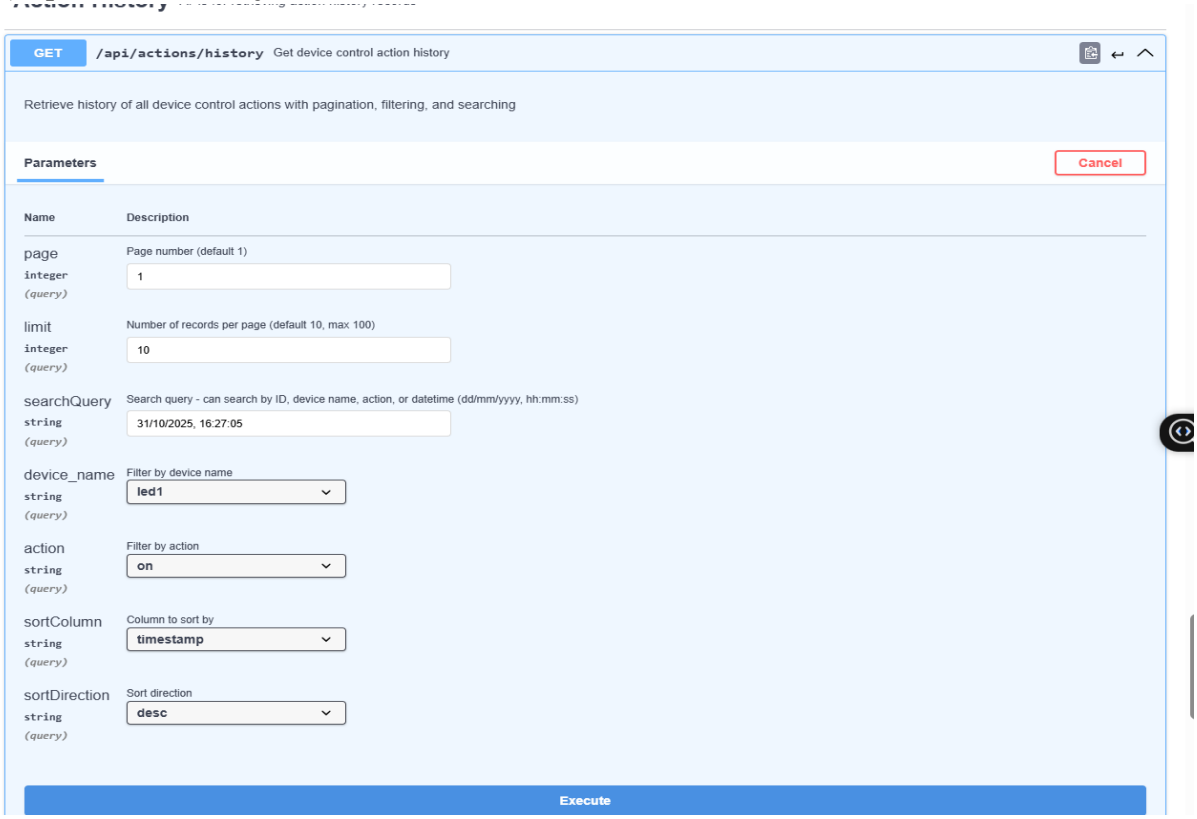


Hình 16 :api/data

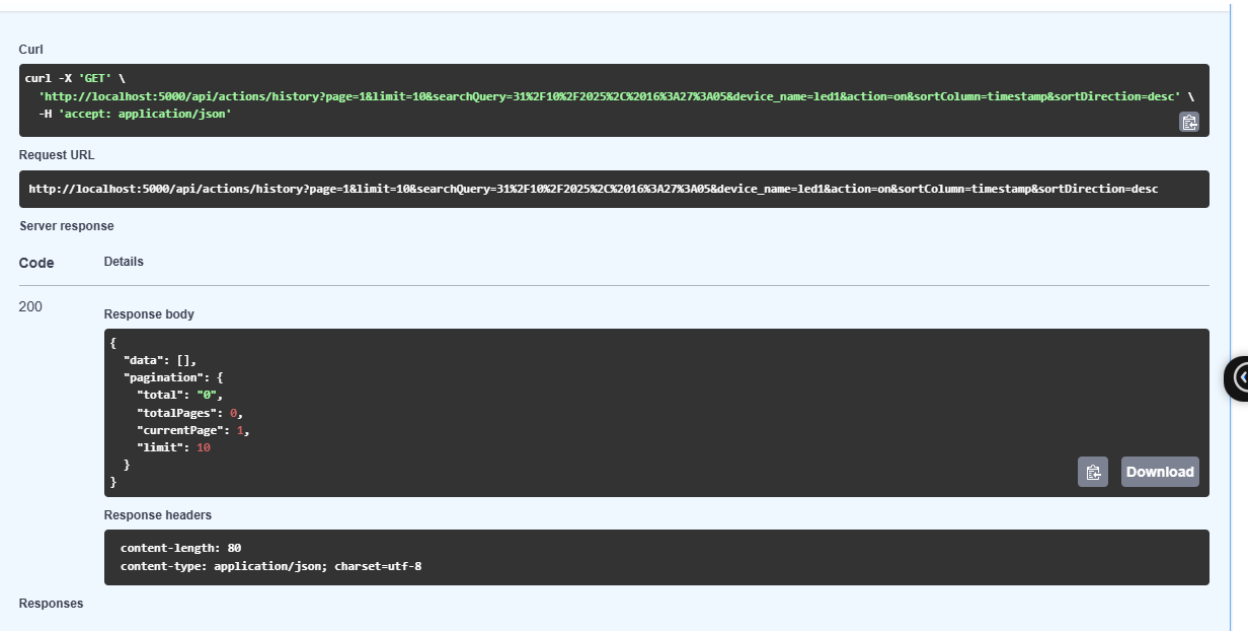


Hình 17:response api/data

*)Api action/history



Hình 18:api action/history



Hình 19:response action/history

*)Api control

{

"deviceName": "fan",

"action": "on"

}

Hình 20: api/control

CHƯƠNG 5:KẾT QUẢ

I. Kết quả triển khai hệ thống

Sau khi hoàn thiện quá trình xây dựng, hệ thống IoT Dashboard đã được triển khai thành công với đầy đủ các thành phần phần cứng và phần mềm. Mô hình thử nghiệm gồm:

- Phần cứng: Vi điều khiển ESP32 kết nối với cảm biến DHT11 (đo nhiệt độ và độ ẩm), cảm biến ánh sáng LDR, và ba đèn LED mô phỏng thiết bị điện trong nhà (quạt, đèn, điều hòa).
- Phần mềm: Hệ thống Backend Node.js kết hợp với MQTT Broker trung gian, cơ sở dữ liệu MySQL, và giao diện Frontend React hiển thị dữ liệu theo thời gian thực.

Sau khi kết nối, ESP32 hoạt động ổn định, truyền dữ liệu cảm biến lên server thông qua giao thức MQTT. Dữ liệu được ghi nhận liên tục vào bảng sensor_data trong MySQL và hiển thị trực tiếp trên Dashboard dưới dạng biểu đồ động.
Khi người dùng gửi lệnh điều khiển thiết bị (ví dụ bật đèn), lệnh được gửi qua API đến Backend, sau đó chuyển đến ESP32 qua MQTT. Thiết bị phản hồi tức thì và trạng thái được cập nhật chính xác trên giao diện web.

Hệ thống đạt được độ trễ trung bình dưới 1 giây, đảm bảo khả năng phản hồi nhanh và chính xác giữa các thành phần.
Tất cả lịch sử thao tác được lưu trong bảng action_history, giúp người dùng dễ dàng tra cứu, quản lý và kiểm soát.

II. Kết quả giao diện vận hành

1. **Giao diện Dashboard:**
    - Hiển thị đầy đủ thông số nhiệt độ, độ ẩm, ánh sáng theo thời gian thực.
    - Biểu đồ đường cập nhật mượt mà và ổn định.
    - Các nút điều khiển thiết bị hoạt động tốt, phản hồi trạng thái chính xác (On/Off).
2. **Giao diện Data Sensor:**
    - Dữ liệu cảm biến được hiển thị dưới dạng bảng, có thể tìm kiếm và lọc theo thời gian hoặc ID.
    - Các giá trị thay đổi theo thời gian được cập nhật đúng với thực tế cảm biến đo được.
3. **Giao diện Action History:**
    - Hiển thị chính xác lịch sử bật/tắt thiết bị theo thứ tự thời gian.
    - Tính năng tìm kiếm giúp truy xuất nhanh các thao tác cũ.

4. **Giao diện Profile:**
    - Hoạt động ổn định, hiển thị đúng thông tin sinh viên thực hiện.
    - Các nút liên kết như GitHub, API Documentation và Update Profile đều hoạt động tốt.

III. Đánh giá hiệu quả và độ tin cậy

Tiêu chí	Kết quả đạt được	Nhận xét
Tốc độ truyền dữ liệu	< 1 giây	Nhanh, ổn định, không bị ngắt kết nối
Giao diện người dùng	Trực quan, dễ thao tác	Thiết kế rõ ràng, phù hợp cho người mới
Tính năng điều khiển	Hoạt động đúng yêu cầu	Phản hồi tức thì, đồng bộ với thiết bị
Lưu trữ dữ liệu	Chính xác, có phân trang	Dễ dàng truy xuất và phân tích
Bảo mật và xác thực	Có sử dụng JWT/Token	Đảm bảo an toàn khi truy cập hệ thống

Hệ thống đạt **mức độ ổn định cao** trong quá trình thử nghiệm, không xảy ra lỗi mất kết nối hay sai lệch dữ liệu. Việc sử dụng MQTT giúp tối ưu băng thông, đồng thời cơ chế WebSocket đảm bảo giao tiếp hai chiều giữa máy chủ và giao diện người dùng diễn ra liên tục.

IV. Hạn chế và hướng phát triển

Hạn chế:

- Hệ thống hiện mới chỉ thử nghiệm trên quy mô nhỏ (một ESP32, một cụm cảm biến).
- Chưa tích hợp cơ chế cảnh báo tự động khi vượt ngưỡng nhiệt độ/độ ẩm.
- Chưa có module quản lý đa người dùng hoặc phân quyền truy cập.

Hướng phát triển:

- Tích hợp thêm các cảm biến khác như khí gas, chuyển động, độ ồn để mở rộng phạm vi giám sát.
- Xây dựng tính năng **"**Automation Rule**"** (tự động bật quạt khi nhiệt độ > 30°C).
- Phát triển ứng dụng di động (Android/iOS) để người dùng có thể điều khiển từ xa mọi lúc, mọi nơi.

- Kết nối với nền tảng đám mây (Firebase, AWS IoT, ThingSpeak) để lưu trữ và phân tích dữ liệu ở quy mô lớn.
- Ứng dụng trí tuệ nhân tạo (AI) để dự đoán xu hướng nhiệt độ, độ ẩm và đề xuất điều khiển tối ưu.

V. Kết luận

Dự án IoT Dashboard đã hoàn thành đúng mục tiêu đề ra: xây dựng một hệ thống giám sát và điều khiển thiết bị thông minh dựa trên công nghệ IoT.
Hệ thống thể hiện tính hiệu quả, khả năng mở rộng, và tính ứng dụng thực tế cao, có thể triển khai trong các mô hình nhà thông minh, lớp học thông minh hoặc phòng thí nghiệm.
Dự án không chỉ củng cố kiến thức về phần cứng – phần mềm IoT mà còn giúp sinh viên nắm vững quy trình thiết kế hệ thống tích hợp, từ cảm biến đến giao diện web, tạo tiền đề cho các nghiên cứu và phát triển ứng dụng IoT trong tương lai.