HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN 1

BÁO CÁO HỌC PHẦN

ĐỀ TÀI: Xây dựng hệ thống IOT giám sát và điều khiển thiết bị.

Học phần	**: IoT và Ứng dụng**
Giảng viên hướng dẫn	**: Nguyễn Quốc Uy**
Sinh viên thực hiện	**: Nông Thanh Hải**
Mã sinh viên	**: B22DCCN264**

Hà Nội – 2025

Contents

DANH MỤC HÌNH ẢNH

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

Nông Thanh Hải

CHƯƠNG I: GIỚI THIỆU ĐỀ TÀI

1. Giới thiệu chung

Trong thời đại công nghệ 4.0, **Internet of Things (IoT)** trở thành một xu hướng quan trọng, cho phép các thiết bị vật lý kết nối và trao đổi dữ liệu thông qua Internet.
Việc áp dụng IoT giúp con người **giám sát, điều khiển và tự động hóa** các thiết bị trong nhà, trong công nghiệp và trong môi trường làm việc, nâng cao năng suất và sự tiện nghi.

Xuất phát từ nhu cầu đó, đề tài **"Hệ thống IoT Dashboard giám sát và điều khiển thiết bị"** được xây dựng với mục tiêu:

- Tạo một **nền tảng web trực quan** cho phép giám sát nhiệt độ, độ ẩm, ánh sáng theo thời gian thực.
- Cung cấp **khả năng điều khiển thiết bị từ xa** như quạt, đèn, điều hòa,…
- Lưu trữ và hiển thị **lịch sử hoạt động và dữ liệu cảm biến** để người dùng có thể theo dõi, phân tích.

Đề tài giúp người thực hiện **nâng cao kỹ năng lập trình fullstack (Node.js – MySQL – JavaScript)**, hiểu được cơ chế **truyền dữ liệu thời gian thực bằng MQTT và Socket.IO**, đồng thời củng cố kiến thức về **mạng máy tính và hệ thống nhúng.**

2. Mục tiêu nghiên cứu

- Xây dựng hệ thống IoT có khả năng thu thập dữ liệu môi trường từ cảm biến.
- Tạo giao diện web dashboard trực quan, cập nhật dữ liệu realtime.
- Cho phép điều khiển thiết bị IoT từ xa qua Internet.
- Lưu trữ dữ liệu và lịch sử thao tác trên cơ sở dữ liệu MySQL.
- Tích hợp công nghệ MQTT, Node.js, Socket.IO và Chart.js trong cùng một hệ thống.

3. Phạm vi và đối tượng nghiên cứu

- **Đối tượng:** Hệ thống IoT phục vụ giám sát môi trường và điều khiển thiết bị.
- **Phạm vi**
  - Hệ thống thu thập dữ liệu từ cảm biến DHT11 (nhiệt độ, độ ẩm) và BH1750 (ánh sáng).
  - Thiết bị điều khiển gồm: quạt, đèn, điều hòa mô phỏng (dạng LED).
  - Giao diện web hiển thị dữ liệu và cho phép điều khiển.

4. Ý nghĩa thực tiễn

Đề tài giúp người thực hiện hiểu rõ:

- Quy trình thu thập dữ liệu cảm biến, xử lý và hiển thị trên web.
- Cách giao tiếp giữa **thiết bị IoT – Server – Giao diện người dùng.**
- Ứng dụng được kiến thức các môn học: **Mạng máy tính, Lập trình web, Cơ sở dữ liệu, IoT và Ứng dụng.**

CHƯƠNG II: THIẾT BỊ VÀ CÔNG NGHỆ SỬ DỤNG

1. Thiết bị phần cứng sử dụng

a. ESP8266 NodeMCU

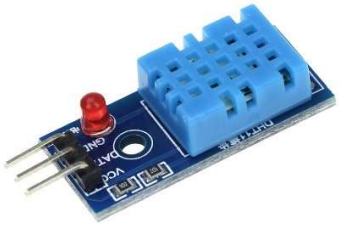
Hình 1:NodeMCU

ESP8266 là vi điều khiển tích hợp WiFi, được sử dụng làm trung tâm của hệ thống IoT.

Thông số kỹ thuật cơ bản:

- Vi xử lý: Tensilica L106 32-bit, tốc độ 80 MHz
- Bộ nhớ Flash: 4 MB
- Điện áp hoạt động: 3.3V
- Giao tiếp: WiFi, UART, SPI, I2C, PWM
- Số chân GPIO: 11
- Ngôn ngữ lập trình: Arduino C / C++

b. Cảm biến nhiệt độ, độ ẩm DHT11



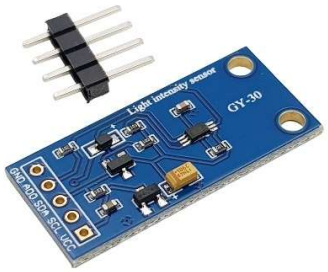
Hình 2: Cảm biến DHT11

Cảm biến đo nhiệt độ và độ ẩm sử dụng tín hiệu số, dễ lập trình và ổn định trong môi trường trong nhà.

Thông số kỹ thuật cơ bản:

- Dải đo nhiệt độ: 0 – 50°C (sai số ±2°C)
- Dải đo độ ẩm: 20 – 90% RH (sai số ±5%)
- Điện áp hoạt động: 3.3V – 5V
- Chu kỳ đo: 1 giây/lần
- Giao tiếp: Digital single wire

c. Cảm biến ánh sáng BH1750



Hình 3: Cảm biến BH1750

Cảm biến đo cường độ ánh sáng kỹ thuật số, giao tiếp I2C, độ chính xác cao.

Thông số kỹ thuật cơ bản:

- Dải đo: 1 – 65535 Lux
- Giao tiếp: I2C (SDA, SCL)
- Nguồn hoạt động: 3.3V – 5V
- Tốc độ đo: 16 ms
- Độ phân giải: 1 Lux

d. Đèn led mô phỏng

Hình 4: Đèn LED

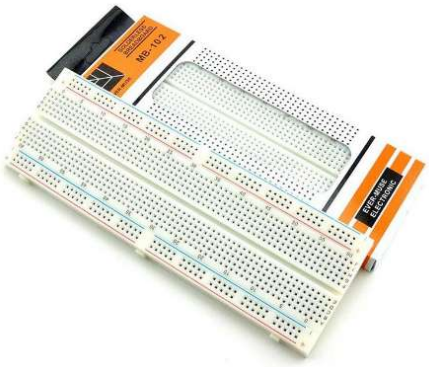
Các đèn LED được điều khiển trực tiếp từ NodeMCU thông qua các chân digital output, mô phỏng đèn và quạt trong mô hình nhà thông minh.

Thông số cơ bản:

- LED: điện áp hoạt động 3.3V, dòng 20 mA
- Quạt mini: điện áp 5V
- Điều khiển qua: GPIO D6, D7

e. Các linh kiện khác

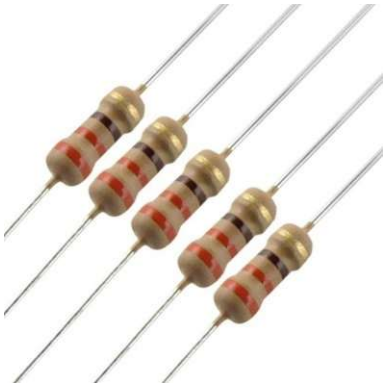
BreadBoard:



Hình 5: Breadboard

- **Loại:** Bảng mạch thử nghiệm (Prototype board)
- **Chức năng:** Kết nối và cố định linh kiện mà không cần hàn
- **Đặc điểm:**
  - Gồm 830 lỗ, có hai hàng nguồn và 63 hàng kết nối
  - Dễ dàng tháo lắp, phù hợp cho mô hình thử nghiệm nhỏ

Điện trở:



Hình 6: Điện trở

- **Loại:** Điện trở 220Ω và 10kΩ
- **Chức năng:** Giới hạn dòng điện qua LED, đảm bảo an toàn cho linh kiện; dùng làm điện trở kéo lên (pull-up) hoặc kéo xuống (pull-down) cho tín hiệu digital

- **Thông số chính:**
  - Trở kháng: 220Ω, 10kΩ
  - Công suất: 0.25W

Dây jumper(Dây nối):

Hình 7:Dây nối

- **Loại: Dây jumper đực-đực và đực-cái**
- **Chức năng: Kết nối các linh kiện với nhau trên breadboard**
- **Đặc điểm:**
  - Sử dụng nhiều màu để dễ phân biệt (VCC: đỏ, GND: đen, DATA: màu khác)
  - Chiều dài trung bình 10–20 cm, dễ thay thế và mở rộng

2. Mạch kết nối và điều khiển

Cấu trúc kết nối:

- **LED và quạt mô phỏng thiết bị điều khiển:**
  - Kết nối với các chân GPIO của ESP8266 thông qua điện trở giới hạn dòng 220Ω để tránh cháy LED.
  - Chân dương (Anode) của LED nối với chân GPIO (D5, D6 và D7).
  - Chân âm (Cathode) nối với GND.
- **Cảm biến DHT11 (đo nhiệt độ và độ ẩm):**
  - Chân VCC nối 3.3V của ESP8266.
  - Chân DATA nối GPIO D4.
  - Chân GND nối GND.
- **Cảm biến BH1750 (đo cường độ ánh sáng):**
  - Chân VCC nối 3.3V của ESP8266.
  - Chân GND nối GND.
  - Chân SDA nối GPIO D2.
  - Chân SCL nối GPIO D1.
- **Nguồn và kết nối mạch:**

- Toàn bộ mạch được lắp trên breadboard 830 lỗ.
- Dây jumper đực–cái và đực–đực được sử dụng để kết nối các chân tín hiệu.
- Nguồn cấp 5V từ cổng USB máy tính cho NodeMCU.

Nguyên lý hoạt động:

- Cảm biến DHT11 và BH1750 liên tục thu thập dữ liệu môi trường (nhiệt độ, độ ẩm, ánh sáng).
- Dữ liệu được ESP8266 đọc và gửi đến **MQTT Broker** thông qua kết nối WiFi.
- Server Node.js nhận dữ liệu qua MQTT, lưu vào **cơ sở dữ liệu MySQL** và gửi realtime lên **web dashboard** bằng Socket.IO.
- Khi người dùng bật/tắt thiết bị trên web dashboard, server gửi lệnh điều khiển ngược về MQTT Broker.
- ESP8266 nhận lệnh, xuất tín hiệu **HIGH (3.3V)** tại chân điều khiển để bật LED hoặc quạt, và **LOW (0V)** để tắt.
- Các điện trở 220Ω mắc nối tiếp với LED giúp hạn chế dòng điện, tránh hư hại linh kiện.

3. Công nghệ phần mềm được sử dụng

a. HTML và CSS

HTML (HyperText Markup Language) và CSS (Cascading Style Sheets) là hai công nghệ nền tảng trong phát triển giao diện web. Trong hệ thống này, **HTML** được sử dụng để xây dựng cấu trúc của trang web dashboard, bao gồm các thành phần hiển thị dữ liệu cảm biến, biểu đồ và nút điều khiển thiết bị.

CSS được dùng để định dạng bố cục, màu sắc, font chữ và hiệu ứng, giúp giao diện trở nên trực quan, dễ nhìn và phù hợp trên nhiều kích thước màn hình khác nhau.

Sự kết hợp giữa HTML và CSS giúp tạo ra một giao diện **đơn giản, hiện đại và thân thiện**, góp phần nâng cao trải nghiệm người dùng khi giám sát và điều khiển thiết bị IoT.

b. Node.js và Express.js

Phần mềm máy chủ được xây dựng bằng **Node.js**, một môi trường chạy JavaScript phía server sử dụng mô hình **non-blocking I/O**. Node.js giúp hệ thống hoạt động nhanh, xử lý đồng thời nhiều yêu cầu kết nối mà không gây nghẽn.

Trên nền Node.js, framework **Express.js** được sử dụng để phát triển REST API, định tuyến các đường dẫn (route) và xử lý các yêu cầu từ phía client. Express giúp việc xây dựng server trở nên đơn giản, có cấu trúc rõ ràng và dễ mở rộng.

c. MQTT

Giao thức **MQTT (Message Queuing Telemetry Transport)** là cầu nối truyền thông giữa phần cứng và phần mềm. Đây là giao thức truyền dữ liệu nhẹ, hoạt động theo mô hình "Publish – Subscribe" gồm ba thành phần chính: Publisher, Broker và Subscriber.

Trong hệ thống này, ESP8266 đóng vai trò Publisher (gửi dữ liệu cảm biến), Mosquitto là Broker (trung gian truyền tải dữ liệu), và Node.js server là Subscriber (nhận dữ liệu và xử lý).

MQTT được lựa chọn vì ưu điểm gọn nhẹ, tốc độ truyền nhanh, tiêu tốn ít băng thông và hoạt động tốt trong điều kiện mạng không ổn định – rất phù hợp với các thiết bị IoT.

d. Socket.IO

Để hiển thị dữ liệu thời gian thực trên dashboard web, hệ thống sử dụng **Socket.IO** – thư viện cho phép truyền dữ liệu hai chiều giữa server và client qua WebSocket.

Khi dữ liệu cảm biến thay đổi, server tự động gửi dữ liệu mới đến giao diện web mà không cần tải lại trang. Điều này giúp người dùng quan sát sự thay đổi của nhiệt độ, độ ẩm, ánh sáng một cách tức thì.

e. MySQL và SequelizeORM

Dữ liệu thu thập từ các cảm biến được lưu trữ trong **MySQL**, một hệ quản trị cơ sở dữ liệu quan hệ phổ biến. Cấu trúc dữ liệu được thiết kế gồm các bảng lưu thông tin cảm biến, thiết bị và lịch sử thao tác.

Để làm việc với MySQL một cách thuận tiện hơn, dự án sử dụng **Sequelize**, một thư viện ORM cho Node.js. Sequelize giúp ánh xạ các bảng dữ liệu thành các đối tượng trong mã nguồn, giúp việc truy vấn, thêm hoặc sửa dữ liệu trở nên dễ dàng và an toàn hơn.

f. Chart.js

Phần giao diện dashboard hiển thị biểu đồ sử dụng thư viện **Chart.js**, cho phép vẽ biểu đồ đường, cột hoặc tròn bằng HTML5 Canvas.

Trong hệ thống này, Chart.js được dùng để hiển thị biểu đồ biến thiên của các giá trị cảm biến (nhiệt độ, độ ẩm, ánh sáng) theo thời gian thực. Thư viện hỗ trợ cập nhật dữ liệu động, giúp người dùng dễ dàng theo dõi sự thay đổi của các thông số môi trường.

g. Các công cụ hỗ trợ phát triển

- Visual Studio Code (lập trình Node.js)
- Arduino IDE (nạp code cho ESP8266)
- Postman (kiểm thử API)
- XAMPP (quản lý cơ sở dữ liệu MySQL)

CHƯƠNG III: PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

1. Phân tích yêu cầu hệ thống

a. Yêu cầu chúc năng

Hệ thống IoT Dashboard giám sát và điều khiển thiết bị cần đảm bảo các chức năng chính sau:

- **Giám sát dữ liệu cảm biến**:

Hệ thống thu thập và hiển thị các giá trị **nhiệt độ, độ ẩm, ánh sáng** từ các cảm biến DHT11 và BH1750 theo thời gian thực.

- **Cập nhật dữ liệu thời gian thực (Realtime)**:

Khi cảm biến gửi dữ liệu mới, giao diện web phải tự động cập nhật mà không cần tải lại trang.

- **Điều khiển thiết bị IoT từ xa**:

Người dùng có thể bật/tắt **đèn LED, quạt mini hoặc điều hòa mô phỏng** trực tiếp trên dashboard, và trạng thái thiết bị được đồng bộ lại trên giao diện.

- **Lưu trữ lịch sử dữ liệu**:

Mỗi lần cảm biến gửi dữ liệu hoặc người dùng thực hiện thao tác điều khiển, hệ thống phải ghi nhận vào **cơ sở dữ liệu MySQL** để phục vụ việc thống kê, phân tích sau này.

- **Hiển thị biểu đồ thống kê**:

Các thông số nhiệt độ, độ ẩm và ánh sáng được hiển thị dưới dạng biểu đồ theo thời gian, giúp người dùng dễ dàng quan sát và đánh giá xu hướng.

- **Quản lý kết nối MQTT**:

Server Node.js cần đảm bảo kết nối ổn định với MQTT Broker, xử lý khi mất kết nối hoặc khi thiết bị ngoại tuyến.

b. Yêu cầu phi chức năng

- **Hiệu năng**:

Dữ liệu cảm biến được cập nhật liên tục mỗi vài giây mà không gây trễ hoặc giật lag trên giao diện.

- **Bảo mật**:

Các API REST cần có cơ chế xác thực token, giới hạn truy cập trái phép.

- **Khả năng phục hồi**:

Hệ thống cần đảm bảo hoạt động ổn định khi có lỗi kết nối hoặc ngắt tạm thời, và tự động kết nối lại khi mạng ổn định.

- **Tính thân thiện**:

Giao diện web thiết kế trực quan, hiển thị dữ liệu rõ ràng, dễ sử dụng với người không chuyên kỹ thuật.

2. Kiến trúc hệ thống

Hệ thống được thiết kế theo **mô hình ba lớp (Three-tier architecture)**, gồm:

Tầng thiết bị IoT (Device Layer)

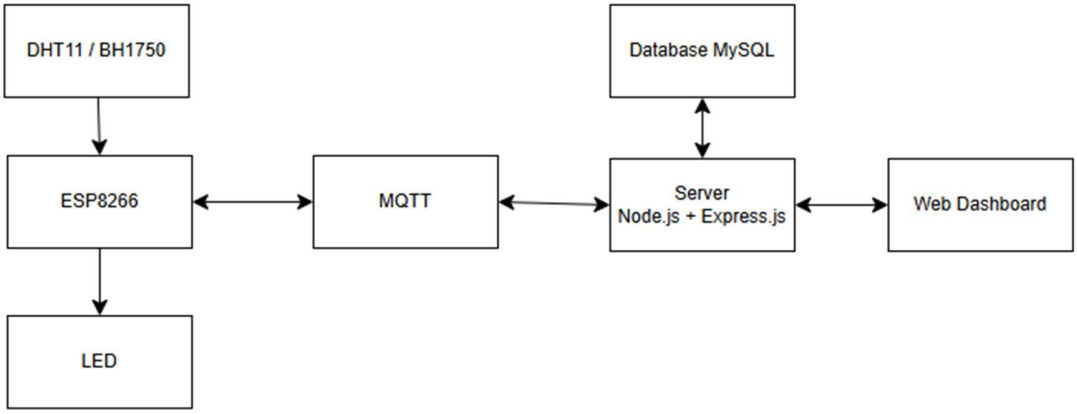
- Bao gồm các cảm biến DHT11, BH1750 và thiết bị điều khiển như LED, quạt.
- ESP8266 NodeMCU đóng vai trò trung tâm, đọc dữ liệu từ cảm biến và truyền qua **MQTT Broker** đến server.
- Đồng thời nhận lệnh điều khiển từ server để thay đổi trạng thái thiết bị.

Tầng máy chủ (Server Layer)

- Sử dụng **Node.js + Express** làm nền tảng chính.
- Nhận dữ liệu từ MQTT Broker, xử lý và lưu vào **MySQL Database** thông qua **Sequelize ORM**.
- Gửi dữ liệu realtime đến web thông qua **Socket.IO**.
- Cung cấp các API REST cho phép truy xuất dữ liệu lịch sử, trạng thái thiết bị.

Tầng giao diện người dùng (Web Layer)

- Giao diện web hiển thị thông tin cảm biến, biểu đồ thống kê và các nút điều khiển thiết bị.
- Dữ liệu được cập nhật tức thời nhờ Socket.IO.
- Người dùng có thể thao tác trực tiếp với thiết bị chỉ bằng một cú click chuột.



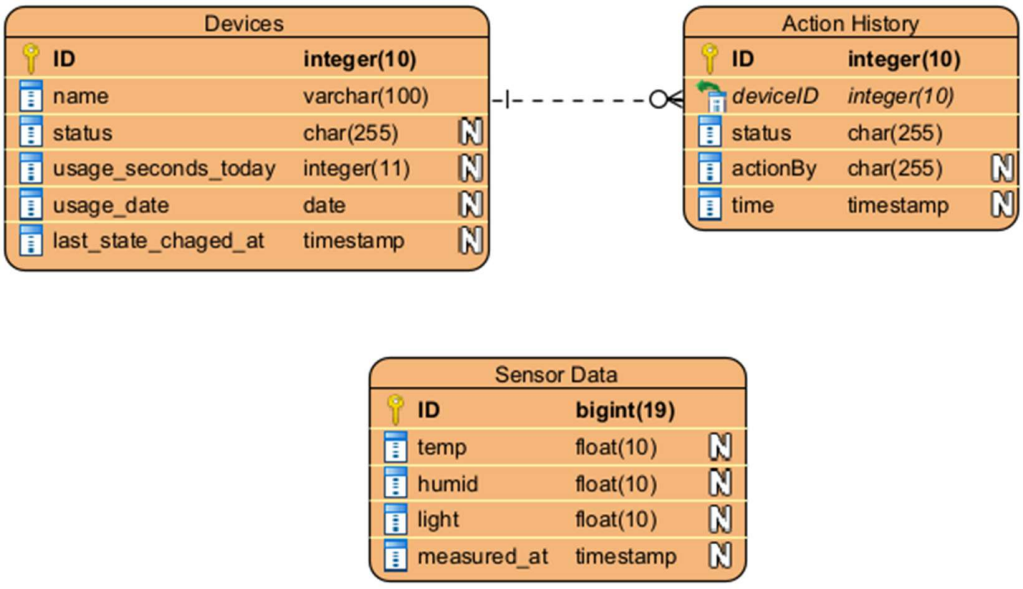
Hình 8: Kiến trúc hệ thống

Quy trình hoạt động

- ESP8266 đọc giá trị cảm biến và gửi dữ liệu qua MQTT topic:
- Server Node.js nhận dữ liệu, lưu vào MySQL và phát sự kiện realtime lên web.
- Dashboard cập nhật biểu đồ và thông số hiển thị.
- Khi người dùng nhấn nút bật/tắt, server gửi lệnh MQTT đến ESP8266 qua topic:
- ESP8266 nhận lệnh, thay đổi trạng thái thiết bị (LED/Quạt) và gửi phản hồi trạng thái lại server.

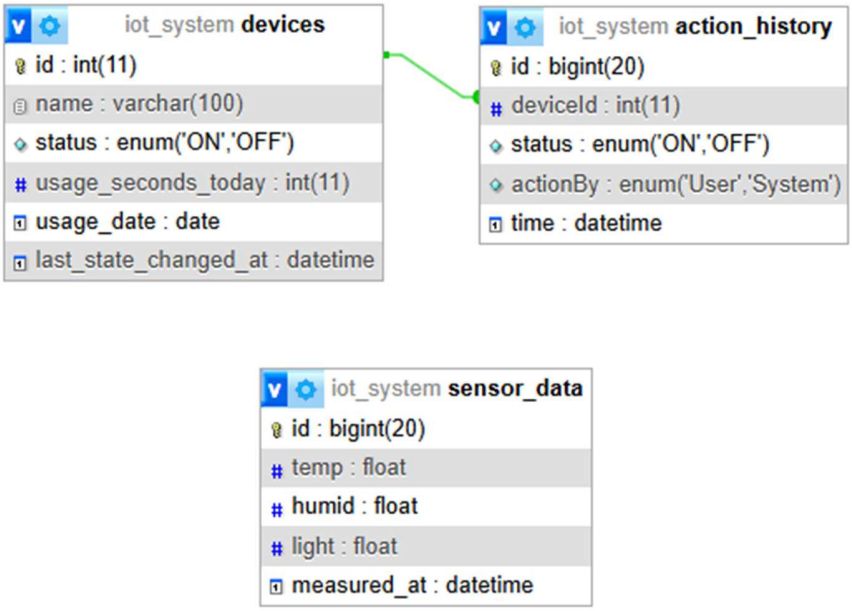
3. Thiết kế cơ sở dữ liệu

a. Lược đồ ERD



Hình 9: Lược đồ ERD

b. Cơ sở dữ liệu



Hình 10: Sơ đồ CSDL

Bảng sensor_data

Mục đích: Lưu dữ liệu thu thập được từ các cảm biến IoT.

- **id** (bigint(20)): Khóa chính, dùng để định danh duy nhất cho mỗi bản ghi dữ liệu cảm biến.
- **temp** (float): Lưu giá trị nhiệt độ (°C) đo được từ cảm biến DHT11.
- **humidity** (float): Lưu giá trị độ ẩm (%) đo được từ cảm biến DHT11.
- **light** (float): Lưu cường độ ánh sáng (lux) đo được từ cảm biến BH1750.
- **time** (datetime): Thời điểm ghi nhận dữ liệu từ cảm biến.

Bảng devices

Mục đích: Lưu danh sách, trạng thái, thời gian hoạt động của các thiết bị.

- **id** (integer(11)) Khóa chính, định danh duy nhất cho mỗi thiết bị.
- **name** (varchar(100)): Tên thiết bị được điều khiển, ví dụ "LED" hoặc "FAN".
- **status** (enum('ON', 'OFF')): Trạng thái hiện tại của thiết bị, ví dụ "ON" hoặc "OFF".
- **usage_second_today**(integer(11)): Thời gian sử dụng của thiết bị trong ngày (giây).
- **usage_date** (date): Ngày tính thời gian sử dụng.
- **last_state_changed_at** (datetime): Thời điểm hành động được thực hiện.

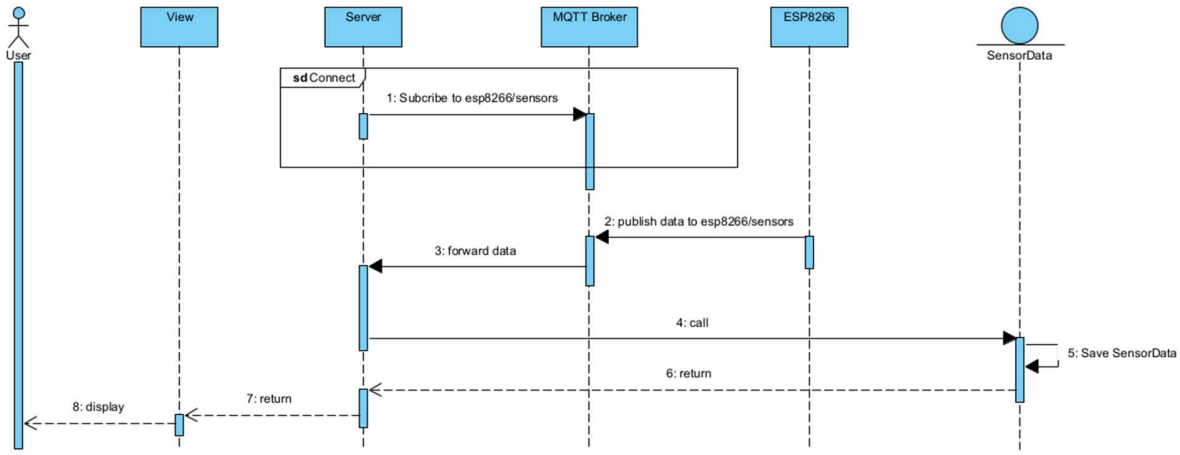
Bảng action_history

Mục đích: Lưu các hành động điều khiển thiết bị (ví dụ: bật/tắt đèn, quạt...).

- **id** (bigint(20)): Khóa chính, định danh duy nhất cho mỗi hành động điều khiển.
- **deviceID** (integer(11)):ID thiết bị được điều khiển, liên kết khóa ngoại với bảng Devices.
- **status** (enum('ON', 'OFF')): Hành động được thực hiện trên thiết bị, ví dụ "ON" hoặc "OFF".
- **actionBy**(enum('User', 'System')):): Người thực hiện hành động điều khiển, ví dụ "User" hoặc "System".
- **time** (datetime): Thời điểm hành động được thực hiện.

4. Biểu đồ tuần tự(Sequence Diagram):

a. Sequence hiển thị thông số cảm biến

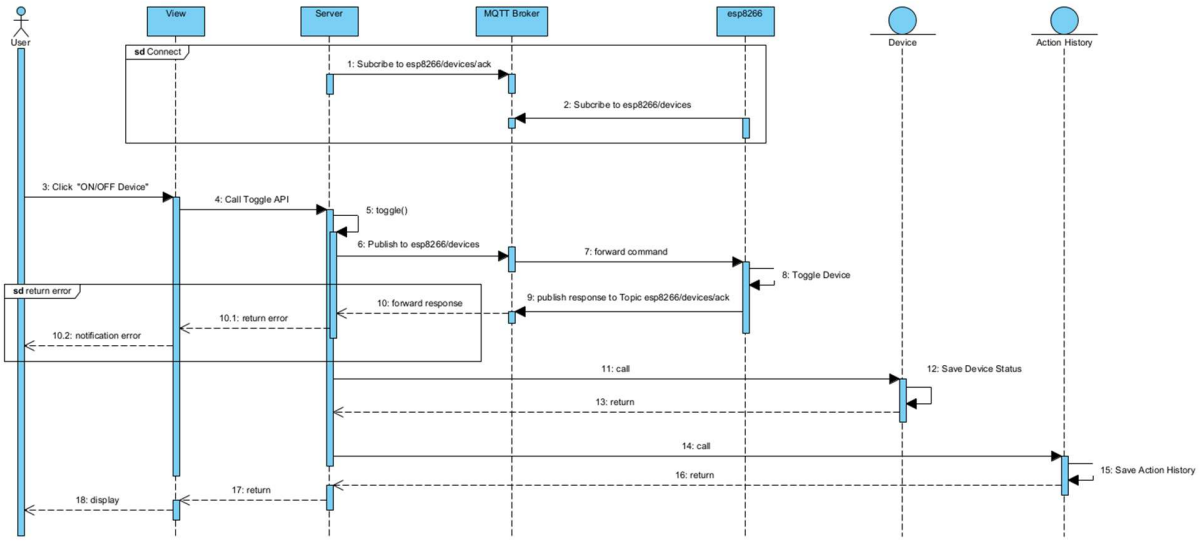


Hình 11: Sequence hiển thị thông số cảm biến

- **Scenario chuẩn:**
  1. Lớp Server subcrice vào topic "esp8266/sensors" của Lớp MQTT Broker
  2. Cảm biến đo các thông số, ESP8266 publish dữ liệu đo được vào topic "esp8266/sensors" lớp MQTT Broker.
  3. Lớp MQTT Broker chuyển tiếp dữ liệu vào Server.
  4. Lớp Server gọi lớp SensorData để lưu dữ liệu cảm biến.
  5. Lớp SensorData thực hiện lưu dữ liệu cảm biến.
  6. Lớp SensorData trả kết quả về Server.
  7. Lớp Server trả kết quả cho view
  8. Lớp View hiển thị cho người dùng

b. Sequence điều khiển thiết bị

- **Scenario chuẩn:**



Hình 12: Sequence Điều khiển thiết bị

1. Lớp Server subcrice vào topic ‘esp8266/devices/ack’ của Lớp MQTT Broker để nhận phản hồi của các yêu cầu điều khiển.
2. Lớp ESP8266 subcribes vào topic ‘esp8266/devices’ của Lớp MQTT Broker để nhận yêu cầu điều khiển thiết bị?
3. Người dùng click vào nút ON/OFF của thiết bị ở Lớp View
4. Lớp View call API điều khiển tới lớp Server.
5. Lớp Server thực hiện phương thức toggle().
6. Phương thức toggle() publish yêu cầu điều khiển vào topic ‘esp8266/devices’ của lớp MQTT Broker.
7. Lớp MQTT Broker chuyển tiếp lệnh đến ESP8266.
8. Lớp ESP8266 thực hiện yêu cầu điều khiển thiết bị.
9. Lớp ESP8266 publish phản hồi vào topic ‘esp8266/devices/ack’ lớp MQTT Broker
10. Lớp MQTT Broker chuyển tiếp phản hồi từ topic về phương thức toggle()
11. Lớp Server gọi đến lớp Device thực hiện phương thức để lưu trạng thái thiết bị.
12. Lớp Device thực hiện lưu trạng thái thiết bị
13. Lớp Device trả kết quả về lớp Java-Client
14. Lớp Server gọi đến lớp Action History thực hiện phương thức để lưu lịch sử điều khiển.
15. Lớp Action History thực hiện lưu lịch sử.

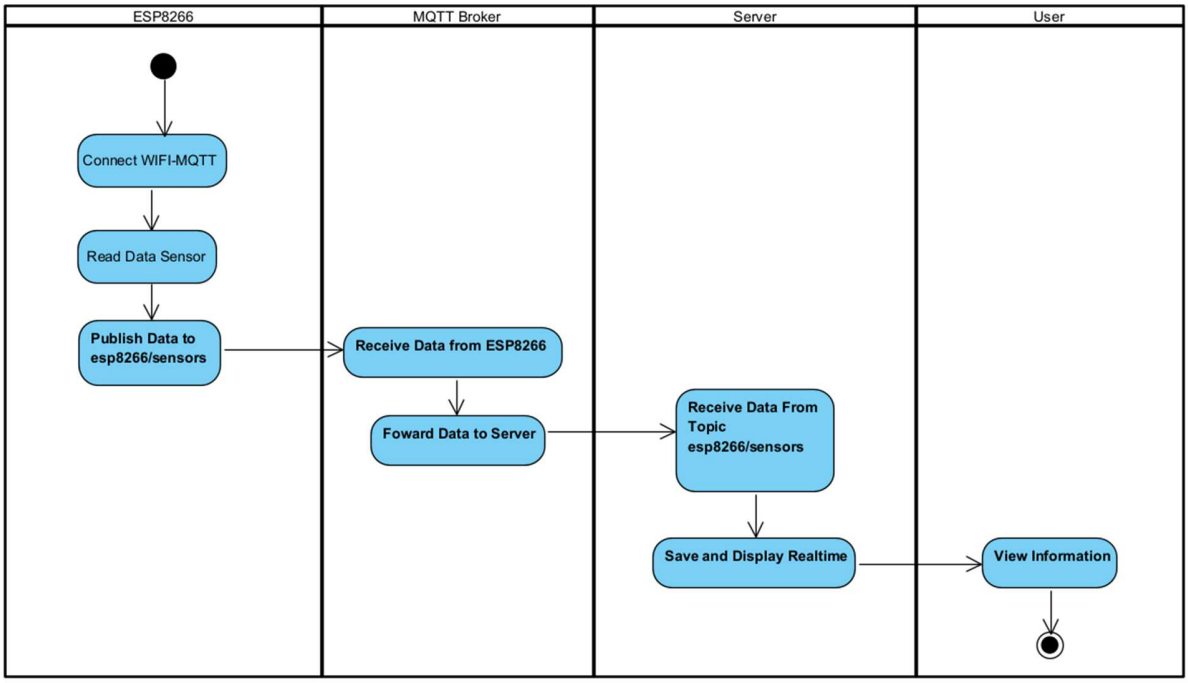
16. Lớp Action History trả kết quả về lớp Server.
17. Server trả kết quả về cho View.
18. Lớp View hiển thị kết quả thực hiện yêu cầu về cho người dùng

- **Scenario ngoại lệ:**

10. Phương thức toggle() nhận về lỗi không thực hiện được yêu cầu.
10.1 Lớp Server trả thông báo lỗi về View.
10.2 Lớp View hiển thị thông báo cho người dùng.

5. Activity Diagram

a. Activity hiển thị thông số cảm biến



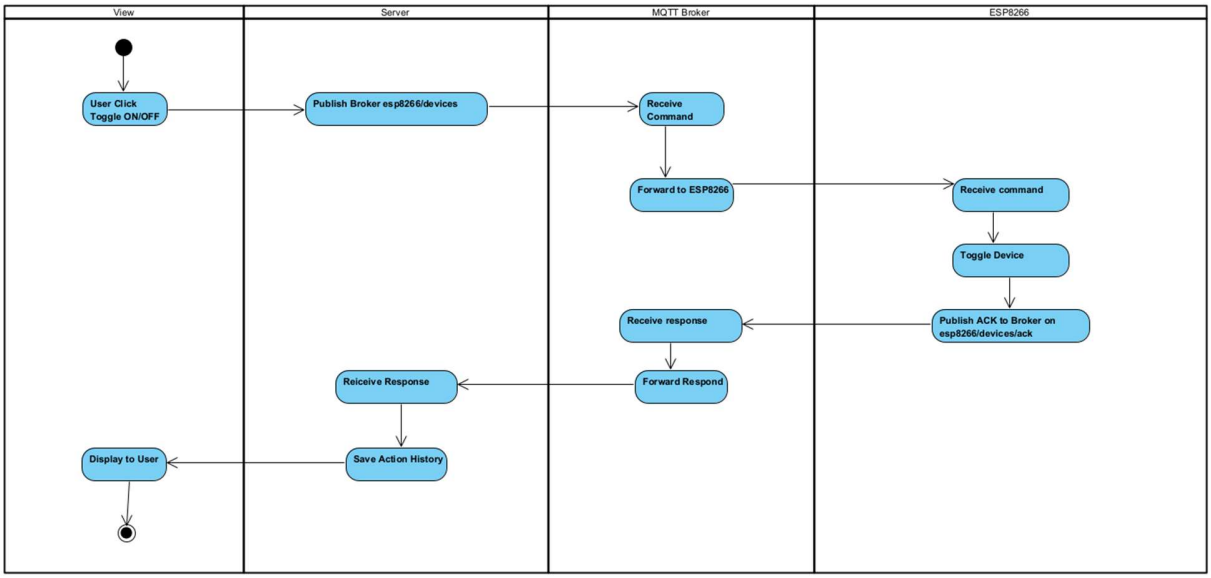
Hình 13: Activity hiển thị thông số cảm biến

1) **ESP8266 kết nối WiFi – MQTT**
   - Vi điều khiển ESP8266 khởi tạo kết nối đến mạng WiFi và kết nối với MQTT Broker để chuẩn bị truyền dữ liệu.
2) **Đọc dữ liệu từ các cảm biến**
   - ESP8266 đọc giá trị từ các cảm biến (nhiệt độ, độ ẩm, ánh sáng).
3) **Publish dữ liệu lên Broker theo topic**
   - ESP8266 đóng gói dữ liệu thành JSON (bao gồm nhiệt độ, ánh sáng, độ ẩm).
   - VD:{“temp”:29, “humid”:45, “light”:24}

- Gửi (publish) dữ liệu đến MQTT Broker với topic định nghĩa trước (ví dụ: esp8266/sensors).

4) **MQTT Broker nhận dữ liệu từ ESP8266**
   - MQTT Broker tiếp nhận dữ liệu do ESP8266 gửi lên.
5) **MQTT Broker chuyển tiếp dữ liệu đến Client (Server)**
   - Broker phân phối dữ liệu đã nhận cho các client đã subscribe topic esp8266/sensors.
6) **Server nhận dữ liệu từ Broker**
   - Server lắng nghe topic esp8266/sensors, nhận dữ liệu cảm biến từ MQTT Broker.
7) **Lưu và hiển thị dữ liệu thời gian thực**
   - Server lưu dữ liệu vào cơ sở dữ liệu (bảng DataSensor).
   - Đồng thời cập nhật và hiển thị dữ liệu trên giao diện web/app theo thời gian thực.
8) **Người dùng quan sát thông tin trên giao diện**
   - Người dùng cuối có thể theo dõi giá trị cảm biến (nhiệt độ, độ ẩm, ánh sáng) trực tiếp trên giao diện hệ thống.

b. Activity điều khiển thiết bị



Hình 14:Activity Điều khiển thiết bị

1) **Người dùng click bật/tắt thiết bị trên giao diện (View)**
   - Người dùng thực hiện thao tác điều khiển, ví dụ bật/tắt quạt hoặc đèn.

2) **Server publish lệnh điều khiển lên Broker**
- Server gửi một message đến MQTT Broker với topic esp8266/devices.
- Message có định dạng JSON, ví dụ:
- {"id":1,"status":"OFF","actionBy":"User"}

3) **MQTT Broker nhận data và chuyển tiếp đến ESP8266**
- Broker tiếp nhận dữ liệu từ Server.
- Chuyển tiếp dữ liệu này đến ESP8266 (đã subscribe topic esp8266/devices).

4) **ESP8266 nhận lệnh và thực thi**
- ESP8266 đọc message nhận được.
- Tiến hành thực hiện lệnh điều khiển thiết bị (bật/tắt).

5) **Xác định kết quả thực thi**
- ESP8266 chuyển đổi trạng thái thiết bị và publish một message ACK về Broker (esp8266/devices/ack).
- Ví dụ: {"id":1,"status":"OFF","actionBy":"User"}

6) **MQTT Broker nhận phản hồi và chuyển tiếp về Server**
- Broker nhận message phản hồi từ ESP8266.
- Chuyển tiếp phản hồi này đến Server (client subscriber).

7) **Server xử lý phản hồi**
- Server lưu bản ghi vào bảng ActionHistory.

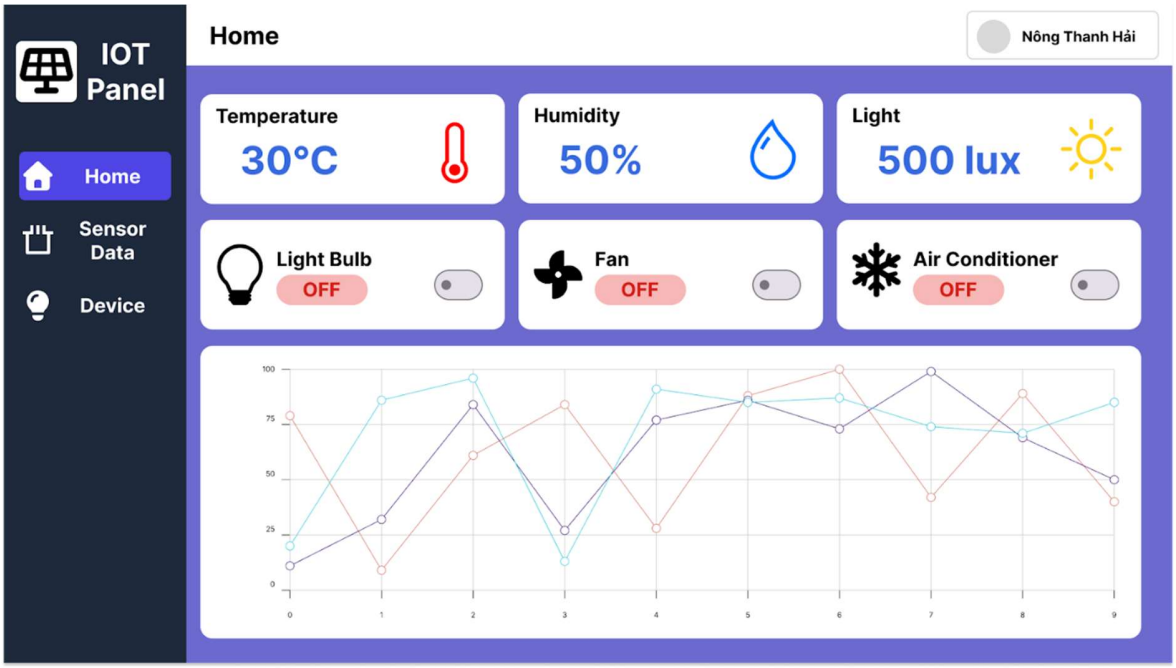
8) **Server hiển thị kết quả đến người dùng (Client)**
- Gửi phản hồi cuối cùng về giao diện.
- Người dùng quan sát trạng thái thiết bị hoặc thông báo lỗi ngay trên ứng dụng.

6. Thiết kế giao diện người dùng (Figma)

a. Trang Dashboard

Hiển thị các thông số **Nhiệt độ – Độ ẩm – Ánh sáng** dạng số và biểu đồ thời gian thực.

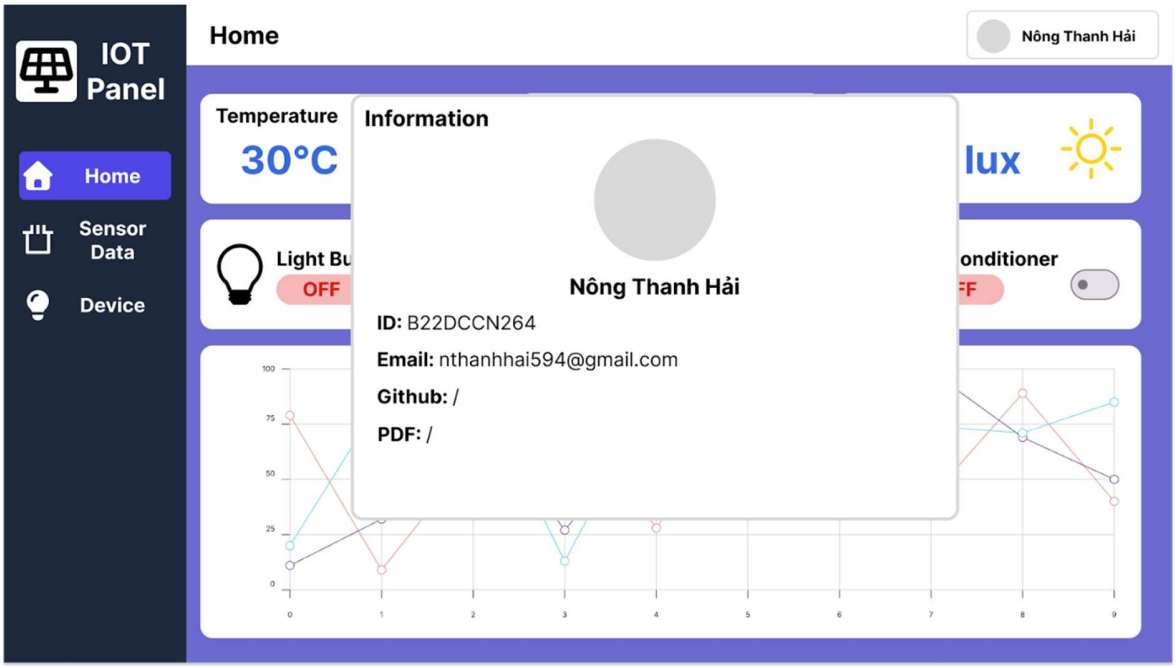
Mỗi thiết bị có một thẻ riêng với biểu tượng và trạng thái **ON/OFF**. Khi người dùng nhấn vào biểu tượng, lệnh được gửi ngay đến server.



Hình 15: Thiết kế trang Dashboard

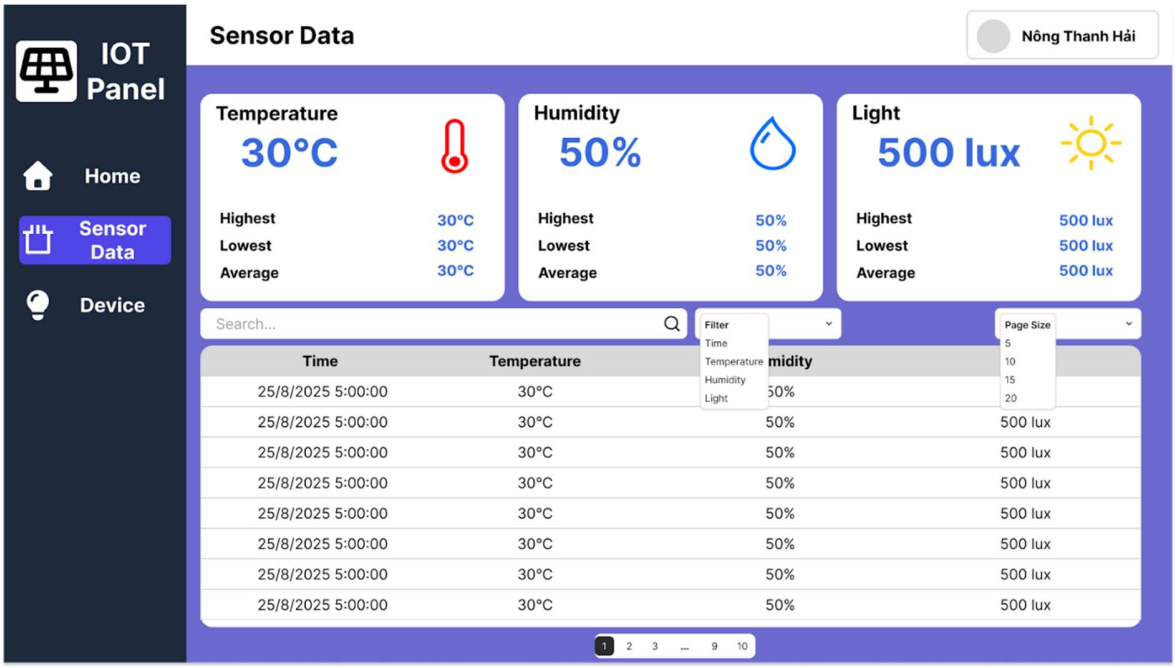
b. Trang Profile

Hiển thị thông tin cá nhân: bao gồm ảnh, MSV, email,…



Hình 16: Thiết kế trang Profile

c. Trang SensorData



Hình 17: Thiết kế trang SensorData

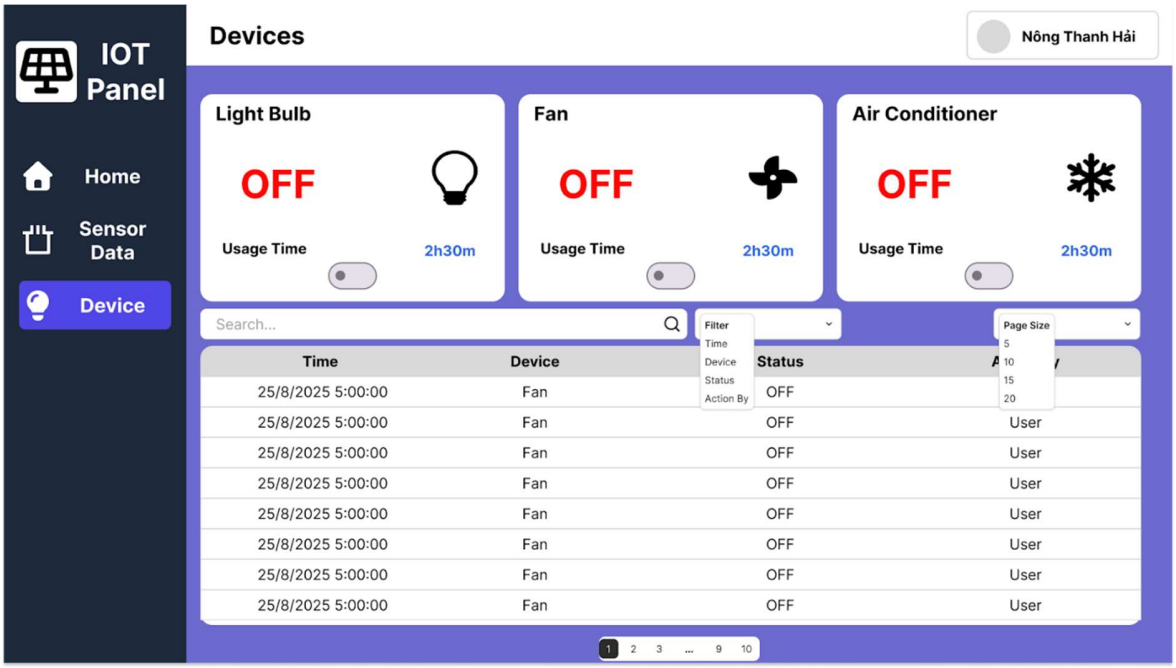
Hiển thị các thông số Nhiệt độ – Độ ẩm – Ánh sáng dạng số và biểu đồ thời gian thực.

Danh sách các dữ liệu cảm biến theo thời gian.

d. Trang Devices

Mỗi thiết bị có một thẻ (card) riêng với biểu tượng và trạng thái ON/OFF. Khi người dùng nhấn vào biểu tượng, lệnh được gửi ngay đến server.

Danh sách các thao tác bật/tắt thiết bị gần nhất cùng thời gian và người thực hiện.



Hình 18: Thiết kế Trang Devices

CHƯƠNG IV: XÂY DỰNG VÀ TRIỂN KHAI HỆ THỐNG

1. Tổng quan quá trình xây dựng

Dựa trên phân tích và thiết kế đã trình bày ở chương 3, hệ thống được xây dựng hoàn chỉnh bao gồm ba thành phần chính:

a. **Phần cứng IoT (Thiết bị và cảm biến)**
- ESP8266 NodeMCU kết nối cảm biến DHT11 (đo nhiệt độ, độ ẩm), cảm biến BH1750 (đo ánh sáng), và các thiết bị điều khiển (LED, quạt mini).
- Dữ liệu được gửi lên server thông qua giao thức MQTT.

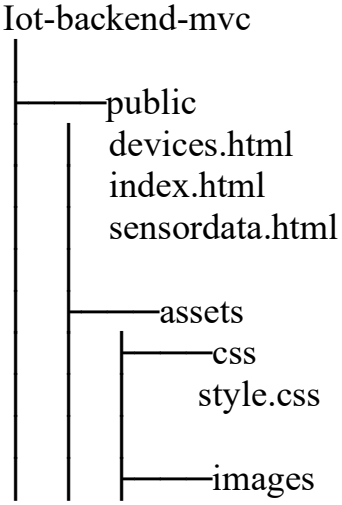
b. Phần mềm máy chủ (Server)
- Xây dựng bằng **Node.js + Express**, nhận dữ liệu từ thiết bị, lưu vào **MySQL** và phát sự kiện thời gian thực lên web bằng **Socket.IO**.

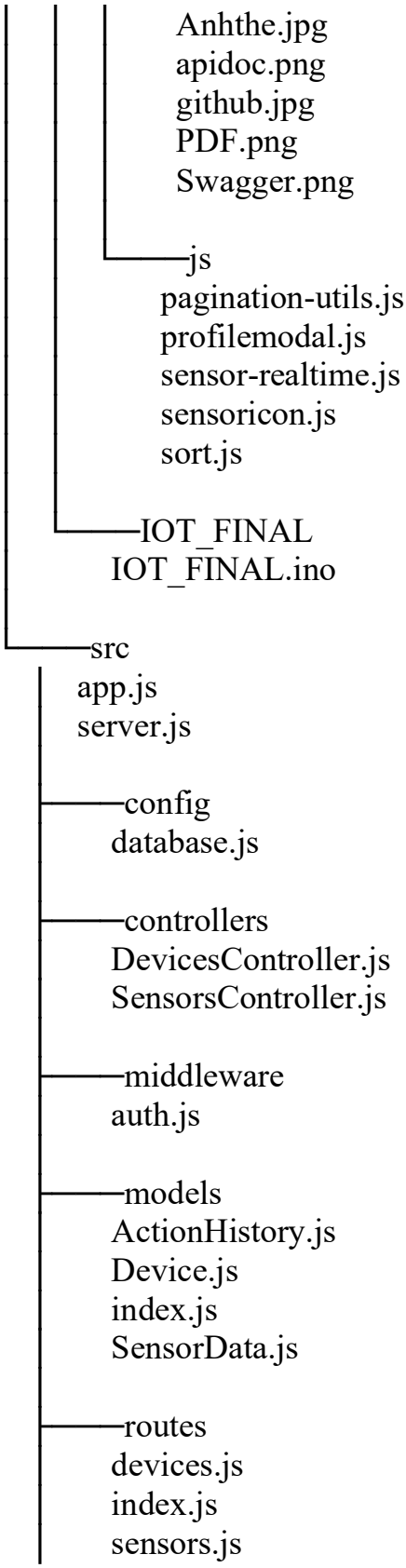
c. Giao diện người dùng (Frontend)
- Xây dựng bằng **HTML, CSS, JavaScript, Chart.js** để hiển thị dữ liệu cảm biến và cho phép điều khiển thiết bị trực quan.

Quá trình triển khai gồm các bước:

1) Thiết lập môi trường phát triển (Node.js, MySQL, MQTT Broker).
2) Viết và nạp chương trình cho ESP8266.
3) Xây dựng backend (API, MQTT subscriber, socket server).
4) Phát triển giao diện web hiển thị dữ liệu realtime.
5) Kiểm thử, hiệu chỉnh và triển khai trên mạng LAN.

2. Cấu trúc thư mục dự án




Anhthe.jpg
apidoc.png
github.jpg
PDF.png
Swagger.png
js
pagination-utils.js
profilemodal.js
sensor-realtime.js
sensoricon.js
sort.js
IOT_FINAL
IOT_FINAL.ino
src
app.js
server.js
config
database.js
controllers
DevicesController.js
SensorsController.js
middleware
auth.js
models
ActionHistory.js
Device.js
index.js
SensorData.js
routes
devices.js
index.js
sensors.js

```
├──services
│     mqttService.js
│
└──utils
```

3. Phần cứng và chương trình ESP8266

Hệ thống sử dụng **ESP8266 NodeMCU** làm trung tâm, kết nối cảm biến **DHT11** (đo nhiệt độ, độ ẩm), **BH1750** (đo ánh sáng) và ba **thiết bị điều khiển mô phỏng** bằng LED.

ESP8266 truyền dữ liệu cảm biến và nhận lệnh điều khiển thông qua **MQTT Broker (Mosquitto)**.

a. Cấu hình kết nối

```
// ==== WiFi & MQTT Config ====
const char* ssid       = "ThanhHai";
const char* password   = "Thanhhai59@";
const char* mqtt_server= "192.168.0.213";
const char* mqtt_user  = "ThanhHai";
const char* mqtt_pass  = "thanhhai2004";

WiFiClient espClient;
PubSubClient client(espClient);

// ==== DHT11 Config ====
#define DHTPIN 2      // GPIO2 (D4)
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

// ==== BH1750 Config ====
BH1750 lightMeter;

// ==== LED/Relay Pins ====
#define LED1 14   // GPIO14 (D5)
#define LED2 12   // GPIO12 (D6)
#define LED3 13   // GPIO13 (D7)
```

Hình 19: Cấu hình kết nối ESP8266

b. Xử lý lệnh điều khiển (MQTT Callback)

```
// ===== MQTT callback =====
void callback(char* topic, byte* payload, unsigned int length) {
  String message;
  for (unsigned int i = 0; i < length; i++) message += (char)payload[i];
  message.trim();

  Serial.print("MQTT IN ["); Serial.print(topic); Serial.print("] ");
  Serial.println(message);

  if (String(topic) == devicesTopic) {
    StaticJsonDocument<128> doc;
    DeserializationError error = deserializeJson(doc, message);
    if (error) {
      Serial.print("JSON parse failed: ");
      Serial.println(error.c_str());
      return;
    }

    int id = doc["id"] | 0;
    const char* status = doc["status"] | "";
    const char* actionBy = doc["actionBy"] | "System";
    bool on = (String(status) == "ON");

    switch (id) {
      case 1:
        digitalWrite(LED1, on ? HIGH : LOW);
        Serial.println(on ? "LED1 đã bật" : "LED1 đã tắt");
        publishAck(1, LED1, actionBy);
        break;
      case 2:
        digitalWrite(LED2, on ? HIGH : LOW);
        Serial.println(on ? "LED2 đã bật" : "LED2 đã tắt");
        publishAck(2, LED2, actionBy);
        break;
      case 3:
        digitalWrite(LED3, on ? HIGH : LOW);
        Serial.println(on ? "LED3 đã bật" : "LED3 đã tắt");
        publishAck(3, LED3, actionBy);
        break;
      default:
        Serial.println("ID không hợp lệ (chỉ 1-3)");
        break;
    }
  }
}
```

Hình 20: Hàm xử lý lệnh điều khiển

c. Đọc Sensor và gửi dữ liệu(Loop)

```
// ===== Loop =====
void loop() {
  if (!client.connected()) reconnect();
  client.loop();

  // Đọc và publish cảm biến định kỳ
  static unsigned long lastSend = 0;
  const unsigned long interval = 2000;
  unsigned long now = millis();
  if (now - lastSend >= interval) {
    lastSend = now;

    float h   = dht.readHumidity();
    float t   = dht.readTemperature();
    float lux = lightMeter.readLightLevel();

    if (isnan(h) || isnan(t) || isnan(lux)) {
      Serial.println("Sensor read error");
      return;
    }

    StaticJsonDocument<128> doc;
    doc["temp"]  = t;
    doc["humid"] = h;
    doc["light"] = lux;

    char payload[128];
    serializeJson(doc, payload);
    bool ok = client.publish(sensorsTopic, payload, true);
    Serial.print("PUB sensors: "); Serial.print(payload);
    Serial.println(ok ? " [OK]" : " [FAIL]");
  }
}
```

Hình 21: Hàm đọc Sensor và gửi dữ liệu

4. Cấu hình cơ sở dữ liệu MySQL

```
CREATE TABLE IF NOT EXISTS sensor_data (
  id BIGINT AUTO_INCREMENT PRIMARY KEY,
  temp  FLOAT NOT NULL,
  humid FLOAT NOT NULL,
  light FLOAT NOT NULL,
  measured_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
```

```
  INDEX idx_measured_at (measured_at)
);

CREATE TABLE IF NOT EXISTS devices (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  status ENUM('ON','OFF') NOT NULL DEFAULT 'OFF',
  usage_seconds_today INT NOT NULL DEFAULT 0,
  usage_date DATE NOT NULL DEFAULT (CURRENT_DATE),
  last_state_changed_at DATETIME NULL
);

CREATE TABLE IF NOT EXISTS action_history (
  id BIGINT AUTO_INCREMENT PRIMARY KEY,
  deviceId INT NOT NULL,
  status ENUM('ON','OFF') NOT NULL,
  actionBy ENUM('User','System') NOT NULL DEFAULT 'System',
  time DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
   CONSTRAINT fk_action_device FOREIGN KEY (deviceId) REFERENCES
devices(id),
  INDEX idx_device_time (deviceId, time)
);
```

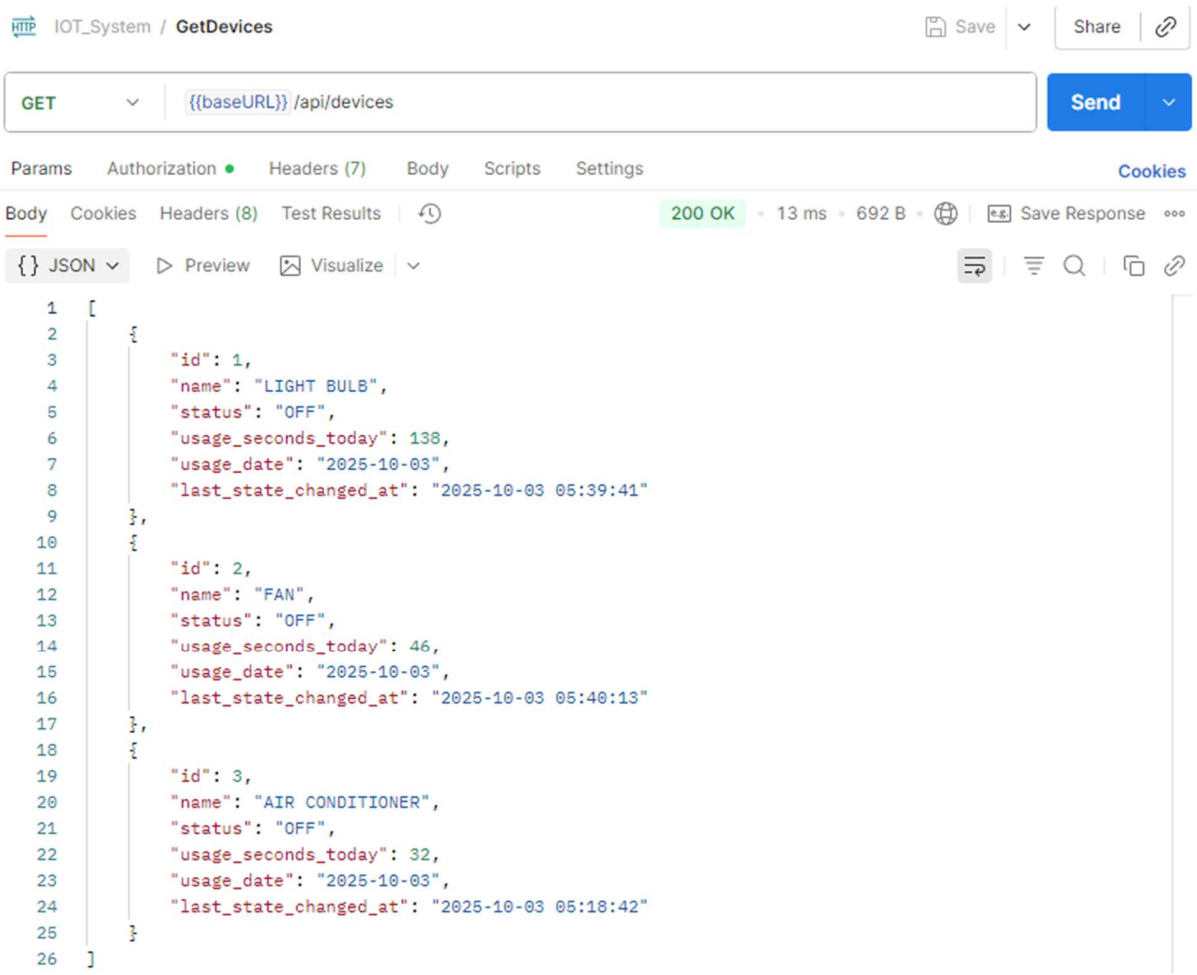
5. Tài liệu API (POSTMAN)

a. GET /api/devices

Lấy danh sách toàn bộ thiết bị.

Input: không yêu cầu

Output:

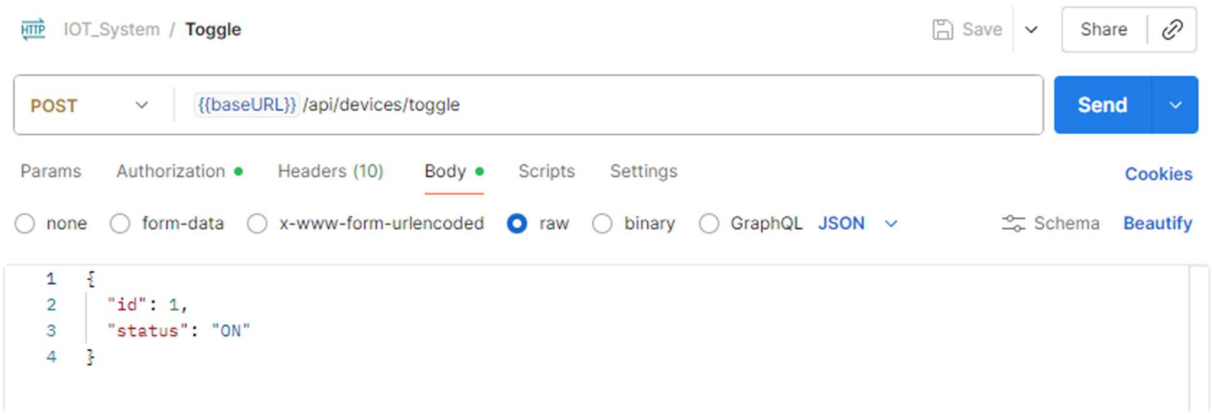


Hình 22: Output GET /api/devices

b. POST /api/devices/toggle

Điều khiển thiết bị

Input:



Hình 23: Input POST/api/devices/toggle

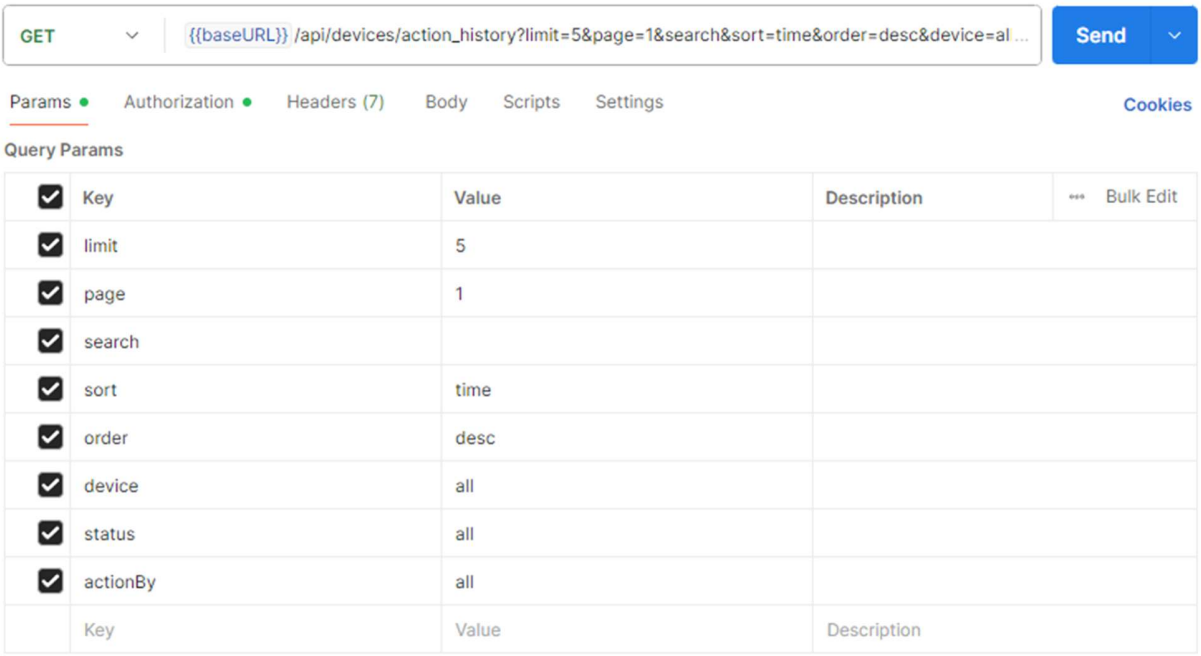
Output:

Hình 24: Output POST /api/devices/toggle

c. GET /api/devices/action_history

Lấy lịch sử điều khiển thiết bị.

Input:



Hình 25: Input GET /api/devices/action_history

Output:


```
{
    "data": [
        {
            "id": 350,
            "deviceId": 2,
            "deviceName": "FAN",
            "status": "OFF",
            "actionBy": "User",
            "time": "2025-10-03 05:40:13"
        },
        {
            "id": 349,
            "deviceId": 2,
            "deviceName": "FAN",
            "status": "ON",
            "actionBy": "User",
            "time": "2025-10-03 05:40:04"
        },
        {
            "id": 348,
            "deviceId": 1,
            "deviceName": "LIGHT BULB",
            "status": "OFF",
            "actionBy": "User",
            "time": "2025-10-03 05:39:41"
        },
        {
            "id": 347,
            "deviceId": 1,
            "deviceName": "LIGHT BULB",
            "status": "ON",
            "actionBy": "User",
            "time": "2025-10-03 05:39:40"
        },
        {
            "id": 346,
            "deviceId": 1,
            "deviceName": "LIGHT BULB",
            "status": "OFF",
            "actionBy": "User",
            "time": "2025-10-03 05:18:44"
        }
    ],
    "pagination": {
        "page": 1,
        "limit": 5,
        "total": 350,
        "totalPages": 70
    }
}
```

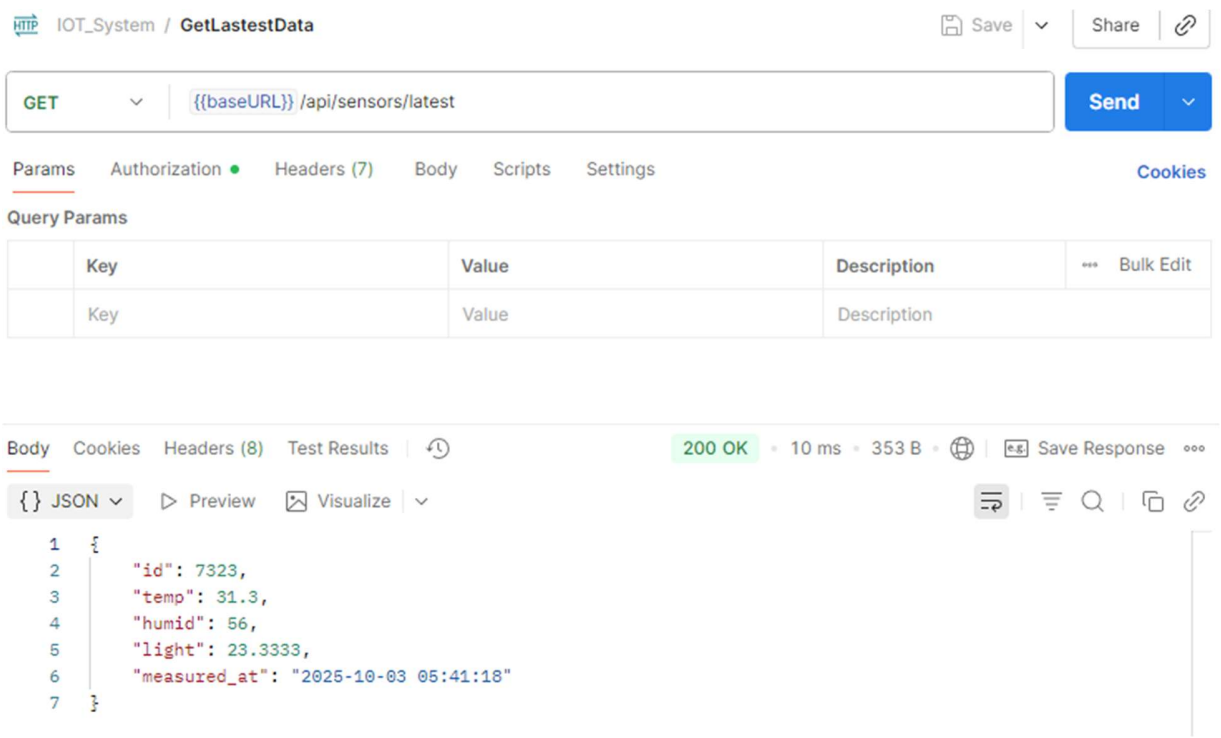

Hình 26: Output GET /api/devices/action_history

d. GET /api/sensors/latest

Lấy dữ liệu cảm biến mới nhất

Input: Không yêu cầu

Output:



Hình 27: Output GET /api/sensors/lastest

e. GET /api/sensors/today

Lấy dữ liệu cảm biến trong ngày

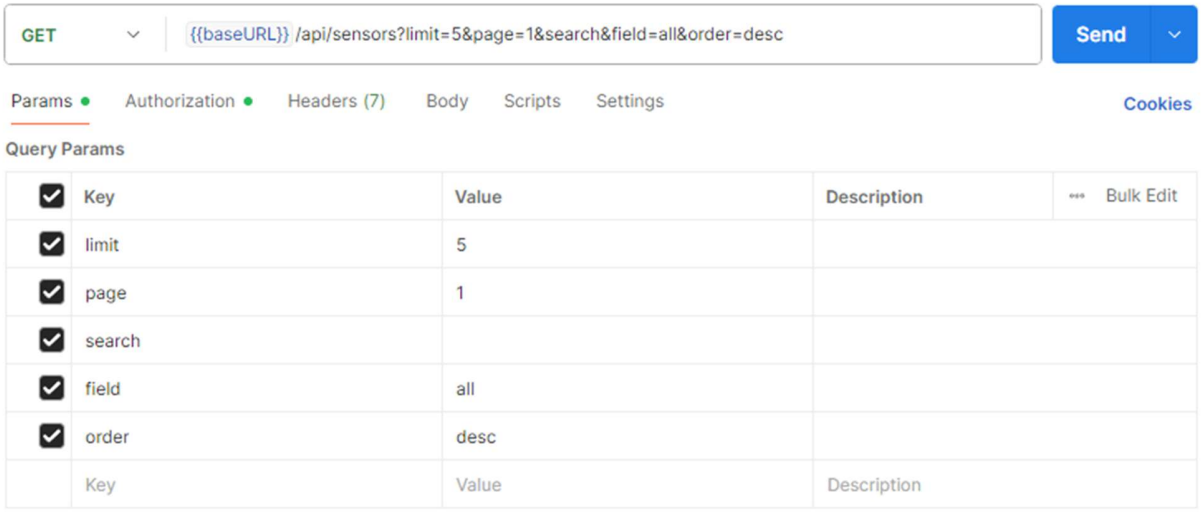
Input: Không yêu cầu

Output:

f. GET /api/sensors

Lấy danh sách dữ liệu cảm biến (có phân trang, tìm kiếm, sắp xếp).

Input:



Hình 28: Input GET /api/sensors

Output:

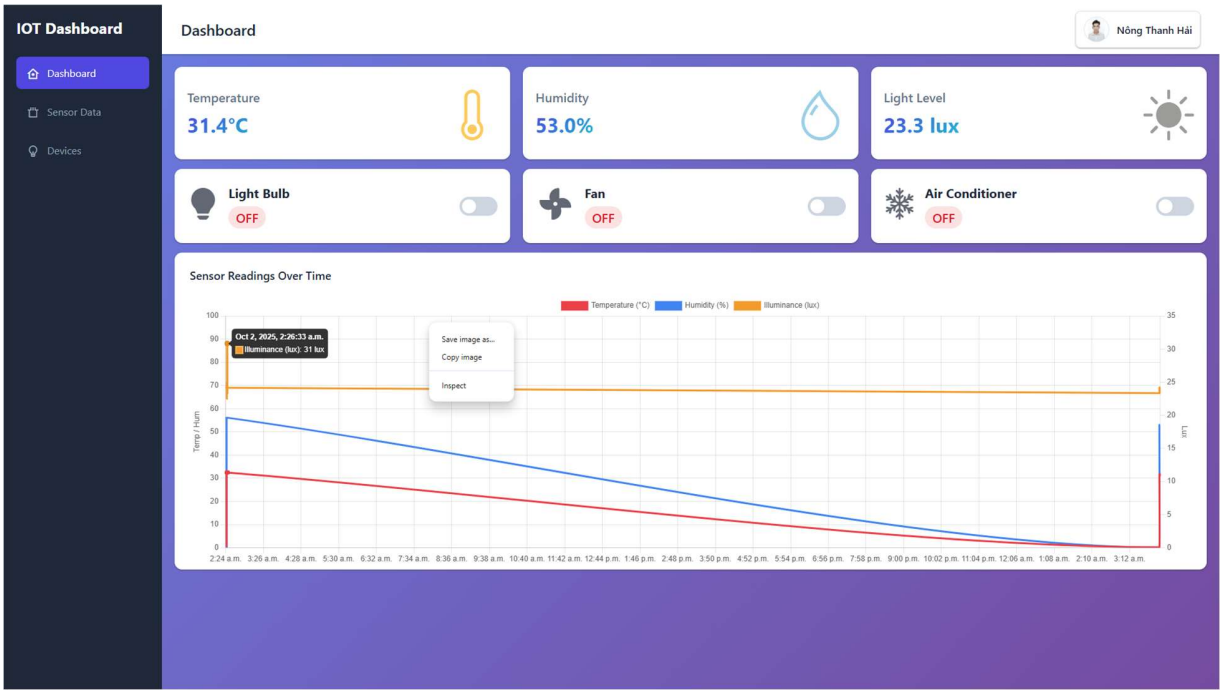

```
{
    "data": [
        {
            "id": 7323,
            "temp": 31.3,
            "humid": 56,
            "light": 23.3333,
            "measured_at": "2025-10-03 05:41:18"
        },
        {
            "id": 7322,
            "temp": 31.2,
            "humid": 56,
            "light": 23.3333,
            "measured_at": "2025-10-03 05:41:16"
        },
        {
            "id": 7321,
            "temp": 31.1,
            "humid": 56,
            "light": 23.3333,
            "measured_at": "2025-10-03 05:41:14"
        },
        {
            "id": 7320,
            "temp": 31.4,
            "humid": 56,
            "light": 23.3333,
            "measured_at": "2025-10-03 05:41:12"
        },
        {
            "id": 7319,
            "temp": 31.6,
            "humid": 56,
            "light": 23.3333,
            "measured_at": "2025-10-03 05:41:10"
        }
    ],
    "pagination": {
        "page": 1,
        "limit": 5,
        "total": 6272,
        "totalPages": 1255
    }
}
```


Hình 29:Output GET /api/sensors

6. Xây dựng giao diện người dùng(Front End)

a. Trang Dashboard

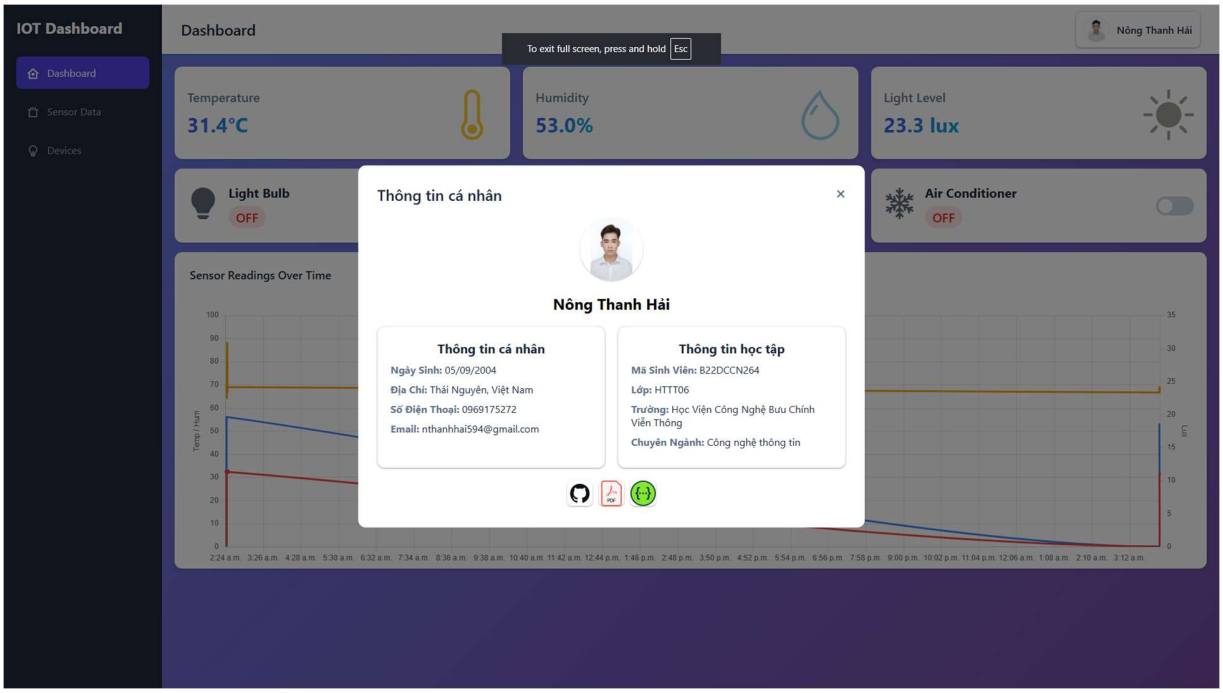
• Hiển thị các thông số nhiệt độ, độ ẩm, ánh sáng thời gian thức đến cho người dùng từ thiết bị điều khiển và cảm biến.
• Hiển thị biểu đồ tổng quan của các thông số đến người dùng để dễ dàng xem sự biến đổi.
• Giao diện điều khiển ON/OFF các thiết bị.



Hình 30: Giao diện trang Dashboard

b. Trang Profile

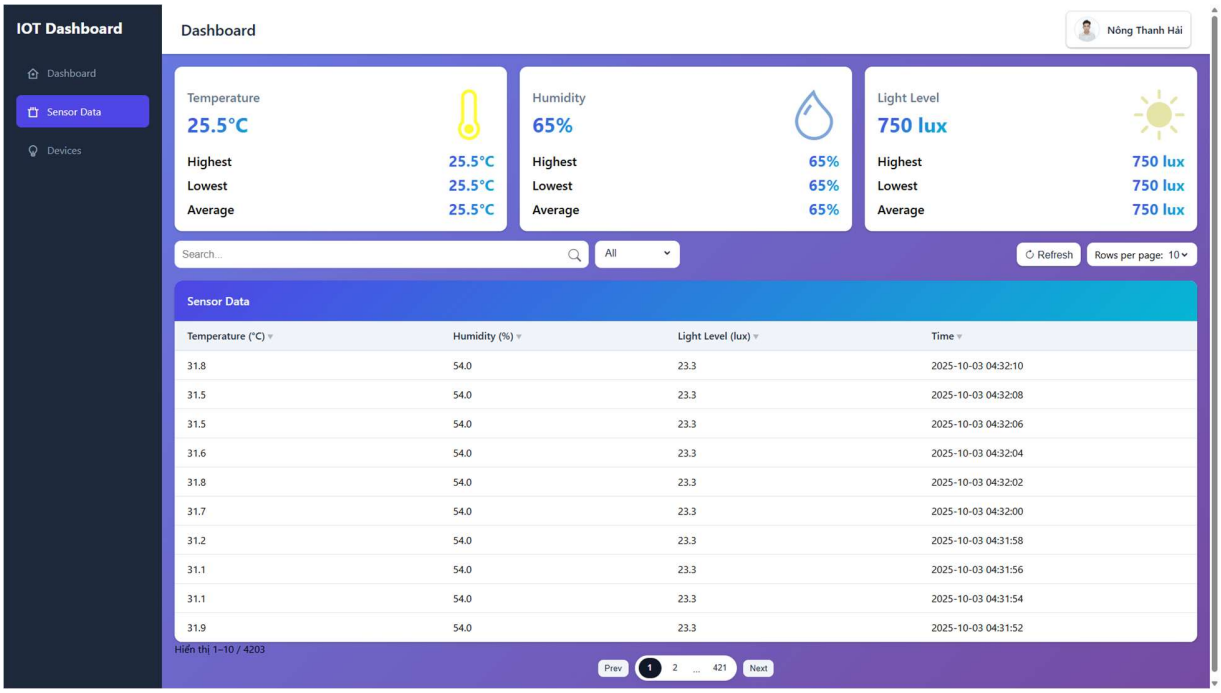
• Hiển thị thông tin cá nhân và thông tin học tập của người điều khiển cùng với các thông tin về dự án như: Github, Swapgger, Pdf.



Hình 31:Giao diện trang Profile

c. Trang DataSensor

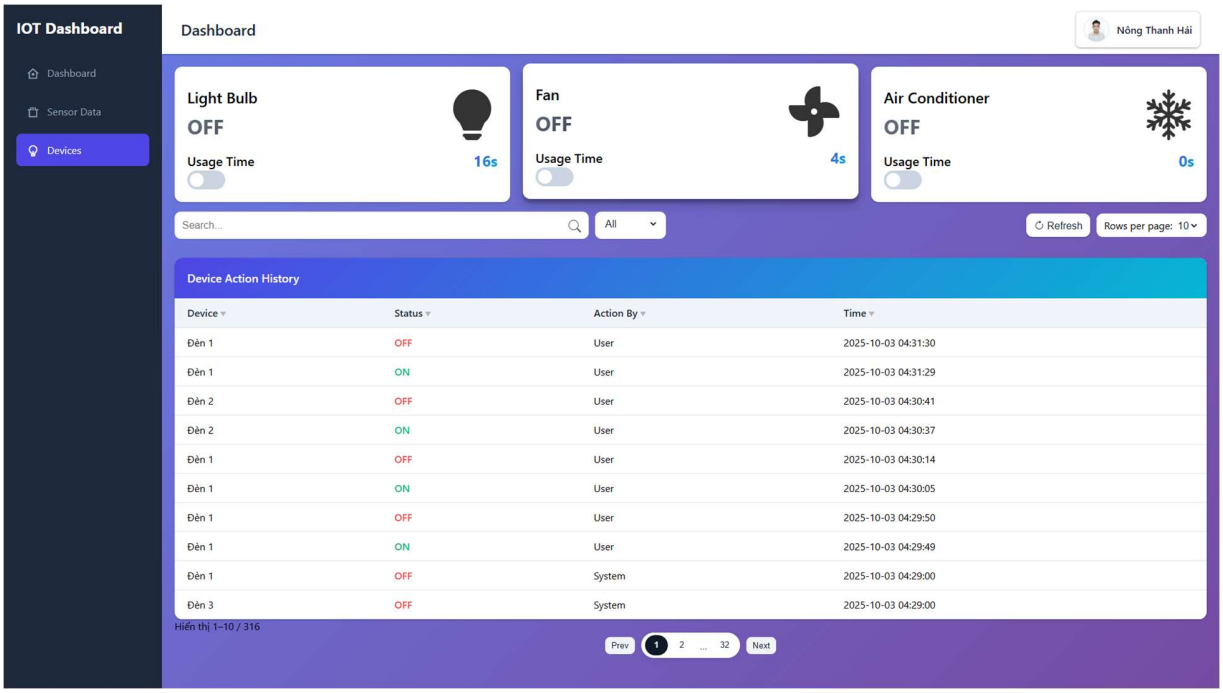
- Giao diện hiển thị, thông tin cảm biến hiện tại, các mốc cao nhất, thấp nhất và trung bình trong ngày, lịch sử biến đổi các thông số đo từ thiết bị.
- Thanh thực hiện tìm kiếm theo yêu cầu như: theo thời gian, số liệu chính xác từ người dùng.



Hình 32:Giao diện trang SensorData

d. Trang Devices

- Hiển thị trạng thái ,điều khiển các thiết bị
- 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 33: Giao diện trang Devices

7. Kiểm thử hệ thống

Trường hợp kiểm thử	Mô tả	Kết quả mong đợi	Kết quả thực tế
ESP8266 gửi dữ liệu cảm biến	Gửi dữ liệu 2s/lần qua MQTT	Server nhận và lưu MySQL	Đạt
Dashboard hiển thị realtime	Cập nhật dữ liệu mà không tải lại trang	Biểu đồ và giá trị cập nhật mượt	Đạt
Bật/tắt thiết bị	Nhấn nút ON/OFF trên web	LED thay đổi trạng thái ngay	Đạt
Mất kết nối MQTT	Ngắt WiFi tạm thời	Tự động kết nối lại sau khi ổn định	Đạt
Lưu lịch sử hoạt động	Mỗi thao tác được ghi lại vào DB	Bảng action_history có bản ghi mới	Đạt

8. Triển khai hệ thống

- **MQTT Broker:** cài đặt trên máy tính nội bộ bằng Mosquitto.
- **Node.js Server:** chạy ở cổng http://localhost:3000.
- **Giao diện Web:** truy cập trực tiếp tại http://localhost:3000/.
- **ESP8266:** kết nối cùng mạng WiFi nội bộ, publish/subcribe MQTT qua địa chỉ IP của máy chủ.

Kết quả thực nghiệm:

- Thời gian phản hồi trung bình 200ms.
- Cảm biến hoạt động ổn định, dữ liệu hiển thị trơn tru.
- Người dùng có thể giám sát và điều khiển thiết bị ngay trên giao diện web.

CHƯƠNG V: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

1. Kết luận

Sau quá trình nghiên cứu và xây dựng, đề tài **"Hệ thống IoT Dashboard giám sát và điều khiển thiết bị"** đã được hoàn thiện và vận hành thành công. Hệ thống cho phép **giám sát thông số môi trường theo thời gian thực** và **điều khiển thiết bị IoT từ xa** thông qua giao diện web trực quan.

Kết quả đạt được cụ thể như sau:

- **Hoàn thiện phần cứng IoT:**

  ESP8266 NodeMCU kết nối với cảm biến **DHT11** và **BH1750** để đo các giá trị **nhiệt độ, độ ẩm, ánh sáng**. Các thiết bị mô phỏng (LED, quạt, điều hòa) được điều khiển qua chân GPIO, hoạt động ổn định và phản hồi tức thì.

- **Hoàn thiện phần mềm máy chủ (Server):**

  Sử dụng **Node.js + Express** làm nền tảng backend, **MQTT** để giao tiếp dữ liệu với ESP8266, **MySQL + Sequelize** để lưu trữ dữ liệu cảm biến và lịch sử điều khiển.
  Dữ liệu được xử lý và cập nhật **thời gian thực (Realtime)** bằng **Socket.IO**.

- **Hoàn thiện giao diện người dùng (Frontend):**

  Giao diện web được xây dựng bằng **HTML, CSS, JavaScript, Chart.js**, hiển thị trực quan các thông số cảm biến và biểu đồ theo thời gian. Người dùng có thể bật/tắt thiết bị ngay trên dashboard và quan sát phản hồi tức thời.

- **Kết nối hệ thống ổn định:**

  Thời gian phản hồi trung bình dưới 200ms, dữ liệu cảm biến cập nhật đều đặn 2s/lần, tự động khôi phục khi mất kết nối mạng.

Nhìn chung, hệ thống đạt được mục tiêu đề ra:

"Xây dựng một mô hình IoT hoàn chỉnh cho phép giám sát môi trường và điều khiển thiết bị qua Internet."

2. Hướng phát triển

Để hoàn thiện và nâng cao hơn trong tương lai, hệ thống có thể được mở rộng theo các hướng sau:

1) **Phát triển tính năng người dùng:**

- Thêm **đăng nhập, phân quyền** (Admin/User).
- Lưu lịch sử hoạt động theo từng tài khoản.

2) **Tích hợp cảnh báo thông minh:**

- Gửi thông báo khi nhiệt độ, độ ẩm hoặc ánh sáng vượt ngưỡng cho phép.
- Tự động bật/tắt thiết bị khi điều kiện môi trường thay đổi (ví dụ: bật quạt khi nhiệt độ > 30°C).

3) **Lưu trữ và phân tích dữ liệu:**

- Chuyển dữ liệu cảm biến lên **nền tảng đám mây** như **Firebase, AWS IoT hoặc ThingsBoard** để giám sát từ xa mọi lúc, mọi nơi.
- Kết hợp **trí tuệ nhân tạo (AI)** để dự đoán xu hướng thay đổi môi trường.

4) **Phát triển ứng dụng di động:**

- Xây dựng **mobile app (React Native hoặc Flutter)** để người dùng có thể điều khiển và theo dõi thiết bị dễ dàng qua điện thoại.

5) **Cải thiện phần cứng:**

- Thay cảm biến **DHT11 → DHT22** để tăng độ chính xác.
- Bổ sung cảm biến **khí gas, độ ồn, chuyển động PIR** cho các ứng dụng Smart Home.
- Dùng **relay module** để điều khiển thiết bị điện thực tế.

3. Kinh nghiệm và bài học đạt được

Trong quá trình thực hiện đề tài, người thực hiện đã tích lũy được nhiều kinh nghiệm quan trọng, bao gồm:

- **Hiểu rõ mô hình hoạt động của hệ thống IoT:**

Từ phần cứng, giao thức truyền dữ liệu, đến backend và frontend web.

- **Nắm vững kỹ năng lập trình Node.js và Arduino:**

  Biết cách sử dụng **MQTT**, **Socket.IO**, **Express**, và **MySQL** trong việc xây dựng hệ thống realtime.

- **Kỹ năng thiết kế hệ thống và giao diện:**

  Thiết kế dashboard trực quan, tối ưu hiển thị dữ liệu cảm biến và biểu đồ động bằng **Chart.js**.

- **Kỹ năng phân tích – khắc phục lỗi:**

  Biết cách debug lỗi kết nối MQTT, lỗi WiFi, và xử lý mất kết nối tự động trên ESP8266.

- **Hiểu được quy trình phát triển một dự án IoT hoàn chỉnh:**

  Từ thiết kế sơ đồ mạch, viết chương trình nhúng, đến triển khai backend và giao diện người dùng.

4. Tổng kết

Đề tài **IoT Dashboard giám sát và điều khiển thiết bị** không chỉ giúp người thực hiện hiểu sâu hơn về lĩnh vực **Internet of Things**, mà còn rèn luyện tư duy **phân tích, thiết kế, lập trình và triển khai hệ thống thực tế**.

Sản phẩm có tính ứng dụng cao, có thể mở rộng thành mô hình **nhà thông minh (Smart Home)** hoặc **hệ thống giám sát môi trường (Environmental Monitoring System)**, góp phần thúc đẩy việc áp dụng công nghệ IoT trong đời sống hàng ngày.