

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CƠ BẢN I

BỘ MÔN IOT VÀ ỨNG DỤNG



IOT VÀ ỨNG DỤNG

Giảng viên hướng dẫn	: NGUYỄN QUỐC UY
Họ và tên sinh viên	: VŨ ĐỨC VUI
Mã sinh viên	: B22DCCN921
Nhóm	: 16

Hà Nội – 2025

MỤC LỤC

1	Đặt vấn đề	4
2	Tổng quan dự án.....	4
2.1	Mục đích của dự án	4
2.2	Các tính năng của hệ thống	4
2.3	Khả năng mở rộng.....	5
2.4	Công nghệ sử dụng.....	5
2.5	Các thiết bị sử dụng.....	6
2.	Tổng quan giao diện.....	12
2.1.	Trang Dashboard	12
2.2.	Trang Data Sensor	12
2.3.	Trang Action History	13
2.4.	Trang Profile Settings.....	13
3.	Thiết kế chi tiết.....	14
3.1	Sơ đồ kiến trúc hệ thống.....	14
3.1	Biểu đồ tuần tự	15
3.3	Database	18
4.	Code	19
4.1	APIs routing	19
4.2	API dữ liệu cảm biến /api/data	20
4.3	API lịch sử hành động /api/actions/history	21
4.4	API điều khiển thiết bị /api/command.....	21
4.5	API cung cấp thông tin trạng thái kết nối /api/mqtt-status	22
5.	Swagger.....	23

DANH MỤC HÌNH ẢNH

Figure 1: ESP WROOM 32	8
Figure 2: DHT11 Views.....	9
Figure 3: Cảm biến quang trở.....	10
Figure 4: Led	11
Figure 5: Bảng Action Logs	18
Figure 6: Bảng Action History	19
Figure 7 /api/actions/history:	23
Figure 8: response /api/actions/history	23
Figure 9: /api/command.....	24
Figure 10: response /api/command.....	24
Figure 11: /api/data	24
Figure 12: response /api/data	25
Figure 13: /api/data/history	25
Figure 14: response /api/data/history	26
Figure 15: /api/mqtt-status.....	26
Figure 16: response /api/mqtt-status.....	26

1 Đặt vấn đề

Trong bối cảnh công nghệ ngày càng phát triển, việc xây dựng một hệ thống thời gian thực đóng vai trò quan trọng trong việc thu thập và truyền tải dữ liệu từ các cảm biến đến người dùng cuối một cách nhanh chóng, chính xác và liên tục. Hệ thống này không chỉ đảm bảo dữ liệu được hiển thị trên giao diện người dùng theo thời gian thực, mà còn tạo điều kiện để người dùng có thể tương tác ngược lại, gửi các lệnh điều khiển hoặc phản hồi trực tiếp đến thiết bị. Đây là mô hình cơ bản của một hệ thống Internet vạn vật (IoT) đơn giản, nơi các thiết bị cảm biến, bộ xử lý và giao diện người dùng được kết nối thông qua mạng internet, hình thành một mạng lưới thông minh nhằm mục đích tối ưu hóa việc giám sát, quản lý và điều khiển từ xa. Hệ thống IoT như vậy không chỉ nâng cao hiệu quả hoạt động mà còn cải thiện đáng kể trải nghiệm của người dùng thông qua sự kết nối liền mạch giữa con người và thiết bị.

2 Tổng quan dự án

2.1 Mục đích của dự án

Dự án được phát triển nhằm mục đích tạo ra một giải pháp công nghệ toàn diện, giúp xây dựng một hệ thống giám sát và điều khiển thông minh cho không gian sống và làm việc. Bằng cách kết hợp sức mạnh của công nghệ Internet of Things (IoT) với một giao diện người dùng trực quan, dự án mang đến cho người dùng khả năng quản lý môi trường trong nhà một cách dễ dàng, hiệu quả và chủ động hơn bao giờ hết.

Dự án sẽ xây dựng nên một hệ thống cho phép người dùng giám sát các thông số môi trường như nhiệt độ, độ ẩm, ánh sáng trong thời gian thực, đồng thời điều khiển từ xa các thiết bị như đèn, quạt, và điều hòa. Hệ thống có khả năng phân tích dữ liệu lịch sử để thiết lập các quy tắc tự động hóa, giúp các thiết bị tự điều chỉnh theo điều kiện môi trường. Tất cả được quản lý qua một giao diện dashboard trực quan, dễ sử dụng, nhằm mang lại sự tiện nghi, tiết kiệm năng lượng và nâng cao chất lượng cuộc sống.

2.2 Các tính năng của hệ thống

- **Hiển thị dữ liệu thời gian thực:** Các thông số quan trọng như nhiệt độ, độ ẩm, và cường độ ánh sáng được cập nhật liên tục và hiển thị qua các thẻ thông tin (cards) nổi bật. Bên cạnh đó, các biểu đồ đường (line charts) trực quan hóa sự thay đổi của các thông số này theo thời gian trong ngày, giúp người dùng nắm bắt xu hướng một cách nhanh chóng.

- Điều khiển thiết bị từ xa: Ngay trên dashboard, người dùng có thể dễ dàng bật/tắt các thiết bị như quạt, điều hòa, đèn thông qua các nút gạt (toggles). Trạng thái hiện tại (đang bật hay đang tắt) của từng thiết bị cũng được hiển thị rõ ràng.
- Lịch sử Dữ liệu Cảm biến (Data Sensor): Cung cấp một bảng ghi lại toàn bộ dữ liệu thô từ cảm biến theo từng mốc thời gian cụ thể. Với các công cụ tìm kiếm và phân trang, người dùng có thể dễ dàng tra cứu lại các chỉ số trong quá khứ để phục vụ cho việc phân tích chuyên sâu.
- Nhật ký Hoạt động (Action History): Lưu lại tất cả các hành động điều khiển thiết bị (bật/tắt) đã được thực hiện. Chức năng này rất quan trọng để kiểm tra, giám sát và quản lý hệ thống, đảm bảo mọi thay đổi đều được ghi nhận.

2.3 Khả năng mở rộng

- Gửi thông báo đẩy (push notification) đến điện thoại hoặc email của người dùng khi có sự kiện bất thường xảy ra. Ví dụ: Cảnh báo khi độ ẩm quá cao (>85%) để chống nấm mốc, hoặc khi nhiệt độ quá thấp/cao so với ngưỡng an toàn.
- Tích hợp tính năng theo dõi thời gian hoạt động của các thiết bị (đèn, điều hòa, quạt) để ước tính lượng điện năng tiêu thụ. Hệ thống có thể tạo ra các báo cáo hàng tuần/tháng, giúp người dùng quản lý chi phí và sử dụng năng lượng hiệu quả hơn.

2.4. Công nghệ sử dụng

- Về phía backend, hệ thống được xây dựng trên nền tảng Node.js – môi trường chạy JavaScript phía máy chủ nổi bật với hiệu năng cao và khả năng xử lý bất đồng bộ, rất phù hợp cho các ứng dụng IoT. Đi cùng với đó là Express.js – một framework web tối giản nhưng mạnh mẽ, giúp xây dựng hệ thống API RESTful một cách nhanh chóng và có cấu trúc. Để tương tác với cơ sở dữ liệu, dự án sử dụng Prisma ORM, một công cụ hiện đại giúp việc truy vấn dữ liệu trở nên an toàn, trực quan và dễ bảo trì hơn so với viết SQL thuần.
- Ở phía frontend, dự án lựa chọn React.js làm thư viện chính để xây dựng một giao diện người dùng (UI) sinh động, hiện đại và có khả năng phản hồi nhanh. Để quản lý việc điều hướng giữa các trang chức năng, React Router được tích hợp, cho phép tạo ra một ứng dụng đơn trang (SPA) mượt mà. Dữ liệu cảm biến được trực quan hóa thành các biểu đồ sống động nhờ thư viện Chart.js. Trong khi đó, Axios được dùng để thực hiện

các cuộc gọi API đến backend, và đặc biệt, Socket.IO (dựa trên WebSocket) được triển khai để tạo ra kết nối hai chiều, giúp cập nhật dữ liệu trên dashboard theo thời gian thực mà không cần tải lại trang.

- Về cơ sở dữ liệu, dự án áp dụng một chiến lược lưu trữ kép để tối ưu hóa hiệu suất. MySQL (hoặc PostgreSQL), một hệ quản trị cơ sở dữ liệu quan hệ mạnh mẽ, được sử dụng để lưu trữ các dữ liệu có cấu trúc như thông tin người dùng, cấu hình thiết bị và lịch sử hành động. Đồng thời, một cơ sở dữ liệu chuỗi thời gian như InfluxDB được khuyến nghị để lưu trữ dữ liệu cảm biến (nhiệt độ, độ ẩm, ánh sáng) vốn có tần suất ghi rất cao, giúp việc truy vấn và phân tích theo thời gian trở nên cực kỳ hiệu quả.
- Trong phần giao thức truyền thông, MQTT (Message Queuing Telemetry Transport) đóng vai trò chủ đạo cho việc giao tiếp giữa thiết bị IoT và máy chủ. Với đặc tính nhẹ và mô hình publish/subscribe linh hoạt, MQTT đảm bảo dữ liệu được truyền đi một cách tin cậy và tức thời. Song song với đó, giao thức HTTP/HTTPS là nền tảng cho việc giao tiếp giữa frontend và backend thông qua các API RESTful đã xây dựng.
- Về phần cứng, trái tim của hệ thống là vi điều khiển ESP32 với khả năng kết nối WiFi mạnh mẽ. Thiết bị này được lập trình thông qua môi trường Arduino IDE hoặc PlatformIO, có nhiệm vụ thu thập dữ liệu từ các cảm biến như DHT11/DHT22 (đo nhiệt độ và độ ẩm), LDR (đo cường độ ánh sáng) và điều khiển các thiết bị ngoại vi như đèn, quạt thông qua module Relay.
- Cuối cùng, để đảm bảo quy trình phát triển chuyên nghiệp, dự án sử dụng Git để quản lý phiên bản mã nguồn và GitHub làm nền tảng lưu trữ, cộng tác. Công cụ Postman được tận dụng để kiểm thử và tài liệu hóa API, giúp đảm bảo hệ thống hoạt động ổn định và chính xác.

2.5 Các thiết bị sử dụng

2.5.1 CHIP WIFI ESP WROOM32

ESP WROOM32 là một module vi điều khiển cao tích hợp, đặc biệt là khả năng đáp ứng đầy đủ cho các dự án IoT và các ứng dụng không dây. Bàn nền trên nền vi xử lý ESP32 khỏe mạnh, module này áp dụng giải pháp kết nối đầy đủ với chi phí ít nhất.

- Điểm nổi bật tạo nên sự vượt trội:

- + Tích hợp đồng thời WiFi (802.11 b/g/n) và Bluetooth/BLE
- + Đầy đủ hỗ trợ giao thức TCP/IP và MQTT dành cho IoT
- + Bộ nhớ diện tích rộng, thỏa thích các ứng dụng phức tạp

- + Đa dạng giao tiếp ngoại vi, dễ mở rộng
 - + Tiêu dùng điện năng thấp, dùng được cho thiết bị di động
- Thông số kỹ thuật của ESP WROOM 32:
- + Điện áp nguồn (USB): 5V DC
 - + Đầu vào/Đầu ra điện áp: 3.3V DC
 - + Dòng điện : 5 μ A trong hệ thống treo chế độ
 - + In/OUT : 24
 - + Mô hình : ESP32 38 chân
 - + Thạch anh : 240Mhz
 - + Các chuẩn giao tiếp : I²C, SPI, UART / USART, USB, CAN
 - + Nhiệt độ hoạt động : -40°C ~ 85°C
 - + Cổng ADC : 12bit – 18 Kênh
 - + Chip USB-Serial: CP2102
 - + Loại: Wifi + Bluetooth Module
 - + Hiệu suất: Lên đến 600 DMIPS
 - + Tần số: lên đến 240MHz
 - + Wifi: 802.11 B/g/n/E/I (802.11N @ 2.4 GHz lên đến 150 Mbit/S)
 - + Bluetooth: 4.2 BR/EDR BLE 2 chế độ điều khiển
 - + Ăng ten: PCB
 - + Bảo mật: IEEE 802.11, bao gồm cả WFA, WPA/WPA2 và WAPI
 - + Phần cứng tăng tốc mật mã học: AES, SHA-2, RSA, hình elip mật mã Đường Cong (ECC), số ngẫu nhiên Máy phát điện (RNG)

Dựa trên lợi thế nổi bật này, ESP WROOM32 thực sự là lựa chọn ưu tiên cho các nhà phát triển khi chỉ đạo dự án IoT, từ nhỏ đến các ứng dụng thương mại. Module này không chỉ mang đến hiệu suất mà còn bao gồm tính linh hoạt và khả năng tích hợp mạnh mẽ, góp phần rút ngắn thời gian phát triển và tối ưu hóa chi phí.

ESP32 module common features

All ESP32 modules share these features (only a summary):

- CPU cores (one or two)
- Internal memory (ROM, SRAM)
- External SRAM
- Timers and watchdogs
 - Four general-purpose 64-bit timers
 - Three watchdog timers (used to recover from faults)
- RTC clock
- 2.4 GHz receiver and transmitter radio
- Wifi, 802.11 b/g/n
- Bluetooth, classic and BLE
- RTC (co-processor) and Low-Power management with multiple power modes.
- 34 GPIO pins
- Analog to Digital Converter (ADC)
- Hall Sensor, capable to detect a magnetic field without additional hardware
- Digital to Analog Converter (DAC)
- Touch sensor via 10 capacitive-sensing pins.
- Ethernet MAC interface.
- SD/SDIO/MMC host controller
- SDIO/SPI slave controller
- UART
- I²C
- Infrared Remote Controller
- Pulse Counter
- Pulse Width Modulation (PWM)
- LED PWM
- SPI
- Hardware acceleration of algorithms such as AES, RSA and ECC



ESP32 for busy people

Tech
Explorations

Figure 1: ESP WROOM 32

2.5.2 Cảm biến nhiệt độ độ ẩm DHT11

Cảm biến nhiệt độ và độ ẩm DHT11 là một giải pháp giá rẻ và phổ biến, được thiết kế để đo và theo dõi các thông số môi trường. Với khả năng đo nhiệt độ trong khoảng 0-50°C (độ chính xác $\pm 2^{\circ}\text{C}$) và độ ẩm từ 20-90% RH (độ chính xác $\pm 5\%$ RH), DHT11 đáp ứng tốt nhu cầu giám sát môi trường cơ bản.

- Về mặt cấu tạo, DHT11 có 3 chân kết nối chính:

- + VCC: Điện áp nguồn (3.3V-5V)
- + GND: Chân nối đất
- + DATA: Chân truyền dữ liệu (cần điện trở pull-up 4.7k Ω -10k Ω)

DHT11 sử dụng giao thức truyền thông 1-wire đơn giản, cho phép trao đổi dữ liệu với vi điều khiển qua một dây tín hiệu duy nhất. Cảm biến truyền dữ liệu dưới dạng tín hiệu số với độ phân giải 1°C cho nhiệt độ và 1% cho độ ẩm. Tần suất đọc dữ liệu tối đa là 1 lần/giây, phù hợp cho hầu hết các ứng dụng thông thường.

- Các đặc điểm kỹ thuật chính của cảm biến bao gồm:

- + Phạm vi đo nhiệt độ: 0°C đến 50°C với độ chính xác $\pm 2^{\circ}\text{C}$.
- + Phạm vi đo độ ẩm: 20% đến 90% RH với độ chính xác $\pm 5\%$ 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.

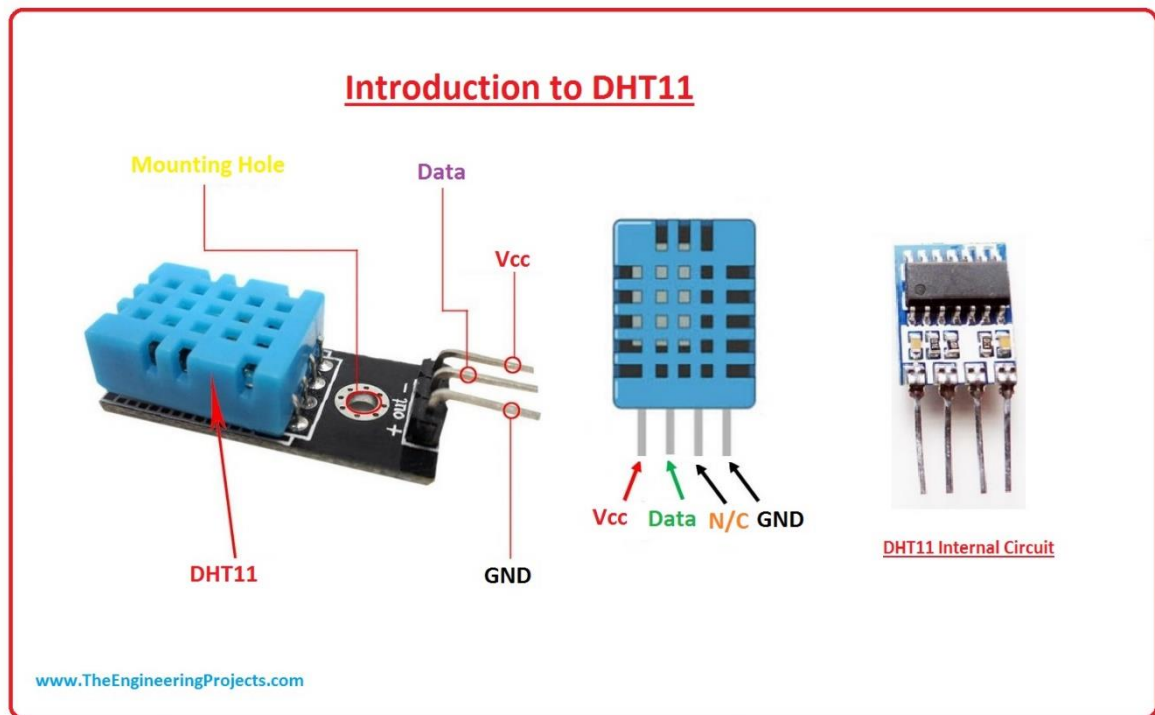


Figure 2: DHT11 Views

2.5.3 Cảm biến quang trở 4 chân

Cảm Biến Ánh Sáng Quang Trở 4 Chân nhạy nhất với ánh sáng xung quanh, thường được sử dụng để phát hiện độ sáng của môi trường xung quanh, kích hoạt vi điều khiển hoặc mô-đun chuyển tiếp..

Khi độ sáng của môi trường xung quanh không đạt đến ngưỡng cài đặt, đầu DO sẽ tạo ra mức cao. Khi độ sáng của môi trường xung quanh vượt quá ngưỡng đã đặt, đầu DO sẽ tạo ra mức thấp

- Thông số kỹ thuật :

- + Điện áp làm việc: 3.3~5VDC
- + Sử dụng LM393
- + Sử dụng quang trở có độ nhạy cảm cao
- + Có biến trở để điều chỉnh
- + Kích thước : 32x14mm

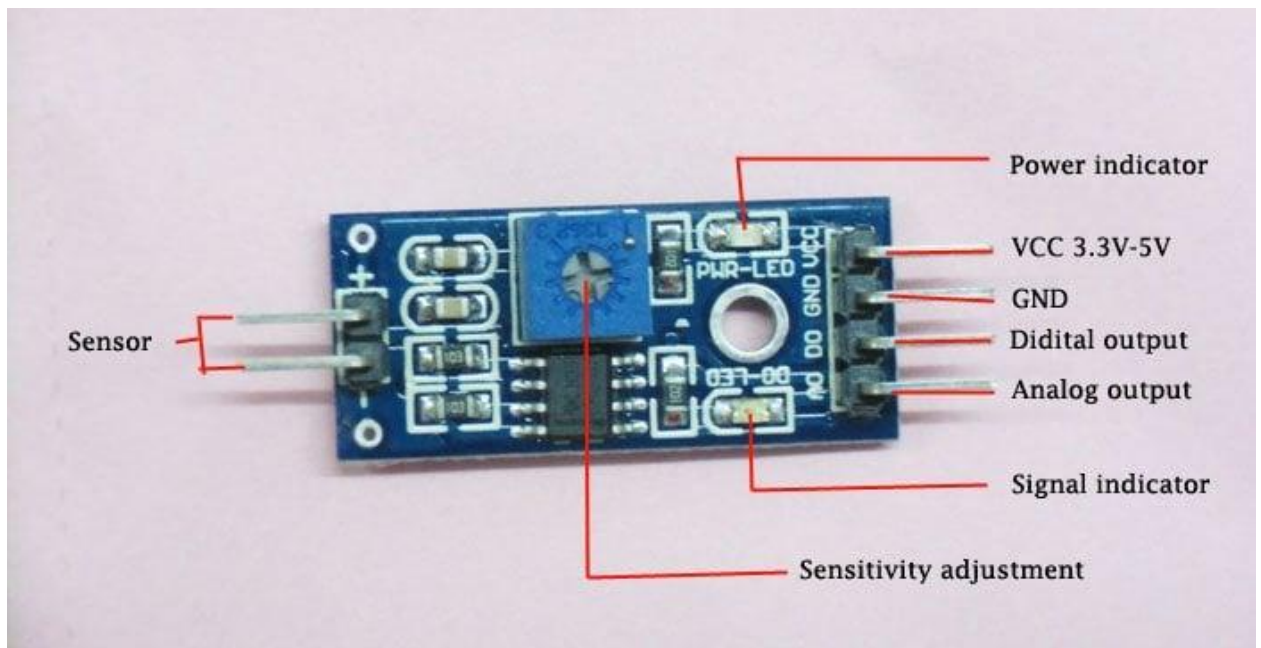


Figure 3: Cảm biến quang trở

2.5.4 Led

LED (Light Emitting Diode) là một linh kiện bán dẫn hai chân, rất phổ biến trong các ứng dụng điện tử và dự án IoT nhờ hiệu suất và tính linh hoạt. Về cấu tạo, LED gồm chân dương (Anode), dài hơn, kết nối với nguồn điện dương và chân âm (Cathode), ngắn hơn, nối với mass, cả hai đều được bọc trong vỏ nhựa epoxy trong suốt để bảo vệ và khuếch tán ánh sáng.

Khi có dòng điện chạy theo chiều thuận từ Anode đến Cathode, LED sẽ phát sáng nhờ quá trình tái hợp của các electron và lỗ trống trong vật liệu bán dẫn, với điện áp hoạt động thường từ 1.8V đến 3.3V và dòng điện tối ưu khoảng 20mA. Để bảo vệ LED khỏi hư hỏng, mỗi LED cần có điện trở hạn dòng đặt nối tiếp, nhập điện trở phụ thuộc vào điện áp nguồn và loại LED cụ thể được sử dụng.

LED có khả năng phát ra nhiều màu sắc như đỏ, xanh lá, xanh dương, vàng và trắng, với cường độ sáng từ 100 đến 5000 mcd tùy thuộc vào loại vật liệu bán dẫn. Những ưu điểm nổi bật của LED bao gồm tiêu thụ điện năng thấp, tuổi thọ cao lên đến hàng chục nghìn giờ, và khả năng khởi động nhanh ngay kể cả trong điều kiện nhiệt độ khắc nghiệt.

Trong các dự án này LED được sử dụng để báo trạng thái. Có thể dễ dàng điều khiển LED thông qua chân GPIO của vi điều khiển, sử dụng mức logic HIGH/LOW để bật hoặc tắt, hoặc chế độ PWM để điều chỉnh độ sáng và tạo hiệu ứng. Ngoài việc chú ý đến cực tính của LED khi kết nối, cần đảm bảo rằng dòng điện không vượt quá định mức an toàn để tránh làm hỏng LED, điều này đặc biệt quan trọng khi thực hiện thiết kế mạch cho các ứng dụng khác nhau.



Figure 4: Led

Kết luận:

Dự án Hệ thống Giám sát và Điều khiển Thông minh được phát triển với mục đích tạo ra một giải pháp công nghệ toàn diện, kết hợp sức mạnh của Internet of Things (IoT), nhằm mang lại khả năng quản lý môi trường sống và làm việc một cách dễ dàng, hiệu quả và chủ động; hệ thống cho phép người dùng giám sát các thông số môi trường (nhiệt độ, độ ẩm, ánh sáng) trong thời gian thực và điều khiển từ xa các thiết bị (đèn, quạt, điều hòa) thông qua một giao diện dashboard trực quan. Về mặt tính năng, hệ thống cung cấp khả năng hiển thị dữ liệu thời gian thực qua biểu đồ, điều khiển thiết bị bằng nút gạt, lưu trữ chi tiết Lịch sử Dữ liệu Cảm biến và Nhật ký Hoạt động để phục vụ việc phân tích và giám sát; đồng thời, dự án có khả năng mở rộng để tích hợp các tính năng cao cấp như gửi thông báo đẩy khi có sự kiện bất thường và ước tính lượng điện năng tiêu thụ. Về mặt công nghệ, kiến trúc dự án được xây dựng vững chắc với Node.js/Express.js (backend), React.js (frontend) cùng với Socket.IO và giao thức MQTT để đảm bảo việc truyền dữ liệu tức thời, tối ưu hóa lưu trữ bằng cách sử dụng MySQL/PostgreSQL cho dữ liệu cấu trúc và InfluxDB cho dữ liệu chuỗi thời gian. Cốt lõi phần cứng là vi điều khiển ESP32 (ESP WROOM32), một module tích hợp Wi-Fi và Bluetooth, kết hợp với các cảm biến phổ biến như DHT11 (Nhiệt độ/Độ ẩm) và Cảm biến Quang trở (Ánh sáng), cho phép thu thập dữ liệu và điều

hiển thiết bị ngoại vi thông qua module Relay. Tóm lại, dự án này đã thiết lập một nền tảng kỹ thuật vững chắc, linh hoạt và có khả năng mở rộng cao, đáp ứng đầy đủ mục tiêu về tiện nghi, tiết kiệm năng lượng và nâng cao chất lượng cuộc sống.

2. Tổng quan giao diện

2.1. Trang Dashboard

Giao diện Dashboard đóng vai trò là trung tâm điều khiển của toàn bộ hệ thống, được thiết kế hiện đại và trực quan để người dùng có thể giám sát và tương tác một cách hiệu quả.

Giao diện bao gồm:

- Thông số thời gian thực: Ba khối thông tin được đặt nổi bật ở phía trên cùng, hiển thị các dữ liệu môi trường quan trọng nhất được cập nhật liên tục. Mỗi khối hiển thị một thông số riêng biệt là nhiệt độ, độ ẩm và ánh sáng, đi kèm với trạng thái đánh giá và biểu tượng trực quan để người dùng nhận biết nhanh.
- Biểu đồ giám sát: Nằm ở khu vực trung tâm, gồm hai biểu đồ đường trực quan hóa xu hướng dữ liệu theo thời gian. Biểu đồ đầu tiên là sự kết hợp giữa nhiệt độ và độ ẩm, cho phép người dùng so sánh mối tương quan giữa hai chỉ số này. Biểu đồ thứ hai dành riêng cho việc theo dõi sự biến thiên của cường độ ánh sáng trong ngày.
- Điều khiển thiết bị: Khu vực phía dưới cùng cho phép người dùng quản lý trực tiếp các thiết bị thông minh như Quạt, Điều hòa và Đèn. Mỗi thiết bị đều có biểu tượng nhận diện và một công tắc để người dùng có thể bật/tắt từ xa một cách thuận tiện.

2.2. Trang Data Sensor

Giao diện Data Sensor được thiết kế chuyên biệt cho việc tra cứu và phân tích dữ liệu cảm biến chi tiết, giao diện bao gồm:

- Thanh công cụ: Nằm ở phía trên cùng, bao gồm một ô tìm kiếm để người dùng có thể nhập từ khóa tra cứu, cùng với các nút chức năng để lọc và thực hiện tìm kiếm.
- Bảng dữ liệu cảm biến: Là thành phần trung tâm, hiển thị dữ liệu được ghi lại một cách có hệ thống. Các cột trong bảng bao gồm mã định danh của bản ghi, các giá trị nhiệt độ, độ ẩm, ánh sáng và mốc thời gian chính xác khi dữ liệu được thu thập.

- Phân trang và thông tin: Nằm ở cuối bảng, khu vực này cung cấp thông tin về số lượng kết quả đang hiển thị và các nút điều hướng trang, giúp người dùng dễ dàng duyệt qua một khối lượng lớn dữ liệu.

2.3. Trang Action History

Giao diện Action History được thiết kế để cung cấp một nhật ký đầy đủ và minh bạch về tất cả các hành động điều khiển thiết bị đã được thực hiện trong hệ thống, giao diện bao gồm hai phần chính:

- Thanh công cụ lọc: Nằm ở phía trên, cho phép người dùng nhanh chóng lọc và tìm kiếm lịch sử. Công cụ bao gồm một menu thả xuống để chọn xem lịch sử của một thiết bị cụ thể, một ô tìm kiếm và một nút để thực thi lệnh tìm.
- Bảng lịch sử hành động: Hiển thị danh sách các hành động đã được ghi lại, được sắp xếp theo thời gian giảm dần. Các cột thông tin bao gồm mã định danh, tên thiết bị đã được tác động, hành động đã thực hiện như bật hoặc tắt, và thời gian chính xác khi hành động diễn ra. Trạng thái hành động được thể hiện bằng màu sắc để dễ dàng phân biệt.

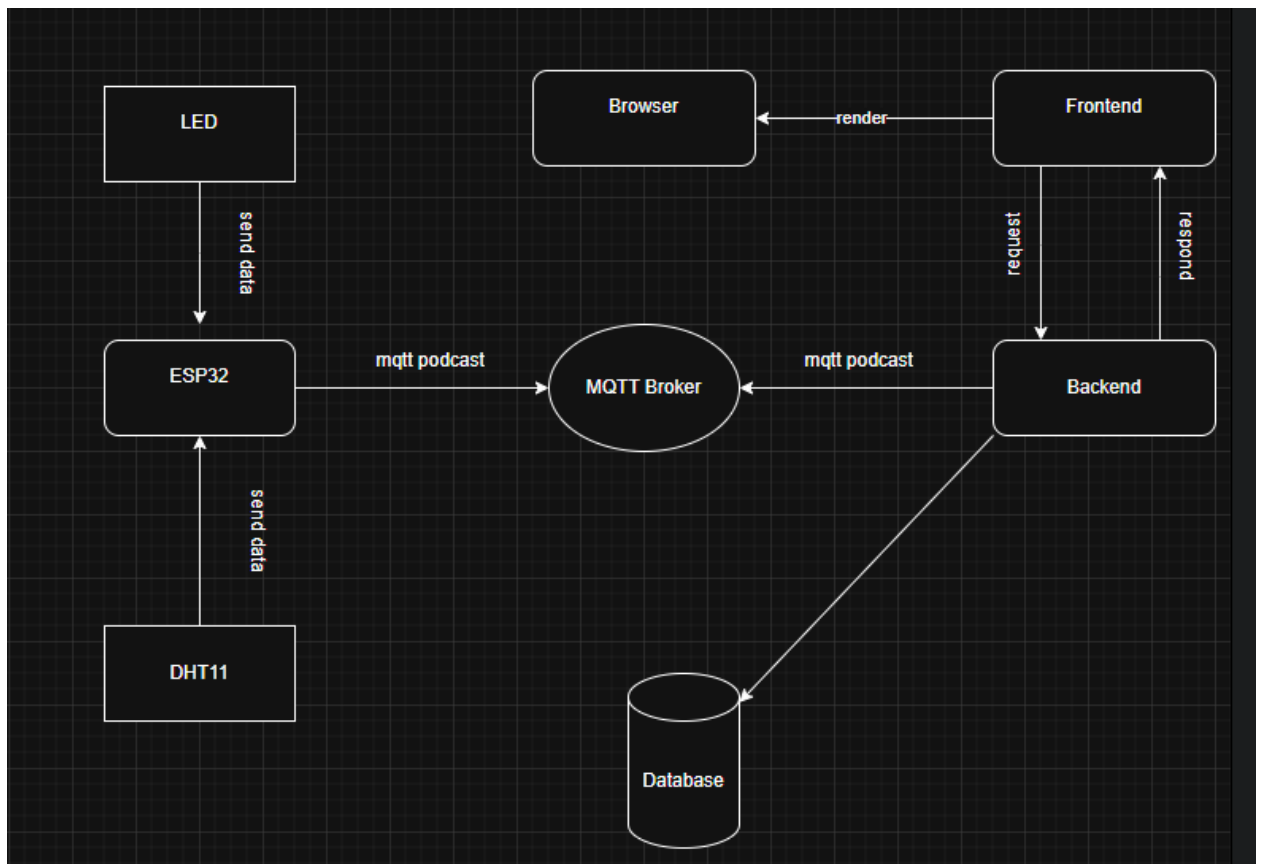
2.4. Trang Profile Settings

Bao gồm các thành phần sau:

- Thông tin định danh: Phía trên cùng của thẻ là một biểu tượng avatar người dùng, ngay dưới đó là tên đầy đủ và vai trò của họ trong hệ thống, giúp xác định nhanh chóng danh tính người dùng.
- Thông tin chi tiết: Phần thân thẻ liệt kê các thông tin liên hệ và cá nhân quan trọng như email, số điện thoại, tài khoản GitHub và vị trí. Mỗi mục thông tin đều đi kèm với một biểu tượng trực quan tương ứng để tăng tính dễ đọc.

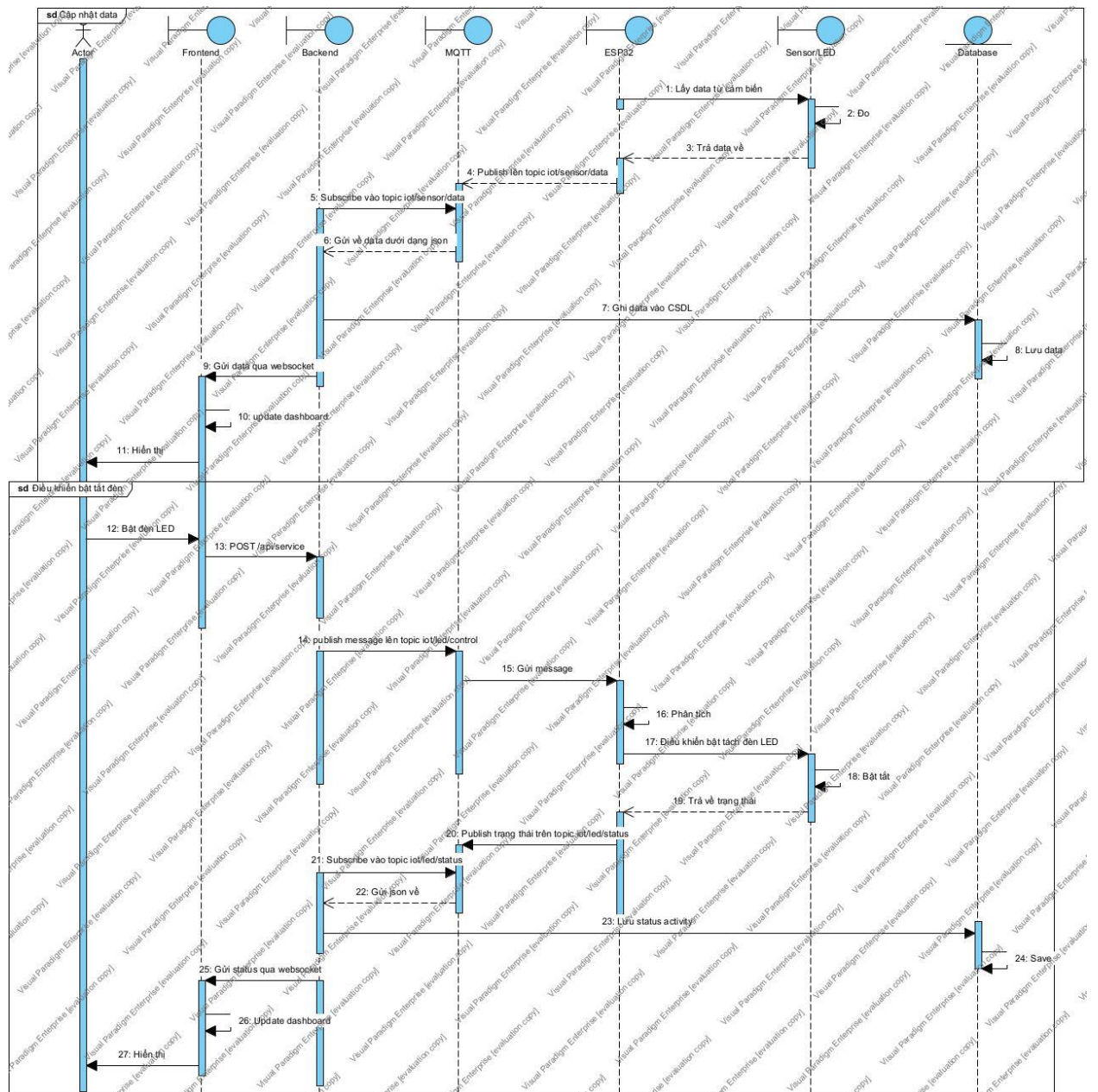
3. Thiết kế chi tiết

3.1 Sơ đồ kiến trúc hệ thống



Hình 1: Sơ đồ luồng

3.1 Biểu đồ tuần tự



- Luồng 1: Đồng bộ dữ liệu từ thiết bị

1. Hardware gửi dữ liệu gốc đến MQTT.

2. MQTT gửi dữ liệu này lên hệ thống Backend.
3. Backend đăng ký một chủ đề liên quan đến dữ liệu với Data Server.
4. Backend yêu cầu Data Server lưu trữ dữ liệu vừa nhận.
5. Data Server thực hiện xong và gửi tín hiệu phản hồi lại cho Backend.
6. Backend trả kết quả về cho MQTT.

- Luồng 2: Lấy dữ liệu cảm biến hiển thị lên web

7. User thực hiện hành động trên MQTT.
8. MQTT gửi yêu cầu lấy dữ liệu (get/use) tới Backend.
9. Backend gọi API để lấy dữ liệu cảm biến từ Data Server.
10. Data Server trả về dữ liệu cảm biến cho Backend.
11. Backend trả dữ liệu về cho MQTT.
12. MQTT xử lý dữ liệu vừa nhận.
13. MQTT hiển thị dữ liệu lên giao diện cho User.

- Luồng 3: Điều khiển bật đèn

14. User nhấn nút bật đèn trên MQTT.
15. MQTT gọi API điều khiển bật/tắt đèn tới Backend.
16. Backend gửi yêu cầu điều khiển API tới Action Server.
17. Action Server xác nhận đã nhận lệnh và phản hồi lại cho Backend.
18. Backend trả kết quả về cho MQTT.
19. MQTT hiển thị thông báo đã bật đèn cho User.

- Luồng 4: Phản hồi từ thiết bị và cập nhật trạng thái

20. Backend gửi lệnh điều khiển tới Hardware.
21. Hardware đăng ký để lắng nghe lệnh từ Backend.
22. Hardware thực hiện hành động vật lý là bật/tắt đèn.
23. Hardware xuất bản trạng thái mới của mình lên Backend.
24. Backend gửi dữ liệu trạng thái mới này đến MQTT.
25. MQTT gửi phản hồi xác nhận đã nhận cho Backend.
26. MQTT hiển thị trạng thái mới nhất của thiết bị cho User.

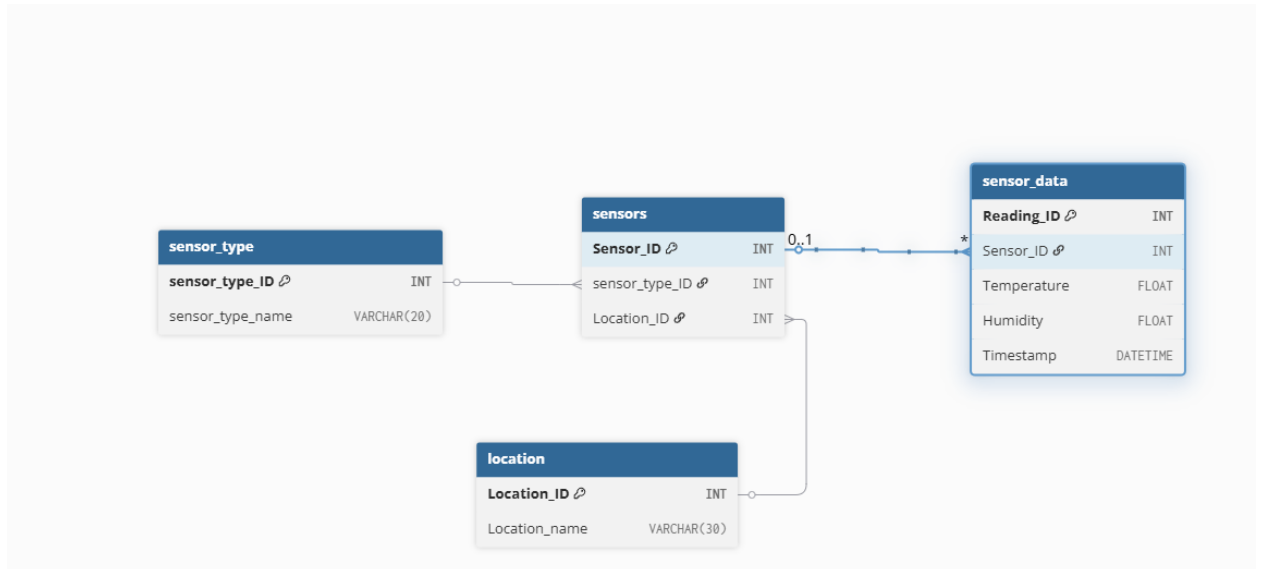
- Luồng 5: Xem lịch sử hoạt động

27. User mở mục Lịch sử trên MQTT.
28. MQTT gọi API lấy lịch sử từ Backend.
29. Backend gửi yêu cầu truy vấn lịch sử tới Data Server.
30. Data Server trả về dữ liệu lịch sử cho Backend.
31. Backend trả kết quả lịch sử về cho MQTT.
32. MQTT xử lý dữ liệu lịch sử nhận được.
33. MQTT hiển thị danh sách lịch sử cho User.

- Luồng 6: Xem dữ liệu chi tiết

34. User chọn mở một mục dữ liệu cụ thể trên MQTT.
35. MQTT gọi API để lấy dữ liệu chi tiết từ Backend.
36. Backend tiếp tục gọi API để lấy dữ liệu cảm biến chi tiết từ Data Server.
37. Data Server trả về dữ liệu được yêu cầu cho Backend.
38. Backend trả dữ liệu chi tiết về cho MQTT.
39. MQTT xử lý dữ liệu chi tiết.

40. MQTT hiển thị thông tin chi tiết này cho User.



- Như được thể hiện trong sơ đồ Entity Relationship ở trên, các mối quan hệ là như sau:

- Một - Nhiều (1 : N) Nhiều Sensor có thể thuộc về một SensorType.
- Một - Nhiều (1 : N) Nhiều Reading có thể được lấy từ một Sensor.
- Một - Nhiều (1 : N) Một Location có thể chứa nhiều Sensor.

3.3 Database

id	device	action	created_at
1	FAN	ON	2025-10-10 09:19:13
2	AIR_CONDITIONER	ON	2025-10-10 09:19:15
3	FAN	ON	2025-10-10 09:21:49
4	AIR_CONDITIONER	ON	2025-10-10 09:21:50
5	ALL_DEVICES	OFF	2025-10-10 09:35:52
6	ALL_DEVICES	ON	2025-10-10 09:45:23
7	ALL_DEVICES	OFF	2025-10-10 09:45:25
8	FAN	ON	2025-10-10 09:49:18
9	FAN	ON	2025-10-10 09:58:23
10	FAN	ON	2025-10-10 09:58:24
11	AIR_CONDITIONER	ON	2025-10-10 09:58:25
12	LED	ON	2025-10-10 09:58:27
13	LED	ON	2025-10-10 09:58:28
14	ALL_DEVICES	ON	2025-10-10 09:58:31
15	ALL_DEVICES	OFF	2025-10-10 09:58:33
16	FAN	ON	2025-10-10 09:59:31
17	AIR_CONDITIONER	ON	2025-10-10 09:59:34
18	FAN	ON	2025-10-10 10:01:40
19	FAN	ON	2025-10-10 10:01:42
20	LED	ON	2025-10-10 10:01:43
21	ALL_DEVICES	OFF	2025-10-10 10:01:45
22	FAN	ON	2025-10-10 10:02:09
23	FAN	ON	2025-10-10 10:02:11

Figure 5: Bảng Action Logs

action_logs: Ghi lại lịch sử hoạt động của các thiết bị, chẳng hạn như hành động bật hoặc tắt được thực hiện vào thời gian nào.

	id	temperature	humidity	light	created_at
▶	1	28.50	85.00	2940	2025-10-10 00:13:10
	2	28.50	85.00	2951	2025-10-10 00:13:12
	3	28.50	85.00	2758	2025-10-10 00:13:14
	4	28.50	85.00	2963	2025-10-10 00:13:37
	5	28.50	85.00	2956	2025-10-10 00:13:37
	6	28.50	85.00	2919	2025-10-10 00:13:37
	7	28.50	85.00	2909	2025-10-10 00:13:37
	8	28.60	85.00	2928	2025-10-10 00:13:37
	9	28.50	85.00	2873	2025-10-10 00:13:37
	10	28.50	85.00	2992	2025-10-10 00:13:37
	11	28.50	85.00	3003	2025-10-10 00:13:37
	12	28.50	85.00	2992	2025-10-10 00:13:37
	13	28.50	85.00	2994	2025-10-10 00:13:37
	14	28.50	85.00	3008	2025-10-10 00:13:38
	15	28.50	85.00	3023	2025-10-10 00:13:38
	16	28.50	85.00	3017	2025-10-10 00:13:40
	17	28.50	85.00	3017	2025-10-10 00:13:42
	18	28.50	85.00	2986	2025-10-10 00:13:44
	19	28.50	85.00	3037	2025-10-10 00:13:46
	20	28.50	85.00	3039	2025-10-10 00:13:48
	21	28.50	85.00	3061	2025-10-10 00:13:50
	22	28.50	85.00	3056	2025-10-10 00:13:52
	23	28.50	85.00	3061	2025-10-10 00:13:54
	24	28.50	85.00	2897	2025-10-10 00:13:56
	25	28.50	85.00	3067	2025-10-10 00:13:58
	26	28.50	85.00	3069	2025-10-10 00:14:00
	27	28.50	85.00	3087	2025-10-10 00:14:02
	28	28.50	85.00	3095	2025-10-10 00:14:04
	29	28.50	85.00	3094	2025-10-10 00:14:06
	30	28.50	85.00	3111	2025-10-10 00:14:08

Figure 6: Bảng Action History

sensor_readings: Chứa dữ liệu lịch sử từ các cảm biến, bao gồm nhiệt độ, độ ẩm và ánh sáng được ghi lại theo thời gian

4. Code

4.1 APIs routing

```
app.use('/api/data', dataRoutes(db, currentLedStatus));
app.use('/api/command', commandRoutes(db, client, COMMAND_TOPIC,
() => isEsp32DataConnected));
app.use('/api/actions', actionRoutes(db));
app.use('/api', statusRoutes(() => isMqttConnected, () =>
isEsp32DataConnected));
```

4.2 API dữ liệu cảm biến /api/data

```
import express from 'express';

const router = express.Router();

export default (db, currentLedStatus) => {
  router.get('/', async (req, res) => {
    try {
      const [rows] = await db.query('SELECT * FROM
sensor_readings ORDER BY created_at 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' });
    }
  });

  router.get('/history', async (req, res) => {
    try {
      const page = parseInt(req.query.page) || 1;
      const limit = 10;
      const offset = (page - 1) * limit;
      const search = req.query.search || '';

      let searchClause = '';
      let values = [];

      if (search) {
        searchClause = 'WHERE created_at LIKE ?';
        values.push(`%${search}%`);
      }

      const countSql = `SELECT COUNT(*) AS totalItems FROM
sensor_readings ${searchClause}`;
      const [[{ totalItems }]] = await db.query(countSql,
values);

      const totalPages = Math.ceil(totalItems / limit);

      const dataSql = `
        SELECT * FROM sensor_readings
        ${searchClause}
        ORDER BY created_at DESC
        LIMIT ? OFFSET ?
      `;

      values.push(limit, offset);
      const [data] = await db.query(dataSql, values);

      res.json({
        totalItems,
```

```

        totalPages,
        currentPage: page,
        data,
    });
} catch (error) {
    console.error('Error fetching history:', error.message);
    res.status(500).json({ error: 'Failed to fetch sensor
history' });
}
});

return router;});

```

4.3 API lịch sử hành động /api/actions/history

```

import express from 'express';

const router = express.Router();

export default (db) => {
    router.get('/history', async (req, res) => {
        try {
            const [actions] = await db.query('SELECT * FROM
action_logs ORDER BY created_at DESC');
            res.json(actions);
        } catch (error) {
            console.error('Error fetching action history:',
error.message);
            res.status(500).json({ error: 'Failed to fetch action
history' });
        }
    });

    return router;
};

```

4.4 API điều khiển thiết bị /api/command

```

import express from 'express';

const router = express.Router();

export default (db, client, COMMAND_TOPIC, isEsp32DataConnected) =>
{
    router.post('/', async (req, res) => {
        const { command } = req.body;

        if (!command) {
            return res.status(400).json({ error: 'Command is
required' });
        }

        if (!isEsp32DataConnected) {
            return res.status(503).json({ error: 'Device is
disconnected. Cannot send command.' });
        }
    });
}

```

```

    }

    let device = null;
    let action = null;

    if (command === 'allon' || command === 'alloff') {
        device = 'ALL DEVICES';
        action = command === 'allon' ? 'ON' : 'OFF';
    } else {
        const led = command.slice(0, 4);
        const state = command.slice(4).toUpperCase();

        device = {
            led1: 'FAN',
            led2: 'AIR_CONDITIONER',
            led3: 'LED',
        }[led];

        action = state;
    }

    if (device && action) {
        try {
            const sql = 'INSERT INTO action_logs (device,
action) VALUES (?, ?)';
            await db.query(sql, [device, action]);
            console.log(`Action logged: ${device} ->
${action}`);
        } catch (error) {
            console.error('Failed to log action:',
error.message);
        }
    }

    client.publish(COMMAND_TOPIC, command, (err) => {
        if (err) {
            console.error('MQTT publish error:', err.message);
            return res.status(500).json({ error: 'Failed to send
command (MQTT Broker issue)' });
        }

        console.log(`Sent command via MQTT: ${command}`);
        return res.status(200).send('Command sent
successfully');
    });
});

return router;
};

```

4.5 API cung cấp thông tin trạng thái kết nối /api/mqtt-status

```

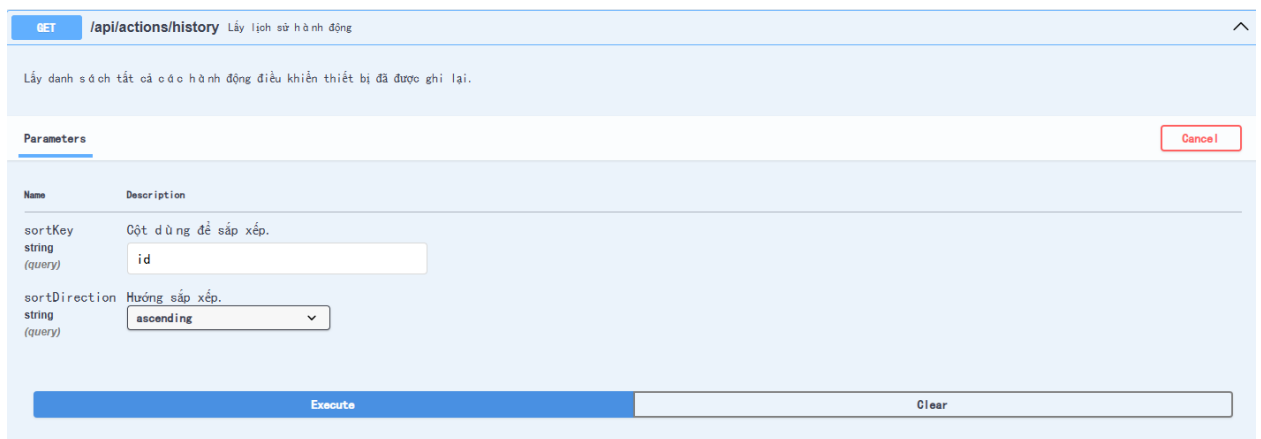
import express from 'express';

const router = express.Router();

```

```
export default (isMqttConnected, isEsp32DataConnected) => {
  router.get('/mqtt-status', (req, res) => {
    res.json({ isConnected: isMqttConnected,
isEsp32DataConnected: isEsp32DataConnected });
  });
  return router;
}
```

5. Swagger



GET /api/actions/history Lấy lịch sử hành động

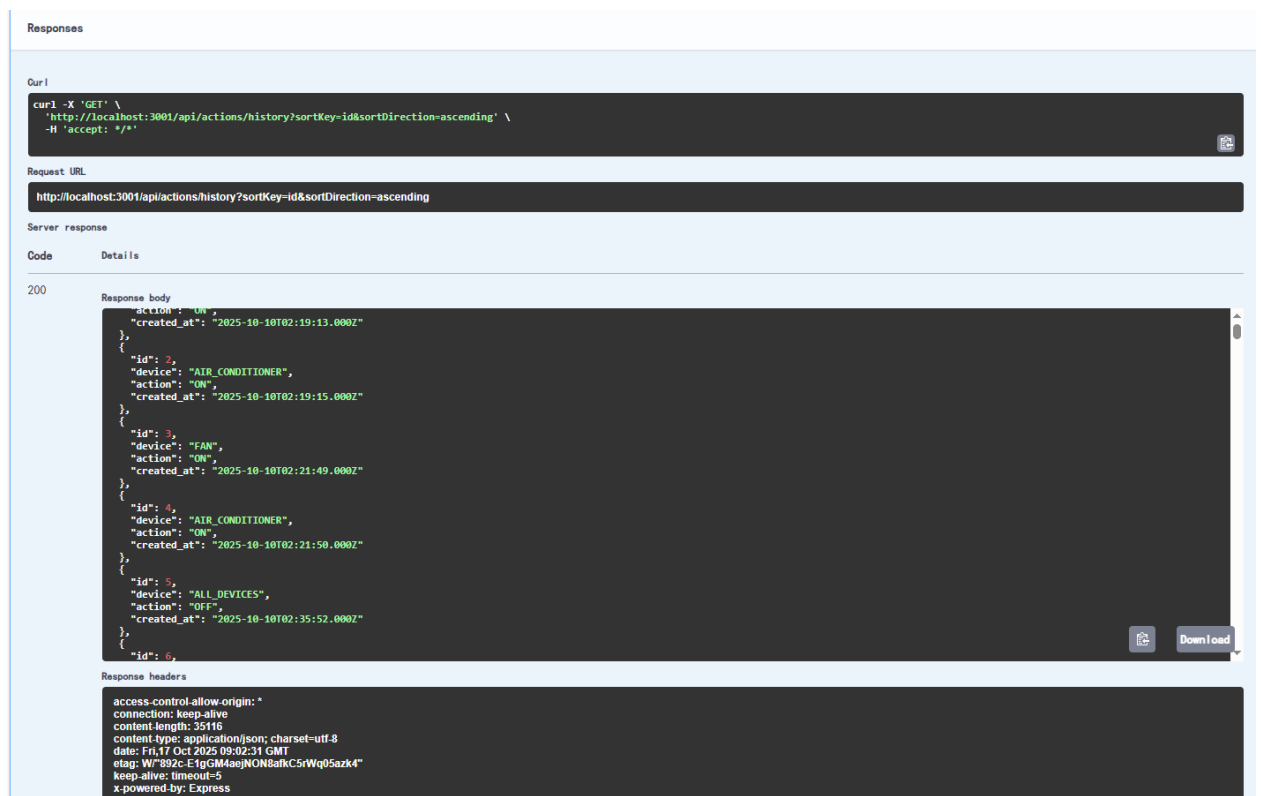
Lấy danh sách tất cả các hành động điều khiển thiết bị đã được ghi lại.

Parameters

Name	Description
sortKey string (query)	Cột dùng để sắp xếp. <input type="text" value="id"/>
sortDirection string (query)	Hướng sắp xếp. <input type="text" value="ascending"/>

Execute Clear

Figure 7 /api/actions/history:



Responses

Curl

```
curl -X 'GET' \
  'http://localhost:3001/api/actions/history?sortKey=id&sortDirection=ascending' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:3001/api/actions/history?sortKey=id&sortDirection=ascending
```

Server response

Code Details

200

Response body

```
{
  "action": "ON",
  "created_at": "2025-10-10T02:19:13.000Z"
},
{
  "id": 2,
  "device": "AIR_CONDITIONER",
  "action": "ON",
  "created_at": "2025-10-10T02:19:15.000Z"
},
{
  "id": 3,
  "device": "FAN",
  "action": "ON",
  "created_at": "2025-10-10T02:21:49.000Z"
},
{
  "id": 4,
  "device": "AIR_CONDITIONER",
  "action": "ON",
  "created_at": "2025-10-10T02:21:50.000Z"
},
{
  "id": 5,
  "device": "ALL_DEVICES",
  "action": "OFF",
  "created_at": "2025-10-10T02:35:52.000Z"
},
{
  "id": 6,

```

Response headers

```
access-control-allow-origin: *
connection: keep-alive
content-length: 35116
content-type: application/json; charset=utf-8
date: Fri, 17 Oct 2025 09:02:31 GMT
etag: W/"892c-E1gGM4aejNON8afkC5rWq05azk4"
keep-alive: timeout=5
x-powered-by: Express
```

Figure 8: response /api/actions/history

POST /api/command Gửi lệnh điều khiển thiết bị

Gửi một lệnh tới thiết bị thông qua MQTT.

Parameters Cancel Reset

No parameters

Request body required application/json

Edit Value | Schema

```
{
  "command": "led1off"
}
```

Execute Clear

Figure 9: /api/command

Responses

Url

```
curl -X 'POST' \
  'http://localhost:3001/api/command' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "command": "led1off"
  }'
```

Request URL

http://localhost:3001/api/command

Server response

Code	Details
200	<p>Response body</p> <p>Command sent successfully</p> <p>Download</p> <p>Response headers</p> <pre>access-control-allow-origin: * connection: keep-alive content-length: 25 content-type: text/html; charset=utf-8 date: Fri, 17 Oct 2025 09:11:16 GMT etag: W/"19-C4Xab6Kz2wKWBLcfJeh5VtJS4" keep-alive: timeout=5 x-powered-by: Express</pre>

Figure 10: response /api/command

GET /api/data Lấy dữ liệu cảm biến mới nhất

Truy xuất thông số cảm biến gần nhất và trạng thái hiện tại của các đèn LED.

Parameters Cancel

No parameters

Execute Clear

Figure 11: /api/data

Responses

Curl

```
curl -X 'GET' \
'http://localhost:3001/api/data' \
-H 'accept: */*'
```

Request URL

http://localhost:3001/api/data

Server response

Code Details

200

Response body

```
{
  "sensors": {
    "id": 8701,
    "temperature": "24.10",
    "humidity": "48.00",
    "light": 376,
    "created_at": "2025-10-17T09:12:49.000Z"
  },
  "leds": {
    "led1": "off",
    "led2": "off",
    "led3": "off"
  }
}
```

Response headers

```
access-control-allow-origin: *
connection: keep-alive
content-length: 164
content-type: application/json; charset=utf-8
date: Fri, 17 Oct 2025 09:12:50 GMT
etag: W/"a4-y8-d7x8GbzOopNbojTVIaxDtyNU"
keep-alive: timeout=5
x-powered-by: Express
```

Figure 12: response /api/data

GET /api/data/history Lấy dữ liệu cảm biến theo trang

Lấy danh sách dữ liệu cảm biến trong quá khứ, có hỗ trợ phân trang, sắp xếp và tìm kiếm.

Parameters

Cancel

Name	Description
page integer (query)	Số trang để phân trang. <input type="text" value="1"/>
search string (query)	Từ khóa tìm kiếm. <input type="text" value="17"/>
sortKey string (query)	Cột dùng để sắp xếp. <input type="text" value="id"/>
sortDirection string (query)	Hướng sắp xếp. <input type="text" value="ascending"/>

Execute Clear

Figure 13: /api/data/history

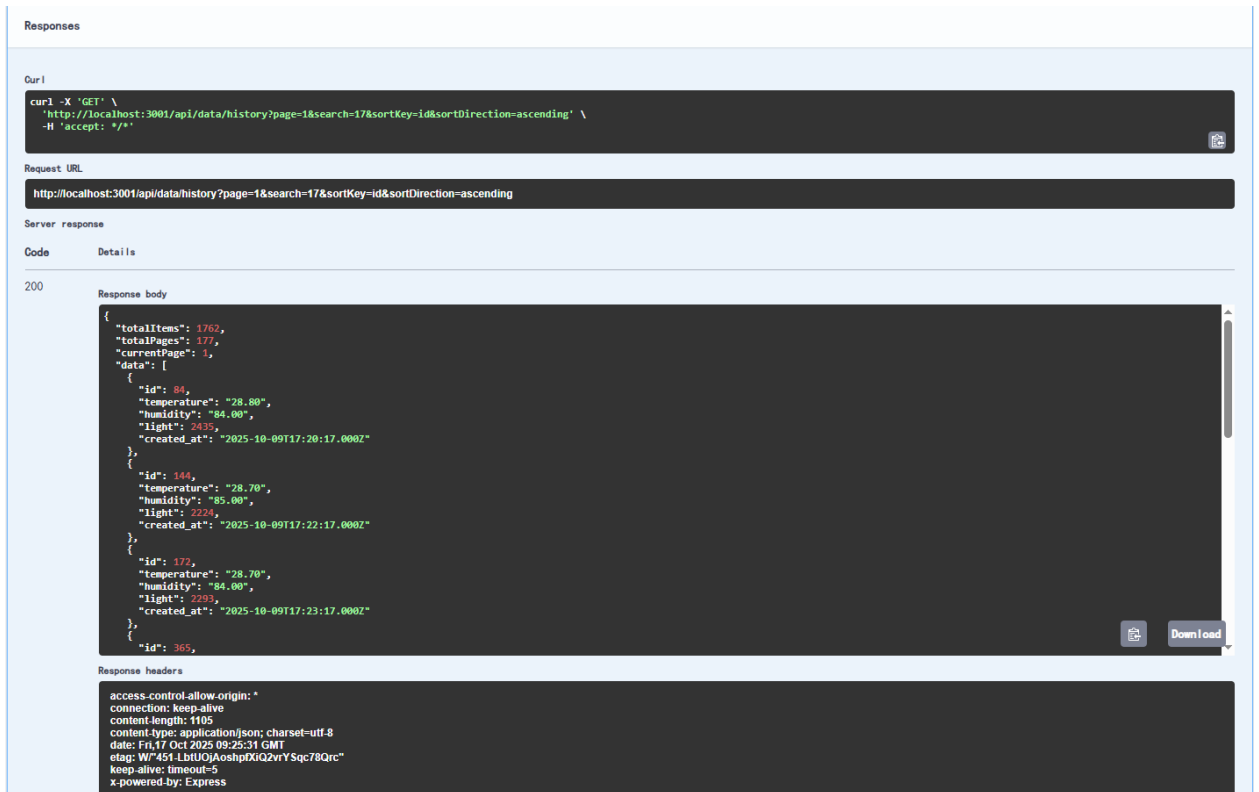


Figure 14: response /api/data/history

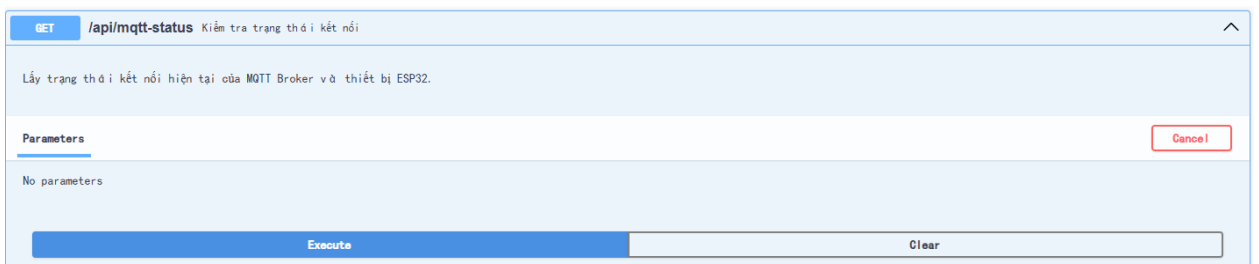


Figure 15: /api/mqtt-status

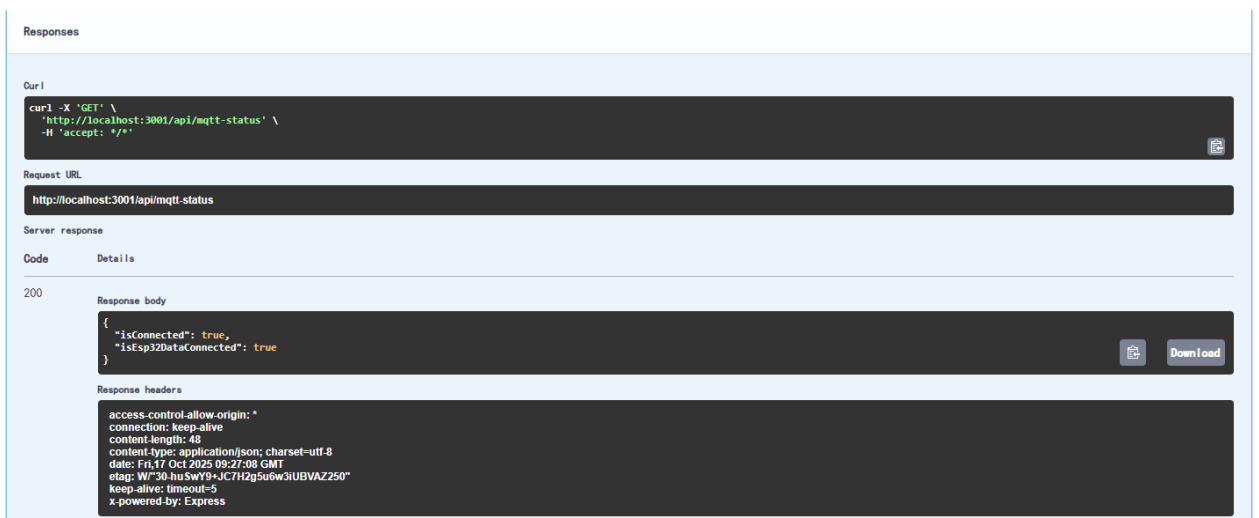


Figure 16: response /api/mqtt-status

6. Kết Quả

6.1 Kết quả triển khai hệ thống

Sau quá trình triển khai và hoàn thiện, hệ thống IoT Dashboard đã được đưa vào vận hành chính thức với đầy đủ các thành phần phần cứng và phần mềm theo thiết kế.

Mô hình hệ thống bao gồm các thành phần chính sau: Khối phần cứng sử dụng vi điều khiển ESP32 tích hợp với cảm biến nhiệt độ - độ ẩm DHT11, cảm biến ánh sáng LDR và ba module đèn LED đóng vai trò mô phỏng các thiết bị điện trong gia đình (quạt, đèn, điều hòa). Ở phía server, hệ thống được xây dựng trên nền tảng Backend Node.js kết hợp với MQTT Broker đảm nhận vai trò trung gian truyền tin, cơ sở dữ liệu MySQL phục vụ lưu trữ, và giao diện Frontend React cung cấp khả năng hiển thị dữ liệu thời gian thực.

Kết quả triển khai cho thấy ESP32 hoạt động ổn định, thực hiện truyền dữ liệu cảm biến lên server thông qua giao thức MQTT một cách liên tục và tin cậy. Toàn bộ dữ liệu thu thập được được lưu trữ tự động vào bảng `sensor_data` trong cơ sở dữ liệu MySQL và hiển thị trực quan trên Dashboard thông qua các biểu đồ động được cập nhật theo thời gian thực.

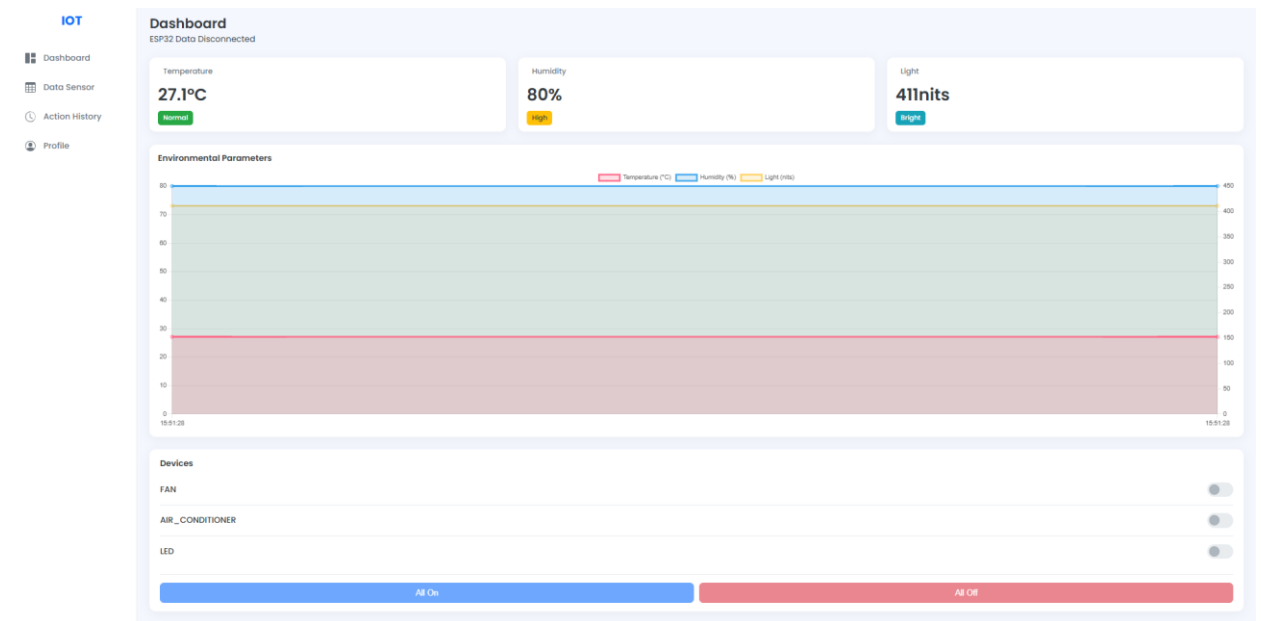
Quá trình điều khiển thiết bị được thực hiện thông qua cơ chế: khi người dùng gửi lệnh điều khiển (ví dụ: bật/tắt đèn), lệnh sẽ được chuyển đến Backend thông qua API, sau đó chuyển tiếp tới ESP32 qua kênh MQTT. Thiết bị phản hồi ngay lập tức và trạng thái mới được cập nhật chính xác trên giao diện web.

Hệ thống đạt được hiệu suất đáng tin cậy với độ trễ trung bình dưới 2 giây trong toàn bộ quy trình vận hành, đảm bảo khả năng phản hồi nhanh chóng và chính xác giữa các thành phần. Bên cạnh đó, mọi thao tác điều khiển đều được ghi nhận đầy đủ trong bảng `action_history`, cung cấp khả năng theo dõi, tra cứu và quản lý toàn diện cho người sử dụng.

Kết quả triển khai chứng minh hệ thống đã đáp ứng được các yêu cầu kỹ thuật đề ra, vận hành ổn định và mang lại hiệu quả thiết thực trong việc giám sát và điều khiển thiết bị từ xa.

6.2 Kết quả giao diện vận hành

6.2.1. Giao diện Dashboard



Giao diện Dashboard thể hiện hiệu suất ổn định trong việc hiển thị toàn bộ các thông số môi trường bao gồm nhiệt độ, độ ẩm và cường độ ánh sáng theo thời gian thực. Hệ thống biểu đồ đường (line chart) được tích hợp hoạt động mượt mà, đảm bảo cập nhật liên tục và đồng bộ với dữ liệu đầu vào. Các nút điều khiển thiết bị (On/Off) phản hồi nhanh chóng và chính xác, đồng bộ hóa trạng thái giữa giao diện người dùng và trạng thái thực tế của thiết bị.

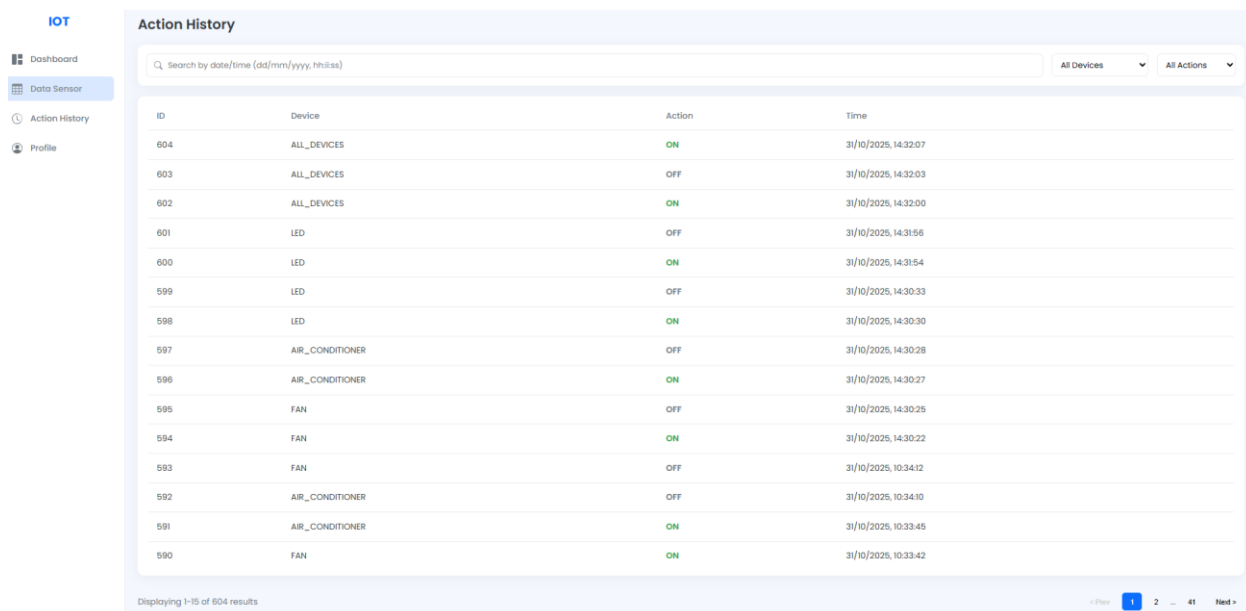
6.2.2. Giao diện Data Sensor

The screenshot shows the IOT Data Sensor interface. It features a search bar at the top with the placeholder 'Search by date/time (dd/mm/yyyy hh:mm:ss)' and dropdowns for 'Sort by Time' and 'Descending'. Below the search bar is a table with the following columns: ID, Temperature (°C), Humidity (%), Light (nits), and Time. The table contains 18 rows of data. At the bottom, there is a status bar that says 'Displaying 1-18 of 18907 results' and a pagination control showing 'Page 1' of '1128'.

ID	Temperature (°C)	Humidity (%)	Light (nits)	Time
16907	27.10	80.00	411	31/10/2025, 15:09:51
16906	27.10	80.00	405	31/10/2025, 15:09:49
16905	27.30	80.00	470	31/10/2025, 15:04:47
16904	27.30	80.00	533	31/10/2025, 15:04:45
16903	27.30	80.00	541	31/10/2025, 15:04:43
16902	27.30	80.00	453	31/10/2025, 15:04:41
16901	27.30	80.00	537	31/10/2025, 15:04:39
16900	27.30	80.00	547	31/10/2025, 15:04:37
16899	27.30	80.00	539	31/10/2025, 15:04:35
16898	27.30	80.00	439	31/10/2025, 15:04:33
16897	27.30	80.00	497	31/10/2025, 15:04:31
16896	27.30	80.00	469	31/10/2025, 15:04:29
16895	27.30	80.00	466	31/10/2025, 15:04:27
16894	27.30	80.00	528	31/10/2025, 15:04:25
16893	27.30	80.00	436	31/10/2025, 15:04:24

Giao diện hiển thị dữ liệu cảm biến được tổ chức khoa học dưới dạng bảng, cung cấp đầy đủ chức năng tìm kiếm và lọc dữ liệu theo các tiêu chí thời gian hoặc ID cụ thể. Dữ liệu hiển thị trên giao diện được xác nhận có sự tương đồng cao với giá trị thực tế đo được từ cảm biến, đảm bảo độ tin cậy trong việc theo dõi và phân tích.

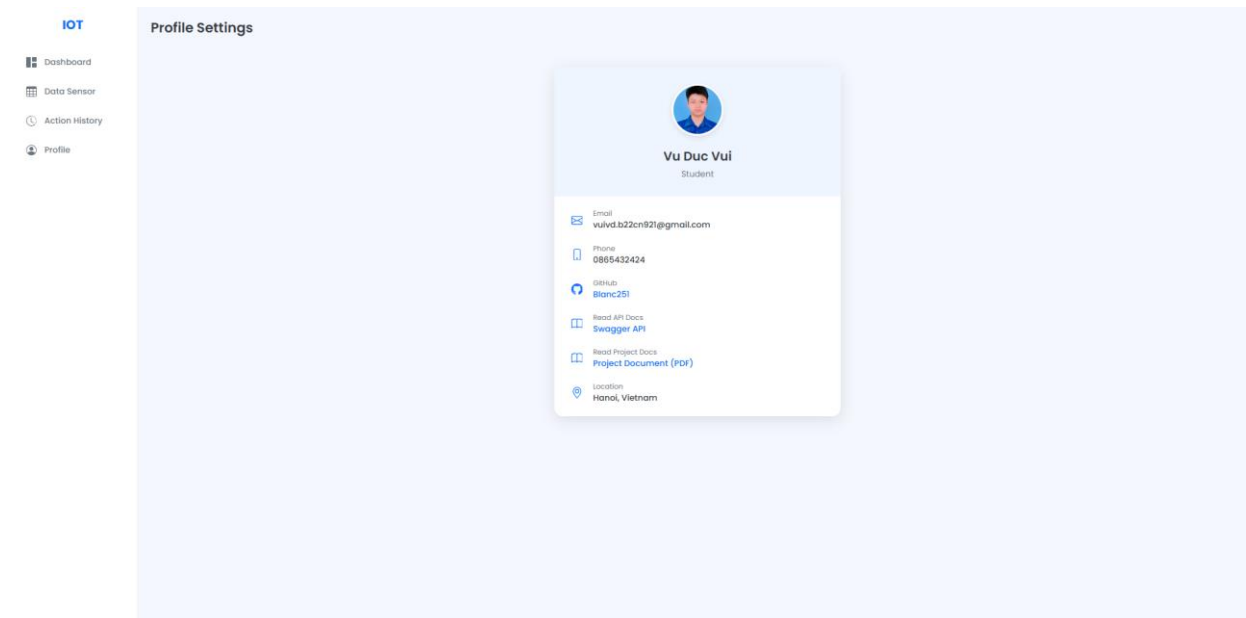
6.2.3. Giao diện Action History



ID	Device	Action	Time
604	ALL_DEVICES	ON	3/10/2025, 14:32:07
603	ALL_DEVICES	OFF	3/10/2025, 14:32:03
602	ALL_DEVICES	ON	3/10/2025, 14:32:00
601	LED	OFF	3/10/2025, 14:31:56
600	LED	ON	3/10/2025, 14:31:54
599	LED	OFF	3/10/2025, 14:30:33
598	LED	ON	3/10/2025, 14:30:30
597	AIR_CONDITIONER	OFF	3/10/2025, 14:30:28
596	AIR_CONDITIONER	ON	3/10/2025, 14:30:27
595	FAN	OFF	3/10/2025, 14:30:25
594	FAN	ON	3/10/2025, 14:30:22
593	FAN	OFF	3/10/2025, 10:34:12
592	AIR_CONDITIONER	OFF	3/10/2025, 10:34:10
591	AIR_CONDITIONER	ON	3/10/2025, 10:33:45
590	FAN	ON	3/10/2025, 10:33:42

Giao diện lịch sử thao tác hoạt động hiệu quả trong việc ghi nhận và hiển thị chính xác toàn bộ lịch sử bật/tắt thiết bị theo trình tự thời gian. Tính năng tìm kiếm được triển khai tối ưu, cho phép người dùng nhanh chóng truy xuất các thao tác trong quá khứ mà không gặp phải độ trễ đáng kể.

6.2.4. Giao diện Profile



Profile Settings

Vu Duc Vui
Student

Email: vuvd.b22cn921@gmail.com

Phone: 0865432424

GitHub: [Blanc251](#)

Read API Docs: [Swagger API](#)

Read Project Docs: [Project Document \(PDF\)](#)

Location: Hanoi, Vietnam

Giao diện cá nhân hoạt động ổn định, hiển thị chính xác thông tin của sinh viên thực hiện dự án. Tất cả các liên kết chức năng quan trọng bao gồm GitHub, Tài liệu API và Cập nhật Thông tin đều được vận hành trơn tru, đáp ứng đầy đủ các yêu cầu thiết kế ban đầu.

Kết luận: Toàn bộ hệ thống giao diện của IoT Dashboard đã đáp ứng được các tiêu chí kỹ thuật đề ra, thể hiện qua khả năng vận hành ổn định, tốc độ phản hồi nhanh và độ chính xác cao trong hiển thị dữ liệu và điều khiển thiết bị.

6.3 Đánh giá hiệu quả và độ tin cậy

Về tốc độ truyền dữ liệu: Hệ thống đạt được chỉ số ấn tượng với độ trễ trung bình dưới 1 giây cho toàn bộ quá trình truyền nhận dữ liệu. Điều này được thể hiện qua khả năng duy trì kết nối liên tục và ổn định giữa thiết bị phần cứng ESP32 và server, không ghi nhận bất kỳ sự cố gián đoạn nào trong suốt thời gian vận hành thử nghiệm.

Về giao diện người dùng: Giao diện được thiết kế tập trung vào tính trực quan và dễ sử dụng. Bố cục được sắp xếp hợp lý với các thành phần hiển thị rõ ràng, giúp người dùng - kể cả những người mới sử dụng lần đầu - có thể dễ dàng làm quen và thao tác mà không cần hướng dẫn phức tạp.

Về tính năng điều khiển: Các chức năng điều khiển thiết bị hoạt động hoàn toàn chính xác theo thiết kế. Mọi thao tác bật/tắt đều được phản hồi tức thì từ phía thiết bị, đồng thời trạng thái hiển thị trên giao diện luôn đồng bộ với trạng thái thực tế của thiết bị, đảm bảo độ tin cậy cao.

Về lưu trữ dữ liệu: Hệ thống quản lý lưu trữ dữ liệu một cách hiệu quả với độ chính xác tuyệt đối. Tính năng phân trang được tích hợp đầy đủ, cho phép người dùng dễ dàng truy xuất thông tin lịch sử và thực hiện các thao tác phân tích dữ liệu một cách thuận tiện.

Về bảo mật và xác thực: Cơ chế bảo mật được triển khai bài bản thông qua việc sử dụng JWT/Token, cung cấp lớp bảo vệ an toàn cho mọi truy cập hệ thống. Giải pháp này đảm bảo tính toàn vẹn và bảo mật cho dữ liệu trong suốt quá trình vận hành.

Đánh giá tổng quan: Hệ thống duy trì được mức độ ổn định cao trong mọi điều kiện vận hành. Việc ứng dụng giao thức MQTT đã chứng minh hiệu quả trong việc tối ưu hóa băng thông, trong khi cơ chế WebSocket đảm bảo kênh giao tiếp hai chiều giữa server và client luôn thông suốt. Kết quả thử nghiệm cho thấy hệ thống hoàn toàn đáp ứng được các yêu cầu kỹ thuật khắt khe, từ tốc độ phản hồi, độ tin cậy đến tính bảo mật, hứa hẹn một giải pháp IoT hoàn chỉnh cho các ứng dụng thực tế.

6.4 Hạn chế và hướng phát triển

1. Những hạn chế hiện tại

Hệ thống IoT Dashboard trong giai đoạn hiện tại vẫn tồn tại một số điểm cần được cải thiện. Thứ nhất, hệ thống mới chỉ được thử nghiệm ở quy mô nhỏ với cấu hình tối thiểu gồm một board ESP32 và một cụm cảm biến, điều này chưa phản ánh đầy đủ khả năng vận hành trong môi trường thực tế với số lượng thiết bị lớn. Thứ hai, hệ thống còn thiếu cơ chế cảnh báo tự động khi các thông số môi trường như nhiệt độ hoặc độ ẩm vượt ngưỡng an toàn, làm giảm tính chủ động trong giám sát. Cuối cùng, việc chưa tích hợp module quản

lý đa người dùng và phân quyền truy cập khiến hệ thống chưa phù hợp cho các ứng dụng yêu cầu chia sẻ quyền điều khiển và bảo mật ở nhiều cấp độ khác nhau.

2. Định hướng phát triển trong tương lai

Để khắc phục các hạn chế và nâng cao giá trị ứng dụng, hệ thống sẽ được phát triển theo các hướng trọng tâm sau:

Về mở rộng phạm vi giám sát, hệ thống sẽ được tích hợp thêm nhiều loại cảm biến đa dạng như cảm biến khí gas, cảm biến chuyển động và cảm biến độ ồn, cho phép giám sát toàn diện môi trường.

Về nâng cao tính tự động hóa, tính năng "Automation Rule" sẽ được nghiên cứu phát triển, cho phép hệ thống tự động kích hoạt các hành động dựa trên ngữ cảnh, ví dụ như tự động bật quạt khi nhiệt độ vượt quá 30°C, từ đó tối ưu hóa trải nghiệm và tiết kiệm năng lượng.

Về đa nền tảng, một ứng dụng di động trên cả hai hệ điều hành Android và iOS sẽ được xây dựng, mang lại cho người dùng khả năng giám sát và điều khiển hệ thống từ xa một cách linh hoạt và thuận tiện.

Về hạ tầng dữ liệu, hệ thống sẽ được kết nối với các nền tảng đám mây chuyên biệt cho IoT như Firebase, AWS IoT hoặc ThingSpeak. Việc này không chỉ giải quyết bài toán lưu trữ và xử lý dữ liệu ở quy mô lớn mà còn mở ra khả năng phân tích dữ liệu chuyên sâu.

Về trí tuệ hệ thống, định hướng dài hạn là tích hợp các thuật toán Trí tuệ nhân tạo (AI) và Học máy (Machine Learning). Các thuật toán này sẽ phân tích dữ liệu lịch sử để dự đoán xu hướng biến đổi của nhiệt độ, độ ẩm và từ đó đưa ra các đề xuất điều khiển tối ưu, biến hệ thống từ "thụ động" thành "chủ động" trong việc quản lý môi trường.

6.5 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.