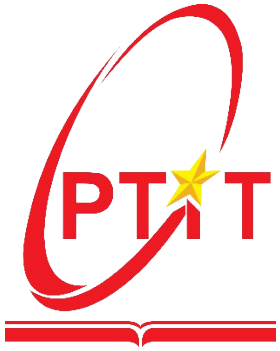
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG



BÁO CÁO BÀI TẬP LỚN
HỌC PHẦN IOT VÀ ỨNG DỤNG

ĐỀ TÀI:

XÂY DỰNG HỆ THỐNG CHĂN NUÔI GIA SÚC THÔNG MINH

Nhóm: 16

Sinh viên thực hiện:

Lương Tiến Đạt B22DCCN190

I – Giới thiệu:

Trong bối cảnh ngành chăn nuôi đang ngày càng chịu tác động từ biến đổi khí hậu, dịch bệnh và nhu cầu thị trường, việc đảm bảo môi trường sống ổn định và phù hợp cho vật nuôi trở thành một yêu cầu cấp thiết. Các yếu tố như nhiệt độ, độ ẩm và ánh sáng có ảnh hưởng trực tiếp đến sức khỏe, khả năng sinh trưởng và năng suất của đàn gia súc. Tuy nhiên, trong thực tế, phần lớn các hộ chăn nuôi vẫn quản lý những yếu tố này theo phương pháp thủ công, dẫn đến sự thiếu chính xác và khó duy trì tính ổn định trong thời gian dài. Điều này có thể làm tăng nguy cơ phát sinh dịch bệnh, ảnh hưởng đến chất lượng và hiệu quả chăn nuôi.

Để giải quyết vấn đề trên, dự án **"Xây dựng hệ thống chăn nuôi gia súc thông minh"** được thực hiện với mục tiêu ứng dụng công nghệ cảm biến và tự động hóa vào quản lý chuồng trại. Hệ thống được thiết kế để giám sát liên tục các thông số môi trường như nhiệt độ, độ ẩm và ánh sáng. Các cảm biến sẽ truyền dữ liệu về bộ xử lý trung tâm, từ đó đưa ra các quyết định điều khiển phù hợp, chẳng hạn như:

- Bật hoặc tắt đèn khi mức ánh sáng không đạt yêu cầu, giúp vật nuôi có điều kiện sinh hoạt và nghỉ ngơi hợp lý.
- Kích hoạt hệ thống quạt hoặc phun sương khi nhiệt độ, độ ẩm vượt ngưỡng cho phép, tạo môi trường thoáng mát và ổn định.
- Lưu trữ dữ liệu trong cơ sở dữ liệu để phục vụ cho việc thống kê, phân tích xu hướng và tối ưu quá trình chăn nuôi.

Nhờ vậy, người chăn nuôi có thể giảm bớt khối lượng công việc thủ công, tiết kiệm chi phí vận hành, đồng thời đảm bảo môi trường chuồng trại luôn trong trạng thái tối ưu cho sự phát triển của đàn gia súc.

I.1 – Mục tiêu :

- Xây dựng một hệ thống giám sát và điều khiển môi trường chuồng trại hoạt động theo thời gian thực, dựa trên dữ liệu thu được từ cảm biến.
- Duy trì ổn định các yếu tố nhiệt độ, độ ẩm và ánh sáng trong phạm vi tối ưu nhằm nâng cao sức khỏe, hạn chế bệnh tật và cải thiện tốc độ sinh trưởng của gia súc.
- Tự động hóa các quy trình cơ bản (chiếu sáng, thông gió, làm mát) để giảm thiểu sự phụ thuộc vào sức lao động thủ công, đồng thời tăng tính chính xác và ổn định.
- Tạo nền tảng để quản lý dữ liệu chăn nuôi một cách khoa học, hỗ trợ phân tích, so sánh và đưa ra quyết định cho các giai đoạn sản xuất tiếp theo.

I.2 – Phạm vi :

- **Đối tượng áp dụng**: Hệ thống được xây dựng hướng tới chuồng trại chăn nuôi gia súc có quy mô vừa và nhỏ, phù hợp với điều kiện của nhiều hộ gia đình và trang trại.
- **Chức năng chính**: Thu thập dữ liệu từ cảm biến ánh sáng, nhiệt độ, độ ẩm; điều khiển thiết bị chiếu sáng, quạt thông gió và hệ thống phun sương theo ngưỡng đã thiết lập.
- **Giới hạn**: Hệ thống bước đầu chưa bao gồm các chức năng phức tạp như tự động cho ăn, phát hiện dịch bệnh, theo dõi trọng lượng hoặc phân tích hành vi vật nuôi. Những tính năng này có thể được mở rộng trong giai đoạn phát triển sau.
- **Công nghệ sử dụng**: Kết hợp phần cứng (cảm biến, vi điều khiển, thiết bị điện) với phần mềm điều khiển, có thể tích hợp giao diện giám sát trên máy tính hoặc thiết bị di động.

II – Thiết kế hệ thống:

II.1 – Phần mềm :

Để xây dựng hệ thống chăn nuôi gia súc thông minh, em lựa chọn kết hợp nhiều công nghệ ở cả phía phần cứng và phần mềm. Trong đó, trọng tâm của phần mềm là các thành phần: **Node.js, HTML, TailwindCSS và MQTT Cloud**.

1. Node.js:

Node.js là môi trường chạy JavaScript phía máy chủ. Khác với JavaScript vốn chỉ được dùng trong trình duyệt, Node.js cho phép sử dụng cùng một ngôn ngữ để phát triển ứng dụng phía server.

Trong dự án, Node.js đảm nhiệm vai trò **backend**, bao gồm các chức năng:

- **Xử lý dữ liệu**: nhận dữ liệu từ cảm biến thông qua MQTT Cloud, sau đó lọc, chuẩn hóa và lưu trữ.
- **Điều khiển thiết bị**: dựa trên logic định sẵn (ví dụ nhiệt độ > 30°C thì bật quạt), Node.js gửi lệnh trở lại MQTT Cloud để điều khiển phần cứng.
- **Xây dựng API**: cung cấp giao diện lập trình ứng dụng (REST API hoặc WebSocket) để web frontend lấy dữ liệu cảm biến và trạng thái thiết bị theo thời gian thực.
- **Quản lý kết nối**: do MQTT và các yêu cầu HTTP có thể xảy ra đồng thời, Node.js tận dụng cơ chế bất đồng bộ để xử lý nhiều kết nối mà không gây nghẽn.

Nhờ Node.js, hệ thống có thể duy trì hoạt động ổn định và linh hoạt khi lượng dữ liệu cảm biến hoặc số lượng yêu cầu từ người dùng tăng lên.

2. HTML

HTML (HyperText Markup Language) là ngôn ngữ đánh dấu cơ bản để xây dựng cấu trúc của trang web. Trong hệ thống, HTML không chỉ là “bộ khung” cho giao diện,

mà còn đóng vai trò quan trọng trong việc tổ chức và hiển thị thông tin từ backend đến người dùng.

Các ứng dụng cụ thể của HTML trong dự án:

- **Hiển thị dữ liệu cảm biến**: bố trí bảng hiển thị các thông số nhiệt độ, độ ẩm, ánh sáng theo thời gian.
- **Biểu đồ trực quan**: kết hợp HTML với thư viện JavaScript (ví dụ Chart.js) để minh họa dữ liệu dạng biểu đồ.
- **Tương tác người dùng**: các nút bấm (bật/tắt thiết bị), biểu mẫu chọn ngày tháng, và các thành phần điều hướng đều được định nghĩa bằng HTML.

Như vậy, HTML chính là nền tảng để người dùng có thể tiếp cận và quản lý hệ thống chăn nuôi thông qua giao diện web.

3. TailwindCss:

TailwindCSS là framework CSS với triết lý “utility-first”. Thay vì viết các tệp CSS riêng biệt cho từng thành phần, lập trình viên sử dụng trực tiếp các lớp tiện ích trong HTML để định dạng.

Vai trò của TailwindCSS trong hệ thống:

- **Xây dựng bố cục nhanh chóng**: cho phép định nghĩa grid, flexbox, margin, padding chỉ bằng các class ngắn gọn.
- **Thiết kế trực quan và thống nhất**: giúp duy trì phong cách giao diện gọn gàng, dễ đọc, không rườm rà.
- **Responsive**: dễ dàng tùy chỉnh để giao diện hiển thị tốt trên nhiều loại thiết bị (máy tính, máy tính bảng, điện thoại).
- **Tiết kiệm thời gian**: giảm bớt việc viết CSS thủ công, giúp nhóm tập trung vào chức năng và dữ liệu.

Trong bối cảnh dự án, TailwindCSS giúp hệ thống có giao diện đủ rõ ràng để người chăn nuôi quan sát dữ liệu, thao tác đơn giản mà không cần quan tâm nhiều đến kỹ thuật.

4. MQTT Cloud:

MQTT (Message Queuing Telemetry Transport) là giao thức truyền thông nhẹ, được thiết kế cho các hệ thống IoT. Dữ liệu không được gửi trực tiếp từ thiết bị đến server, mà đi qua một **Broker** trung gian. MQTT Cloud chính là dịch vụ broker được triển khai trên nền tảng đám mây.

Ứng dụng trong dự án:

- **Truyền dữ liệu cảm biến**: thiết bị (ví dụ ESP32) sẽ publish dữ liệu về nhiệt độ, độ ẩm, ánh sáng lên MQTT Cloud. Node.js đóng vai trò subscriber để nhận dữ liệu này.
- **Gửi lệnh điều khiển**: Node.js sẽ publish lệnh (bật đèn, bật quạt, kích hoạt phun sương) lên MQTT Cloud, sau đó thiết bị ở chuồng trại sẽ nhận lệnh và thực hiện.
- **Tính linh hoạt**: MQTT Cloud đảm bảo các bên có thể giao tiếp ngay cả khi không nằm trong cùng mạng cục bộ, nhờ vậy hệ thống có thể được truy cập và điều khiển từ xa.
- **Cơ chế pub/sub**: cho phép dễ dàng mở rộng. Nếu sau này thêm nhiều cảm biến hoặc thiết bị, chỉ cần publish/subscribe vào các topic mới mà không cần thay đổi cấu trúc cốt lõi.

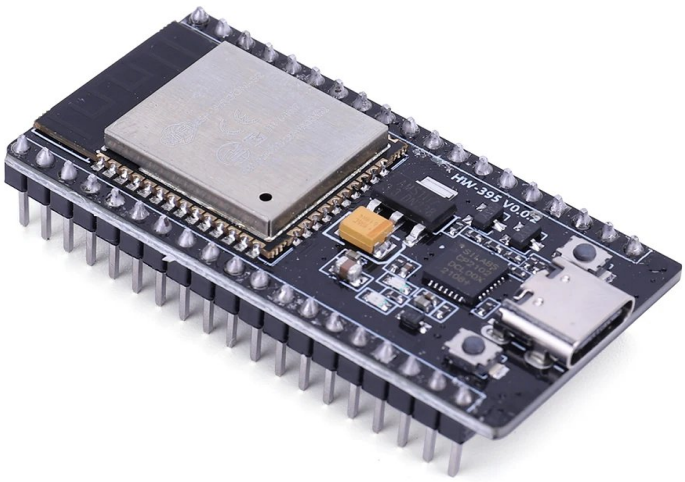
MQTT Cloud đóng vai trò như "cầu nối" giữa phần cứng và phần mềm, giúp toàn bộ hệ thống hoạt động mạch lạc.

II.2 – Phần cứng:

Bên cạnh các công nghệ phần mềm như Node.js, HTML, TailwindCSS và MQTT Cloud, hệ thống cần đến các thành phần phần cứng để thu thập dữ liệu từ môi

trường và thực hiện các thao tác điều khiển. Các thiết bị được lựa chọn đều có khả năng tương tác với phần mềm và kết nối mạng, đảm bảo quá trình giám sát và điều khiển diễn ra liền mạch.

1. ESP32-DEVKIT



Hình 1 : ESP32-DEVKIT

ESP32 là một bộ vi điều khiển thuộc danh mục vi điều khiển trên chip công suất thấp và tiết kiệm chi phí. Hầu hết tất cả các biến thể ESP32 đều tích hợp Bluetooth và Wi-Fi chế độ kép, làm cho nó có tính linh hoạt cao, mạnh mẽ và đáng tin cậy cho nhiều ứng dụng.

Nó là sự kế thừa của vi điều khiển NodeMCU ESP8266 phổ biến và cung cấp hiệu suất và tính năng tốt hơn. Bộ vi điều khiển ESP32 được sản xuất bởi Espressif Systems và được sử dụng rộng rãi trong nhiều ứng dụng khác nhau như IoT, robot và tự động hóa.

ESP32 cũng được thiết kế để tiêu thụ điện năng thấp, lý tưởng cho các ứng dụng chạy bằng pin. Nó có hệ thống quản lý năng lượng cho phép nó hoạt động ở chế độ ngủ và chỉ thức dậy khi cần thiết, điều này có thể kéo dài tuổi thọ pin rất nhiều.

Trong dự án này, ESP32 đóng vai trò trung tâm xử lý ở phía phần cứng:

- Nhận dữ liệu từ các cảm biến (DHT11, BH1750).
- Gửi dữ liệu thu thập được lên MQTT Cloud để truyền về hệ thống phần mềm.
- Nhận lệnh điều khiển từ MQTT Cloud (do Node.js gửi lên) và thực hiện thao tác qua module Relay.

ESP32 là cầu nối quan trọng giữa thế giới vật lý (môi trường chuồng trại) và phần mềm quản lý trên server.

a. Tổng quan:

- Cầu USB-to-UART: Cầu USB-UART chip đơn có thể cung cấp tốc độ truyền lên đến 3 Mbps.
- Giao diện: Giao diện TYPE-C, có thể được sử dụng làm nguồn điện cho bảng mạch hoặc làm giao diện giao tiếp giữa các mô-đun PC và ESP32-WROOM-32.
- Giao tiếp: Wi Fi 2,4 GHz + Bluetooth
- Bộ xử lý: Bộ xử lý lõi kép Xtensa @ 32 Bit LX6 hiệu suất cao, bộ xử lý co công suất cực thấp
- Nhiều thiết bị ngoại vi: với hiệu suất tiêu thụ điện năng tốt nhất, hiệu suất RF, độ ổn định, tính linh hoạt và độ tin cậy, phù hợp với các tình huống ứng dụng khác nhau và các yêu cầu tiêu thụ điện năng khác nhau.

Bảng phát triển có thể chọn từ bất kỳ phương pháp nào trong số ba phương pháp cung cấp điện sau:

- Nguồn cung cấp giao diện micro-usb (mặc định)

- Nguồn điện chân 5V / GND
- Nguồn điện chân 3V3 / GND

b. Thông số kỹ thuật:

Vi xử lý: Dual-core Tensilica Xtensa LX6, xung nhịp tối đa 240 MHz.

RAM: 520 KB SRAM.

ROM: 448 KB.

Flash: 4 MB (thông thường).

Kết nối: Wi-Fi 802.11 b/g/n, Bluetooth v4.2 + BLE.

Nguồn cấp:

- 5V qua cổng microUSB hoặc chân VIN.
- 3.3V qua chân 3V3.

GPIO: 30 chân (trong đó có nhiều chân đa năng).

ADC: 12-bit, tối đa 18 kênh.

DAC: 2 kênh (GPIO25, GPIO26).

PWM: hầu hết các GPIO.

UART/SPI/I2C/I2S: nhiều kênh, dễ cấu hình.

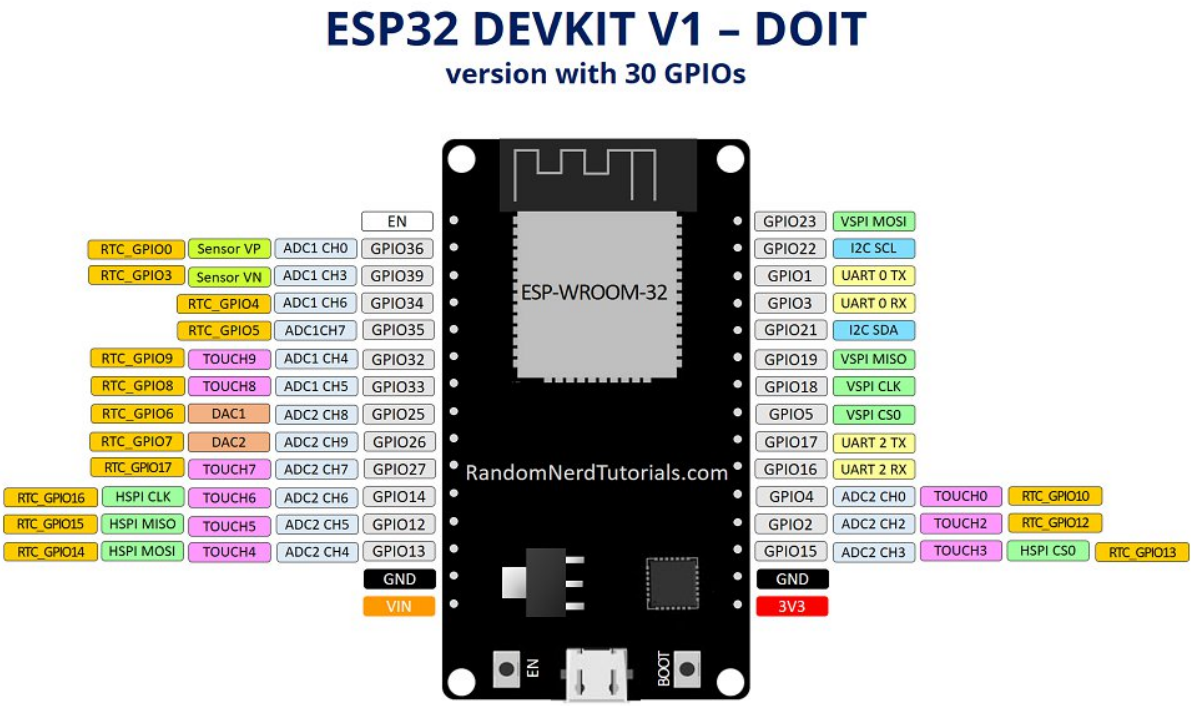
Dòng tiêu thụ:

- Bình thường: 160–260 mA.
- Deep sleep: ~10 µA.

Kích thước: 52*29mm

c. Sơ đồ chân:

DEVKIT hệ thống sử dụng có 30 chân kết nối, trong đó có 25 GPIO , 1 chân EN(rs), 2 chân GND và 2 chân nguồn



Hình 2 : Sơ dồ chân ESP32-DEVKIT (30 chân)

2. Cảm biến DHT11

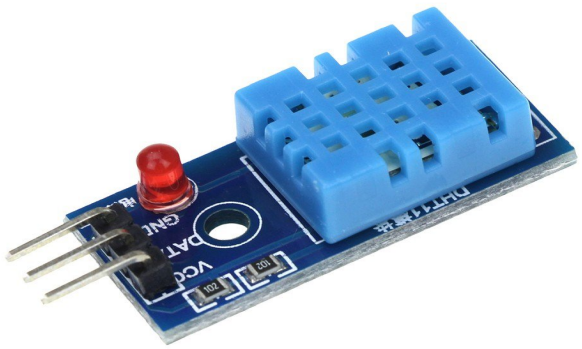
DHT11 là một cảm biến kỹ thuật số giá rẻ để cảm nhận nhiệt độ và độ ẩm. Cảm biến này có thể dễ dàng giao tiếp với bất kỳ bộ vi điều khiển vi nào như Arduino, Raspberry Pi, ... để đo độ ẩm và nhiệt độ ngay lập tức.

DHT11 là một cảm biến độ ẩm tương đối. Để đo không khí xung quanh, cảm biến này sử dụng một điện trở nhiệt và một cảm biến độ ẩm điện dung.

Trong hệ thống, nó cung cấp các thông số môi trường cơ bản nhưng rất quan trọng cho việc chăn nuôi:

- Nhiệt độ quá cao hoặc quá thấp sẽ ảnh hưởng đến sức khỏe và khả năng sinh trưởng của gia súc.
- Độ ẩm cần duy trì ở mức hợp lý để tránh môi trường quá khô hoặc quá ẩm, dễ phát sinh bệnh.

Dữ liệu từ DHT11 được gửi về ESP32, sau đó truyền lên phần mềm để lưu trữ và xử lý.



Hình 3 : Module cảm biến độ ẩm, nhiệt độ DHT11

a. Tổng quan:

Cảm biến DHT11 bao gồm một phần tử cảm biến độ ẩm điện dung và một điện trở nhiệt để cảm nhận nhiệt độ. Tụ điện cảm biến độ ẩm có hai điện cực với chất nền giữ ẩm làm chất điện môi giữa chúng. Thay đổi giá trị điện dung xảy ra với sự

thay đổi của các mức độ ẩm. IC đo, xử lý các giá trị điện trở đã thay đổi này và chuyển chúng thành dạng kỹ thuật số.

Để đo nhiệt độ, cảm biến này sử dụng một nhiệt điện trở có hệ số nhiệt độ âm, làm giảm giá trị điện trở của nó khi nhiệt độ tăng. Để có được giá trị điện trở lớn hơn ngay cả đối với sự thay đổi nhỏ nhất của nhiệt độ, cảm biến này thường được làm bằng gốm bán dẫn hoặc polymer.

b. Thông số kỹ thuật:

- Điện áp hoạt động: 3V - 5V DC

- Dòng điện tiêu thụ: 2.5mA

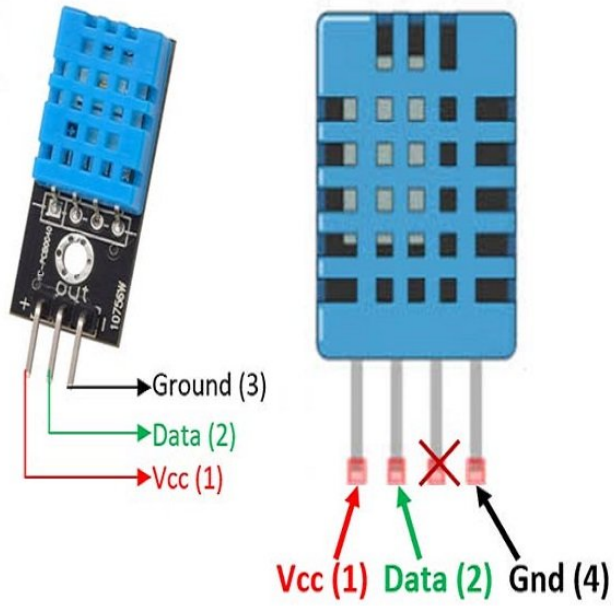
- Phạm vi cảm biến độ ẩm: 20% - 90% RH, sai số ±5%RH

- Phạm vi cảm biến nhiệt độ: 0°C ~ 50°C, sai số ±2°C

- Tần số lấy mẫu tối đa: 1Hz (1 giây 1 lần)

- Kích thước: 23 * 12 * 5 mm

c. Sơ đồ chân:



Hình 4: Sơ đồ chân Module DHT11

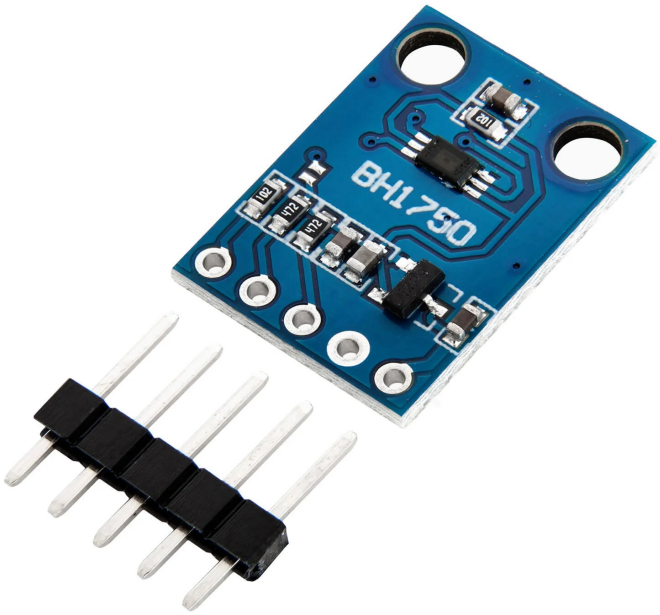
3. Module BH1750

BH1750 là cảm biến ánh sáng xung quanh kỹ thuật số thường được sử dụng trong điện thoại di động để điều chỉnh độ sáng màn hình dựa trên ánh sáng môi trường. Cảm biến này có thể đo chính xác giá trị LUX của ánh sáng lên tới 65535lx.

Trong hệ thống, BH1750 giúp đánh giá mức độ chiếu sáng trong chuồng trại để:

- Điều chỉnh đèn chiếu sáng khi ánh sáng tự nhiên không đủ.
- Duy trì chu kỳ sáng – tối hợp lý, hỗ trợ quá trình sinh hoạt và phát triển của vật nuôi.

Thông tin từ BH1750 cũng được truyền về ESP32 rồi gửi lên hệ thống phần mềm để đưa ra quyết định điều khiển.



Hình 5: Module cảm biến ánh sáng BH1750

a. Tổng quan:

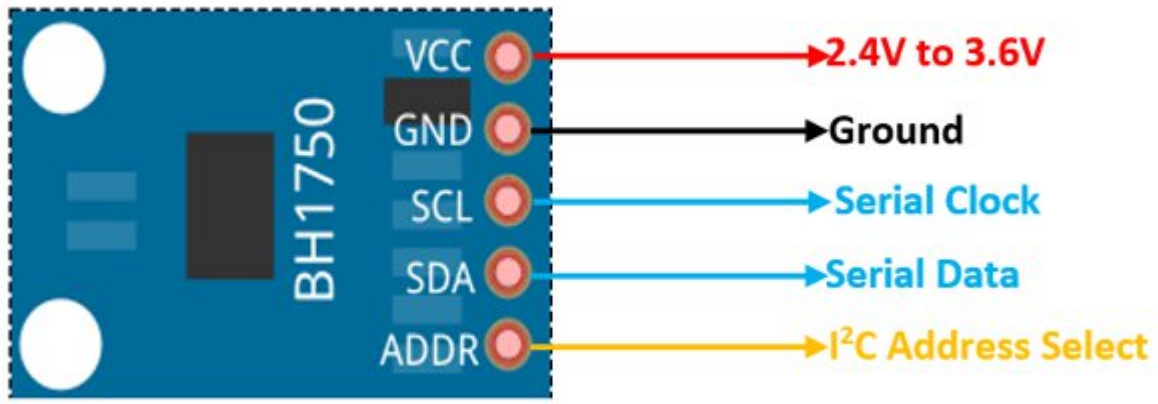
- Giao tiếp với vi điều khiển thông qua chuẩn I²C (hai dây SDA và SCL).
- Độ phân giải cao (có thể đo từ ánh sáng rất yếu đến ánh sáng mạnh ngoài trời).
- Trả về dữ liệu số trực tiếp, không cần chuyển đổi tín hiệu analog sang digital.

b. Thông số kỹ thuật:

- Nguồn điện: 2.4V-3.6V (thường là 3.0V)
- Tiêu thụ dòng điện ít hơn: 0,12mA
- Dải đo: 1-65535lx
- Giao tiếp: bus I2C
- Độ chính xác: +/-20%

- Bộ chuyển đổi A/D tích hợp để chuyển đổi độ sáng analog trong dữ liệu số.
- Ảnh hưởng rất nhỏ của bức xạ hồng ngoại
- Độ phản hồi cao gần với mắt người.

c. Sơ đồ chân:



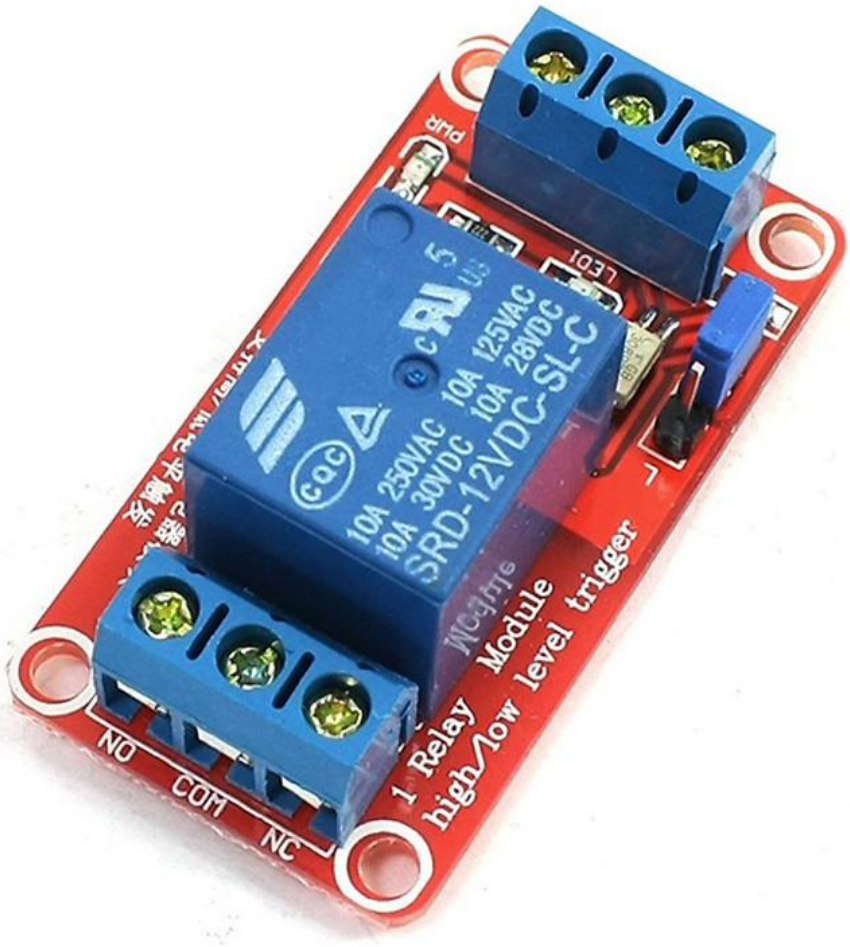
Hình 6: Sơ đồ chân cảm biến ánh sáng BH1750

4. Module Relay:

Module Relay là thiết bị trung gian giúp ESP32 điều khiển các tải điện xoay chiều hoặc một số thiết bị công suất lớn. Trong dự án, relay được sử dụng để:

- Bật/tắt đèn chiếu sáng.
- Kích hoạt quạt thông gió hoặc hệ thống phun sương.

Relay đóng vai trò "cầu nối" giữa tín hiệu điều khiển điện tử của ESP32 và các thiết bị điện thực tế trong chuồng trại.



Hình 7: Module Relay

a. Tổng quan

- Có khả năng cách ly giữa mạch điều khiển (ESP32) và mạch tải điện áp cao.
- Hoạt động như một công tắc: bật hoặc tắt thiết bị theo tín hiệu từ vi điều khiển.

- Có thể điều khiển nhiều loại tải khác nhau: đèn chiếu sáng, quạt, máy phun sương...

b. Thông số kỹ thuật:

Điện áp hoạt động: 5V DC (một số loại có phiên bản 3.3V hoặc 12V).

Dòng tiêu thụ cuộn dây: khoảng 70 – 80 mA khi kích hoạt.

Điện áp tín hiệu điều khiển:

- Mức thấp (0V) → relay tắt.
- Mức cao (3.3V hoặc 5V tùy loại) → relay bật.

Ngõ ra tiếp điểm (điều khiển tải):

- Điện áp tối đa: 250V AC hoặc 30V DC.
- Dòng tải tối đa: 10A (AC) hoặc 10A (DC).

Cấu trúc tiếp điểm:

- Thường có 3 chân: COM (chung), NO (thường hở), NC (thường đóng).
- COM – NO: đóng mạch khi relay kích hoạt.
- COM – NC: đóng mạch khi relay không kích hoạt.

Cách ly tín hiệu: một số module có opto-coupler để cách ly giữa mạch điều khiển (ESP32/vi điều khiển) và mạch tải điện áp cao.

Kích thước module: nhỏ gọn, thường khoảng 44 x 26 x 18 mm (có thể thay đổi theo nhà sản xuất).

Chỉ báo trạng thái: LED báo relay đang bật/tắt.

II.3 – Giao diện

1. Giao diện DashBoard

Chức năng chính là hiển thị kết quả thực tế thu được từ các cảm biến theo thời gian thực. Giao diện có ba ô chỉ số đặt ở vị trí nổi bật: ánh sáng hiển thị bằng đơn vị lux, nhiệt độ hiển thị bằng độ C, và độ ẩm hiển thị bằng phần trăm. Bên dưới ba ô này là biểu đồ trực quan, giúp người dùng quan sát sự thay đổi của ánh sáng, nhiệt độ và độ ẩm trong một khoảng thời gian nhất định. Biểu đồ sử dụng các đường màu khác nhau để phân biệt từng thông số, cho phép theo dõi xu hướng biến động theo thời gian. Ngoài ra, ở phần cuối trang có ba công tắc điều khiển thiết bị gồm bóng đèn, điều hòa và quạt. Các công tắc này cho phép người dùng bật hoặc tắt thiết bị trực tiếp từ giao diện nhằm điều chỉnh và cân bằng các chỉ số môi trường theo nhu cầu.

2. Giao diện Bảng dữ liệu

Chức năng chính là cung cấp dữ liệu cảm biến dưới dạng bảng chi tiết. Mỗi dòng trong bảng gồm đầy đủ các thông tin: độ ẩm (%), ánh sáng (lux), nhiệt độ (°C) và thời gian ghi nhận dữ liệu. Bảng được sắp xếp theo thứ tự thời gian và có thêm các công cụ hỗ trợ tìm kiếm, lọc hoặc sắp xếp để người dùng dễ dàng quản lý dữ liệu. Ngoài ra, bảng còn hỗ trợ phân trang để hiển thị số lượng lớn bản ghi, tránh quá tải màn hình. Với chức năng này, người dùng có thể xem lại các giá trị cụ thể mà hệ thống đã thu thập được, đối chiếu từng mốc thời gian với các chỉ số cảm biến thực tế. Đây là phần quan trọng để kiểm chứng và phân tích xu hướng từ dữ liệu đã lưu.

3, Giao diện Lịch sử thao tác

Chức năng chính là ghi lại và hiển thị toàn bộ lịch sử điều khiển các thiết bị. Dữ liệu được trình bày dưới dạng bảng, bao gồm các cột: tên thiết bị (ví dụ lightLed, humiLed, tempLed), trạng thái hoạt động (ON hoặc OFF) và thời gian chính xác thao tác được thực hiện. Người dùng có thể dễ dàng biết được ai đã bật/tắt thiết bị

nào và vào thời điểm nào. Bảng cũng được tích hợp công cụ sắp xếp, tìm kiếm và lọc theo điều kiện, giúp truy xuất nhanh chóng các thao tác đã thực hiện trước đó. Tính năng này hỗ trợ việc theo dõi và đánh giá mức độ tương tác với hệ thống, đồng thời đảm bảo minh bạch khi kiểm tra lại quá trình vận hành.

4. Giao diện Profile

Chức năng chính là hiển thị thông tin cá nhân của người dùng. Giao diện gồm các trường cơ bản: họ và tên, số điện thoại, file PDF. Ngoài ra, có liên kết đến tài khoản GitHub và tài liệu API để phục vụ cho việc quản lý, truy cập và đồng bộ dữ liệu dự án. Ảnh đại diện người dùng được hiển thị bên góc phải, giúp nhận diện trực quan. Các thành phần được sắp xếp thành từng khối rõ ràng, dễ quan sát và thuận tiện cho việc cập nhật hoặc tra cứu thông tin cá nhân.

II.4 – Thiết kế CSDL

1. Các lớp thực thể:

a. Lớp User - Lớp chứa thông tin người dùng.

- UserID : Id người dùng
- name : tên người dùng
- phoneNumber : số điện thoại người dùng

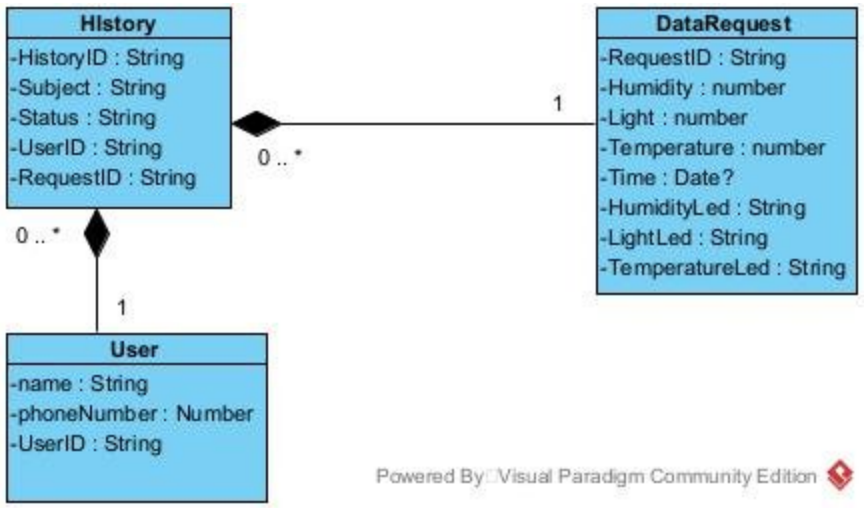
b. Lớp History - lớp lưu trạng thái các đối tượng điều khiển, gắn với **User** và **DataRequest**.

- HistoryID : id lịch sử thao tác
- Subject : đối tượng là người dùng thao tác
- Status : trạng thái hiện tại của đối tượng mà người dùng thao tác
- ResquestID : id của lớp DataResquest - lớp nhận dữ liệu từ thiết bị cảm biến

c. Lớp DataResquest - Lớp chứa dữ liệu cảm biến được gửi về (độ ẩm, ánh sáng, nhiệt độ).

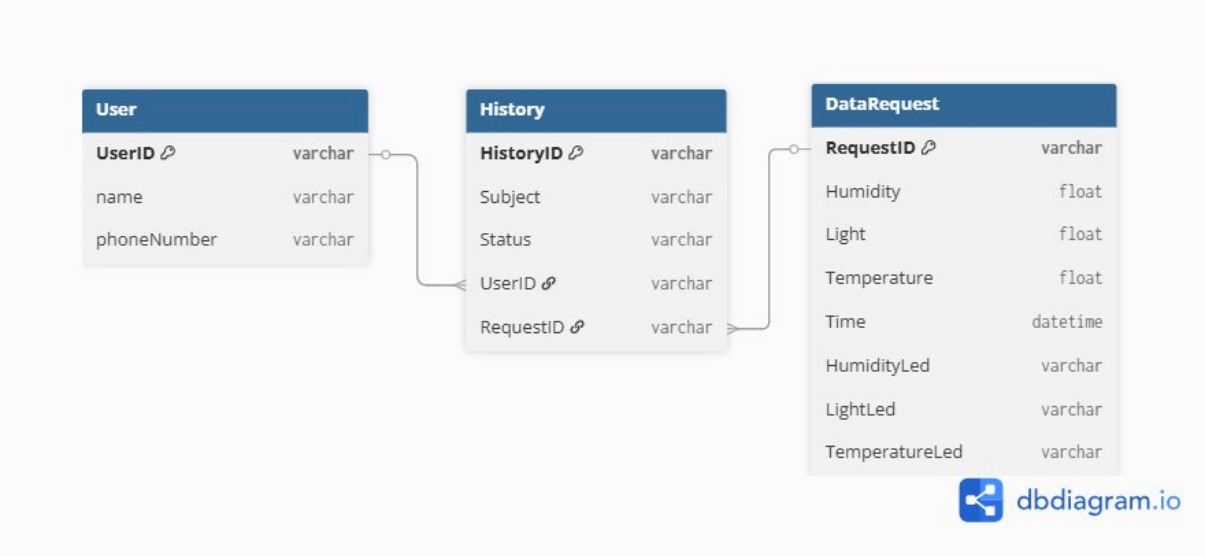
- ResquestID : id mỗi lần thiết bị cảm biến truyền vào sever
- Humidity : giá trị của độ ẩm
- HumidityLed : đèn bật/tắt độ ẩm
- Light : giá trị của ánh sáng
- LightLed : đèn bật/tắt ánh sáng
- Temperature: giá trị của nhiệt độ
- TemperatureLed : đèn bật/tắt của nhiệt độ
- Time : thời gian mà server nhận được

2. Mối quan hệ thực thể:

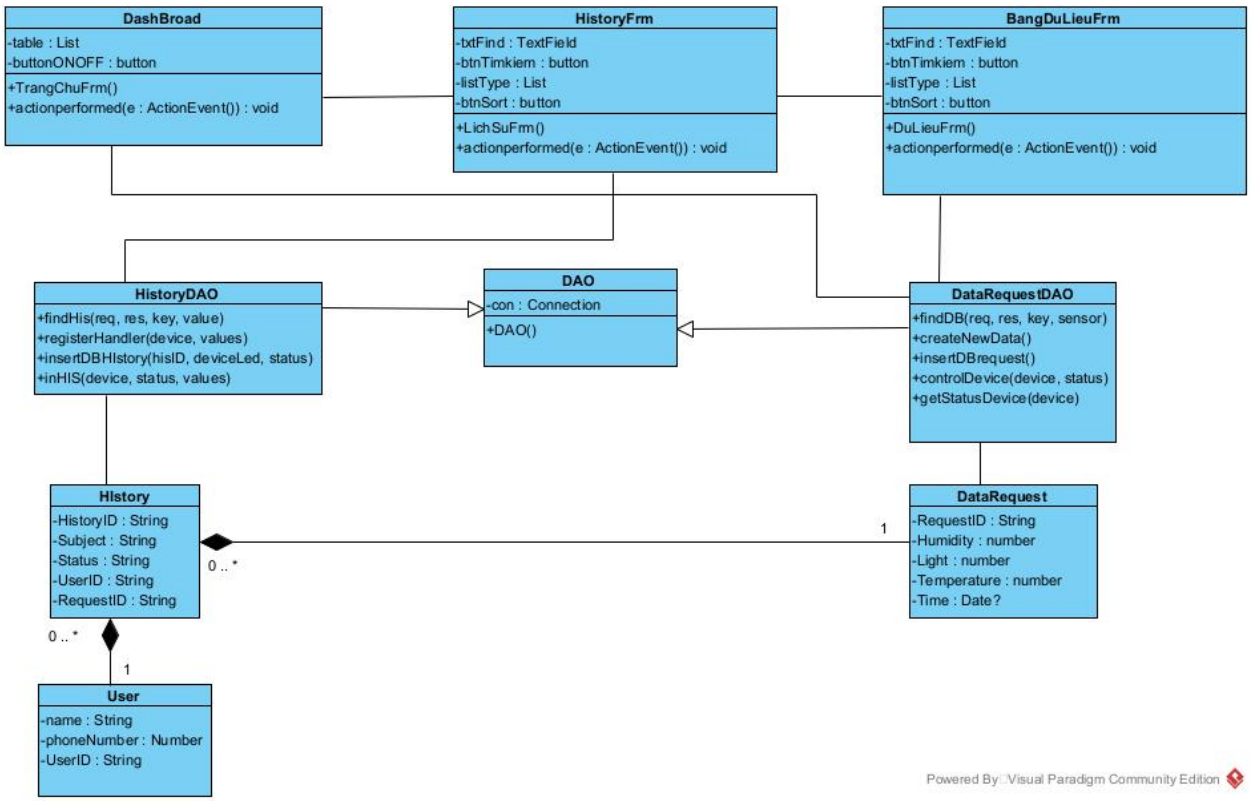


- Một User có nhiều lịch sử điều khiển → Lớp User có quan hệ một nhiều với lớp History
- Một yêu cầu dữ liệu có nhiều bản ghi trạng thái → Lớp DataRequest có quan hệ một nhiều với lớp History

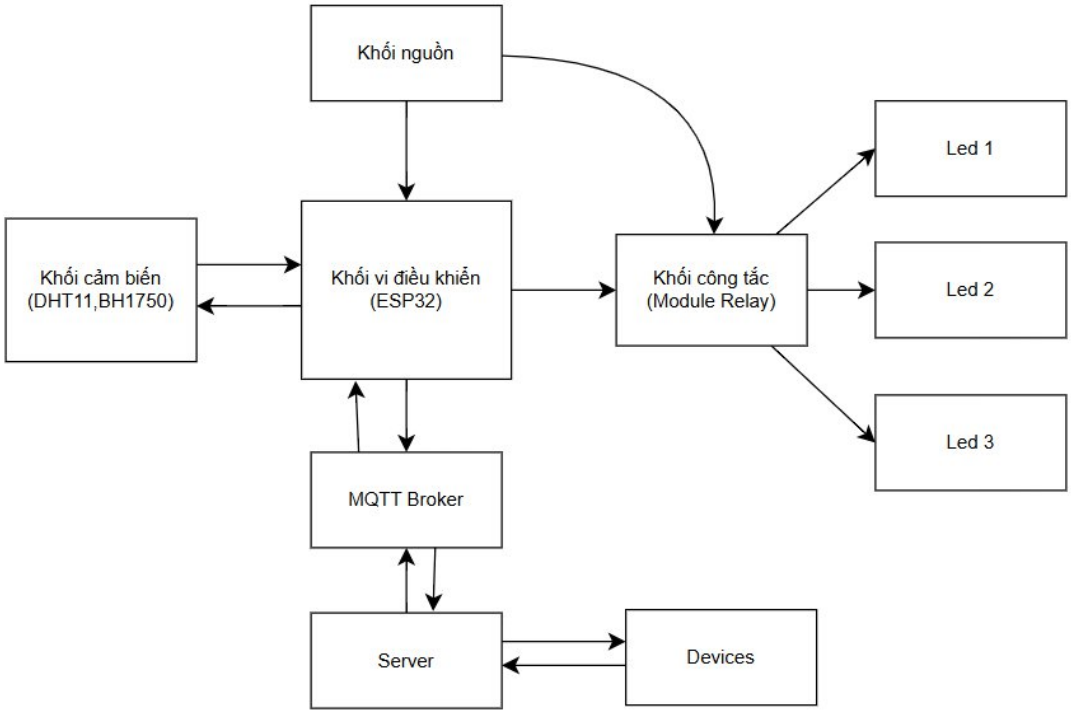
3. Biểu đồ ERD



4. Mô hình MVC



5. Sơ đồ khối:



- Khối nguồn : Cung cấp năng lượng cho toàn bộ hệ thống, bao gồm vi điều khiển ESP32, các cảm biến, module relay và các tải đầu ra (LED).
- Khối cảm biến (DHT11, BH1750) :
    - Cảm biến DHT11 được dùng để đo nhiệt độ và độ ẩm môi trường.
    - Cảm biến BH1750 được dùng để đo cường độ ánh sáng.
    - Các tín hiệu thu thập từ cảm biến sẽ được gửi về **khối vi điều khiển** để xử lý.
- Khối vi điều khiển (ESP32):
    - Đóng vai trò trung tâm xử lý, nhận dữ liệu từ các cảm biến.
    - Thực hiện xử lý dữ liệu, đưa ra quyết định điều khiển.

- Gửi tín hiệu điều khiển đến **khối công tắc (module relay)** để bật/tắt các thiết bị đầu ra (LED).

- Đồng thời, ESP32 cũng giao tiếp với **Server** và ngược lại thông qua khối MQTT Broker, giúp người dùng giám sát và điều khiển hệ thống từ xa thông qua thiết bị kết nối mạng.

- Khối công tắc (Module Relay):

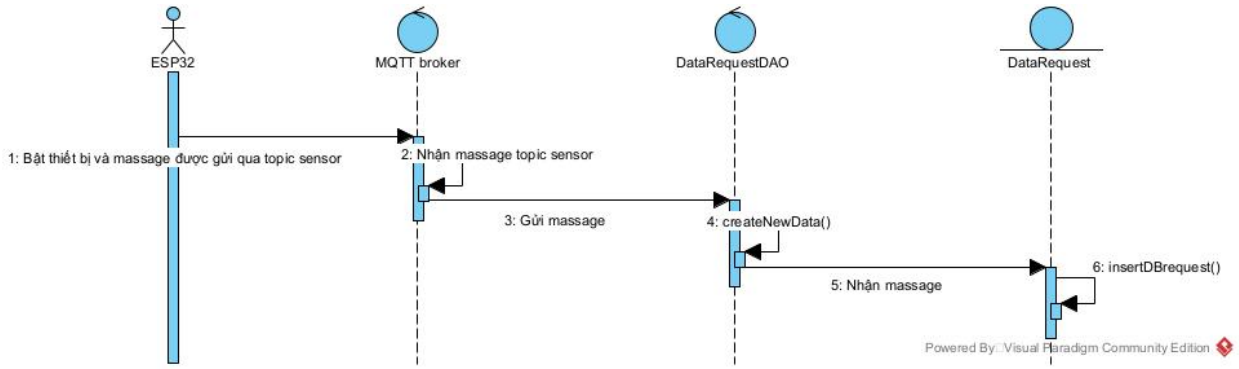
  - Đóng vai trò như mạch đóng/ngắt, nhận lệnh từ ESP32.
  - Điều khiển trực tiếp các tải đầu ra (LED 1, LED 2, LED 3).

- Các thiết bị đầu ra (LED1, LED2, LED3): là các tải mô phỏng, được điều khiển bật/tắt tùy theo tín hiệu ESP32 thông qua module relay
- Server và Device:

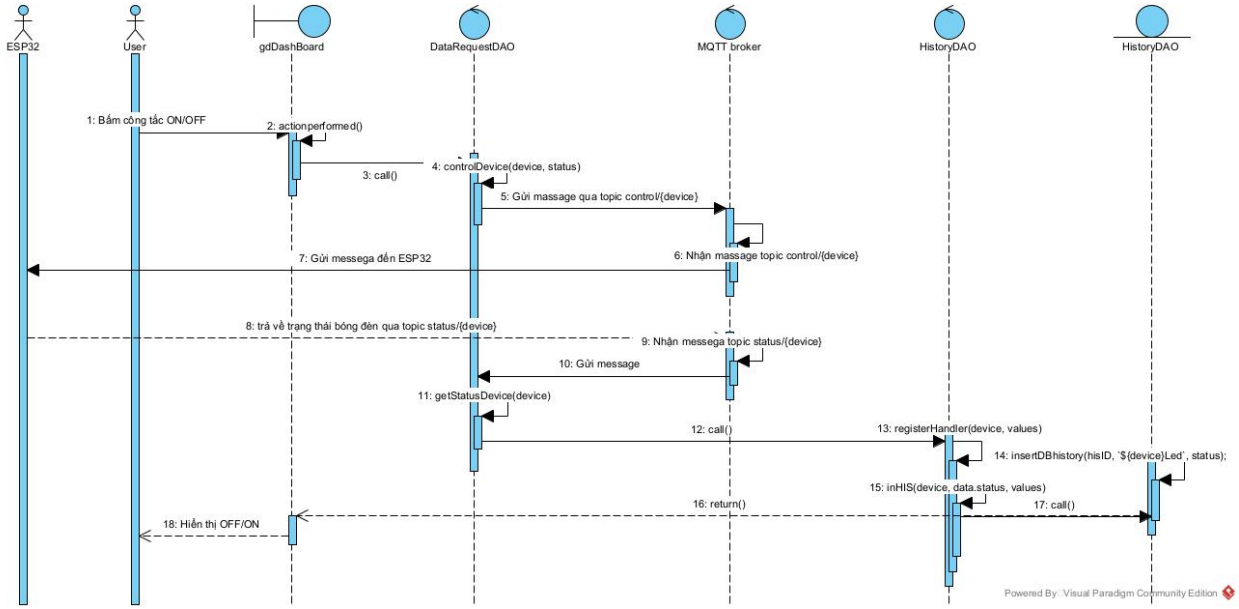
  - ESP32 kết nối với **Server** thông qua MQTT broker để lưu trữ và quản lý dữ liệu vào DataBase.
  - Người dùng có thể sử dụng **Devices** (máy tính, smartphone, v.v.) để giám sát và điều khiển hệ thống từ xa.

II.5 – Luồng

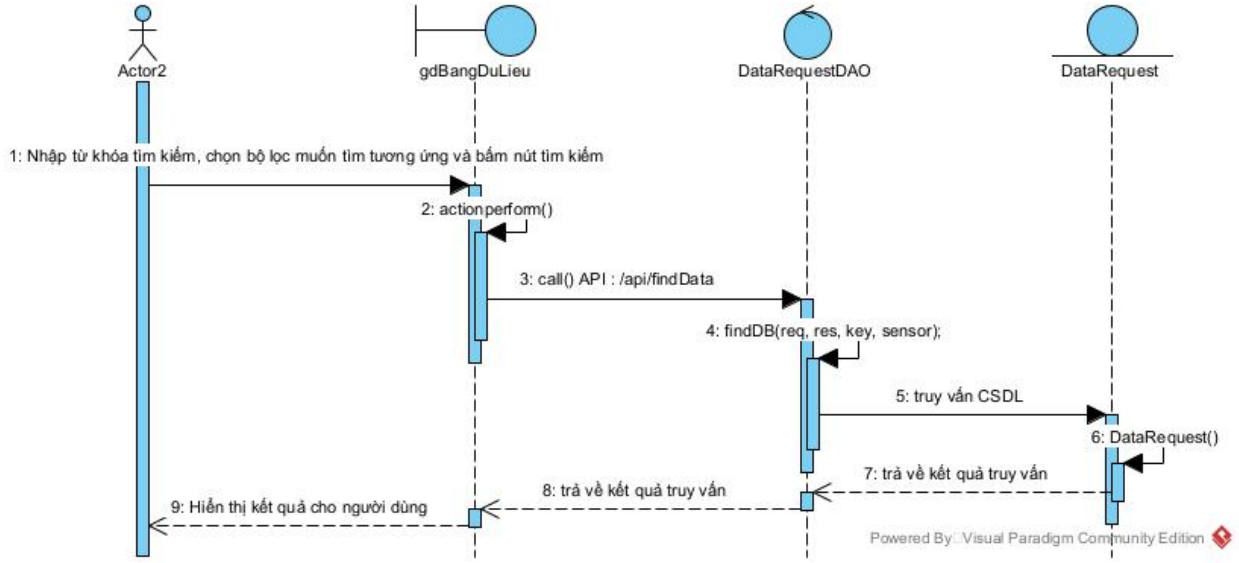
1. Nhận dữ liệu từ ESP32:



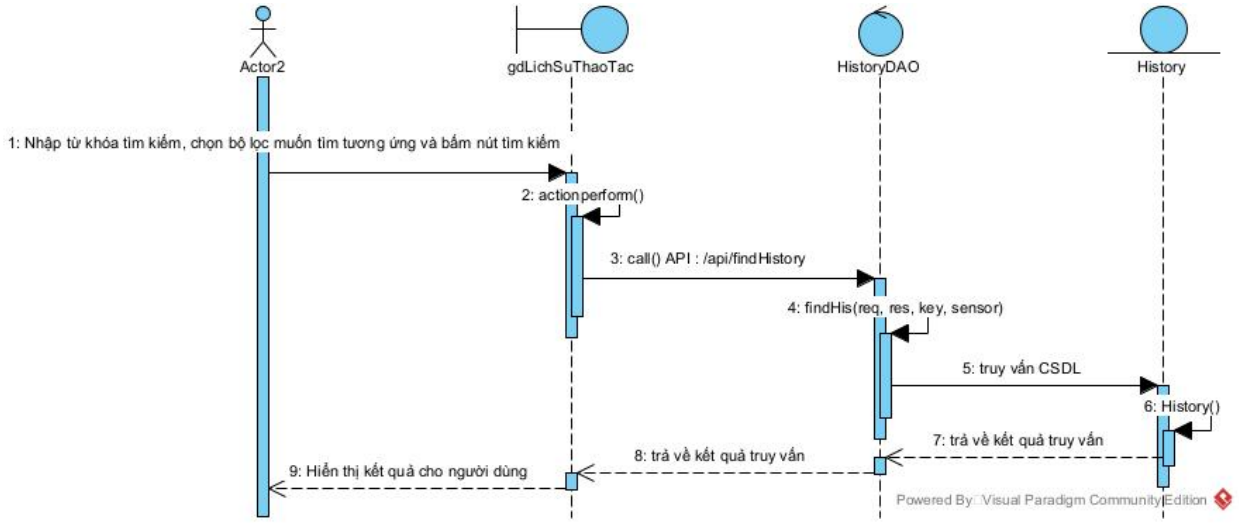
2. Bật/tắt đèn:



3. Tìm kiếm dữ liệu:



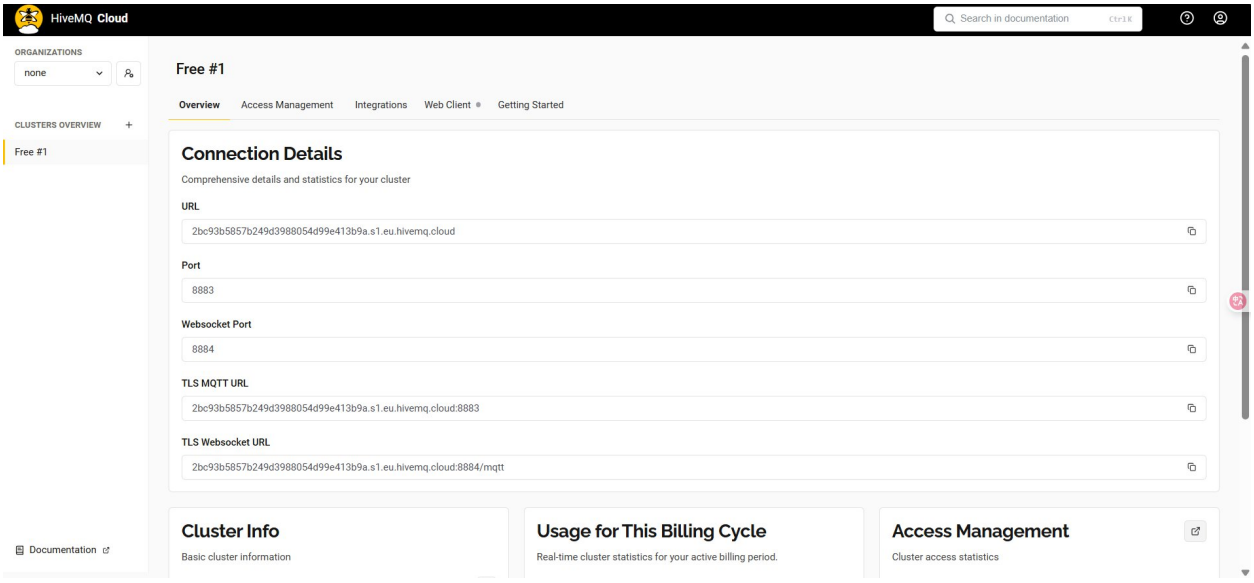
4. Tìm kiếm lịch sử thao tác:



III – Xây dựng hệ thống:

III.1 – MQTT broker (MQTT Cloud):

- Đầu tiên truy cập vào đường link này để tạo tài khoản: HiveMQ Cloud – Fully-Managed MQTT Platform on the Cloud | Free Version Available . Sau khi tạo xong tài khoản, chúng ta sẽ có một broker instance miễn phí



- Sau đó tạo username và password mới chọn Access Management → Add new credentials, ghi nhớ để kết nối

- Phía backend sẽ kết nối với MQTT Could:

```
const mqtt = require("mqtt");

const client = mqtt.connect(process.env.MQTT_BROKER, {
  port: process.env.MQTT_PORT,
  username: process.env.MQTT_USERNAME,
  password: process.env.MQTT_PASSWORD,
  protocol: "mqtts", // Bắt buộc dùng TLS
  rejectUnauthorized: false, // Chấp nhận chứng chỉ self-signed
  reconnectPeriod: 1000, // Thử reconnect mỗi 1s nếu mất kết nối
});
```

```
#connect MQTT
MQTT_BROKER=mqtts://2bc93b5857b249d3988054d99e413b9a.s1.eu.hivemq.cloud
MQTT_PORT=8883
MQTT_USERNAME=esp32
MQTT_PASSWORD=Ltdat2004@
MQTT_TOPIC=sensor
```

- Kết nối với ESP32:

```
const char* mqtt_server = "2bc93b5857b249d3988054d99e413b9a.s1.eu.hivemq.cloud";
const int mqtt_port = 8883;
const char* mqtt_user = "esp32";
const char* mqtt_pass = "Ltdat2004@";
const char* mqtt_topic = "sensor";
```

```
void reconnectMQTT() {
  while (!client.connected()) {
    Serial.print("Đang kết nối MQTT...");
    if (client.connect("ESP32_Client", mqtt_user, mqtt_pass)) {
      Serial.println("Kết nối thành công!");
      client.subscribe(mqtt_control_light_topic);
      client.subscribe(mqtt_control_temperature_topic);
      client.subscribe(mqtt_control_humidity_topic);
    } else {
      Serial.print("Thất bại, mã lỗi = ");
      Serial.print(client.state());
      Serial.println(" -> thử lại sau 5s");
      delay(5000);
    }
  }
}
```

- Khi kết nối cả 2 thành công thì ta có thể thực hiện được tiếp các công việc tiếp theo

III.2 – Nạp code vào ESP32 từ Arduno:

Giải thích đoạn code:

1.Import thư viện :

```
#include <Wire.h>
#include <BH1750.h>
#include "DHT.h"
#include <WiFi.h>
#include <WiFiManager.h>
#include <PubSubClient.h>
#include <WiFiClientSecure.h>
```

- **Wire.h**: giao tiếp I²C (dùng cho BH1750).
- **BH1750.h**: thư viện cảm biến ánh sáng BH1750.

- **DHT.h**: thư viện cảm biến nhiệt độ/độ ẩm DHT11.
- **WiFi.h**: quản lý kết nối WiFi.
- **WiFiManager.h**: giúp cấu hình WiFi dễ dàng, tạo AP để nhập SSID/Password.
- **PubSubClient.h**: client MQTT để publish/subscribe.
- **WiFiClientSecure.h**: client hỗ trợ TLS/SSL khi kết nối MQTT qua port 8883.
    -

2.Khai báo cảm biến:

```
#define DHTPIN 4
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

BH1750 lightMeter;
```

- DHT11 kết nối chân data vào GPIO 4.
- Khởi tạo đối tượng dht để đọc nhiệt độ/độ ẩm.
- lightMeter: đối tượng đọc dữ liệu từ cảm biến ánh sáng BH1750 (giao tiếp I^2C).

3. Khai báo relay:

```
#define RELAY_TEMP 33
#define RELAY_HUMI 32
#define RELAY_LIGHT 25
```

- Relay điều khiển thiết bị (ví dụ đèn, quạt, máy phun sương) thông qua 3 GPIO:
    - GPIO 33 → relay cho nhiệt độ.
    - GPIO 32 → relay cho độ ẩm.
    - GPIO 25 → relay cho ánh sáng.

4. Cấu hình MQTT:

```
const char* mqtt_server = "2bc93b5857b249d3988054d99e413b9a.s1.eu.hivemq.cloud";
const int mqtt_port = 8883;
const char* mqtt_user = "esp32";
const char* mqtt_pass = "Ltdat2004@";
```

- Broker chạy trên HiveMQ Cloud (cổng 8883, có TLS).
- mqtt_user và mqtt_pass: thông tin đăng nhập đã tạo trong HiveMQ Cloud.

```
const char* mqtt_topic = "sensor";
const char* mqtt_status_led_light = "status/light";
const char* mqtt_status_led_temperature = "status/temp";
const char* mqtt_status_led_humidity = "status/humi";
const char* mqtt_control_light_topic = "control/light";
const char* mqtt_control_temperature_topic = "control/temp";
const char* mqtt_control_humidity_topic = "control/humi";
```

Topic chia thành 2 loại:

- **Publish sensor data**: sensor (ESP32 gửi dữ liệu).
- **Status topic**: báo trạng thái relay (status/...).
- **Control topic**: nhận lệnh bật/tắt từ backend hoặc app (control/...).

5. Biến trạng thái Relay:

```
bool tempLed = false;
bool humiLed = false;
bool lightLed = false;
```

- 3 biến bool lưu trạng thái hiện tại của relay (ON/OFF).
- Dùng để cập nhật dữ liệu gửi đi.

6. Callback MQTT

```
void callback(char* topic, byte* payload, unsigned int length) {
  ...
}
```

- Hàm này chạy khi ESP32 **nhận được message từ MQTT**.

- Chuyển đổi payload sang string → kiểm tra topic để biết lệnh cho **light/temp/humi**.
- Nếu message == "ON" → bật relay, publish trạng thái "ON".
- Nếu message == "OFF" → tắt relay, publish trạng thái "OFF".
- Đồng thời in log ra Serial.

7. reconnectMQTT():

```
void reconnectMQTT() {
  while (!client.connected()) {
    if (client.connect("ESP32_Client", mqtt_user, mqtt_pass)) {
      client.subscribe(...);
    } else {
      delay(5000);
    }
  }
}
```

- Nếu mất kết nối broker, ESP32 sẽ **tự động thử lại**.
- Khi kết nối lại thành công → **subscribe lại các topic control** (control/light, control/temp, control/humi).

8. setup()

```
void setup() {
  Serial.begin(115200);

  // WiFiManager autoConnect
  // Khởi động DHT11 + BH1750
  // Cấu hình pin relay
  // Kết nối MQTT
}
```

- **Serial.begin**: bật log debug.
- **WiFiManager.autoConnect("Dat")**: ESP32 tạo AP nếu chưa có WiFi, để người dùng nhập SSID/PASS.
- **Khởi động cảm biến**: Wire.begin(21,22) cho I²C, dht.begin(), lightMeter.begin().
- **Relay setup**: pinMode OUTPUT, khởi tạo trạng thái LOW (tắt).
- **MQTT setup**: cấu hình client, gắn callback.

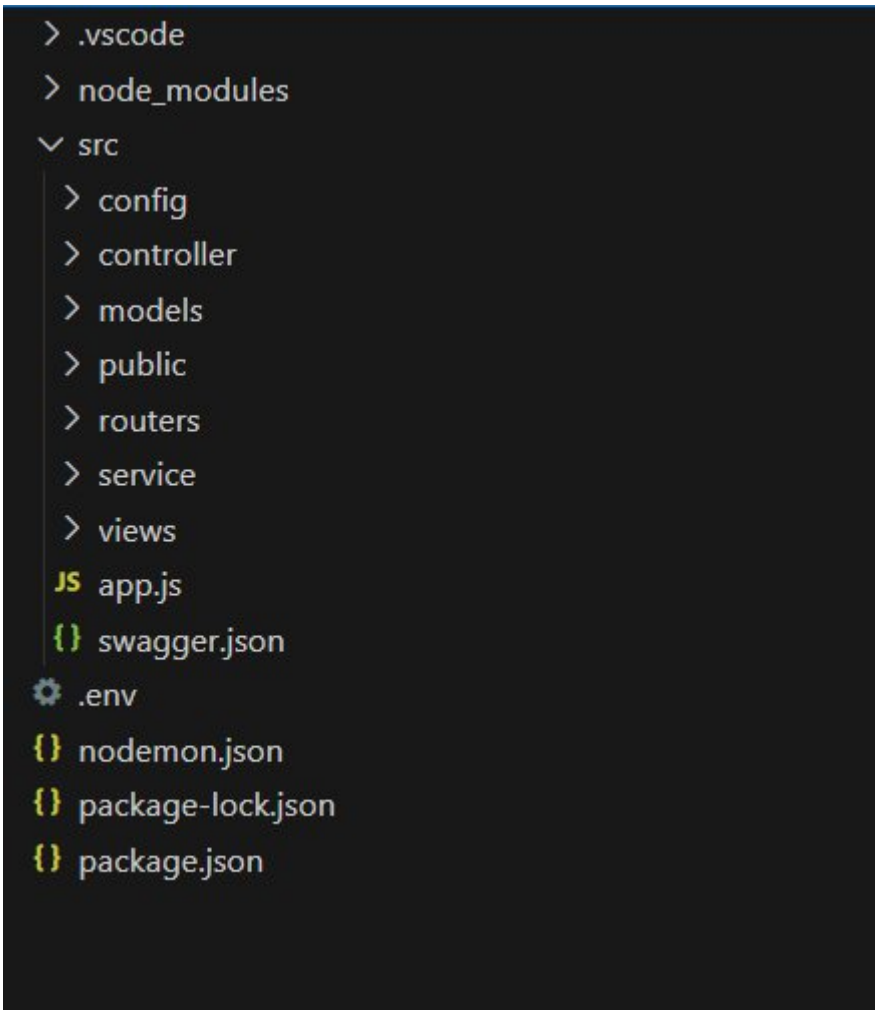
9. loop()

```
void loop() {
  if (!client.connected()) {
    reconnectMQTT();
  }
  client.loop();

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

  String payload = "{ ... }";
  client.publish(mqtt_topic, payload.c_str());

  delay(3000);
}
```

- Kiểm tra kết nối MQTT → nếu mất kết nối thì gọi reconnectMQTT().
- Đọc dữ liệu từ **cảm biến ánh sáng, nhiệt độ, độ ẩm**.
- Tạo JSON payload chứa:
    - temperature, humidity, light.
    - Trạng thái relay (TemperatureLed, HumidityLed, LightLed).
- Publish lên topic sensor.
- In log ra Serial.
- Lặp lại sau 3 giây.

III.3 – Web:

Cấu trúc dự án trên được xây dựng theo mô hình **MVC (Model – View – Controller)** kết hợp với **service layer** và **các cấu hình riêng biệt**, giúp mã nguồn dễ quản lý, mở rộng, và bảo trì. Cách chia thư mục này giúp phân tách rõ ràng từng phần trong ứng dụng, tránh trộn lẫn logic, giao diện và dữ liệu.



1. .vscode/

- Chứa các thiết lập riêng của Visual Studio Code như cấu hình debugger, cài đặt workspace.
- Hỗ trợ team làm việc đồng bộ môi trường phát triển.

2. node_modules/

- Nơi lưu trữ các package cài đặt bằng **npm/yarn**.
- Toàn bộ thư viện phụ thuộc sẽ nằm trong đây.

3. src/

Là thư mục chính chứa toàn bộ mã nguồn của dự án.

Bên trong có các thư mục con:

- **config/**

    - Chứa các file cấu hình (database, JWT, logger, môi trường, …).
    - Giúp dễ dàng thay đổi setting mà không ảnh hưởng đến logic code.
- **controller/**
    - Xử lý request từ client.
    - Nhận dữ liệu từ service, sau đó trả response về cho client.
    - Không chứa logic phức tạp, chủ yếu đóng vai trò trung gian.
- **models/**
    - Định nghĩa cấu trúc dữ liệu (ORM như Sequelize, Mongoose, …).
    - Tương ứng với bảng trong CSDL (User, Product, Order, …).
- **public/**
    - Chứa file tĩnh: hình ảnh, CSS, JS client.
    - Khi chạy server, thư mục này có thể được expose để client truy cập trực tiếp.
- **routers/**
    - Định nghĩa các endpoint (API route).
    - Mỗi route sẽ gọi tới controller tương ứng.
- **service/**
    - Chứa logic nghiệp vụ chính.
    - Controller sẽ gọi service để xử lý (ví dụ: tính toán giá trị, xử lý dữ liệu, gọi model).
- **views/**
    - Chứa template hiển thị (nếu dùng template engine như EJS, Handlebars, Pug).
    - Nếu dự án chỉ dùng API, thư mục này có thể bỏ qua.
- **app.js**
    - File khởi tạo ứng dụng chính.

- Load middleware (body-parser, cors, …), router, kết nối DB, và start server.

4. swagger.json

- Cấu hình **Swagger API Documentation**.
- Dùng để mô tả, kiểm thử, và chia sẻ API cho team/dev khác.

5. .env

- Lưu các biến môi trường (PORT, DB_URL, SECRET_KEY, …).
- Giúp bảo mật thông tin nhạy cảm, không ghi trực tiếp trong code.

6. nodemon.json

- Cấu hình cho **nodemon** – công cụ tự động reload server khi code thay đổi.

7. package.json

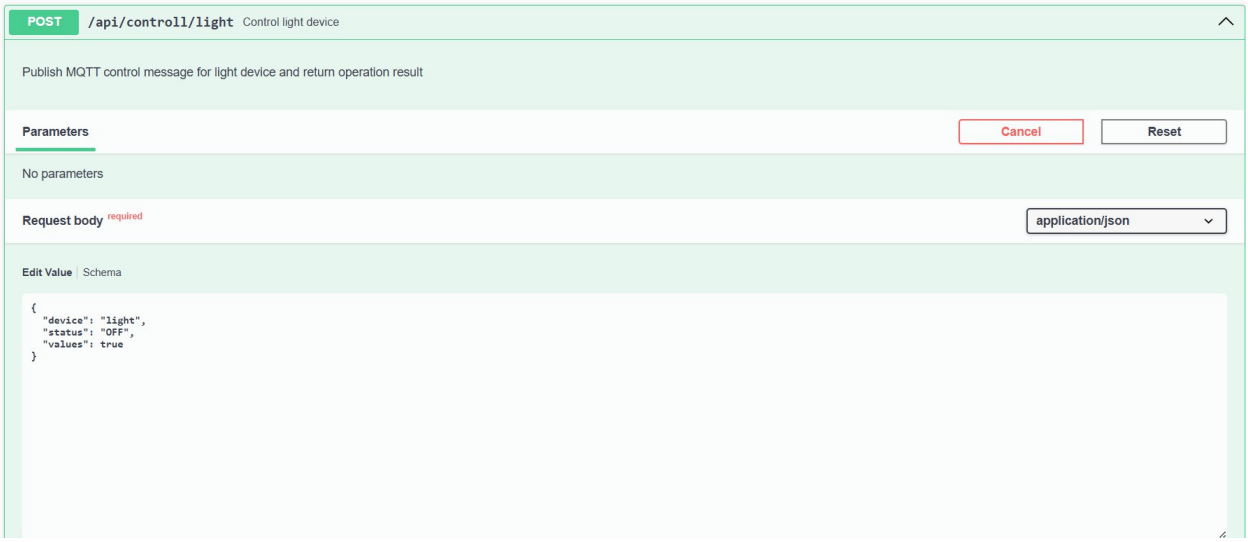
- File mô tả dự án: tên, phiên bản, script, dependencies.
- Là "trái tim" của dự án Node.js.

8. package-lock.json

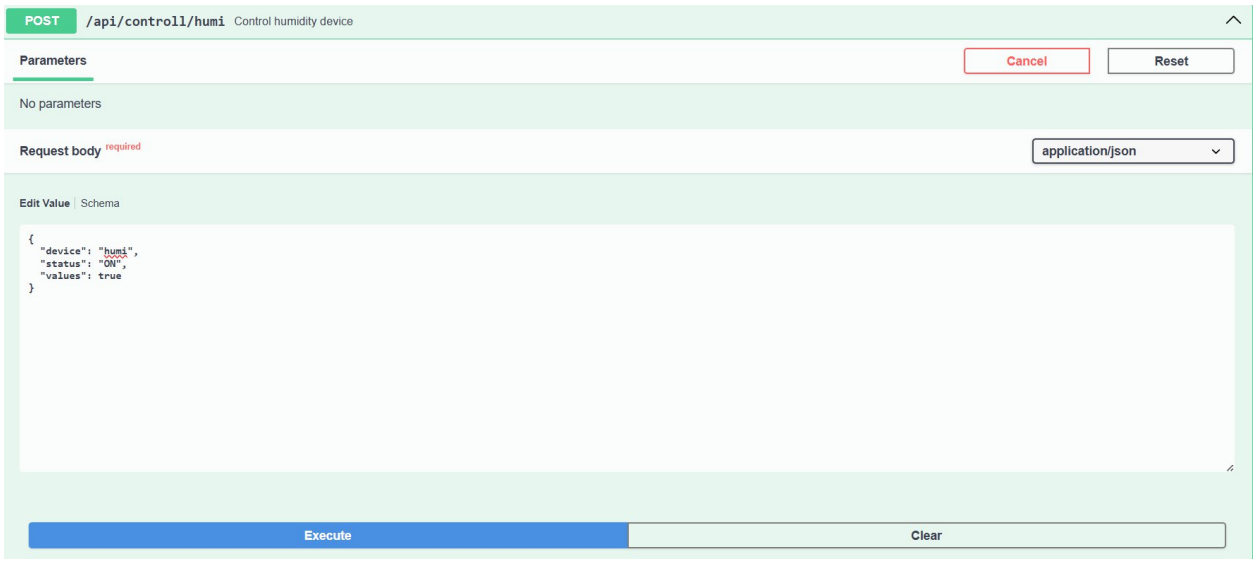
- Ghi lại phiên bản chính xác của các package đã cài.
- Giúp đồng bộ môi trường giữa các máy khác nhau

III.4: API doc

1. Bật tắt đèn:


POST
/api/controll/light
Control light device
Publish MQTT control message for light device and return operation result
Parameters
Cancel
Reset
No parameters
Request body
required
application/json
Edit Value
Schema
{
"device": "light",
"status": "OFF",
"values": true
}


Curl
curl -X 'POST' \
'http://localhost:3000/api/controll/light' \
-H 'accept: */*' \
-H 'Content-Type: application/json' \
-d '{
"device": "light",
"status": "OFF",
"values": true
}'
Request URL
http://localhost:3000/api/controll/light
Server response
Code
Details
200
Response body
{
"message": "Đèn đã được điều khiển!",
"data": {
"device": "light",
"status": "OFF",
"values": true
}
}
Download
Response headers
connection: keep-alive
content-length: 105
content-type: application/json; charset=utf-8
date: Fri,26 Sep 2025 09:01:46 GMT
etag: W/"69-kfPYShTDazXrex30XTyTFHm0wFI"
keep-alive: timeout=5
x-powered-by: Express



Responses

Curl

```
curl -X 'POST' \
  'http://localhost:3000/api/controll/humi' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "device": "humi",
  "status": "ON",
  "values": true
}'
```

Request URL

```
http://localhost:3000/api/controll/humi
```

Server response

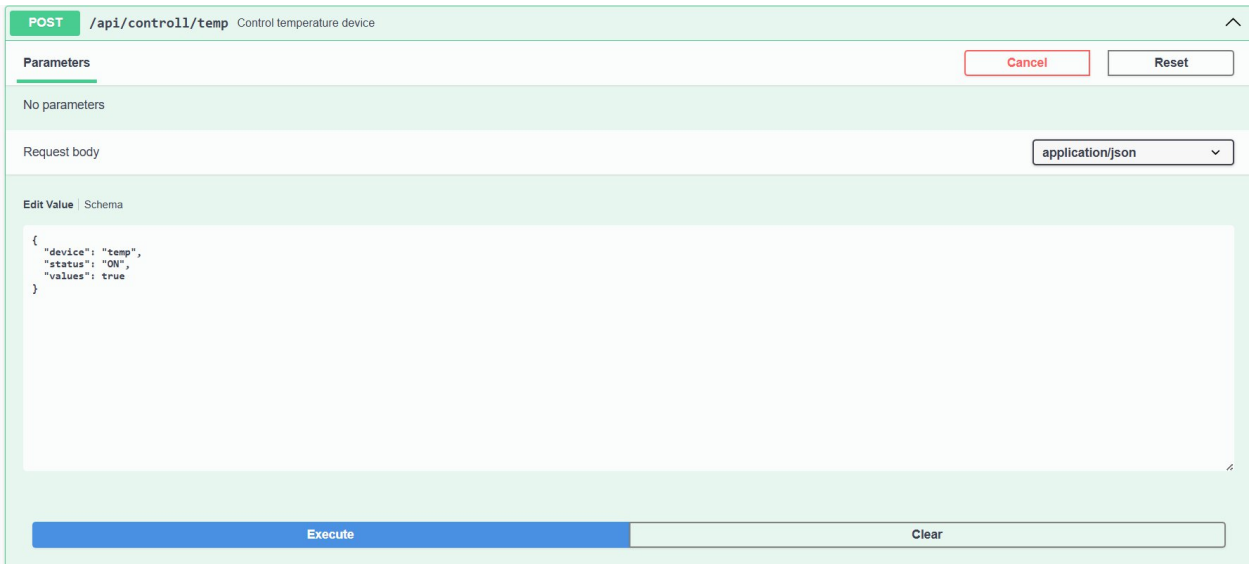
Code	Details
200	Response body

```
{
  "message": "Đèn đã được điều khiển!",
  "data": {
    "device": "humi",
    "status": "ON",
    "values": true
  }
}
```

Download

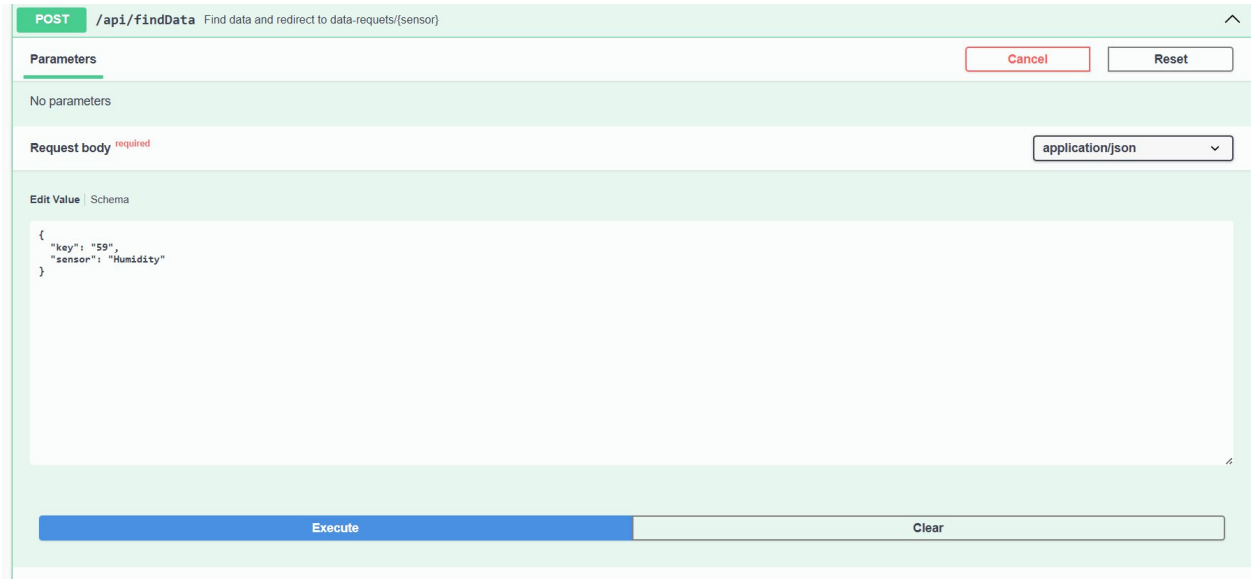
Response headers

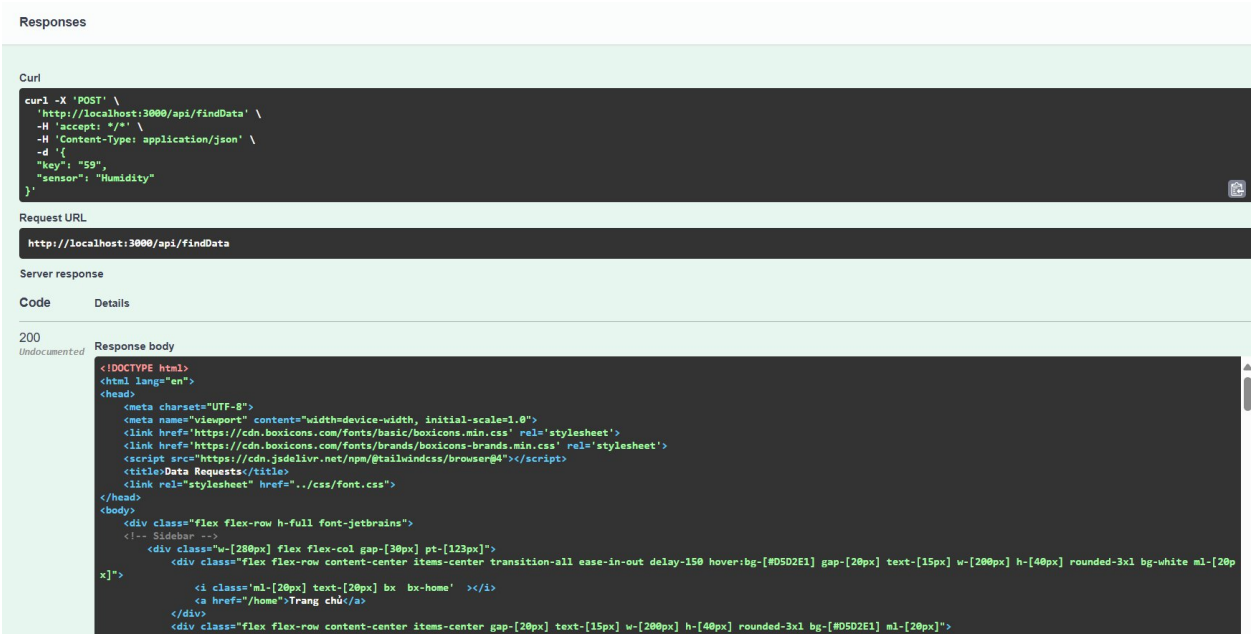
```
connection: keep-alive
content-length: 103
content-type: application/json; charset=utf-8
date: Fri,26 Sep 2025 09:02:27 GMT
etag: W/"67-QjycD+eldMpJZ50WhStNIo2tBzY"
keep-alive: timeout=5
x-powered-by: Express
```

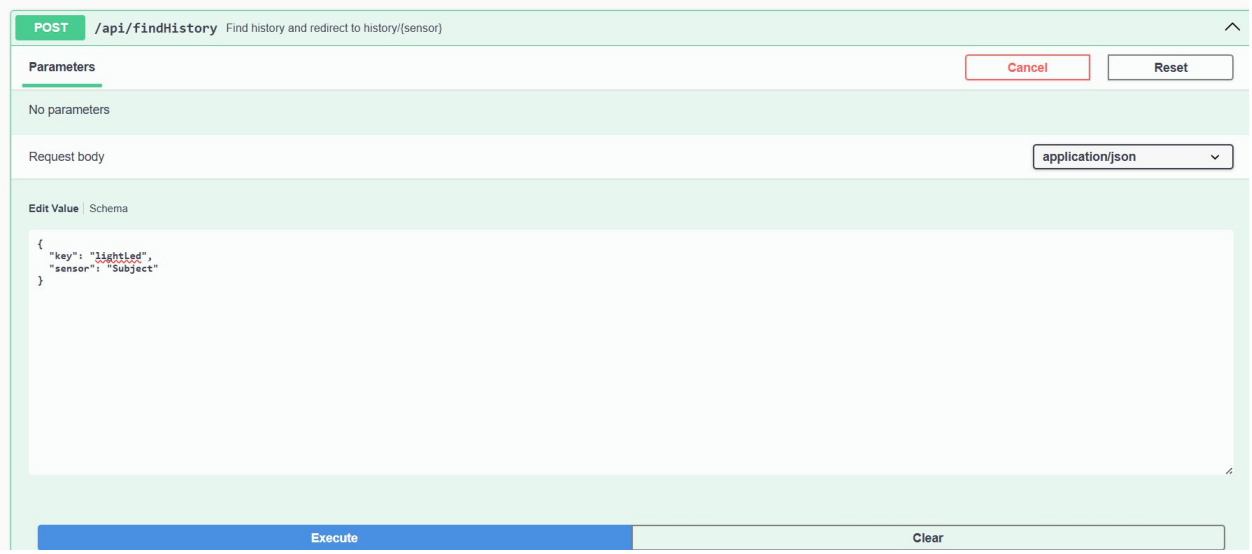

POST
/api/controll/temp
Control temperature device
Parameters
Cancel
Reset
No parameters
Request body
application/json
Edit Value | Schema
{
"device": "temp",
"status": "ON",
"values": true
}
Execute
Clear


Responses
Curl
curl -X 'POST' \
'http://localhost:3000/api/controll/temp' \
-H 'accept: */*' \
-H 'Content-Type: application/json' \
-d '{
"device": "temp",
"status": "ON",
"values": true
}'
Request URL
http://localhost:3000/api/controll/temp
Server response
Code
Details
200
Response body
{
"message": "Đèn đã được điều khiển!",
"data": {
"device": "temp",
"status": "ON",
"values": true
}
}
Download
Response headers

2. Tìm kiếm dữ liệu







Server response

Code Details

200
Undocumented

Response body

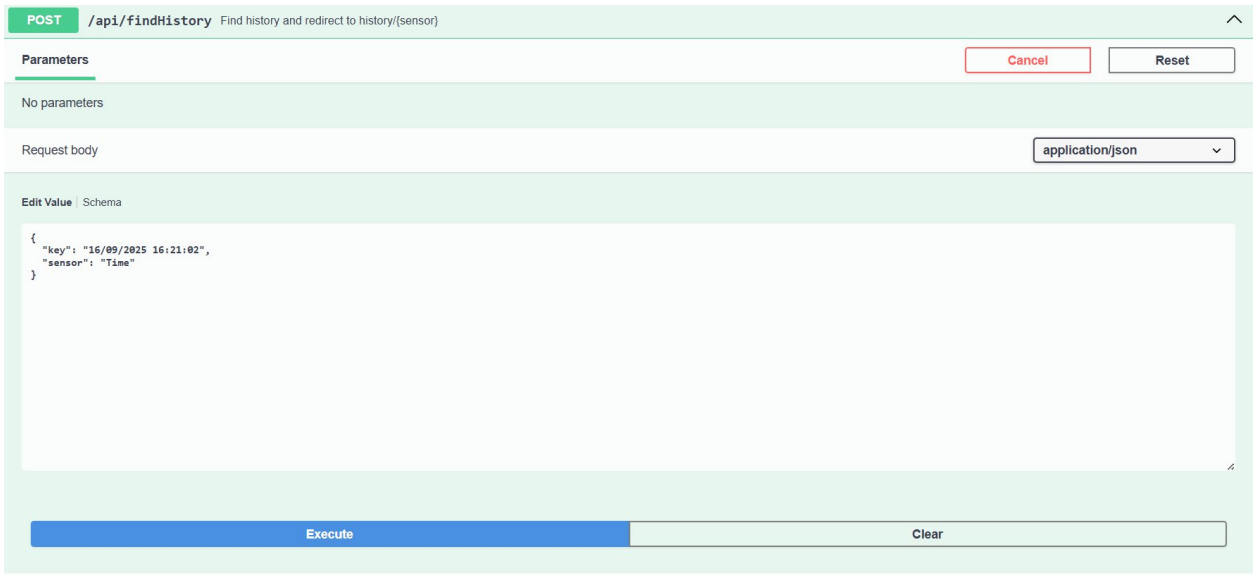
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href='https://cdn.boxicons.com/fonts/basic/boxicons.min.css' rel='stylesheet'>
    <link href='https://cdn.boxicons.com/fonts/brands/boxicons-brands.min.css' rel='stylesheet'>
    <script src="https://cdn.jsdelivr.net/npm/@tailwindcss/browser@4"></script>
    <link rel="stylesheet" href="../css/font.css">
    <title>History</title>
</head>
<body>
    <div class="flex flex-row h-full font-jetbrains">
    <!-- Sidebar -->
        <div class="w-[280px] flex flex-col gap-[30px] pt-[123px]">
            <div class="flex flex-row content-center items-center transition-all ease-in-out delay-150 hover:bg-[#D5D2E1] gap-[20px] text-[15px] w-[200px] h-[40px] rounded-3xl bg-white ml-[20p
x]">
                <i class='ml-[20px] text-[20px] bx  bx-home'  ></i>
                <a href="/home">Trang chủ</a>
            </div>
            <div class="flex flex-row content-center items-center transition-all ease-in-out delay-150 hover:bg-[#D5D2E1] gap-[20px] text-[15px] w-[200px] h-[40px] rounded-3xl bg-white ml-[20p
x]">
                <i class='ml-[20px] text-[20px] bx  bx-menu'  ></i>
                <a href="/data-requets">Bảng dữ liệu</a>
            </div>
            <div class="flex flex-row content-center items-center gap-[20px] text-[15px] w-[200px] h-[40px] rounded-3xl bg-[#D5D2E1] ml-[20px]">
                <i class='ml-[20px] text-[20px] bx  bx-history'  ></i>
```

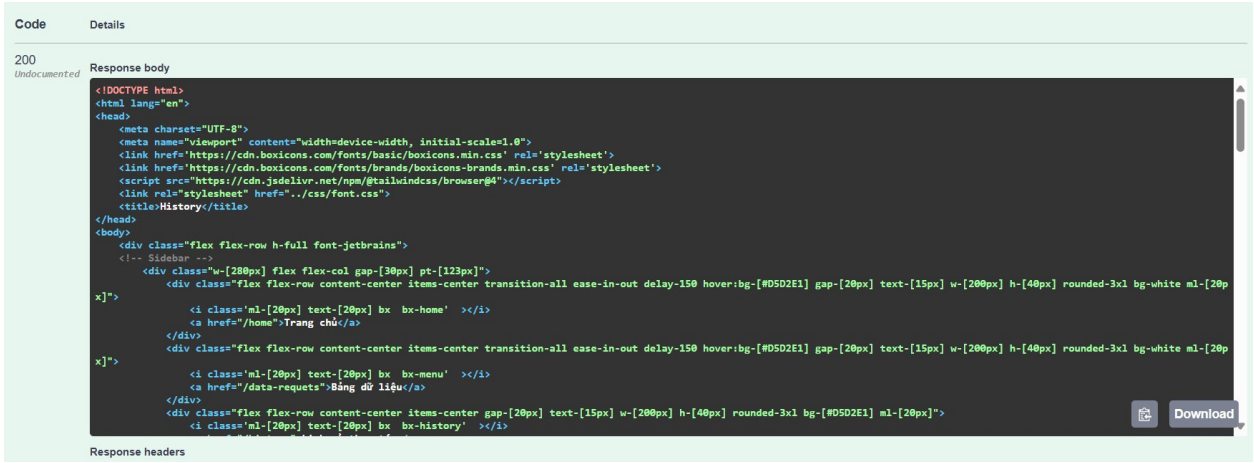
Download

Response headers

```
content-length: 14018
content-type: text/html; charset=utf-8
date: Fri,26 Sep 2025 09:13:55 GMT
etag: W/"36c2-euDKIM6gvQeAjIJoj3EYcNwRVnA"
x-powered-by: Express
```

Responses





IV – Kết luận:

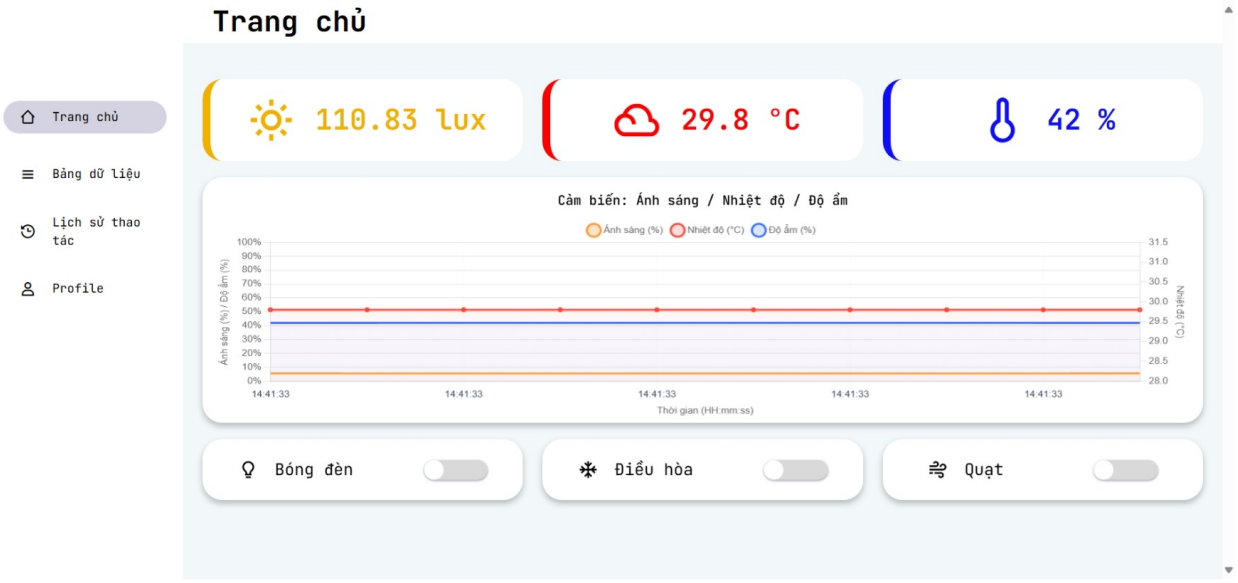
IV.1 - Chức năng đã triển khai:

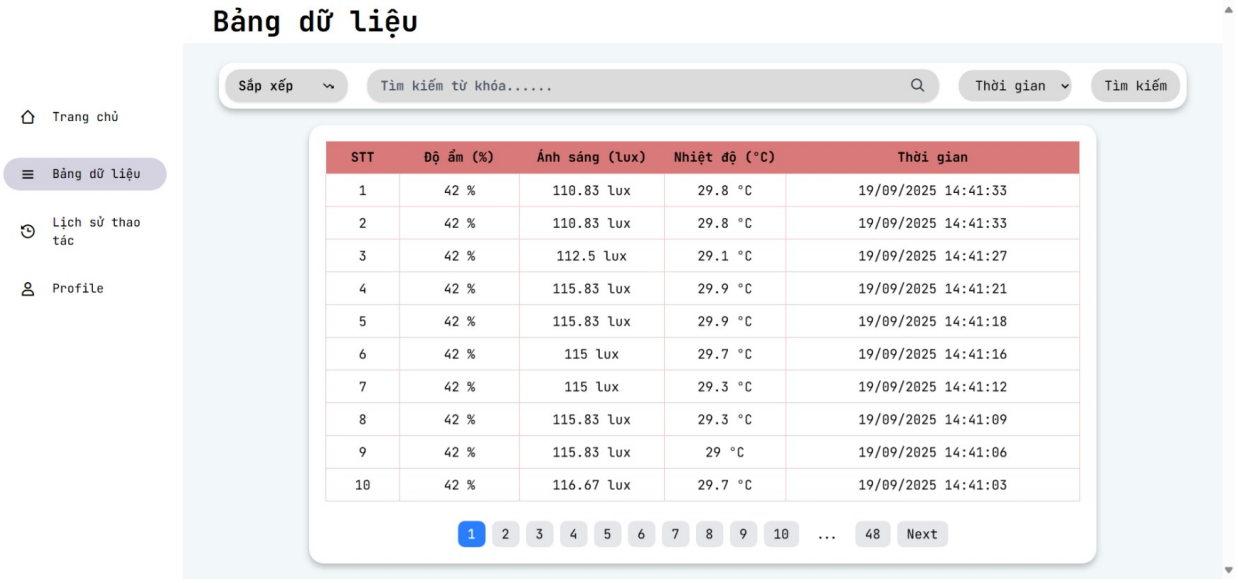
Hệ thống IoT giám sát đã được xây dựng thành công với đầy đủ các chức năng cốt lõi theo thiết kế ban đầu. Các chức năng chính đã được triển khai bao gồm:

- Thu thập dữ liệu cảm biến: Hệ thống đã triển khai thành công việc thu thập dữ liệu từ hai loại cảm biến chính.
- Điều khiển LED từ xa: Chức năng điều khiển LED đã được triển khai hoàn chỉnh với khả năng điều khiển độc lập 3 LED thông qua giao diện web.
- Lưu trữ và quản lý dữ liệu: Cơ sở dữ liệu MySQL đã được tích hợp thành công để lưu trữ dữ liệu cảm biến và lịch sử hành động.

- Hiển thị dữ liệu trực quan: Tích hợp Chart.js đã được thực hiện thành công để hiển thị dữ liệu cảm biến dưới dạng biểu đồ line chart.
- API RESTful: Hệ thống API đã được xây dựng hoàn chỉnh với Nodejs, cung cấp các endpoint cần thiết cho frontend. API được thiết kế theo chuẩn REST với cấu trúc response thống nhất và xử lý lỗi đầy đủ.

IV.2 - Giao diện người dùng:





Trang chủ

Bảng dữ liệu

Lịch sử thao tác

Profile

Lịch sử thao tác

Sắp xếp | Tìm kiếm từ khóa..... | Tất cả | Tìm kiếm

STT	Thiết bị	Trạng thái hoạt động	Thời gian
1	lightLed	ON	16/09/2025 16:21:02
2	humiLed	ON	16/09/2025 16:21:11
3	tempLed	ON	16/09/2025 16:21:14
4	lightLed	OFF	16/09/2025 16:21:26
5	humiLed	OFF	16/09/2025 16:21:30
6	tempLed	OFF	16/09/2025 16:21:33
7	tempLed	ON	16/09/2025 17:15:02
8	humiLed	ON	16/09/2025 17:15:05
9	lightLed	ON	16/09/2025 17:15:08
10	lightLed	OFF	16/09/2025 17:15:23

1 2 3 4 5 6 7 Next

Trang chủ

Bảng dữ liệu

Lịch sử thao tác

Profile

Profile

Họ và tên
Lương Tiến Đạt

Số điện thoại
0378775109

PDF
Đạt Lương IOT

Github
Git hub IOT

Document API
Đạt Lương IOT