HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

BÁO CÁO BÀI TẬP LỚN
HỌC PHẦN IOT VÀ ỨNG DỤNG

ĐỀ TÀI:

TRIỂN KHAI HỆ THỐNG QUẢN LÝ THIẾT BỊ IOT

Nhóm: 16

Sinh viên thực hiện:

Trần Hữu Phúc B22DCCN634

Giảng viên hướng dẫn

Ts. Nguyễn Quốc Uy

HÀ NỘI 2025

MỤC LỤC

DANH MỤC HÌNH ẢNH VÀ BẢNG BIỂU

CHƯƠNG 1: GIỚI THIỆU

1.1. Mục đích dự án

Trong thời đại công nghệ hiện đại, việc giám sát và điều khiển các thiết bị từ xa đã trở thành nhu cầu thiết yếu trong nhiều lĩnh vực như nông nghiệp thông minh, nhà thông minh, và các hệ thống công nghiệp. Dự án "**Hệ Thống Quản Lý Thiết Bị IoT**" được phát triển nhằm tạo ra một giải pháp toàn diện cho việc thu thập, lưu trữ và quản lý dữ liệu từ các thiết bị cảm biến IoT.

Mục đích chính của dự án là xây dựng một hệ thống giám sát thời gian thực cho phép theo dõi các thông số môi trường quan trọng như nhiệt độ, độ ẩm và ánh sáng. Hệ thống được thiết kế để có khả năng điều khiển từ xa các thiết bị LED thông qua giao diện web, tạo ra một môi trường giám sát và điều khiển tích hợp.

Dự án tập trung vào việc cung cấp một nền tảng đáng tin cậy và dễ sử dụng, giúp người dùng có thể theo dõi trạng thái môi trường và thực hiện các thao tác điều khiển một cách hiệu quả. Thông qua việc tích hợp các công nghệ hiện đại như MQTT, MongoDB và REST API, hệ thống đảm bảo tính linh hoạt và khả năng mở rộng trong tương lai.

1.2. Phạm vi ứng dụng

Hệ thống giám sát IoT được thiết kế để phục vụ nhiều lĩnh vực ứng dụng khác nhau. Trong lĩnh vực nông nghiệp thông minh, hệ thống có thể được sử dụng để giám sát điều kiện môi trường trong nhà kính, đảm bảo cây trồng phát triển trong điều kiện tối ưu. Các cảm biến nhiệt độ và độ ẩm giúp theo dõi môi trường sống của cây trồng, trong khi cảm biến ánh sáng hỗ trợ việc điều chỉnh hệ thống chiếu sáng tự động.

Trong môi trường công nghiệp, hệ thống có thể được triển khai để giám sát điều kiện làm việc tại các nhà máy, kho bãi hoặc phòng thí nghiệm. Việc theo dõi liên tục các thông số môi trường giúp đảm bảo an toàn lao động và bảo quản thiết bị trong điều kiện thích hợp. Hệ thống điều khiển LED có thể

được sử dụng để tạo ra các tín hiệu cảnh báo hoặc điều khiển hệ thống chiếu sáng tự động.

Ứng dụng trong nhà thông minh là một lĩnh vực tiềm năng khác của hệ thống. Việc giám sát nhiệt độ và độ ẩm trong nhà giúp điều chỉnh hệ thống điều hòa không khí và thông gió một cách thông minh. Hệ thống điều khiển LED có thể tích hợp với hệ thống chiếu sáng thông minh, tạo ra môi trường sống thoải mái và tiết kiệm năng lượng.

Hệ thống cũng có thể được mở rộng để phục vụ các ứng dụng nghiên cứu khoa học, giáo dục và phát triển sản phẩm. Khả năng lưu trữ và phân tích dữ liệu lịch sử tạo điều kiện cho việc nghiên cứu các xu hướng thay đổi môi trường theo thời gian.

1.3. Công nghệ sử dụng

Hệ thống được xây dựng dựa trên kiến trúc client-server với các thành phần công nghệ được lựa chọn phù hợp với từng lớp chức năng.

1.3.1. Phần cứng và vi điều khiển

Thiết bị ESP32 được chọn làm vi điều khiển chính cho phần cứng của hệ thống. ESP32 là một vi điều khiển 32-bit mạnh mẽ với khả năng kết nối WiFi tích hợp và hỗ trợ Bluetooth. Việc lựa chọn ESP32 dựa trên các ưu điểm như khả năng xử lý cao, tích hợp WiFi, giá thành hợp lý và cộng đồng phát triển lớn.

Cảm biến DHT11 được sử dụng để đo nhiệt độ và độ ẩm môi trường. DHT11 là một cảm biến kỹ thuật số với độ chính xác phù hợp cho các ứng dụng giám sát môi trường cơ bản. Cảm biến ánh sáng được thiết kế sử dụng ADC (Analog-to-Digital Converter) tích hợp của ESP32 để đọc giá trị điện áp từ cảm biến ánh sáng tương tự.

Hệ thống điều khiển LED bao gồm ba LED được kết nối với các chân GPIO của ESP32. Việc sử dụng ba LED độc lập cho phép thực hiện các chức năng điều khiển đa dạng và tạo ra các tín hiệu trạng thái khác nhau.

1.3.2. Giao thức giao tiếp

MQTT (Message Queuing Telemetry Transport) được chọn làm giao thức giao tiếp chính giữa thiết bị ESP32 và backend server. MQTT là một giao thức nhẹ và hiệu quả, đặc biệt phù hợp cho các ứng dụng IoT với khả năng hoạt động tốt trên các kết nối mạng không ổn định.

Việc sử dụng MQTT với kết nối TLS (Transport Layer Security) đảm bảo tính bảo mật trong quá trình truyền dữ liệu. HiveMQ Cloud được chọn làm MQTT broker nhờ khả năng cung cấp dịch vụ ổn định.

1.3.3. Backend và cơ sở dữ liệu

Flask được sử dụng làm framework web để xây dựng REST API backend. Flask được lựa chọn vì tính đơn giản, linh hoạt và khả năng tích hợp tốt với các thư viện Python khác. Việc sử dụng Flask-RESTX giúp tạo ra tài liệu API tự động với Swagger UI, tăng tính tiện dụng cho việc phát triển và bảo trì.

MongoDB được chọn làm cơ sở dữ liệu NoSQL cho việc lưu trữ dữ liệu cảm biến và lịch sử hoạt động. MongoDB phù hợp với đặc điểm dữ liệu của hệ thống IoT, cho phép lưu trữ các document JSON linh hoạt và hỗ trợ các truy vấn phức tạp.

PyMongo được sử dụng làm driver Python để kết nối và thao tác với MongoDB. Thư viện này cung cấp các tính năng đầy đủ để thực hiện các thao tác CRUD và các truy vấn nâng cao.

1.3.4. Frontend và giao diện người dùng

Giao diện web được xây dựng bằng HTML5, CSS3 và JavaScript thuần. Việc sử dụng JavaScript thuần thay vì các framework phức tạp giúp giảm độ phức tạp của hệ thống và tăng tốc độ tải trang.

CHƯƠNG 2: THIẾT KẾ HỆ THỐNG

2.1. Phân tích yêu cầu

2.1.1. Yêu cầu chức năng

Hệ thống cần đáp ứng các yêu cầu chức năng chính bao gồm thu thập dữ liệu cảm biến, lưu trữ và quản lý dữ liệu, điều khiển thiết bị từ xa, và cung cấp giao diện giám sát. Cụ thể, hệ thống phải có khả năng thu thập dữ liệu từ ba loại cảm biến chính: cảm biến nhiệt độ DHT11, cảm biến độ ẩm DHT11, và cảm biến ánh sáng tương tự.

Dữ liệu cảm biến cần được thu thập liên tục với tần suất 1 giây và truyền tải qua giao thức MQTT đến backend server. Hệ thống phải có khả năng lưu trữ dữ liệu vào cơ sở dữ liệu MongoDB với cấu trúc document linh hoạt, cho phép truy vấn và phân tích dữ liệu hiệu quả.

Chức năng điều khiển thiết bị bao gồm khả năng bật/tắt ba LED độc lập thông qua giao diện web. Các lệnh điều khiển được gửi qua MQTT đến thiết bị ESP32 và được thực thi ngay lập tức. Hệ thống cần ghi lại lịch sử tất cả các thao tác điều khiển để phục vụ mục đích giám sát và kiểm tra.

Giao diện web phải cung cấp khả năng hiển thị dữ liệu cảm biến theo thời gian thực, biểu đồ trực quan hóa dữ liệu, bảng dữ liệu với chức năng tìm kiếm và lọc, và các điều khiển LED trực quan.

2.1.2. Yêu cầu phi chức năng

Hệ thống phải đảm bảo tính ổn định và độ tin cậy cao trong việc thu thập và xử lý dữ liệu. Thời gian phản hồi của giao diện web không được vượt quá 2 giây cho các thao tác thông thường. Hệ thống cần có khả năng xử lý ít nhất 1000 bản ghi dữ liệu mỗi giờ mà không bị quá tải.

Về mặt bảo mật, tất cả kết nối MQTT phải sử dụng giao thức TLS để mã hóa dữ liệu trong quá trình truyền tải. Hệ thống cần có cơ chế xác thực và phân

quyền phù hợp để đảm bảo chỉ người dùng được ủy quyền mới có thể truy cập và điều khiển hệ thống.

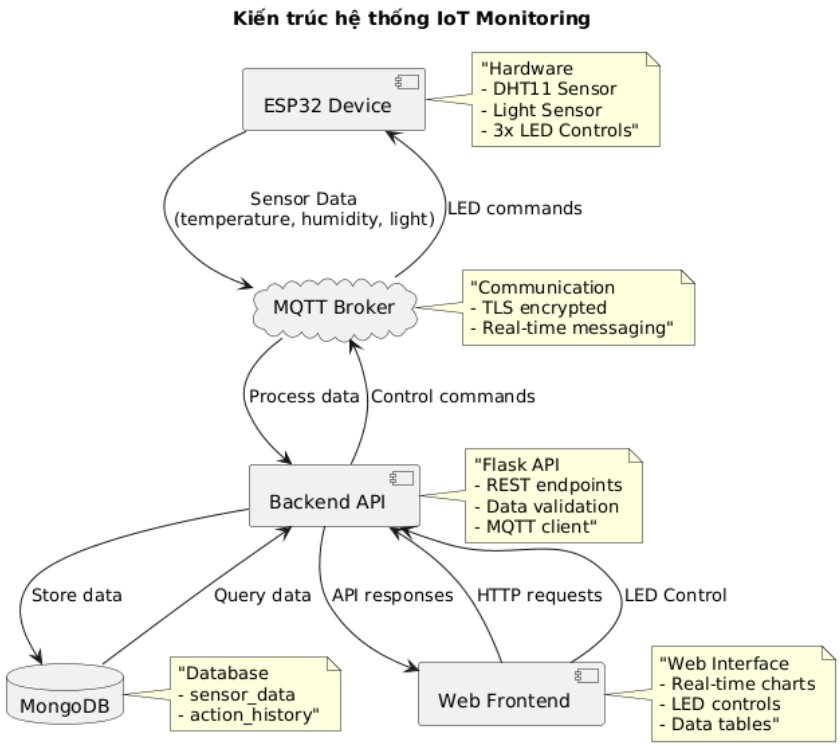
Khả năng mở rộng là một yêu cầu quan trọng khác. Hệ thống phải được thiết kế để có thể dễ dàng thêm mới các loại cảm biến và thiết bị điều khiển mà không cần thay đổi cấu trúc cốt lõi. Kiến trúc microservice được áp dụng để tách biệt các thành phần chức năng, tạo điều kiện cho việc phát triển và bảo trì độc lập.

2.2. Tổng quan kiến trúc hệ thống

2.2.1. Mô hình kiến trúc tổng thể

Hệ thống được xây dựng dựa trên mô hình kiến trúc phân tầng (layered architecture) với các tầng chức năng rõ ràng:

Biểu đồ 1: Mô hình kiến trúc tổng thể



Tầng Hardware (Tầng thiết bị): Bao gồm thiết bị ESP32 tích hợp cảm biến DHT11, cảm biến ánh sáng và hệ thống điều khiển LED. Tầng này thu thập dữ liệu cảm biến và thực thi lệnh điều khiển từ xa.

Tầng Truyền thông: Sử dụng giao thức MQTT (Message Queuing Telemetry Transport) để đảm bảo giao tiếp thời gian thực giữa thiết bị ESP32 và backend. MQTT được lựa chọn do tính năng nhẹ, đáng tin cậy và phù hợp với ứng dụng IoT.

Tầng Backend: Được xây dựng bằng Flask framework với kiến trúc RESTful API, tích hợp MongoDB để lưu trữ dữ liệu và cung cấp các dịch vụ xử lý dữ liệu phức tạp.

Tầng Frontend: Giao diện web được phát triển bằng HTML, CSS và JavaScript thuần, cung cấp trải nghiệm người dùng trực quan và tương tác.

2.2.2. Luồng dữ liệu trong hệ thống

Luồng dữ liệu trong hệ thống được thiết kế theo hướng một chiều từ thiết bị cảm biến đến người dùng cuối:

Bước 1: Thu thập dữ liệu: ESP32 đọc dữ liệu từ cảm biến mỗi giây và đóng gói thành định dạng JSON

Bước 2: Truyền tải: Dữ liệu được gửi qua MQTT broker đến backend API

Bước 3: Xử lý: Backend nhận dữ liệu, kiểm tra tính hợp lệ, tính toán trạng thái và lưu trữ vào MongoDB

Bước 4: Hiển thị: Frontend gọi API để lấy dữ liệu và cập nhật giao diện thời gian thực

2.2.3. Các thành phần công nghệ chính

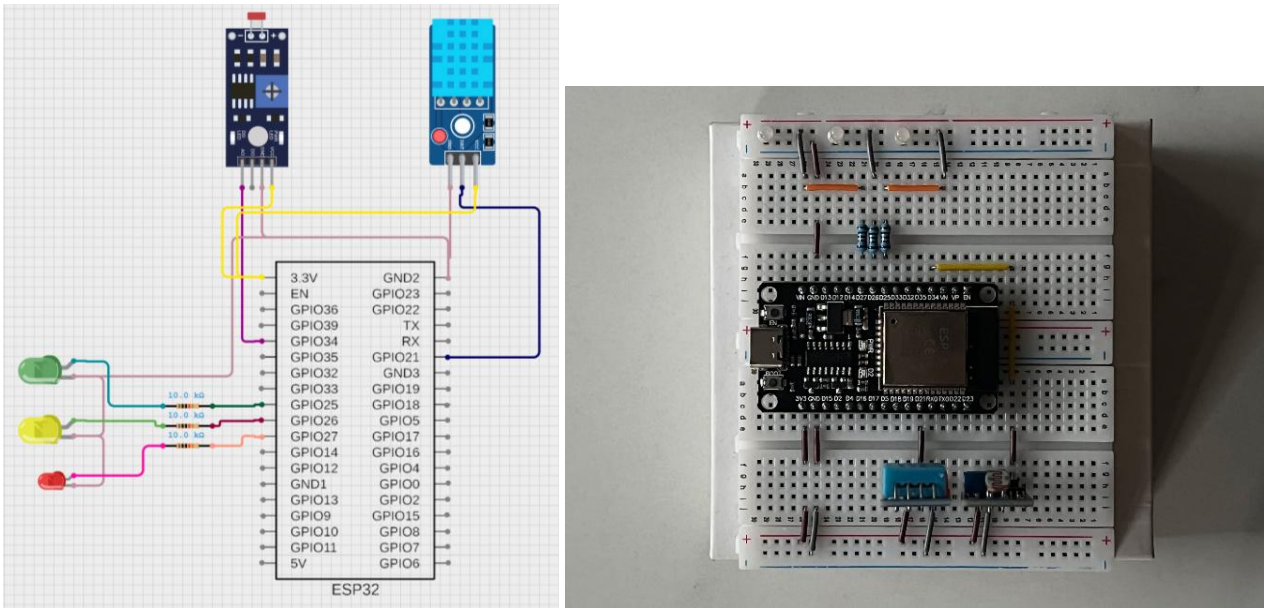
ESP32 (Espressif Systems): Vi điều khiển 32-bit với khả năng kết nối WiFi tích hợp, phù hợp cho ứng dụng IoT. ESP32 cung cấp 34 chân GPIO, hỗ trợ ADC 12-bit và có khả năng xử lý mạnh với dual-core processor.

MQTT (Message Queuing Telemetry Transport): Giao thức truyền thông publish-subscribe nhẹ, được thiết kế đặc biệt cho ứng dụng IoT. MQTT sử dụng mô hình broker-client, đảm bảo giao tiếp đáng tin cậy trong môi trường mạng không ổn định.

Flask (Python Web Framework): Framework web nhẹ và linh hoạt, phù hợp cho phát triển API REST. Flask cung cấp cơ chế routing, xử lý request và tích hợp dễ dàng với thư viện Python khác.

MongoDB (NoSQL Database): Cơ sở dữ liệu hướng tài liệu, phù hợp cho lưu trữ dữ liệu cảm biến có cấu trúc linh hoạt. MongoDB hỗ trợ truy vấn phức tạp, aggregation và mở rộng ngang.

2.3. Thiết kế phần cứng



Hình 1: Thiết kế phần cứng

Các linh kiện được kết nối với ESP trên các chân

- DHT Pin: 21
- ADC Pin: 34
- LED1 Pin: 25
- LED2 Pin: 26
- LED3 Pin: 27

2.4. Giao diện người dùng

2.4.1. Trang chủ (Homepage)

Trang chủ được thiết kế như trung tâm điều khiển chính của hệ thống, cung cấp cái nhìn tổng quan về trạng thái cảm biến và các chức năng điều khiển cơ bản.

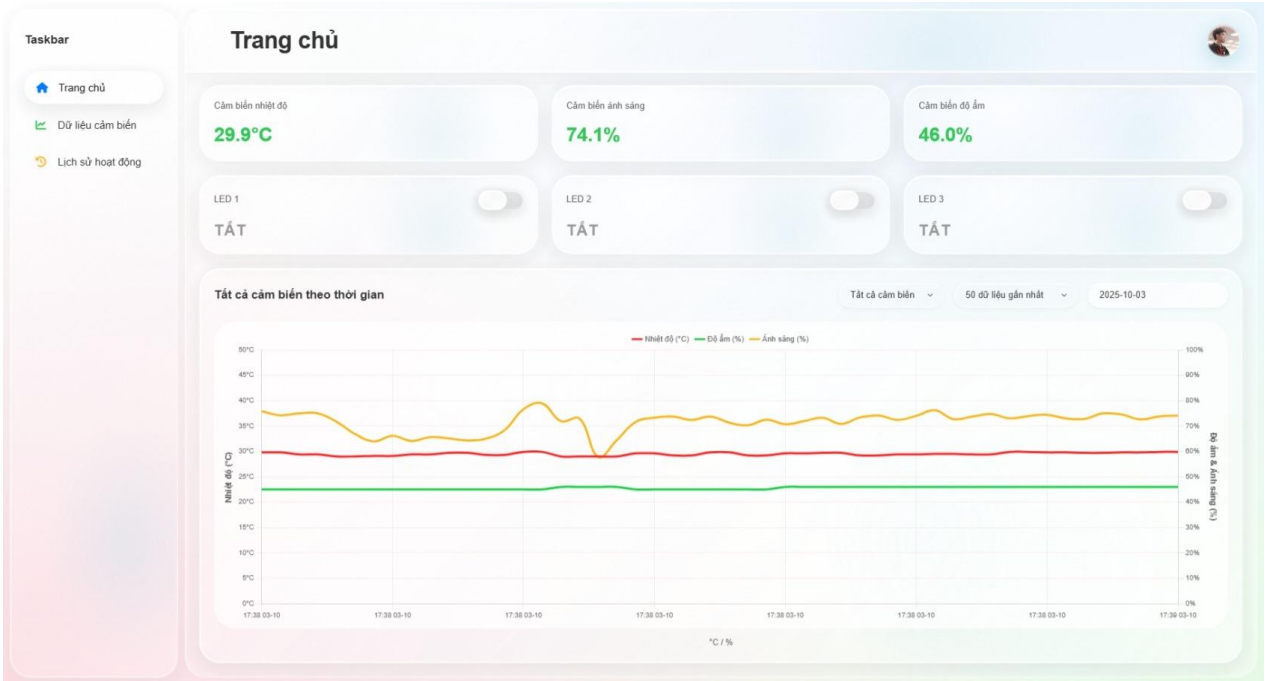
Mỗi loại cảm biến được hiển thị trong card riêng biệt với:

- Tên cảm biến và đơn vị đo
- Giá trị hiện tại được cập nhật thời gian thực
- Màu sắc trạng thái (xanh: bình thường, vàng: cảnh báo, đỏ: nguy hiểm)

Hệ thống điều khiển LED được tích hợp trực tiếp trên trang chủ với:

- Toggle switch cho mỗi LED (LED1, LED2, LED3)
- Trạng thái hiện tại (BẬT/TẮT)

Biểu đồ thời gian thực: Biểu đồ sử dụng Chart.js để hiển thị xu hướng dữ liệu cảm biến trong khoảng thời gian ngắn hoặc hiển thị toàn bộ dữ liệu, giúp người dùng theo dõi thay đổi của thông số môi trường.

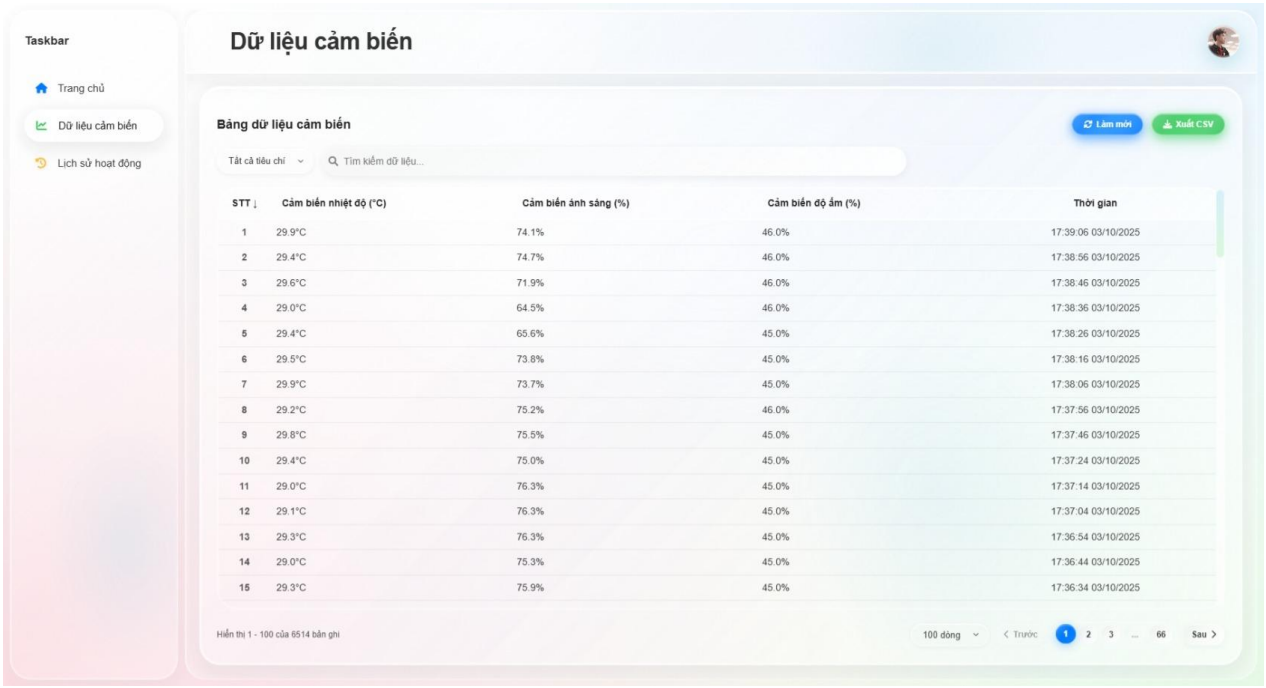


Hình 2: Giao diện Trang chủ

2.4.2. Trang dữ liệu cảm biến (Sensor Data)

Trang dữ liệu cảm biến được thiết kế để quản lý và phân tích dữ liệu lịch sử với CRUD Operations:

- Bảng dữ liệu có thể sắp xếp theo các cột khác nhau
- Hệ thống phân trang với điều khiển kích thước trang
- Tìm kiếm theo nhiều tiêu chí (nhiệt độ, độ ẩm, ánh sáng, thời gian)



Hình 3: Giao diện trang Dữ liệu cảm biến

2.4.3. Trang lịch sử hoạt động (Action History)

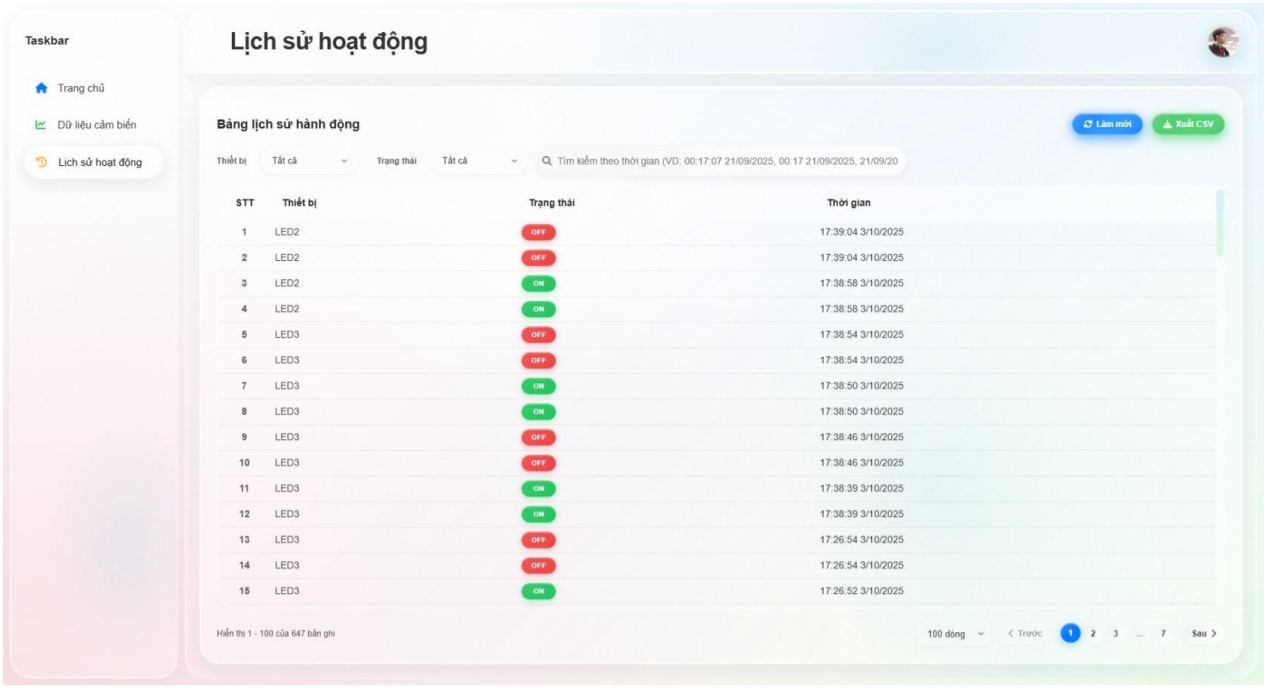
Trang lịch sử hoạt động theo dõi tất cả các thao tác điều khiển LED.

Bảng lịch sử hoạt động:

- Hiển thị chi tiết từng hành động với timestamp
- Thông tin thiết bị (LED1, LED2, LED3)

Lọc nâng cao:

- Lọc theo thiết bị cụ thể
- Lọc theo trạng thái (ON/OFF)
- Tìm kiếm theo thời gian với nhiều định dạng

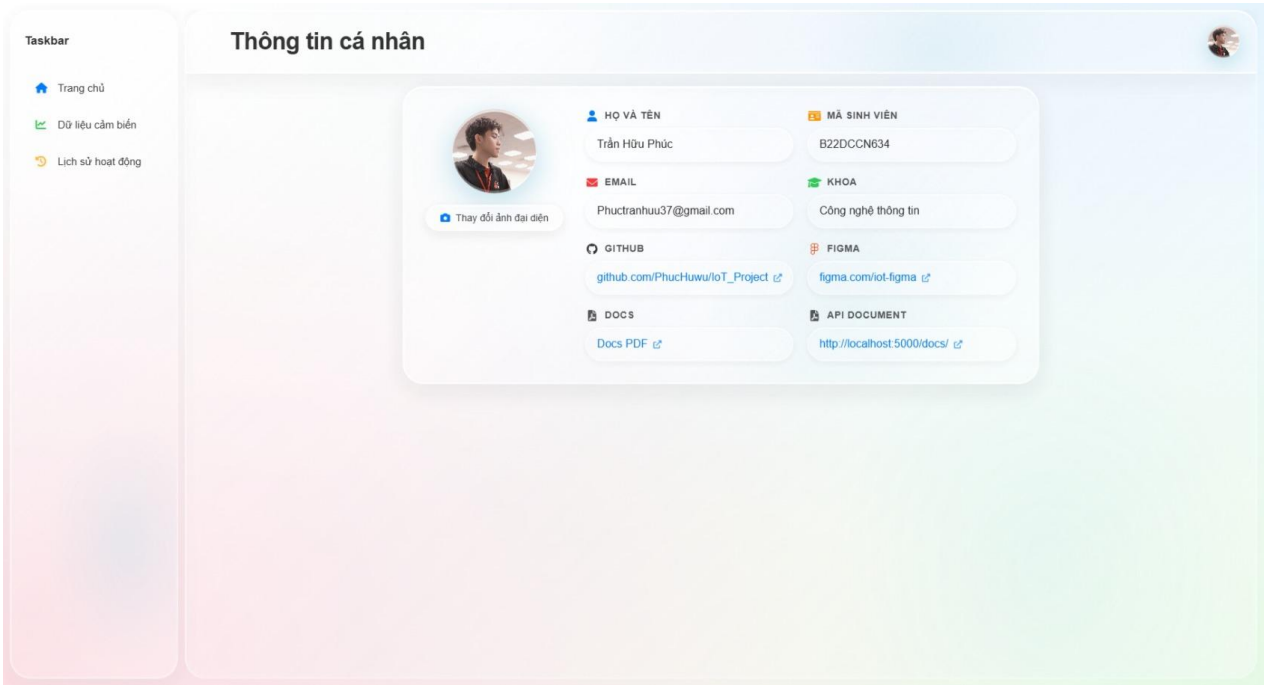


Hình 4: Giao diện trang Lịch sử hoạt động

2.4.4. Trang hồ sơ cá nhân (Profile)

Trang profile cung cấp giao diện hiển thị thông tin cá nhân người dùng:

- Hiển thị thông tin người dùng hiện tại
- Quản lý avatar

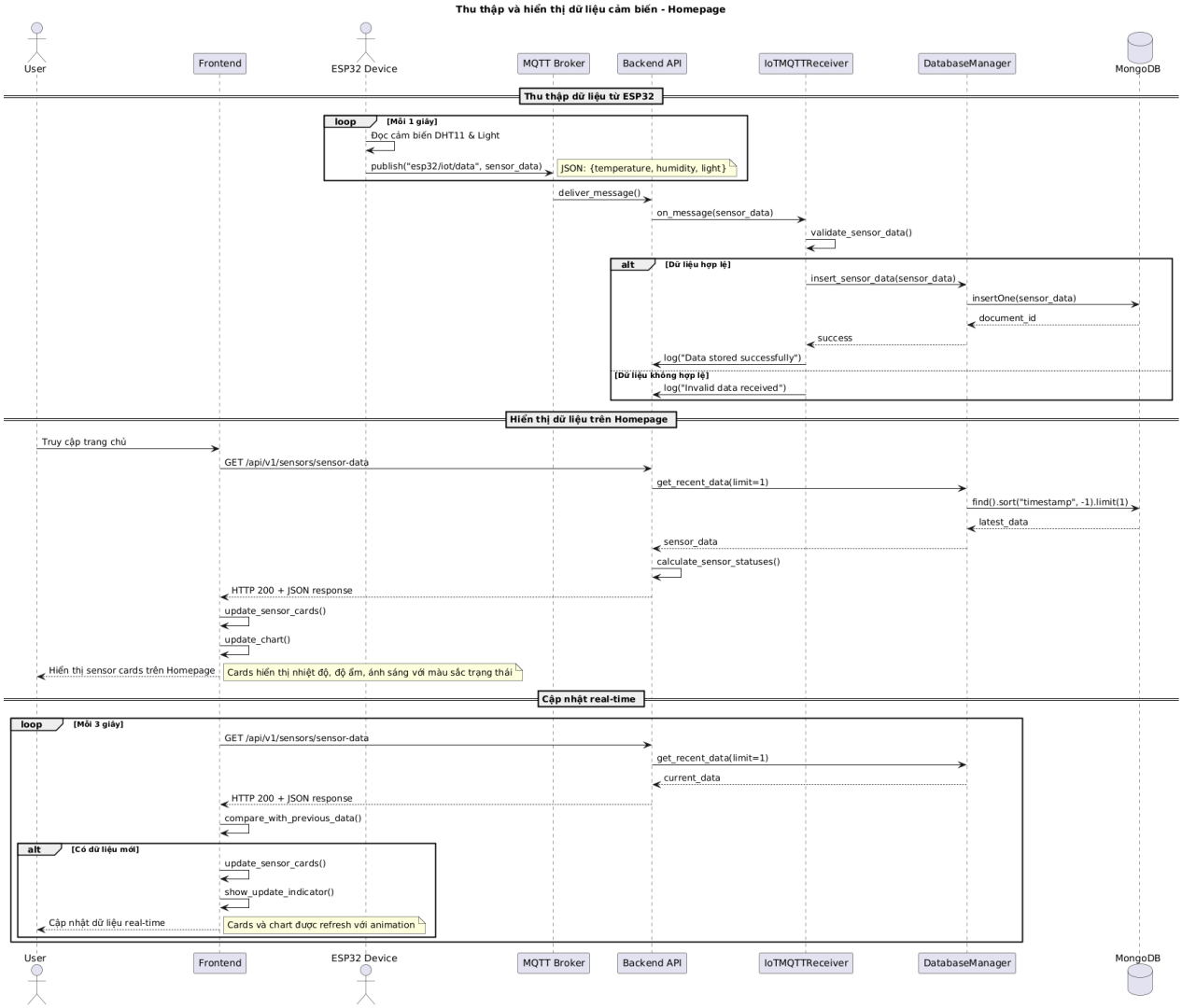


Hình 5: Giao diện trang Thông tin cá nhân

2.5. Luồng xử lý hệ thống

2.5.1. Luồng thu thập và hiển thị dữ liệu cảm biến

Biểu đồ 2: Luồng thu thập và hiển thị dữ liệu cảm biến



Bước 1: Đọc dữ liệu từ cảm biến, ESP32 đọc dữ liệu từ cảm biến mỗi giây trong vòng lặp chính:

- Cảm biến DHT11: Đọc nhiệt độ và độ ẩm
- Cảm biến ánh sáng: Sử dụng ADC 12-bit đọc giá trị analog
- Chuyển đổi ADC thành phần trăm ánh sáng (0-100%)

Bước 2: Kiểm tra và đóng gói dữ liệu

- Kiểm tra tính hợp lệ (không phải NaN)
- Đóng gói thành JSON format

- Publish lên topic "esp32/iot/data"

Bước 3: Backend nhận và xử lý

- MQTTManager nhận message từ broker
- IoTMQTTReceiver xử lý dữ liệu
- DatabaseManager lưu trữ vào MongoDB
- Tự động thêm timestamp và metadata

Bước 4: Hiển thị dữ liệu real-time

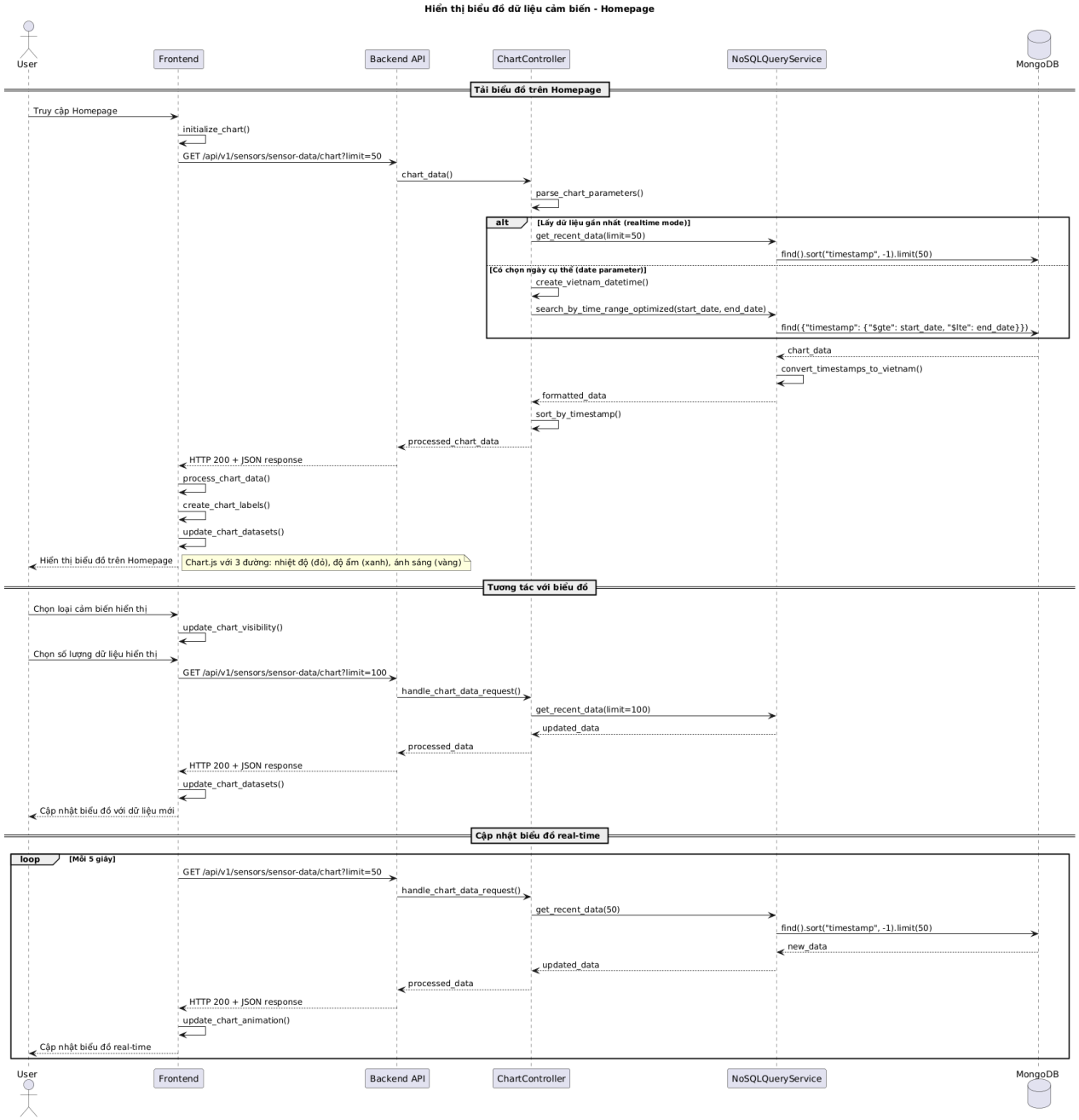
- Frontend gọi API /sensor-data mỗi 3 giây
- Backend trả về dữ liệu mới nhất từ MongoDB
- Frontend cập nhật sensor cards với giá trị mới
- Hiển thị trạng thái cảm biến (NORMAL/WARNING/DANGER)

Bước 5: Cập nhật biểu đồ và thống kê

- Frontend gọi API /sensor-data/chart để lấy dữ liệu biểu đồ
- Chart.js vẽ lại biểu đồ với dữ liệu mới
- Hiển thị xu hướng thay đổi của các thông số môi trường

2.5.2. Luồng hiển thị biểu đồ dữ liệu

Biểu đồ 3: Luồng hiển thị biểu đồ dữ liệu cảm biến



Luồng hiển thị biểu đồ dữ liệu cảm biến trên trang chủ.

Tải biểu đồ ban đầu:

- Frontend gọi API /sensor-data/chart với limit=50
- Backend xử lý request và lấy dữ liệu từ MongoDB
- Dữ liệu được sort theo timestamp và convert timezone

- Frontend sử dụng Chart.js tạo biểu đồ với 3 đường (nhiệt độ, độ ẩm, ánh sáng)

Tương tác với biểu đồ:

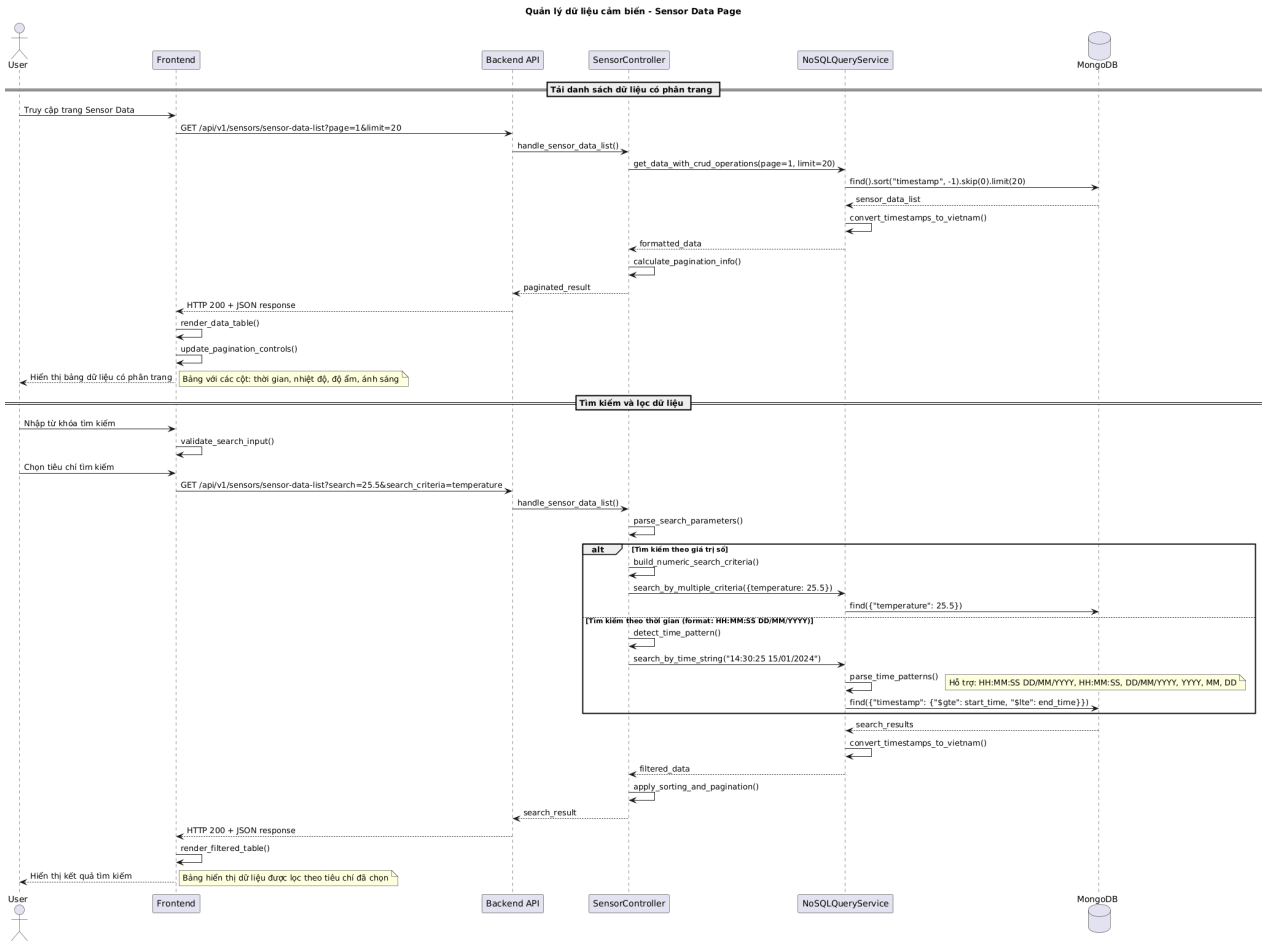
- Người dùng chọn loại cảm biến hiển thị
- Người dùng chọn số lượng dữ liệu (limit)
- Frontend gọi API với parameters mới
- Biểu đồ được cập nhật với dữ liệu mới

Cập nhật real-time:

- Frontend tự động gọi API mỗi 5 giây
- Backend trả về dữ liệu mới nhất

2.5.3. Luồng quản lý dữ liệu cảm biến

Biểu đồ 4: Luồng quản lý dữ liệu cảm biến



Luồng quản lý dữ liệu cảm biến trên trang Sensor Data.

Tải danh sách dữ liệu:

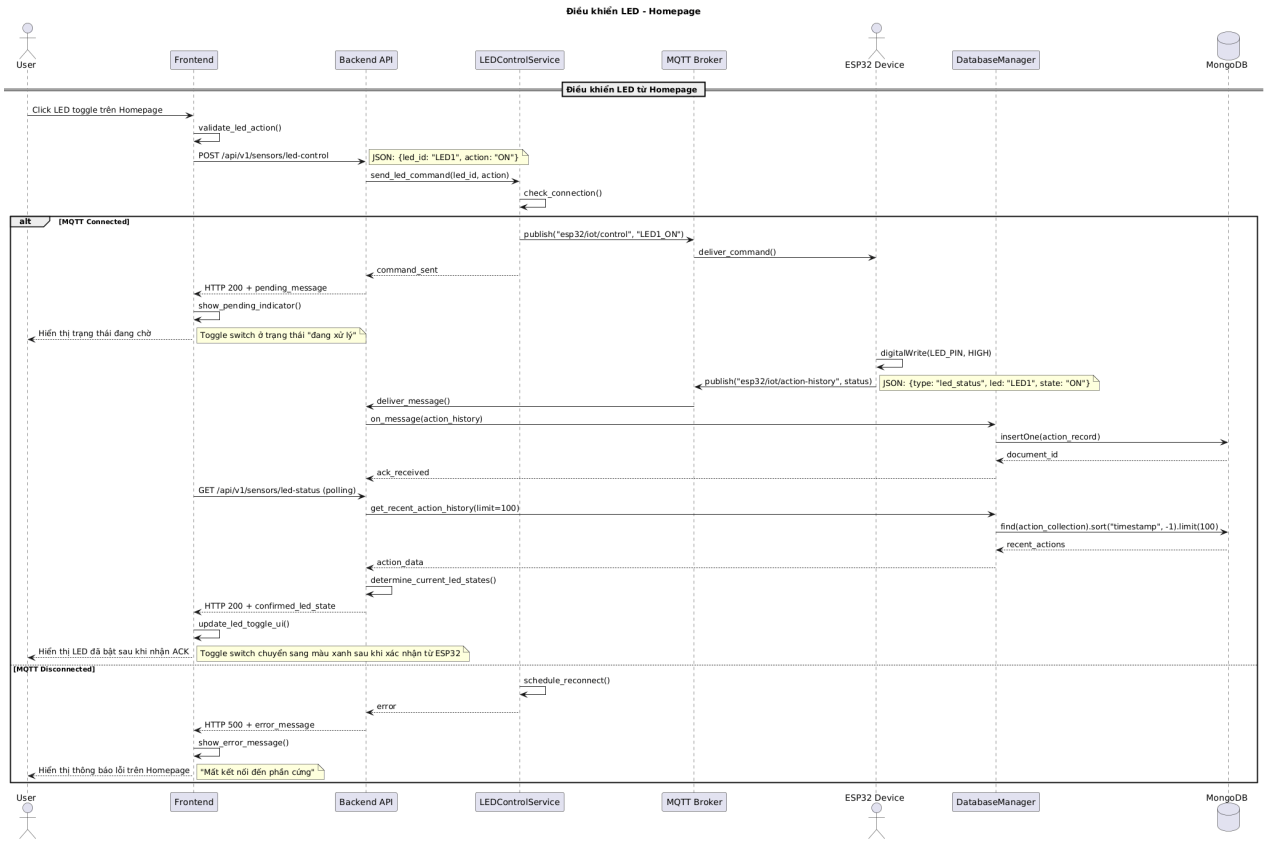
- Frontend gọi API /sensor-data-list với pagination
- Backend query MongoDB với sorting và pagination
- Dữ liệu được format và convert timezone
- Frontend hiển thị bảng với các cột: thời gian, nhiệt độ, độ ẩm, ánh sáng

Tìm kiếm và lọc:

- Tìm kiếm theo giá trị số (nhiệt độ, độ ẩm, ánh sáng)
- Tìm kiếm theo thời gian với nhiều format
- Backend xử lý các tiêu chí tìm kiếm
- Kết quả được hiển thị với pagination và sorting

2.5.4. Luồng điều khiển LED

Biểu đồ 5: Luồng điều khiển LED



Luồng điều khiển LED cho phép người dùng điều khiển từ xa các thiết bị LED thông qua giao diện web.

Bước 1: Người dùng thực hiện thao tác

- Click toggle switch trên giao diện web
- JavaScript gọi API /led-control với led_id và action
- Frontend hiển thị trạng thái pending (đang chờ xử lý)
- Toggle switch chuyển sang chế độ loading/processing

Bước 2: Backend xử lý và gửi lệnh

- API endpoint validate parameters (led_id, action)
- LEDControlService tạo MQTT command
- Publish lệnh lên topic "esp32/iot/control"

Bước 3: ESP32 nhận và thực thi

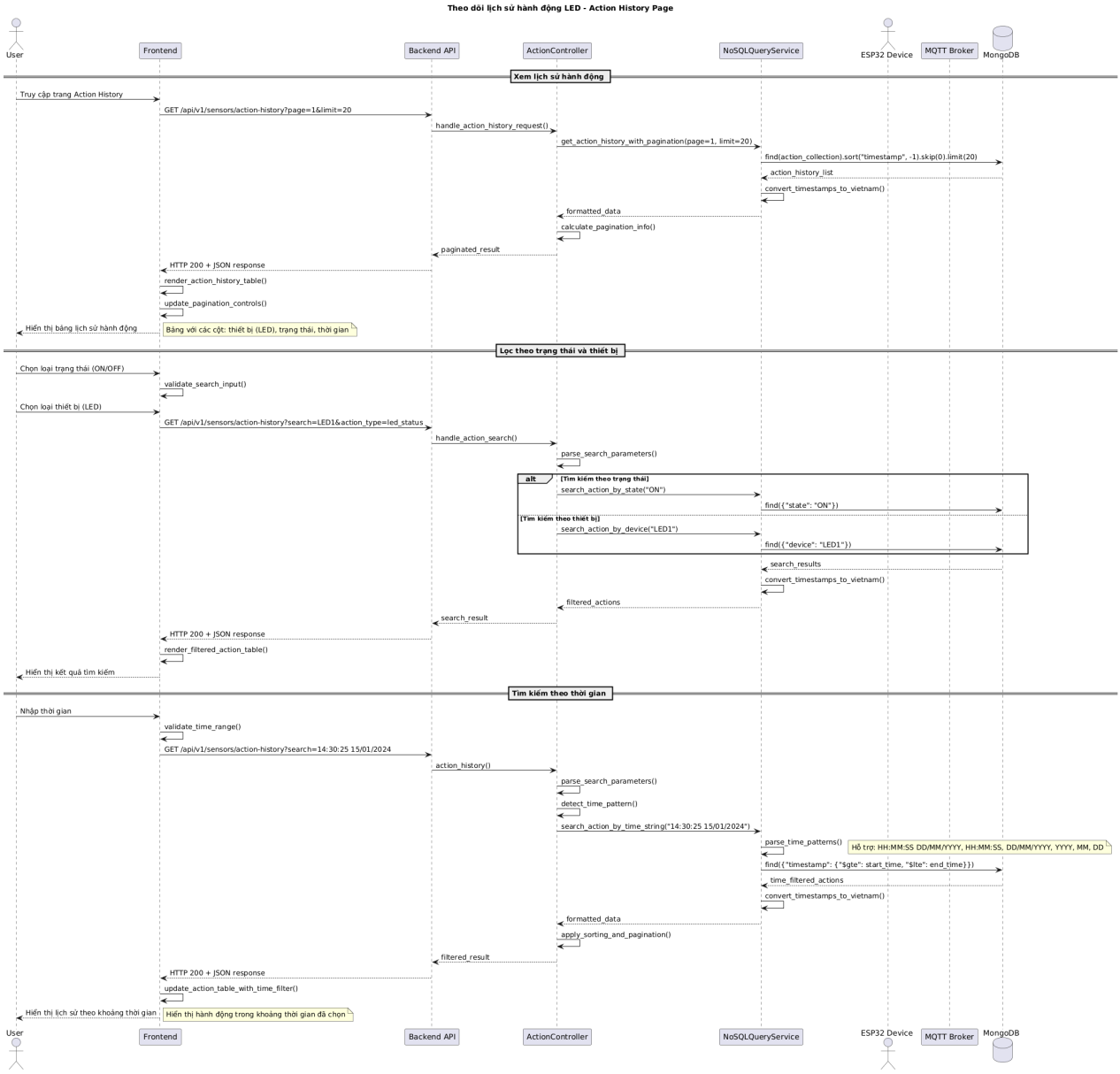
- MQTT callback xử lý message nhận được
- Parse command (LED1_ON, LED2_OFF, v.v.)
- Thực hiện digitalWrite để điều khiển LED vật lý
- Publish lịch sử hoạt động lên topic "esp32/iot/action-history"

Bước 4: Backend nhận và lưu lịch sử hoạt động

- Backend nhận action history từ ESP32
- Validate và lưu vào MongoDB collection action_history
- Dữ liệu chứa thông tin led_id, state và timestamp
- Frontend nhận được trạng thái mới nhất ở lịch sử hoạt động
- Cập nhật toggle switch sang trạng thái cuối cùng (ON/OFF)

2.5.5. Luồng quản lý lịch sử hoạt động

Biểu đồ 6: Luồng theo dõi lịch sử hành động LED



Luồng quản lý và hiển thị lịch sử hành động LED trên trang Action History.

Xem lịch sử hành động:

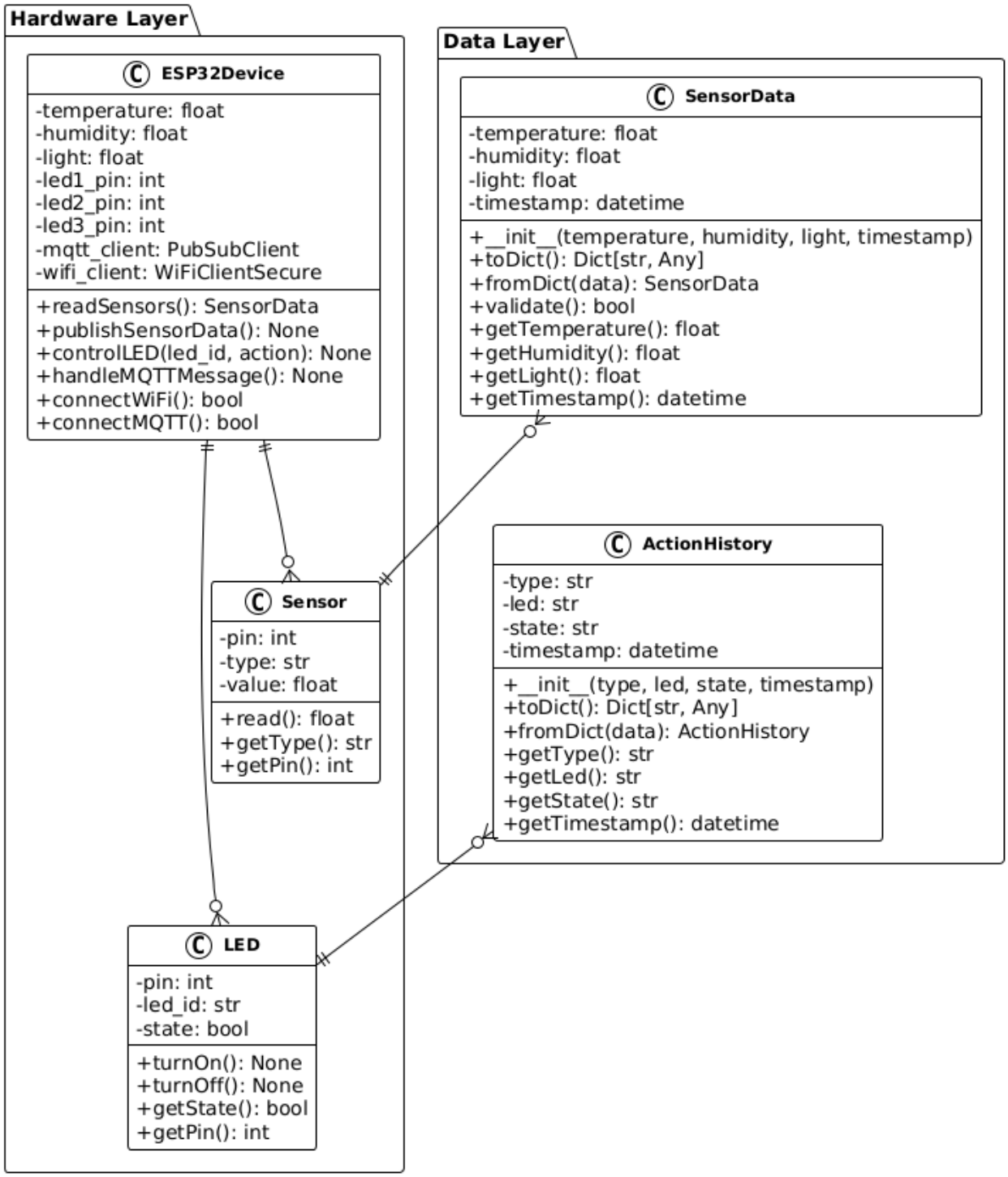
- Frontend gọi API /action-history với pagination (page, limit)
- Backend query MongoDB collection action_history
- Dữ liệu được sort theo timestamp và convert timezone
- Frontend hiển thị bảng với các cột: thiết bị, trạng thái, thời gian

Lọc và tìm kiếm:

- Lọc theo trạng thái (ON/OFF) hoặc thiết bị (LED1, LED2, LED3)
- Tìm kiếm theo thời gian với nhiều format
- Backend xử lý các pattern thời gian khác nhau
- Kết quả được hiển thị với pagination

2.6. Thiết kế cơ sở dữ liệu

Biểu đồ 7: Biểu đồ lớp



2.6.1. Mô hình dữ liệu cảm biến

Mô hình dữ liệu cảm biến được thiết kế để lưu trữ thông tin từ các thiết bị cảm biến một cách hiệu quả và linh hoạt.

Cấu trúc dữ liệu cảm biến được lưu trữ trong MongoDB với các trường chính:

- _id: ObjectId - Mã định danh duy nhất của document
- temperature: Float - Giá trị nhiệt độ (Celsius)
- humidity: Float - Giá trị độ ẩm (phần trăm)
- light: Float - Giá trị ánh sáng (phần trăm)
- timestamp: DateTime - Thời điểm đo dữ liệu

Các trường dữ liệu chính:

- temperature: Giá trị nhiệt độ đo được từ cảm biến DHT11
- humidity: Giá trị độ ẩm đo được từ cảm biến DHT11
- light: Giá trị ánh sáng được tính toán từ ADC
- timestamp: Thời điểm đo dữ liệu

Validation rules:

- Temperature: -50°C đến 80°C
- Humidity: 0% đến 100%
- Light: 0% đến 100%
- Timestamp: Phải là datetime hợp lệ

2.6.2. Mô hình lịch sử hành động

Mô hình lịch sử hành động theo dõi tất cả các thao tác điều khiển và thay đổi trạng thái trong hệ thống.

Cấu trúc dữ liệu lịch sử hành động bao gồm các trường:

- _id: ObjectId - Mã định danh duy nhất của document
- type: String - Loại hành động (led_status, system_event)
- led: String - ID của LED được điều khiển (LED1, LED2, LED3)

- state: String - Trạng thái mới (ON, OFF)
- timestamp: DateTime - Thời điểm thực hiện hành động

Các trường dữ liệu:

- type: Loại hành động ("led_status", "system_event", etc.)
- led: ID của LED được điều khiển ("LED1", "LED2", "LED3")
- state: Trạng thái mới ("ON", "OFF")
- timestamp: Thời điểm thực hiện hành động

2.6.3. Mối quan hệ giữa các bảng

ESP32Device – Sensor: 1 - N (Aggregation)

- Một thiết bị ESP32 có thể chứa nhiều cảm biến
- Một cảm biến chỉ thuộc về một thiết bị ESP32
- Mối quan hệ aggregation: ESP32Device quản lý lifecycle của Sensor

ESP32Device – LED: 1 - N (Aggregation)

- Một thiết bị ESP32 có thể điều khiển nhiều LED
- Một LED chỉ thuộc về một thiết bị ESP32
- Mối quan hệ aggregation: ESP32Device quản lý trạng thái của LED

ESP32Device – SensorData: 1 - N (Association)

- Một thiết bị ESP32 tạo ra nhiều dữ liệu cảm biến theo thời gian
- Một dữ liệu cảm biến chỉ thuộc về một thiết bị ESP32
- Mối quan hệ được xác định thông qua timestamp và device_id

ESP32Device – ActionHistory: 1 - N (Association)

- Một thiết bị ESP32 thực hiện nhiều hành động điều khiển LED
- Một hành động chỉ thuộc về một thiết bị ESP32
- Mối quan hệ được xác định thông qua led_id và timestamp

Sensor – SensorData: 1 - N (Association)

- Một cảm biến tạo ra nhiều dữ liệu đo lường theo thời gian

- Một dữ liệu cảm biến chỉ thuộc về một loại cảm biến
- Mối quan hệ được xác định thông qua sensor type và timestamp

LED – ActionHistory: 1 - N (Association)

- Một LED có nhiều lịch sử thay đổi trạng thái
- Một hành động chỉ ảnh hưởng đến một LED
- Mối quan hệ được xác định thông qua led_id và timestamp

2.7. Giao thức truyền thông MQTT

2.7.1. Tổng quan về MQTT

MQTT (Message Queuing Telemetry Transport) là giao thức truyền thông publish-subscribe nhẹ, được thiết kế đặc biệt cho các ứng dụng IoT và M2M (Machine-to-Machine) communication.

Kiến trúc MQTT:

- Broker: Server trung tâm quản lý connections và message routing
- Client: Thiết bị publish hoặc subscribe messages
- Topic: Địa chỉ logic để phân loại messages
- Message: Payload dữ liệu được truyền tải

2.7.2. Cấu hình MQTT trong hệ thống

Hệ thống sử dụng HiveMQ Cloud làm MQTT broker.

Cài đặt kết nối:

- Host: 9b88959e8c674540989f6ed6cf143c4d.s1.eu.hivemq.cloud
- Port: 8883 (TLS/SSL)
- Protocol: MQTT over TLS
- Authentication: Username/Password authentication
- Keep Alive: 60 seconds

Cấu hình bảo mật:

- TLS 1.2 encryption cho tất cả connections

- Certificate validation disabled cho development
- Username/password authentication
- Client ID uniqueness để tránh conflicts

2.7.3. Topic Structure và Message Format

Hệ thống MQTT sử dụng cấu trúc topic phân cấp như sau:

- esp32/iot/data: Chứa dữ liệu cảm biến được gửi từ ESP32
- esp32/iot/control: Chứa lệnh điều khiển được gửi từ backend
- esp32/iot/action-history: Chứa lịch sử hành động được gửi từ ESP32

Dữ liệu cảm biến được gửi dưới dạng JSON với các trường:

- temperature: 25.5 (nhiệt độ)
- humidity: 60.2 (độ ẩm)
- light: 45.8 (ánh sáng)

Lệnh điều khiển LED được gửi dưới dạng chuỗi văn bản:

- LED1_ON, LED1_OFF
- LED2_ON, LED2_OFF
- LED3_ON, LED3_OFF

Lịch sử hành động được gửi dưới dạng JSON với các trường:

- type: "led_status" (loại hành động)
- led: "LED1" (ID LED)
- state: "ON" (trạng thái)

CHƯƠNG 3: XÂY DỰNG HỆ THỐNG

3.1. Phần cứng (Hardware)

3.1.1. Thiết bị ESP32

ESP32 (Espressif Systems 32) là một vi điều khiển 32-bit được thiết kế đặc biệt cho các ứng dụng Internet of Things (IoT). Vi điều khiển này tích hợp WiFi và Bluetooth, cung cấp khả năng kết nối mạng không dây mạnh mẽ cho các dự án IoT.

Đặc điểm kỹ thuật chính:

- Bộ xử lý: Dual-core Xtensa LX6 32-bit, tần số 240MHz
- Bộ nhớ: 520KB SRAM, 4MB Flash (có thể mở rộng)
- Kết nối: WiFi 802.11 b/g/n, Bluetooth 4.2
- GPIO: 34 chân GPIO có thể cấu hình
- ADC: 12-bit ADC với 18 kênh
- DAC: 8-bit DAC với 2 kênh
- Điện áp hoạt động: 3.3V
- Dòng tiêu thụ: 80-240mA (tùy chế độ hoạt động)

Vai trò trong hệ thống: ESP32 đóng vai trò là tầng phần cứng cốt lõi trong hệ thống. Thiết bị chịu trách nhiệm thu thập dữ liệu từ các cảm biến, xử lý dữ liệu cơ bản, và truyền tải dữ liệu lên backend thông qua giao thức MQTT. Mặt khác, thiết bị còn thực hiện các lệnh điều khiển LED nhận được từ backend.

Hệ thống được cấu hình với các thông số sau:

- MQTT Server: 9b88959e8c674540989f6ed6cf143c4d.s1.eu.hivemq.cloud
- MQTT Port: 8883
- MQTT Client ID: ESP32_IoT_Device

Hệ thống sử dụng HiveMQ Cloud làm MQTT broker với kết nối bảo mật Transport Layer Security (TLS) trên port 8883. Client ID được đặt cố định để thuận tiện cho việc quản lý và debug.

3.1.2. Cảm biến DHT11

DHT11 là cảm biến nhiệt độ và độ ẩm kỹ thuật số với giao tiếp 1-wire. Cảm biến này được sử dụng rộng rãi trong các dự án Internet of Things (IoT) do giá thành thấp và dễ sử dụng.

Đặc điểm kỹ thuật:

- Nhiệt độ: Dải đo -40°C đến +80°C, độ chính xác ±2°C
- Độ ẩm: Dải đo 20% đến 90% RH, độ chính xác ±5% RH
- Giao tiếp: 1-wire digital interface
- Tần số lấy mẫu: Tối đa 1Hz (1 lần/giây)
- Điện áp hoạt động: 3.3V - 5V
- Dòng tiêu thụ: 0.5-2.5mA

Cảm biến được kết nối với chân GPIO 21 của ESP32 với cấu hình:

- DHT Pin: 21
- DHT Type: DHT11
- Thư viện: DHT library

3.1.3. Cảm biến ánh sáng

Cảm biến ánh sáng được thiết kế dựa trên photoresistor (điện trở quang) kết nối với bộ chuyển đổi Analog-to-Digital Converter (ADC) của ESP32.

Nguyên lý hoạt động: Photoresistor có điện trở thay đổi theo cường độ ánh sáng. Khi ánh sáng mạnh, điện trở giảm; khi ánh sáng yếu, điện trở tăng. ESP32 đọc giá trị điện áp qua ADC và chuyển đổi thành giá trị số.

Cảm biến ánh sáng được cấu hình với các thông số:

- ADC Pin: 34
- Độ phân giải: 12-bit
- Số mức: 4096 (0-4095)

ESP32 được cấu hình để sử dụng ADC 12-bit, cung cấp độ phân giải 4096 mức (0-4095). Chân General Purpose Input/Output (GPIO) 34 được sử dụng vì đây là chân chỉ đọc (input-only).

3.1.4. Điều khiển LED

Hệ thống sử dụng 3 Light Emitting Diode (LED) được điều khiển độc lập thông qua các chân GPIO của ESP32.

Hệ thống LED được cấu hình với các thông số:

- LED1 Pin: 25
- LED2 Pin: 26
- LED3 Pin: 27
- Chế độ: OUTPUT
- Trạng thái khởi tạo: LOW (tắt)

Tất cả LED được cấu hình là output và khởi tạo ở trạng thái tắt (LOW). Việc sử dụng 3 LED riêng biệt cho phép thực hiện các chức năng điều khiển phức tạp và tạo hiệu ứng trực quan.

Quá trình xử lý lệnh điều khiển LED được thực hiện qua hàm mqttCallback:

Bước 1: Nhận message từ MQTT topic

Bước 2: Chuyển đổi payload thành string

Bước 3: Kiểm tra topic có phải mqttControlTopic

Bước 4: Xử lý lệnh theo format "LEDx_ACTION":

- LED1_ON: bật LED1 (digitalWrite HIGH)
- LED1_OFF: tắt LED1 (digitalWrite LOW)
- Tương tự cho LED2 và LED3

Bước 5: Gửi trạng thái LED về backend

Hàm callback xử lý các lệnh điều khiển nhận được qua MQTT. Mỗi lệnh được xác định bằng format "LEDx_ACTION" (x là số thứ tự LED, ACTION là ON hoặc OFF).

3.2. Backend API

3.2.1. Kiến trúc Flask

Flask là một micro web framework được viết bằng Python, được thiết kế để xây dựng các ứng dụng web một cách đơn giản và linh hoạt. Trong dự án này, Flask được sử dụng để xây dựng Representational State Transfer (REST) API cho hệ thống IoT.

Hàm create_app() được thiết kế để khởi tạo ứng dụng Flask:

1. Tạo Flask app instance
2. Load cấu hình từ Config class
3. Cấu hình CORS cho phép tất cả origins
4. Tạo Flask-RESTX API với:
   - Version: 1.0
   - Title: IoT Monitoring System API
   - Description: API REST cho hệ thống giám sát IoT
   - Documentation: /docs/
   - Prefix: /api/v1
5. Return app instance

Các thành phần chính:

- Flask Core: Framework chính để xây dựng web application
- Flask-COR: Xử lý Cross-Origin Resource Sharing (CORS) cho frontend
- Flask-RESTX: Tạo REST API với Swagger documentation tự động
- Blueprint: Tổ chức code thành các module độc lập

Các namespace API:

- sensors: Quản lý dữ liệu cảm biến và điều khiển LED

- nosql: Truy vấn NoSQL và thống kê dữ liệu
- system: Thông tin hệ thống và health check

3.2.2. Quản lý cơ sở dữ liệu MongoDB

MongoDB là một cơ sở dữ liệu NoSQL document-oriented, phù hợp với việc lưu trữ dữ liệu IoT do tính linh hoạt và khả năng mở rộng cao.

Class DatabaseManager được thiết kế để quản lý kết nối MongoDB:

Thuộc tính:

- client: MongoDB client instance
- db: Database instance
- sensor_collection: Collection dữ liệu cảm biến
- action_collection: Collection lịch sử hành động

Phương thức connect():

1. Tạo MongoClient với connection string
2. Kết nối đến database theo tên
3. Khởi tạo sensor_collection từ 'sensor_data'
4. Khởi tạo action_collection từ 'action_history'
5. Xử lý exception và log lỗi nếu có

Các thao tác cơ bản:

- insert_sensor_data(): Lưu trữ dữ liệu cảm biến mới
- insert_action_history(): Lưu trữ lịch sử hành động LED
- get_recent_data(): Lấy dữ liệu mới nhất
- get_data_with_crud_operations(): Lấy dữ liệu với phân trang và lọc

Class SensorData được thiết kế để đại diện cho dữ liệu cảm biến:

Thuộc tính:

- temperature: Nhiệt độ
- humidity: Độ ẩm

- light: Ánh sáng
- timestamp: Thời gian
- sensor_statuses: Trạng thái các cảm biến

Phương thức:

- init(): Khởi tạo object với các giá trị đầu vào
- _calculate_statuses(): Tính toán trạng thái cảm biến
- _get_temperature_status(): Xác định trạng thái nhiệt độ
- _get_humidity_status(): Xác định trạng thái độ ẩm
- -_get_light_status(): Xác định trạng thái ánh sáng

3.2.3. Dịch vụ MQTT

MQTT service được thiết kế để nhận và xử lý dữ liệu từ ESP32, đồng thời gửi lệnh điều khiển ngược lại.

Class MQTTManager được thiết kế để quản lý kết nối MQTT:

Thuộc tính:

- mqtt_client: MQTT client instance
- message_callback: Callback xử lý message
- status_callback: Callback xử lý status
- is_connected: Trạng thái kết nối

Phương thức setup_client():

1. Tạo client id duy nhất: python_iot_receiver {pid}_{timestamp}
2. Khởi tạo mqtt.Client với client_id
3. Thiết lập username và password
4. Cấu hình TLS:
   - Tạo SSL context
   - Tắt check hostname
   - Tắt verify mode
   - Áp dụng TLS context cho client

Phương thức _on_message() xử lý các message nhận được:

Quy trình xử lý:

1. Lấy topic và payload từ message
2. Decode payload thành UTF-8 string
3. Kiểm tra loại topic:
   - MQTT_DATA_TOPIC: Xử lý dữ liệu cảm biến
     + Parse JSON payload
     + Validate dữ liệu
     + Gọi message_callback nếu hợp lệ
   - MQTT_ACTION_HISTORY_TOPIC: Xử lý lịch sử hành động
     + Parse JSON payload
     + Gọi status_callback

Phương thức _on_disconnect() xử lý việc ngắt kết nối:

Quy trình xử lý:

1. Đặt is_connected = False
2. Kiểm tra return code (rc):
   - rc = 0: Ngắt kết nối bình thường
   - rc ≠ 0: Ngắt kết nối bất thường
3. Nếu ngắt kết nối bất thường:
   - Tăng số lần thử kết nối lại
   - Log warning message
   - Kích hoạt cơ chế auto-reconnect

3.3. Frontend Web

3.3.1. Gọi API và xử lý dữ liệu

Class SensorDataService cung cấp các phương thức gọi API:

File: api.js

API_BASE_URL: http://localhost:5000/api/v1/sensors

Phương thức getLatestSensorData():

1. Gọi GET /sensor-data
2. Kiểm tra response.ok
3. Parse JSON response
4. Xử lý error và log

Phương thức getChartData(limit):

1. Tạo URL với query parameter limit
2. Gọi GET /sensor-data/chart
3. Kiểm tra response.ok
4. Parse JSON response
5. Xử lý error và log

Phương thức controlLED(ledId, action):

1. Gọi POST /led-control
2. Headers: Content-Type: application/json
3. Body: JSON với led_id và action
4. Kiểm tra response.ok
5. Parse JSON response
6. Xử lý error và log

3.4. Tài liệu API

3.4.1. Endpoint lấy dữ liệu cảm biến

GET /api/v1/sensors/sensor-data - Lấy dữ liệu cảm biến mới nhất từ hệ thống.

Request:

- Method: GET
- Headers: Content-Type: application/json

Response:


```
{
  "_id": "68d9e62c4e25db9407f6606d",
  "temperature": 30.2,
  "humidity": 66,
  "light": 30.5,
  "timestamp": "2025-09-29T08:51:40.214000+07:00",
  "sensor_statuses": {
    "temperature": {
      "status": "bình thường",
      "color_class": "status-normal",
      "value": 30.2
    },
    "humidity": {
      "status": "cảnh báo",
      "color_class": "status-warning",
      "value": 66
    },
    "light": {
      "status": "cảnh báo",
      "color_class": "status-warning",
      "value": 30.5
    }
  },
  "overall_status": {
    "status": "cảnh báo",
    "color_class": "status-warning"
  }
}
```


GET /api/v1/sensors/sensor-data-list - Lấy danh sách dữ liệu cảm biến với phân trang, sắp xếp và tìm kiếm.

- Query Parameters:
- limit: Số lượng bản ghi (mặc định: 5, có thể là "all")
- page: Trang hiện tại (mặc định: 1)
- per_page: Số bản ghi mỗi trang (mặc định: 10)
- sample: Tỷ lệ lấy mẫu dữ liệu (mặc định: 1)

- sort_field: Trường sắp xếp (timestamp, temperature, humidity, light)
- sort_order: Thứ tự sắp xếp (asc, desc)
- search: Từ khóa tìm kiếm
- search_criteria: Tiêu chí tìm kiếm (all, time, temperature, humidity, light)

Response:


```
{
  "status": "success",
  "data": [
    {
      "_id": "68d9e7644e25db9407f66221",
      "temperature": 30.2,
      "humidity": 66,
      "light": 35.5,
      "timestamp": "2025-09-29T08:56:52.226000+07:00"
    }
  ],
  "pagination": {
    "page": 1,
    "per_page": 10,
    "total_count": 1,
    "total_pages": 1,
    "has_prev": false,
    "has_next": false
  },
  "sort": {
    "field": "timestamp",
    "order": "desc"
  },
  "search": {
    "term": "",
    "criteria": "all"
  },
  "count": 1,
  "total_count": 1
}
```

GET /api/v1/sensors/sensor-data/chart - Lấy dữ liệu cho biểu đồ với các tùy chọn lọc.

Query Parameters:

- limit: Số lượng điểm dữ liệu (mặc định: 50)
- date: Ngày cụ thể (format: YYYY-MM-DD)

Response:


```
[
  {
    "_id": "68d9e7094e25db9407f661b8",
    "temperature": 30.1,
    "humidity": 66,
    "light": 34.6,
    "timestamp": "2025-09-29T08:55:21.268000+07:00"
  },
  {
    "_id": "68d9e70a4e25db9407f661b9",
    "temperature": 30.4,
    "humidity": 66,
    "light": 34.8,
    "timestamp": "2025-09-29T08:55:22.236000+07:00"
  }
]
```


3.4.2. Endpoint điều khiển LED

POST /api/v1/sensors/led-control - Gửi lệnh điều khiển LED đến ESP32.

Request Body:


```
{
  "led_id": "LED1",
  "action": "ON"
}
```


Response:


```
{
  "status": "success",
```

```
  "message": "LED control command sent successfully",
  "data": {
    "led_id": "LED1",
    "action": "ON",
    "timestamp": "2024-01-15T10:30:00+07:00"
  }
}
```

GET /api/v1/sensors/led-status - Lấy trạng thái hiện tại của tất cả LED.

Response:

```
{
    "status": "success",
    "data": {
        "led_states": {
            "LED1": "ON",
            "LED2": "OFF",
            "LED3": "ON"
        },
        "pending_commands": {
            "LED1": false,
            "LED2": true,
            "LED3": false
        }
    }
}
```

3.4.3. Endpoint lịch sử hành động

GET /api/v1/sensors/action-history - Lấy lịch sử hành động LED với phân trang và lọc. Endpoint này được sử dụng bởi bảng lịch sử hành động.

Query Parameters:

- page: Số trang (mặc định: 1)
- per_page: Số bản ghi mỗi trang (mặc định: 10)
- sort_field: Trường sắp xếp (timestamp, led, state)
- sort_order: Thứ tự sắp xếp (asc, desc)
- search: Từ khóa tìm kiếm

- device_filter: Lọc theo thiết bị
- state_filter: Lọc theo trạng thái (ON, OFF)
- limit: Giới hạn số bản ghi

Response:


```
{
  "status": "success",
  "data": [
    {
      "_id": "68d9dea6eca8b50d14d947fe",
      "type": "led_status",
      "led": "LED1",
      "state": "ON",
      "timestamp": "2025-09-29T08:19:34.264000+07:00"
    }
  ],
  "pagination": {
    "page": 1,
    "per_page": 1,
    "total_count": 17,
    "total_pages": 17,
    "has_prev": false,
    "has_next": true
  },
  "filters": {
    "device": "LED1",
    "state": "ON",
    "search": ""
  },
  "sort": {
    "field": "timestamp",
    "order": "desc"
  },
  "count": 1,
  "total_count": 17
}
```


3.4.4. Cấu trúc request/response

Tất cả API endpoints đều trả về response theo format chuẩn:

```
{
    "status": "success" | "error",
    "data": <object | array | null>,
    "message": "<string>",
    "pagination": {
        "page": <number>,
        "per_page": <number>,
        "total_count": <number>,
        "total_pages": <number>,
        "has_prev": <boolean>,
        "has_next": <boolean>
    },
    "sort": {
        "field": "<string>",
        "order": "asc" | "desc"
    },
    "search": {
        "term": "<string>",
        "criteria": "<string>"
    },
    "count": <number>,
    "total_count": <number>
}
```

Error Response:

```
{
    "status": "error",
    "message": "Error description",
    "data": null,
    "error_code": "ERROR_CODE"
}
```

HTTP Status Codes:

- 200: OK - Request thành công
- 400: Bad Request - Dữ liệu request không hợp lệ
- 404: Not Found - Không tìm thấy resource
- 500: Internal Server Error - Lỗi server

3.4.5. Ví dụ minh họa API

Ví dụ 1: Lấy dữ liệu cảm biến mới nhất

```
curl -X GET "http://localhost:5000/api/v1/sensors/sensor-data" \
     -H "Content-Type: application/json"
```

Ví dụ 2: Điều khiển LED

```
curl -X POST "http://localhost:5000/api/v1/sensors/led-control" \
     -H "Content-Type: application/json" \
     -d '{"led_id": "LED1", "action": "ON"}'
```

Ví dụ 3: Lấy dữ liệu biểu đồ

```
curl -X GET "http://localhost:5000/api/v1/sensors/sensor-
data/chart?limit=100&date=2025-09-29" \
     -H "Content-Type: application/json"
```

Ví dụ 4: Lấy danh sách dữ liệu với phân trang

```
curl -X GET "http://localhost:5000/api/v1/sensors/sensor-data-
list?page=1&per_page=10&sort_field=timestamp&sort_order=desc" \
     -H "Content-Type: application/json"
```

Ví dụ 5: Lấy lịch sử hành động

```
curl -X GET "http://localhost:5000/api/v1/sensors/action-
history?limit=50&state_filter=ON" \
     -H "Content-Type: application/json"
```

CHƯƠNG 4: KẾT LUẬN VÀ ĐÁNH GIÁ

4.1. Kết quả đạt được

4.1.1. 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 MongoDB 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 Flask-RESTX, 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. Swagger documentation được tích hợp tự động để hỗ trợ việc phát triển và testing.

4.1.2. Giao diện người dùng

Trang chủ (Homepage) được thiết kế làm trung tâm điều khiển chính của hệ thống. Giao diện bao gồm các sensor card hiển thị dữ liệu cảm biến mới nhất với trạng thái màu sắc trực quan. Biểu đồ line chart được đặt ở vị trí nổi bật để hiển thị xu hướng dữ liệu theo thời gian. Khu vực điều khiển LED được bố trí dễ tiếp cận với các nút bấm rõ ràng.

Trang dữ liệu cảm biến (Sensor Data) cung cấp giao diện quản lý dữ liệu chi tiết với bảng dữ liệu có khả năng phân trang, sắp xếp và lọc. Người dùng có thể xem dữ liệu theo các khoảng thời gian khác nhau.

Trang lịch sử hoạt động (Action History) cung cấp bảng dữ liệu chi tiết về mọi thao tác điều khiển LED đã thực hiện. Bảng hỗ trợ tìm kiếm theo từ khóa, lọc theo thiết bị và trạng thái LED. Dữ liệu được sắp xếp theo thời gian với thông tin chi tiết về LED được điều khiển, hành động thực hiện và thời điểm thực hiện.

Trang hồ sơ cá nhân (Profile) được thiết kế để hiển thị thông tin cá nhân của người dùng.

4.2. Đánh giá ưu điểm

Hệ thống được thiết kế theo mô hình kiến trúc phân tầng với sự tách biệt rõ ràng giữa phần cứng, backend và frontend. Điều này tạo ra hệ thống dễ bảo trì, mở rộng và debug. Mỗi tầng có trách nhiệm cụ thể và giao tiếp thông qua các giao diện được định nghĩa rõ ràng.

Giao diện được thiết kế với trọng tâm vào trải nghiệm người dùng. Các chức năng được bố trí logic và dễ tìm kiếm. Màu sắc và icon được sử dụng hiệu quả để truyền tải thông tin trực quan.

Kiến trúc modular cho phép dễ dàng thêm các chức năng mới hoặc thay đổi các thành phần hiện có. API RESTful cung cấp giao diện chuẩn cho việc tích hợp với các hệ thống khác. Cấu trúc database linh hoạt cho phép lưu trữ các loại dữ liệu mới mà không cần thay đổi cấu trúc.

Hệ thống sử dụng TLS cho kết nối MQTT, đảm bảo dữ liệu được mã hóa trong quá trình truyền tải. Validation dữ liệu được thực hiện ở cả frontend và backend để ngăn chặn các lỗi và tấn công cơ bản.

API được tài liệu hóa tự động bằng Swagger, giúp việc phát triển và testing trở nên dễ dàng. Cấu trúc thư mục logic giúp việc bảo trì hiệu quả.

4.3. Hạn chế và khó khăn

ESP32 có giới hạn về số lượng GPIO pins và khả năng xử lý đồng thời nhiều tác vụ. Việc thêm nhiều cảm biến hoặc thiết bị điều khiển có thể gặp khó khăn do hạn chế về tài nguyên. DHT11 có độ chính xác không cao và tần suất lấy mẫu bị giới hạn ở 1Hz.

Hệ thống hoạt động hoàn toàn phụ thuộc vào kết nối WiFi và MQTT broker. Khi mất kết nối mạng, thiết bị ESP32 không thể gửi dữ liệu và nhận lệnh điều khiển. Mặc dù có cơ chế retry, nhưng việc mất kết nối kéo dài có thể dẫn đến mất dữ liệu.

Hệ thống hiện tại chưa có cơ chế xác thực và phân quyền người dùng. Tất cả người dùng đều có quyền truy cập đầy đủ vào hệ thống, điều này có thể gây ra vấn đề bảo mật trong môi trường sản xuất.

MongoDB được cấu hình để lưu trữ tất cả dữ liệu mà không có chính sách retention. Theo thời gian, dung lượng database sẽ tăng lên đáng kể, có thể ảnh hưởng đến hiệu suất hệ thống. Cần có cơ chế tự động xóa dữ liệu cũ hoặc backup định kỳ.

Hệ thống chưa có cơ chế monitoring toàn diện để theo dõi hiệu suất và phát hiện sự cố. Logging cơ bản đã được triển khai nhưng chưa đủ chi tiết để debug các vấn đề phức tạp.

Một số trang như Profile chưa được triển khai đầy đủ chức năng. Giao diện còn thiếu các tính năng nâng cao như cài đặt thông báo, hoặc dashboard tùy chỉnh.

Hệ thống chưa có cơ chế backup dữ liệu tự động và recovery khi có sự cố. Việc mất dữ liệu có thể xảy ra nếu có lỗi phần cứng hoặc phần mềm nghiêm trọng.

Khi lượng dữ liệu tăng lên, hiệu suất truy vấn có thể giảm do thiếu index tối ưu. Frontend có thể gặp vấn đề hiệu suất khi hiển thị lượng lớn dữ liệu trên biểu đồ.