

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG



BÁO CÁO BÀI TẬP CÁ NHÂN
IOT VÀ ỨNG DỤNG

Sinh viên thực hiện	: Lưu Xuân Dũng
Mã sinh viên	: B22DCCN129
Lớp	: D22HTTT05
Giảng viên hướng dẫn	: Nguyễn Quốc Uy

Hà Nội, năm 2025

LỜI CẢM ƠN

Để hoàn thành được báo cáo thực bài tập lớn môn học IOT và ứng dụng nhóm lớp 05 thì em xin cảm ơn tới thầy Nguyễn Quốc Uy đã luôn nhiệt tình chỉ dẫn, giảng dạy cho em những kiến thức về môn học để em có thể hoàn thiện tốt nhất .

Em xin chân thành cảm ơn!

Hà Nội, ngày 3 tháng 10 năm 2025

MỤC LỤC

Chương 1: Giới thiệu và Tổng quan.....	4
1. Giới thiệu:.....	4
2. Tổng quan:.....	4
3. Danh sách các Tính năng chính:.....	6
Chương 2: Thiết kế hệ thống.....	8
1. Thiết kế Tổng quan Kiến trúc.....	8
2. Thiết kế Giao diện (UI/UX Design).....	9
3. Luồng hoạt động (Sequence Diagrams).....	11
4. Thiết kế Cơ sở dữ liệu (Database Design).....	13
Chương 3: Xây dựng hệ thống.....	14
1. Tài liệu API (API Documentation).....	14
2. Xây dựng Phần cứng.....	17
3. Backend.....	24
a) Điểm vào và Luồng khởi tạo (server.js).....	24
b) Lắng nghe và Xử lý Dữ liệu MQTT (services/mqttService.js).....	25
c) Xây dựng API Động (controllers/historyController.js).....	27
d) Logic Cảnh báo Thông minh (services/notificationService.js).....	31
Chương 4: Kết luận.....	32
1. Kết luận.....	32
2. Đánh giá Dự án.....	33
3. Những bài học Kinh nghiệm.....	34
4. Định hướng Phát triển.....	34

Chương 1: Giới thiệu và Tổng quan

1. Giới thiệu:

Trong bối cảnh cuộc Cách mạng Công nghiệp 4.0 đang diễn ra mạnh mẽ, Internet of Things (IoT) đã và đang trở thành một trong những công nghệ trụ cột, định hình lại cách chúng ta tương tác với thế giới vật lý. Từ những ngôi nhà thông minh (Smarthome) có khả năng tự động hóa các tác vụ hàng ngày, đến các nhà máy thông minh (Smart Factory) tối ưu hóa quy trình sản xuất, IoT đã chứng tỏ tiềm năng to lớn trong việc nâng cao chất lượng cuộc sống và hiệu suất công việc.

Nắm bắt xu hướng đó, đề án này tập trung vào việc nghiên cứu và xây dựng một hệ thống IoT hoàn chỉnh, bao trùm từ tầng phần cứng, tầng giao tiếp, xử lý dữ liệu cho đến tầng giao diện người dùng. Mục tiêu của dự án là tạo ra một giải pháp "end-to-end" cho phép người dùng có thể:

- Giám sát các thông số môi trường quan trọng như nhiệt độ, độ ẩm và cường độ ánh sáng theo thời gian thực.
- Điều khiển từ xa các thiết bị điện trong nhà (mô phỏng bằng đèn, quạt, điều hòa) thông qua một giao diện web trực quan.
- Lưu trữ và phân tích dữ liệu lịch sử để theo dõi và đánh giá.
- Nhận cảnh báo thông minh khi có những thay đổi bất thường trong môi trường.

Dự án sử dụng các công nghệ hiện đại và phổ biến trong ngành như vi điều khiển ESP32, giao thức MQTT, nền tảng backend Node.js, cơ sở dữ liệu MongoDB và thư viện frontend React.js để xây dựng một hệ thống không chỉ mạnh mẽ về chức năng mà còn linh hoạt và có khả năng mở rộng cao.

2. Tổng quan:

Hệ thống được thiết kế theo kiến trúc 3 lớp rõ ràng, đảm bảo tính module hóa và dễ dàng bảo trì.

a) Kiến trúc hệ thống:

- Tầng Phần cứng (Hardware Layer):
 - ❖ Bộ não: Vi điều khiển ESP32, có tích hợp sẵn Wifi, chịu trách nhiệm thu thập dữ liệu từ cảm biến và thực thi lệnh điều khiển.
 - ❖ Cảm biến:
 - Cảm biến DHT11/22: Đo nhiệt độ và độ ẩm môi trường.
 - Quang trở (LDR): Đo cường độ ánh sáng, dữ liệu thô được quy đổi sang đơn vị Lux chuẩn.

- ❖ Thiết bị chấp hành: 3 đèn LED được sử dụng để mô phỏng cho các thiết bị thực tế như Đèn, Quạt và Điều hòa.
- Tầng Giao tiếp và Xử lý (Backend & Communication Layer):
 - ❖ Giao thức: MQTT (Message Queuing Telemetry Transport) được sử dụng làm giao thức giao tiếp chính giữa phần cứng và backend. Đây là một giao thức nhẹ, hiệu quả và đáng tin cậy, rất phù hợp cho các ứng dụng IoT.
 - ❖ MQTT Broker: Mosquitto được cài đặt trên máy local, đóng vai trò là một "bưu điện trung tâm", nhận tin nhắn từ các thiết bị và chuyển tiếp đến những nơi đăng ký nhận tin.
 - ❖ Backend Server: Được xây dựng bằng Node.js và framework Express.js. Server này có các nhiệm vụ chính:
 - Kết nối tới MQTT Broker và lắng nghe (subscribe) dữ liệu từ ESP32.
 - Xử lý, phân tích dữ liệu cảm biến để phát hiện các sự kiện bất thường.
 - Cung cấp các API (HTTP Endpoints) để frontend có thể gửi lệnh điều khiển và truy vấn dữ liệu lịch sử.
 - Giao tiếp real-time với frontend qua WebSocket để cập nhật dữ liệu và gửi thông báo tức thời.
 - ❖ Cơ sở dữ liệu: MongoDB Atlas (NoSQL database trên cloud) được sử dụng để lưu trữ toàn bộ dữ liệu cảm biến và lịch sử các hành động điều khiển thiết bị.
- Tầng Giao diện Người dùng (Frontend Layer):
 - ❖ Một ứng dụng Single Page Application (SPA) được xây dựng bằng thư viện React.js.
 - ❖ Giao diện được thiết kế để cung cấp một trải nghiệm người dùng trực quan, cho phép giám sát, điều khiển và xem lại lịch sử một cách dễ dàng.
 - ❖ Giao diện gồm nhiều trang được quản lý bởi React Router, bao gồm Dashboard, Data Sensor, Device Activity và My Profile.
- b) Luồng dữ liệu chính:
 - Luồng Giám sát (Sensor Data Flow): ESP32 đọc cảm biến → Gửi dữ liệu lên MQTT Broker (topic sensors/data) → Backend nhận dữ liệu → Xử lý cảnh báo, lưu vào MongoDB → Đẩy dữ liệu qua WebSocket → Frontend nhận và cập nhật giao diện (biểu đồ, thẻ thông số).

- Luồng Điều khiển (Control Command Flow): Người dùng tương tác trên Frontend → Gửi yêu cầu HTTP tới Backend (API /api/control) → Backend gửi lệnh lên MQTT Broker (topic devices/control) → ESP32 nhận lệnh → Thực thi bật/tắt đèn LED.

3. Danh sách các Tính năng chính:

- a) Giám sát Thời gian thực (Real-time Monitoring)
 - Hiển thị các thông số Nhiệt độ, Độ ẩm, Ánh sáng (Lux) trên Dashboard và tự động cập nhật sau mỗi 3 giây.
 - Biểu đồ đường trực quan biểu diễn sự thay đổi của các thông số trong khoảng thời gian ngắn gần đây
- b) Điều khiển Thiết bị từ xa (Remote Control)
 - Điều khiển bật/tắt độc lập cho từng thiết bị (Đèn, Quạt, Điều hòa) thông qua cần gạt (toggle switch).
 - Gửi lệnh bật/tắt đồng thời tất cả các thiết bị.
 - Hiệu ứng hình ảnh (visual effects) sống động tương ứng với từng hành động bật thiết bị, nâng cao trải nghiệm người dùng.
- c) Lưu trữ và Phân tích Dữ liệu Lịch sử
 - Tự động lưu trữ toàn bộ dữ liệu cảm biến theo thời gian vào cơ sở dữ liệu MongoDB Atlas.
 - Tự động ghi lại nhật ký tất cả các hành động bật/tắt thiết bị (thiết bị nào, hành động gì, vào lúc nào).
 - Giao diện trang "Data Sensor" và "Device Activity" cho phép xem lại dữ liệu lịch sử.
 - Hỗ trợ các chức năng nâng cao trên dữ liệu lịch sử:
 - ❖ Phân trang (Pagination): Duyệt qua các trang dữ liệu.
 - ❖ Sắp xếp (Sorting): Sắp xếp dữ liệu theo nhiều tiêu chí (mới nhất, cũ nhất, giá trị tăng/giảm dần).
 - ❖ Lọc và Tìm kiếm (Filtering & Searching): Tìm kiếm chính xác theo thời gian (đến từng phút) hoặc lọc theo loại thiết bị, hành động.
- d) Hệ thống Cảnh báo Thông minh
 - Tự động phát hiện và gửi cảnh báo đến người dùng khi có các sự kiện bất thường, bao gồm:
 - ❖ Vượt ngưỡng an toàn: Khi một giá trị cảm biến quá cao hoặc quá thấp trong một khoảng thời gian.
 - ❖ Thay đổi đột ngột: Khi giá trị thay đổi quá nhanh trong một chu kỳ đo.

❖ Dao động liên tục: Khi giá trị biến động bất thường trong một khoảng thời gian dài.

- Tự động gửi thông báo "Trở lại bình thường" khi hệ thống ổn định trở lại.
- Hiện thị cảnh báo tức thời dưới dạng thông báo pop-up (toast) và lưu lại trong danh sách để xem lại.
- Cơ chế giới hạn tần suất gửi cảnh báo để tránh làm phiền người dùng.

e) Giao diện và Trải nghiệm Người dùng

- Ứng dụng đa trang, dễ dàng điều hướng giữa các chức năng.
- Giao diện được thiết kế hiện đại, sạch sẽ, lấy cảm hứng từ các dashboard chuyên nghiệp.
- Trang thông tin cá nhân (My Profile) hiển thị thông tin sinh viên và các liên kết quan trọng.
- Hệ thống được bảo mật ở tầng giao tiếp MQTT bằng username và password.

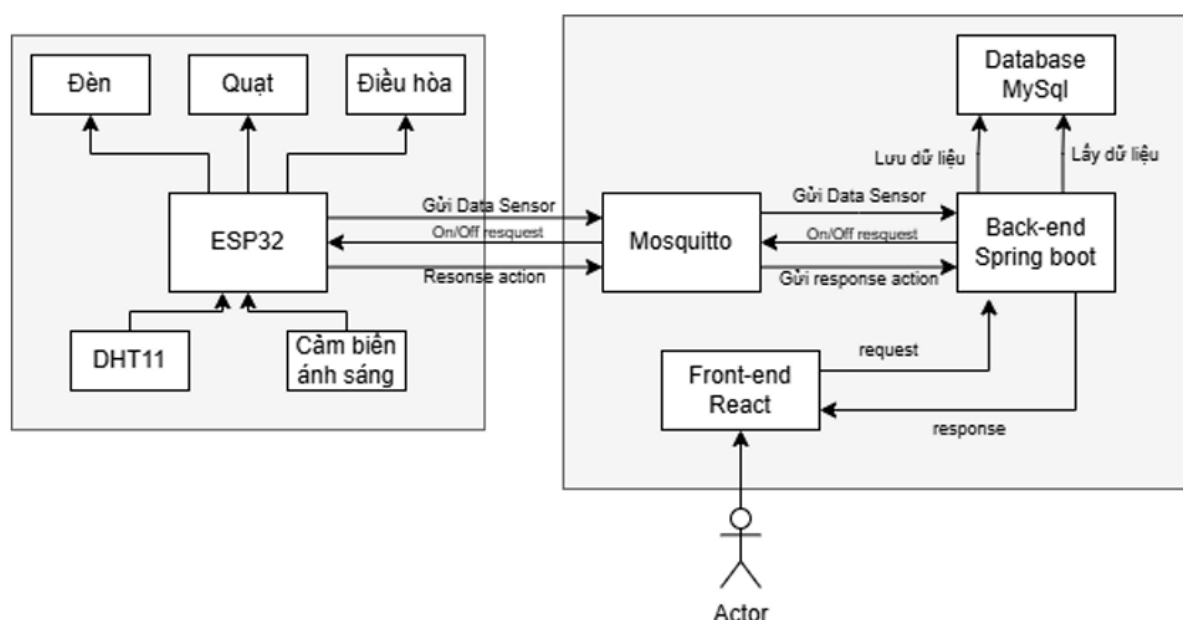
Chương 2: Thiết kế hệ thống

1. Thiết kế Tổng quan Kiến trúc

Để đảm bảo hệ thống có tính module, linh hoạt và khả năng mở rộng, kiến trúc được thiết kế theo mô hình 3 lớp (3-Tier Architecture) kết hợp với mô hình Publish/Subscribe (Pub/Sub) cho việc giao tiếp.

- Tầng Giao diện (Presentation Layer): Là lớp frontend được xây dựng bằng React.js, chịu trách nhiệm hiển thị dữ liệu và nhận tương tác từ người dùng.
- Tầng Ứng dụng (Application/Logic Layer): Là lớp backend được xây dựng bằng Node.js, chứa toàn bộ logic nghiệp vụ, xử lý dữ liệu, quản lý cảnh báo và giao tiếp với các thành phần khác.
- Tầng Dữ liệu (Data Layer): Bao gồm cơ sở dữ liệu MongoDB Atlas để lưu trữ dữ liệu bền vững.
- Kênh Giao tiếp (Communication Channel): MQTT Broker (Mosquitto) đóng vai trò trung gian giao tiếp giữa tầng ứng dụng và các thiết bị phần cứng, tách biệt chúng khỏi nhau.

Kiến trúc này cho phép các thành phần hoạt động độc lập. Ví dụ, chúng ta có thể dễ dàng thay đổi giao diện frontend mà không ảnh hưởng đến backend, hoặc thêm một loại thiết bị phần cứng mới mà không cần sửa đổi logic xử lý chính.



Hình : Thiết kế Tổng quan Kiến trúc

2. Thiết kế Giao diện (UI/UX Design)

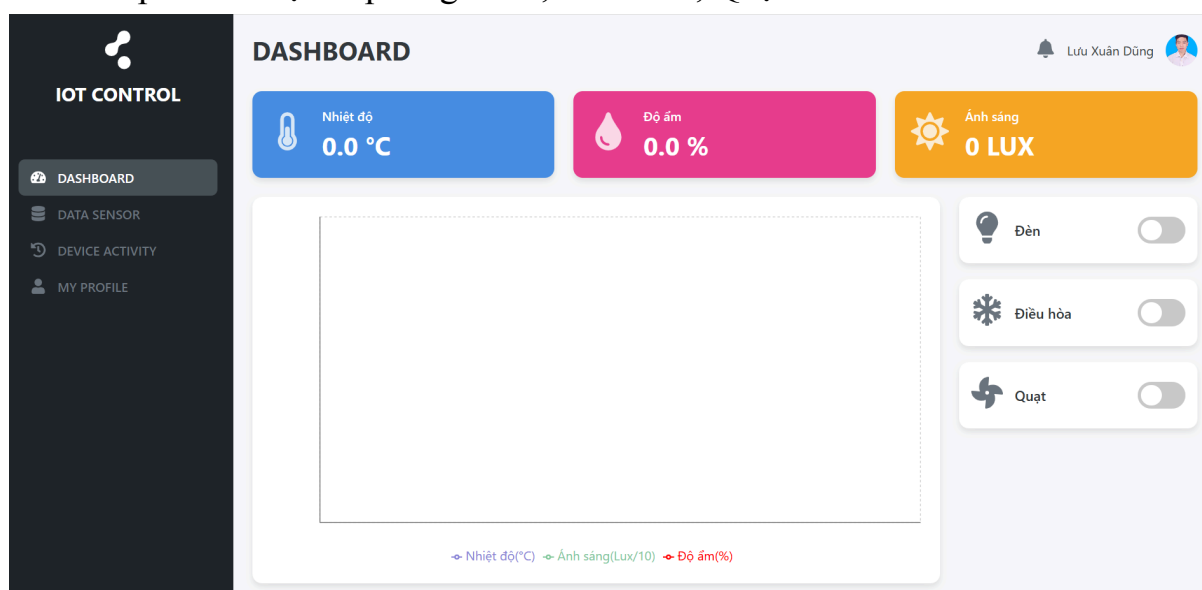
Giao diện người dùng (UI) được thiết kế trên công cụ Figma trước khi tiến hành lập trình. Mục tiêu thiết kế là tạo ra một trải nghiệm (UX) trực quan, hiện đại và dễ sử dụng, tập trung vào việc hiển thị thông tin rõ ràng và cho phép người dùng tương tác một cách nhanh chóng.

Giao diện được chia thành 4 trang chính, có thể truy cập qua một thanh điều hướng (Sidebar) cố định bên trái:

a) Trang Dashboard

Đây là trang tổng quan, cung cấp cái nhìn tức thời về trạng thái của hệ thống.

- Phần trên: Hiển thị 3 thẻ thông số (cards) nổi bật cho Nhiệt độ, Độ ẩm và Ánh sáng với giá trị mới nhất.
- Phần giữa: Một biểu đồ đường (line chart) biểu diễn sự thay đổi của cả 3 thông số trong một khoảng thời gian ngắn gần đây, giúp người dùng nắm bắt xu hướng.
- Phần bên phải: Cung cấp các cần gạt (toggle switch) để điều khiển trực tiếp 3 thiết bị mô phỏng: Đèn, Điều hòa, Quạt.



b) Trang Data Sensor

Trang này cho phép người dùng tra cứu và phân tích dữ liệu cảm biến đã được lưu trữ.

- Thanh công cụ: Bao gồm các bộ lọc (filter) và chức năng tìm kiếm, cho phép người dùng sắp xếp dữ liệu (mới nhất, cũ nhất, giá trị tăng/giảm dần) và tìm kiếm theo loại cảm biến hoặc theo một khoảng thời gian cụ thể.

- Bảng dữ liệu: Hiển thị chi tiết lịch sử dữ liệu cảm biến dưới dạng bảng, bao gồm các cột ID, Ánh sáng, Độ ẩm, Nhiệt độ và Thời gian.
- Phân trang: Các nút điều khiển phân trang (First, Previous, Next, Last, và các số trang) cho phép người dùng duyệt qua một lượng lớn dữ liệu.

- DASHBOARD
- DATA SENSOR**
- DEVICE ACTIVITY
- MY PROFILE

DATA SENSOR

Sắp xếp: Mới nhất
Loại tìm kiếm
Nhập giá trị hoặc thời gian...
Search

ID	Ánh sáng (Lux)	Độ ẩm (%)	Nhiệt độ (°C)	Thời gian
1	722.1281	78.1	30.1	10:39:45 3/10/2025
2	726.8737	78.1	30.1	10:39:42 3/10/2025
3	734.8819	78.1	30.1	10:39:39 3/10/2025
4	749.6163	78.0	30.1	10:39:36 3/10/2025
5	744.6584	78.0	30.1	10:39:33 3/10/2025
6	751.2795	78.0	30.1	10:39:30 3/10/2025
7	757.9847	78.0	30.2	10:39:27 3/10/2025
8	761.3694	78.0	30.2	10:39:24 3/10/2025
9	764.7758	78.0	30.2	10:39:21 3/10/2025
10	775.1263	78.0	30.2	10:39:18 3/10/2025

First Previous 1 2 3 Next Last

c) Trang Device Activity

Tương tự trang Data Sensor, trang này tập trung vào việc tra cứu nhật ký hoạt động của các thiết bị.

- Thanh công cụ: Cho phép lọc theo tên thiết bị (Đèn, Quạt,...) và theo hành động (Bật, Tắt), cũng như tìm kiếm theo thời gian.
- Bảng dữ liệu: Hiển thị lịch sử các hành động đã thực hiện.

- DASHBOARD
- DATA SENSOR
- DEVICE ACTIVITY**
- MY PROFILE

DEVICE ACTIVITY

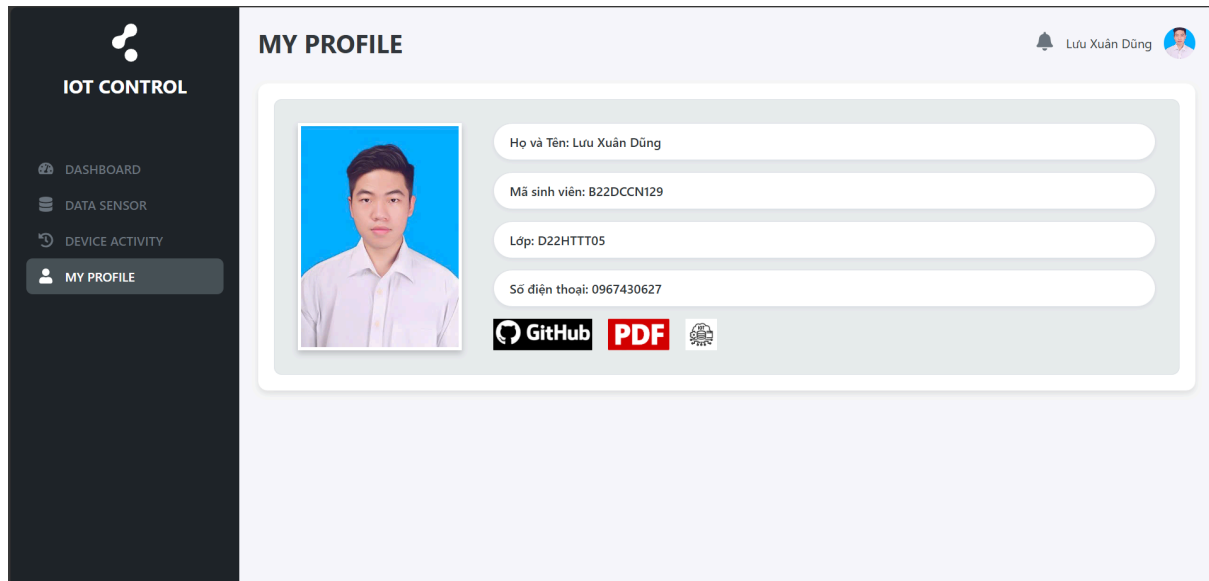
Tất cả thiết bị
Tất cả hành động
HH:mm DD/MM/YYYY
Search

ID	Thiết bị	Hành động	Thời gian
1	Quạt	Tắt	10:39:38 3/10/2025
2	Quạt	Bật	10:36:12 3/10/2025
3	Đèn	Tắt	10:18:54 3/10/2025
4	Đèn	Bật	10:18:52 3/10/2025
5	Điều hòa	Tắt	10:18:52 3/10/2025
6	Điều hòa	Bật	10:18:49 3/10/2025
7	Quạt	Tắt	10:18:49 3/10/2025
8	Quạt	Bật	10:18:48 3/10/2025
9	Đèn	Tắt	10:17:37 3/10/2025
10	Quạt	Tắt	10:17:37 3/10/2025

First Previous 1 2 3 Next Last

d) Trang My Profile

Trang này hiển thị thông tin cá nhân của người thực hiện đồ án, bao gồm ảnh đại diện, họ tên, mã sinh viên và các liên kết quan trọng như GitHub, file PDF báo cáo.



3. Luồng hoạt động (Sequence Diagrams)

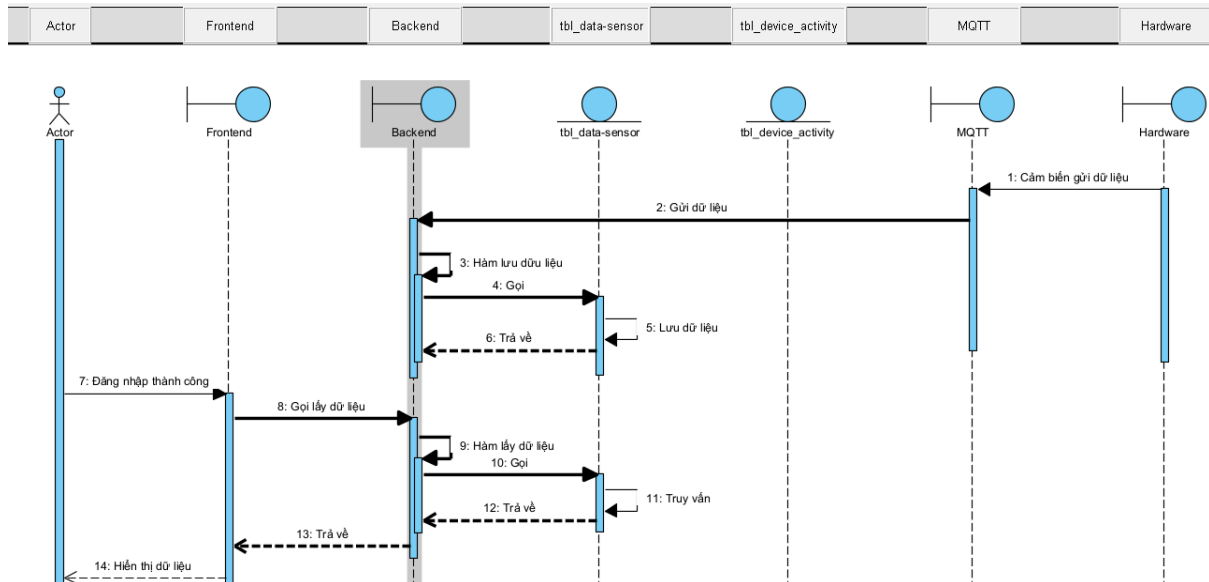
Để làm rõ hơn cách các thành phần tương tác với nhau, chúng ta sẽ xem xét hai luồng hoạt động chính của hệ thống.

a) Luồng Giám sát và Cảnh báo Thời gian thực

1. ESP32: Định kỳ 3 giây, đọc giá trị từ cảm biến DHT và quang trở.
2. ESP32: Đóng gói dữ liệu thành một chuỗi JSON.
3. ESP32: Gửi (Publish) chuỗi JSON đó lên topic sensors/data của MQTT Broker.
4. MQTT Broker: Nhận tin nhắn và chuyển tiếp ngay lập tức tới tất cả các client đang đăng ký (Subscribe) topic này.
5. Backend (Node.js): Nhận được tin nhắn.
6. Backend (Node.js):
 - Lưu bản ghi dữ liệu vào collection sensordatas trong MongoDB Atlas.
 - Đưa dữ liệu vào notificationService để phân tích (kiểm tra vượt ngưỡng, thay đổi đột ngột,...).
 - Nếu phát hiện bất thường, tạo một tin nhắn cảnh báo.
7. Backend (Node.js):
 - Đẩy dữ liệu cảm biến real-time qua WebSocket cho Frontend.
 - Nếu có cảnh báo, đẩy cả tin nhắn cảnh báo qua WebSocket.
8. Frontend (React.js): Nhận dữ liệu qua WebSocket.

9. Frontend (React.js):

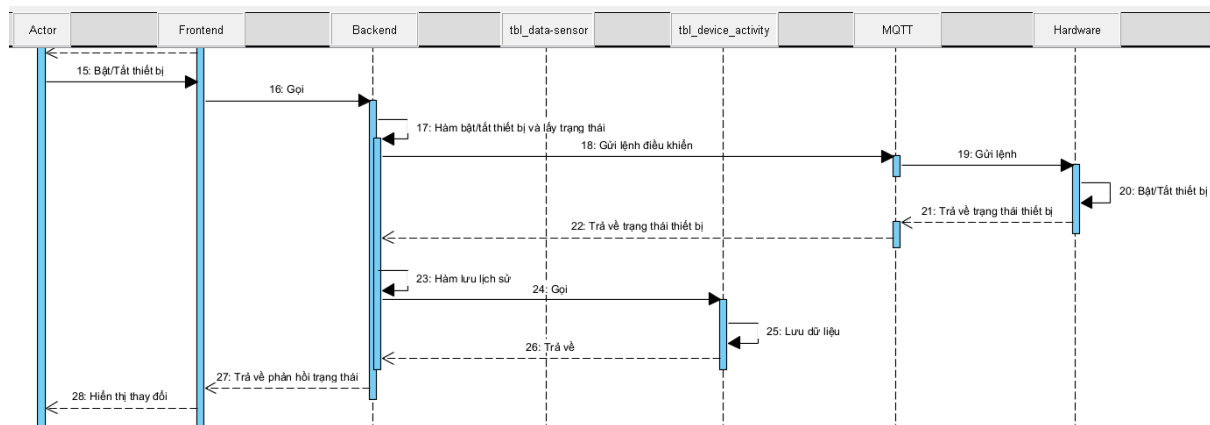
- Cập nhật state, làm cho các thẻ thông số và biểu đồ trên Dashboard được vẽ lại.
- Nếu nhận được cảnh báo, hiển thị thông báo pop-up (toast) và cập nhật danh sách thông báo.



b) Luồng Điều khiển Thiết bị

Luồng này mô tả quá trình khi người dùng bật/tắt một thiết bị trên giao diện.

1. Frontend (React.js): Người dùng click vào cần gạt của một thiết bị (ví dụ: Đèn).
2. Frontend (React.js): Gọi hàm `handleControl`, hàm này gửi một yêu cầu HTTP POST đến API `/api/control` của backend, kèm theo nội dung `{ "device": "led_1", "status": "ON" }`.
3. Backend (Node.js): Nhận được yêu cầu HTTP.
4. Backend (Node.js):
 - Lưu lại hành động này vào collection `deviceactivities` trong MongoDB Atlas.
 - Gửi (Publish) nội dung `{ "device": "led_1", "status": "ON" }` lên topic `devices/control` của MQTT Broker.
5. MQTT Broker: Nhận tin nhắn và chuyển tiếp tới các client đang đăng ký topic này.
6. ESP32: Nhận được tin nhắn từ topic `devices/control`.
7. ESP32: Phân tích nội dung JSON, xác định cần điều khiển chân `LED_1_PIN` sang trạng thái HIGH.
8. ESP32: Thực thi lệnh, đèn LED vật lý sáng lên.



4. Thiết kế Cơ sở dữ liệu (Database Design)






Hệ thống sử dụng cơ sở dữ liệu NoSQL là MongoDB, được triển khai trên nền tảng đám mây MongoDB Atlas. Thiết kế gồm hai bộ sưu tập (collections) chính, tương ứng với hai loại dữ liệu cần lưu trữ.

a) Collection sensordatas

Collection này được thiết kế để lưu trữ các bản ghi dữ liệu cảm biến theo thời gian.

Cấu trúc: Mỗi bản ghi (document) sẽ chứa các trường sau:

- `_id`: (ObjectId) Khóa chính duy nhất do MongoDB tự động tạo.
- `temperature`: (Number) Giá trị nhiệt độ đo được (đơn vị: °C).
- `humidity`: (Number) Giá trị độ ẩm đo được (đơn vị: %).
- `light`: (Number) Giá trị cường độ ánh sáng đo được (đơn vị: Lux).
- `createdAt`: (Date) Dấu thời gian (timestamp) do Mongoose tự động tạo khi bản ghi được chèn vào, cho biết thời điểm dữ liệu được ghi nhận.
- `updatedAt`: (Date) Dấu thời gian do Mongoose tự động tạo khi bản ghi được cập nhật.





tbl_data-sensor		
 Id	integer(10)	
 Lux	float(10)	N
 Temp	float(10)	N
 Humidity	float(10)	N
 Time	timestamp	N

b) Collection deviceactivities

Collection này được thiết kế để lưu trữ nhật ký của tất cả các hành động điều khiển thiết bị.

Cấu trúc: Mỗi bản ghi sẽ chứa các trường sau:

- `_id`: (ObjectId) Khóa chính duy nhất.
- `device`: (String) Tên của thiết bị được điều khiển (ví dụ: "led_1", "led_2").
- `status`: (String) Trạng thái được yêu cầu (ví dụ: "ON", "OFF").
- `createdAt`: (Date) Dấu thời gian cho biết thời điểm hành động được thực hiện.
- `updatedAt`: (Date) Dấu thời gian cập nhật.

tbl_device_activity			
	Id	integer(10)	
	Device	varchar(50)	N
	Action	varchar(50)	N
	Time	timestamp	N

Chương 3: Xây dựng hệ thống

Quá trình triển khai hệ thống được thực hiện dựa trên các bản thiết kế chi tiết đã trình bày ở Phần 2. Phần này sẽ đi sâu vào các khía cạnh kỹ thuật trong việc xây dựng từng thành phần chính của hệ thống, bao gồm tài liệu API, lập trình phần cứng, và kiến trúc phần mềm phía máy chủ (backend).

1. Tài liệu API (API Documentation)

API (Application Programming Interface) đóng vai trò là "bản hợp đồng" giao tiếp, định nghĩa cách thức frontend và backend tương tác với nhau. Hệ thống cung cấp một bộ RESTful API được thiết kế để xử lý các tác vụ cụ thể.

Biến môi trường: `baseUrl`: `http://localhost:3001`

a) API Điều khiển Thiết bị

- Endpoint: `POST /api/control`
- Mục đích: Gửi lệnh điều khiển (bật/tắt) đến một thiết bị cụ thể.
- Phương thức: `POST`
- Body (JSON):

```

1  {
2    "device": "led_1",
3    "status": "ON"
4  }

```

- Phản hồi thành công (200 OK):

```

1  {
2    "success": true,
3    "message": "Command sent and activity logged"
4  }

```

- Phản hồi lỗi (400 Bad Request):

```

1  {
2    "error": "Missing device or status"
3  }

```

b) API Lịch sử Cảm biến

- Endpoint: GET /api/history/sensors
- Mục đích: Lấy danh sách dữ liệu cảm biến đã được lưu trữ, hỗ trợ phân trang, sắp xếp và tìm kiếm.
- Mô tả: Truy vấn dữ liệu cảm biến đã được lưu trữ với khả năng phân trang, sắp xếp và tìm kiếm.
- Phương thức: GET
- Query Parameters (Tùy chọn):
 - ❖ page: Số trang muốn lấy (mặc định: 1).
 - ❖ limit: Số lượng bản ghi trên mỗi trang (mặc định: 10).
 - ❖ sortBy: Sắp xếp theo trường (createdAt, temperature, humidity, light).
 - ❖ order: Thứ tự sắp xếp (asc hoặc desc).
 - ❖ searchType: Loại tìm kiếm (time, temperature, humidity, light).
 - ❖ searchValue: Giá trị tìm kiếm.
- Phản hồi thành công (200 OK):

```

{
  "data": [
    {
      "_id": "68df458175ef6a81f8b05bcd",
      "temperature": 30.1,
      "humidity": 78.1,
      "light": 722.1281,
      "createdAt": "2025-10-03T03:39:45.332Z",
      "updatedAt": "2025-10-03T03:39:45.332Z",
      "__v": 0
    },
    {
      "_id": "68df457e75ef6a81f8b05bcb",
      "temperature": 30.1,
      "humidity": 78.1,
      "light": 726.8737,
      "createdAt": "2025-10-03T03:39:42.361Z",
      "updatedAt": "2025-10-03T03:39:42.361Z",
      "__v": 0
    },
    {
      "_id": "68df457b75ef6a81f8b05bc9",
      "temperature": 30.1,
      "humidity": 78.1,
      "light": 734.8819,
      "createdAt": "2025-10-03T03:39:39.290Z",

```

c) API Lịch sử Hoạt động

- Endpoint: GET /api/history/activity
- Mục đích: Lấy nhật ký hoạt động của thiết bị.
- Mô tả: Truy vấn nhật ký hoạt động của các thiết bị với khả năng phân trang và lọc.
- Phương thức: GET
- Query Parameters (Tùy chọn):
 - ❖ page, limit, sortBy, order.
 - ❖ filterDevice (string): Lọc theo tên thiết bị (led_1, led_2, led_3).
 - ❖ filterStatus (string): Lọc theo hành động (ON, OFF).
 - ❖ searchTime (string): Tìm kiếm theo thời gian.
- Phản hồi thành công (200 OK):


```

1  {
2      "data": [
3          {
4              "_id": "68df457a75ef6a81f8b05bc7",
5              "device": "led_3",
6              "status": "OFF",
7              "createdAt": "2025-10-03T03:39:38.481Z",
8              "updatedAt": "2025-10-03T03:39:38.481Z",
9              "__v": 0
10         },
11         {
12             "_id": "68df44ac75ef6a81f8b05b3d",
13             "device": "led_3",
14             "status": "ON",
15             "createdAt": "2025-10-03T03:36:12.529Z",
16             "updatedAt": "2025-10-03T03:36:12.529Z",
17             "__v": 0
18         },
19         {
20             "_id": "68df409e75ef6a81f8b0583b",
21             "device": "led_1",
22             "status": "OFF",
23             "createdAt": "2025-10-03T03:18:54.881Z",
24             "updatedAt": "2025-10-03T03:18:54.881Z",
25             "__v": 0
26         }
27     ]
28 }

```

d) API Thông tin cá nhân

- Endpoint: GET /api/profile
- Mục đích: Lấy thông tin cá nhân để hiển thị trên trang Profile.
- Mô tả: Lấy thông tin cá nhân tĩnh để hiển thị trên trang Profile.
- Phương thức: GET
- Phản hồi thành công (200 OK):

```

1  {
2      "fullName": "Luu Xuân Dũng",
3      "studentId": "B22DCCN129",
4      "className": "D22HTTT05",
5      "phone": "0967430627",
6      "avatarUrl": "/images/avatar.jpg",
7      "githubUrl": "https://github.com/xdungok/IOT_Project",
8      "pdfUrl": "/files/IOT-Bài thực hành 1-Luu Xuân Dũng.pdf",
9      "apiDocsUrl": "https://docs.google.com/document/d/1FYtPp6mpB8gwxHzUQit3BjDZAs5r3A15Yt6ulBjnbbw8/edit?usp=sharing"
10 }

```

2. Xây dựng Phần cứng

Phần cứng là lớp thu thập dữ liệu và thực thi tại biên. Việc lập trình cho vi điều khiển ESP32 được thực hiện bằng Arduino IDE với các thư viện chuyên dụng.

a) Các bước thực hiện

- Chuẩn bị Linh kiện: Chuẩn bị các linh kiện đã nêu trong thiết kế: 1 bo mạch ESP32, 1 cảm biến DHT11, 1 quang trở, 3 đèn LED, điện trở và dây cắm.

- Lắp ráp Mạch: Kết nối các linh kiện với bo mạch ESP32 theo sơ đồ mạch đã thiết kế. Cảm biến DHT11 được nối vào chân Digital, quang trở được nối vào chân Analog, và các đèn LED được nối vào các chân Digital khác.
- Lập trình Vi điều khiển:
 - ❖ Sử dụng Arduino IDE để lập trình cho ESP32.
 - ❖ Cài đặt các thư viện cần thiết: WiFi.h, PubSubClient.h, ArduinoJson.h, DHT.h.
 - ❖ Viết mã nguồn để thực hiện các chức năng: kết nối Wi-Fi, kết nối MQTT Broker (với xác thực username/password), định kỳ đọc dữ liệu từ cảm biến, quy đổi giá trị ADC của quang trở sang Lux, đóng gói dữ liệu thành JSON và gửi lên topic sensors/data.
 - ❖ Viết mã nguồn để lắng nghe topic devices/control, nhận lệnh, phân tích JSON và điều khiển trạng thái các chân GPIO tương ứng để bật/tắt đèn LED.
- Nạp chương trình và Kiểm tra: Nạp code vào ESP32 và sử dụng Serial Monitor để theo dõi log, đảm bảo thiết bị kết nối và gửi/nhận dữ liệu thành công.
- b) Giải thích các hàm chính trong mã nguồn Arduino:
 - Hàm setup():
 - ❖ Mô tả: Đây là hàm khởi tạo, chỉ chạy một lần duy nhất khi thiết bị khởi động.
 - ❖ Nhiệm vụ:
 1. Khởi tạo giao tiếp Serial để gỡ lỗi (Serial.begin).
 2. Cấu hình các chân GPIO điều khiển LED làm OUTPUT.
 3. Gọi hàm setup_wifi() để kết nối vào mạng Wi-Fi.
 4. Cấu hình thông tin kết nối đến MQTT Broker (client.setServer).
 5. Đăng ký hàm callback để xử lý các tin nhắn MQTT đến (client.setCallback).

```

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

    // Cài đặt các chân pin
    for (int i = 0; i < numDevices; i++) {
        pinMode(devices[i].pin, OUTPUT);
    }

    dht.begin();

    setup_wifi();
    client.setServer(mqtt_server, 1883);
    client.setCallback(callback);
}

```

- Hàm loop():
 - ❖ Mô tả: Là vòng lặp chính của chương trình, được thực thi liên tục sau khi setup() hoàn tất.
 - ❖ Nhiệm vụ:
 1. Kiểm tra và duy trì kết nối MQTT (if (!client.connected()) reconnect();).
 2. Gọi client.loop() để xử lý các tác vụ nền của thư viện MQTT.
 3. Sử dụng một bộ đếm thời gian (millis()) để thực hiện việc đọc và gửi dữ liệu sau mỗi interval (3 giây), tránh việc gửi dữ liệu quá dồn dập.
 4. Trong mỗi chu kỳ, gọi các hàm đọc cảm biến, quy đổi sang Lux, đóng gói thành JSON và publish lên topic sensors/data.

```

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

    unsigned long now = millis();
    if (now - lastMsg > interval) {
        lastMsg = now;

        float h = dht.readHumidity();
        float t = dht.readTemperature();
        int lightAdc = analogRead(LDR_PIN);
        float lightLux = readLux(lightAdc); // Quy đổi giá trị ADC sang Lux

        if (isnan(h) || isnan(t)) {
            Serial.println("Failed to read from DHT sensor!");
            return;
        }

        StaticJsonDocument<200> doc;
        doc["temperature"] = t;
        doc["humidity"] = h;
        doc["light"] = lightLux; // Gửi giá trị Lux đã được quy đổi
        char buffer[256];
        size_t n = serializeJson(doc, buffer);

        client.publish("sensors/data", buffer, n);

        Serial.print("Published message: ");
        Serial.println(buffer);
    }
}

```

- callback(char* topic, byte* payload, unsigned int length):
 - ❖ Mô tả: Hàm này được thư viện PubSubClient tự động gọi mỗi khi có một tin nhắn mới đến từ một topic mà thiết bị đã đăng ký (ở đây là devices/control).
 - ❖ Nhiệm vụ:
 1. Nhận dữ liệu thô (payload).
 2. Phân tích chuỗi JSON để trích xuất thông tin device và status.
 3. Dựa trên các giá trị này, thực hiện logic điều khiển digitalWrite() để thay đổi trạng thái của chân GPIO tương ứng.

```

void callback(char* topic, byte* payload, unsigned int length) {
    Serial.print("Message arrived [");
    Serial.print(topic);
    Serial.print("] ");

    // Tạo một chuỗi từ payload nhận được
    char message[length + 1];
    for (int i = 0; i < length; i++) {
        message[i] = (char)payload[i];
    }
    message[length] = '\0';
    Serial.println(message);

    // Phân tích chuỗi JSON
    StaticJsonDocument<200> doc;
    DeserializationError error = deserializeJson(doc, message);

    if (error) {
        Serial.print(F("deserializeJson() failed: "));
        Serial.println(error.f_str());
        return;
    }

    const char* device = doc["device"]; // Lấy tên thiết bị
    const char* status = doc["status"]; // Lấy trạng thái

```

```

// 1. Xử lý các lệnh điều khiển nhóm ("all")
if (strcmp(device, "all") == 0) {
    if (strcmp(status, "ON") == 0) {
        for (int i = 0; i < numDevices; i++) {
            digitalWrite(devices[i].pin, HIGH);
        }
        Serial.println("All devices turned ON");
    } else if (strcmp(status, "OFF") == 0) {
        for (int i = 0; i < numDevices; i++) {
            digitalWrite(devices[i].pin, LOW);
        }
        Serial.println("All devices turned OFF");
    }
    return;
}

// 2. Nếu không phải lệnh nhóm, tìm và xử lý thiết bị cụ thể
int pinToControl = -1;
for (int i = 0; i < numDevices; i++) {
    if (strcmp(device, devices[i].name) == 0) {
        pinToControl = devices[i].pin;
        break;
    }
}

```

```
// 3. Thực hiện bật/tắt nếu tìm thấy thiết bị hợp lệ
if (pinToControl != -1) {
    if (strcmp(status, "ON") == 0) {
        digitalWrite(pinToControl, HIGH);
        Serial.print(device);
        Serial.println(" turned ON");
    } else if (strcmp(status, "OFF") == 0) {
        digitalWrite(pinToControl, LOW);
        Serial.print(device);
        Serial.println(" turned OFF");
    }
} else {
    Serial.print("Unknown device: ");
    Serial.println(device);
}
}
```

- setup_wifi() và reconnect():
 - ❖ Mô tả: Đây là các hàm phụ trợ quan trọng.
 - ❖ setup_wifi() xử lý quá trình kết nối đến Wi-Fi.
 - ❖ reconnect() xử lý logic kết nối lại với MQTT Broker nếu kết nối bị mất, bao gồm cả việc gửi thông tin xác thực và đăng ký (subscribe) lại các topic cần thiết sau khi kết nối thành công.

```
void setup_wifi() {
    delay(10);
    Serial.println();
    Serial.print("Connecting to ");
    Serial.println(ssid);
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("\nWiFi connected");
    Serial.print("IP address: ");
    Serial.println(WiFi.localIP());
}
```

```

void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    String clientId = "ESP32Client-";
    clientId += String(random(0xffff), HEX);

    if (client.connect(clientId.c_str(), mqtt_user, mqtt_pass)) {
      Serial.println("connected");
      client.subscribe("devices/control");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}
}

```

3. Backend

Backend được xây dựng trên nền tảng Node.js với framework Express.js, theo kiến trúc module hóa để đảm bảo tính rõ ràng, khả năng bảo trì và mở rộng. Dưới đây là phân tích chi tiết về các thành phần và hàm quan trọng.

Giải thích các thành phần chính:

a) Điểm vào và Luồng khởi tạo (server.js)

File server.js là điểm khởi đầu của toàn bộ ứng dụng. Nó thực hiện việc khởi tạo các thành phần theo một trình tự logic chặt chẽ.


```

1  require('dotenv').config();
2
3  const http = require('http');
4  const app = require('./app');
5  const config = require('./config');
6  const websocketService = require('./services/websocketService');
7  const mqttService = require('./services/mqttService');
8  const database = require('./services/database');
9  const notificationService = require('./services/notificationService');
10
11 // Kết nối tới Database
12 database.connect();
13
14 // Khởi tạo WebSocket Service
15 websocketService.init();
16
17 // Khởi tạo Notification Service và truyền hàm broadcast vào
18 notificationService.init(websocketService.broadcast);
19
20 // Khởi tạo MQTT Service và truyền hàm broadcast vào
21 mqttService.init(websocketService.broadcast);
22
23 // Tạo và khởi động HTTP Server
24 const server = http.createServer(app);
25 server.listen(config.server.webServerPort, () => {
26 |   console.log(`HTTP Server is running on http://localhost:${config.server.webServerPort}`);
27 | });

```

Giải thích:

- `require('dotenv').config()`: Lệnh này phải được gọi đầu tiên để nạp các biến từ file `.env` vào `process.env`, giúp các module khác (như `config/index.js`) có thể truy cập được các thông tin nhạy cảm.
- `database.connect()`: Ưu tiên kết nối tới cơ sở dữ liệu trước tiên. Nếu kết nối thất bại, ứng dụng sẽ tự động thoát, đảm bảo hệ thống không chạy trong trạng thái lỗi.
- `websocketService.init()`: Khởi tạo server WebSocket để sẵn sàng nhận kết nối từ các client frontend.
- `mqttService.init(websocketService.broadcast)`: Khởi tạo MQTT client. Bước này minh họa kỹ thuật Dependency Injection (Tiêm phụ thuộc), nơi `mqttService` nhận hàm `broadcast` từ `websocketService`. Điều này cho phép `mqttService` có thể gửi dữ liệu trực tiếp ra frontend mà không cần phải biết về cách `websocketService` hoạt động.

b) Lắng nghe và Xử lý Dữ liệu MQTT (services/mqttService.js)

Đây là cầu nối chính giữa thế giới IoT và thế giới Web. Hàm xử lý sự kiện `message` là trung tâm của mọi hoạt động giám sát.

```

backend > services > JS mqttService.js > init
1  const mqtt = require('mqtt');
2  const config = require('../config');
3  const SensorData = require('../models/sensorData');
4  const notificationService = require('./notificationService');
5
6  let mqttClient;
7
8  // Nhận hàm broadcast từ websocketService
9  function init(broadcastCallback) {
10     mqttClient = mqtt.connect(config.mqtt.brokerUrl, {
11         username: config.mqtt.username,
12         password: config.mqtt.password,
13     });
14
15     mqttClient.on('connect', () => {
16         console.log('Connected to MQTT Broker!');
17         mqttClient.subscribe(config.mqtt.topics.sensorData, (err) => {
18             if (!err) {
19                 console.log(`Subscribed to topic: ${config.mqtt.topics.sensorData}`);
20             }
21         });
22     });
23
24     mqttClient.on('message', async (topic, payload) => {
25         if (topic === config.mqtt.topics.sensorData) {
26             try {
27                 const data = JSON.parse(payload.toString());
28                 console.log('Received sensor data:', data);
29                 notificationService.processSensorData(data);
30             }

```

```

31 // Lưu dữ liệu vào database
32 const newSensorData = new SensorData(data);
33 await newSensorData.save();
34
35 // Đẩy dữ liệu real-time ra frontend
36 broadcastCallback({
37   type: 'sensorData',
38   payload: data,
39   timestamp: new Date().toLocaleTimeString('vi-VN')
40 });
41 } catch (error) {
42   console.error('Error processing sensor data:', error);
43 }
44 }
45 });
46
47 mqttClient.on('error', (error) => console.error('MQTT Client error:', error));
48 }
49
50 function publishControlMessage(device, status) {
51   const message = JSON.stringify({ device, status });
52   mqttClient.publish(config.mqtt.topics.deviceControl, message, (err) => {
53     if (err) {
54       console.error('Failed to publish control message:', err);
55     } else {
56       console.log(`Published to ${config.mqtt.topics.deviceControl}: ${message}`);
57     }
58   });
59 }
60
61 module.exports = {
62   init,
63   publishControlMessage
64 };

```

Giải thích: Hàm này được thiết kế để thực hiện ba tác vụ quan trọng một cách phi đồng bộ mỗi khi có dữ liệu cảm biến mới:

- Phân tích: Dữ liệu được chuyển ngay đến notificationService để phân tích và phát hiện các mẫu bất thường theo thời gian thực.
- Lưu trữ: Lệnh `await newSensorData.save()` đảm bảo dữ liệu được ghi lại một cách an toàn vào MongoDB Atlas trước khi tiếp tục.
- Phát sóng: Dữ liệu được "phát sóng" đến tất cả các client frontend đang kết nối qua WebSocket, phục vụ cho việc cập nhật giao diện Dashboard.

c) Xây dựng API Động (controllers/historyController.js)

Hàm `getSensorHistory`, nằm trong file `controllers/historyController.js`, là một ví dụ điển hình về cách xây dựng một API mạnh mẽ và linh hoạt. Nó được thiết kế để xử lý nhiều tham số đầu vào từ phía client, xây dựng một câu truy vấn MongoDB phức tạp một cách động, và trả về một bộ dữ liệu có cấu trúc rõ ràng, hỗ trợ đầy đủ cho các chức năng phân trang, sắp xếp và tìm kiếm trên giao diện người dùng.

Dưới đây là phân tích chi tiết từng bước logic bên trong hàm.

1) Trích xuất và Chuẩn hóa Tham số (Parameter Extraction and Sanitization)

Bước đầu tiên của bất kỳ hàm API nào là thu thập và chuẩn hóa dữ liệu đầu vào từ yêu cầu (request) của client.

```
const getSensorHistory = async (req, res) => {  
  try {  
    const page = parseInt(req.query.page) || 1;  
    const limit = parseInt(req.query.limit) || 10;  
    const sortBy = req.query.sortBy || 'createdAt';  
    const order = req.query.order === 'asc' ? 1 : -1;  
    const searchType = req.query.searchType;  
    const searchValue = req.query.searchValue;
```

Giải thích:

- req.query: Tất cả các tham số được truyền qua URL (ví dụ: ?page=2&limit=10) được chứa trong đối tượng req.query.
- Cung cấp giá trị Mặc định: Việc sử dụng toán tử || (OR) là một kỹ thuật quan trọng để đảm bảo API luôn hoạt động ổn định. Nếu client không cung cấp một tham số (ví dụ: page), hệ thống sẽ tự động gán một giá trị mặc định (ví dụ: 1). Điều này giúp API trở nên thân thiện và dễ sử dụng hơn.
- Chuẩn hóa Dữ liệu:
 - ❖ parseInt(): Chuyển đổi các giá trị chuỗi ('2') từ URL thành kiểu số nguyên (2) để sử dụng trong các phép tính toán học và truy vấn database.
 - ❖ order === 'asc' ? 1 : -1: Chuyển đổi một chuỗi thân thiện với người dùng ('asc', 'desc') thành giá trị số (1, -1) mà MongoDB yêu cầu cho việc sắp xếp.

2) Xây dựng Đối tượng Truy vấn Động (Dynamic Query Object Construction)

Thay vì viết các câu truy vấn cứng, chúng ta xây dựng một đối tượng điều kiện (queryObject) một cách linh hoạt dựa trên các tham số mà người dùng cung cấp.

```

47 let queryObject = {};
48
49 if (searchType && searchValue) {
50   if (searchType === 'time') {
51     // Dịch chuỗi đầu vào
52     const isoDateString = parseCustomDateString(searchValue);
53
54     if (isoDateString) {
55       const startDate = new Date(isoDateString);
56       let endDate;
57       if (searchValue.split(' ')[0].split(':').length === 2) {
58         // Nếu không có giây, tìm trong cả phút
59         endDate = new Date(startDate.getTime() + 60 * 1000);
60       } else {
61         // Nếu có giây, tìm trong cả giây
62         endDate = new Date(startDate.getTime() + 1000);
63       }
64
65       queryObject.createdAt = {
66         $gte: startDate,
67         $lt: endDate
68       };
69     }
70   } else if (['temperature', 'humidity', 'light'].includes(searchType)) {
71     queryObject[searchType] = { $gte: parseFloat(searchValue) };
72   }
73 }

```

Giải thích:

- queryObject: Ban đầu là một đối tượng rỗng. Nếu không có điều kiện tìm kiếm nào, Model.find({}) sẽ trả về tất cả các bản ghi.
- Tìm kiếm theo Thời gian:
 - ❖ parseCustomDateString: Gọi một hàm tiện ích để chuyển đổi định dạng ngày tháng thân thiện với người dùng (HH:mm DD/MM/YYYY) thành định dạng ISO mà JavaScript có thể hiểu được.
 - ❖ { \$gte: ..., \$lt: ... }: Đây là cú pháp truy vấn của MongoDB, cho phép tìm kiếm trong một khoảng (range). \$gte là "lớn hơn hoặc bằng", \$lt là "nhỏ hơn". Bằng cách này, chúng ta có thể tìm chính xác các bản ghi trong một giây hoặc một phút cụ thể.
- Tìm kiếm theo Giá trị Cảm biến:
 - ❖ queryObject[searchType] = { ... }: Kỹ thuật này cho phép tên của trường cần tìm kiếm (temperature, humidity, hoặc light) được xác định một cách động tại thời điểm chạy. Điều này giúp tránh việc phải viết các khối if/else dài dòng cho từng loại cảm biến, làm cho code ngắn gọn và dễ mở rộng hơn.

3) Thực hiện Truy vấn và Tối ưu hóa Hiệu năng (Query Execution and Performance Optimization)

Sau khi đã có đầy đủ các điều kiện, hệ thống sẽ gửi yêu cầu đến database. Ở đây, việc tối ưu hóa hiệu năng được đặt lên hàng đầu.

```
75     const [data, totalDocuments] = await Promise.all([
76         SensorData.find(queryObject)
77             .sort({ [sortBy]: order })
78             .skip(skip)
79             .limit(limit),
80         SensorData.countDocuments(queryObject)
81     ]);
```

Giải thích:

- **Promise.all:** Đây là một kỹ thuật tối ưu hóa quan trọng. Thay vì thực hiện tuần tự (chờ find xong rồi mới countDocuments), Promise.all cho phép gửi cả hai yêu cầu đến MongoDB cùng một lúc. Database sẽ xử lý chúng song song, và backend sẽ chỉ tiếp tục khi cả hai yêu cầu đều đã có kết quả. Điều này giúp giảm đáng kể thời gian phản hồi của API.
- **Chaining Methods (Nối chuỗi phương thức):** Mongoose cho phép nối các phương thức lại với nhau một cách rất trực quan:
 - ❖ `find(queryObject)`: Lọc ra các bản ghi khớp điều kiện.
 - ❖ `sort({ [sortBy]: order })`: Sắp xếp kết quả đã lọc.
 - ❖ `skip(...)`: Bỏ qua các bản ghi của những trang trước đó.
 - ❖ `limit(...)`: Chỉ lấy đủ số lượng bản ghi cho trang hiện tại.
- **countDocuments(queryObject):** Đây là một phương thức được tối ưu hóa cao của MongoDB, chỉ thực hiện việc đếm mà không cần phải tải toàn bộ dữ liệu về, giúp việc lấy tổng số bản ghi trở nên cực kỳ nhanh chóng.

4) Định dạng Phản hồi (Response Formatting)

Bước cuối cùng là đóng gói kết quả vào một cấu trúc JSON nhất quán và gửi về cho client.

```

83     const totalPages = Math.ceil(totalDocuments / limit);
84
85     res.status(200).json({
86       data,
87       pagination: {
88         currentPage: page,
89         totalPages,
90         totalDocuments,
91         limit
92       }
93     });
94

```

Giải thích: Phản hồi được thiết kế để cung cấp cho frontend mọi thông tin cần thiết trong một lần gọi duy nhất.

- data: Mảng chứa các bản ghi dữ liệu thực tế của trang hiện tại.
- pagination: Một đối tượng riêng chứa tất cả các "siêu dữ liệu" (metadata) về việc phân trang, giúp frontend có thể dễ dàng xây dựng giao diện các nút "First", "Previous", "1, 2, 3...", "Next", "Last" một cách chính xác. Việc tính toán totalPages ở backend giúp giảm tải logic cho frontend.

d) Logic Cảnh báo Thông minh (services/notificationService.js)

Đây là module phức tạp nhất, chịu trách nhiệm phân tích dữ liệu theo chuỗi thời gian. Nó sử dụng một đối tượng sensorStates để duy trì "bộ nhớ" về trạng thái của từng cảm biến.

```

backend > services > JS notificationService.js > ...
17   const sensorStates = {
18     temperature: createInitialState(),
19     humidity: createInitialState(),
20     light: createInitialState()
21   };

```

```
function checkThresholdBreach(sensor, value, state) {
  const now = Date.now();
  const config = THRESHOLDS[sensor];

  if (value < config.low || value > config.high) {
    state.breachCounter++;
    state.normalCounter = 0;
    if (state.breachCounter === 2 && (now - state.lastNotificationTime.threshold > COOLDOWNS.threshold)) {
      const direction = value < config.low ? 'thấp' : 'cao';
      sendNotification('warning', `Cảnh báo: ${sensor} vượt ngưỡng ${direction} (${value.toFixed(1)})`);
      state.lastNotificationTime.threshold = now;
      state.isAlarmActive = true;
    }
  } else {
    state.breachCounter = 0;
    state.normalCounter++;
    if (state.isAlarmActive && state.normalCounter === 3) {
      sendNotification('info', `Thông báo: ${sensor} đã trở lại trạng thái bình thường.`);
      state.isAlarmActive = false; // Reset cờ cảnh báo
    }
  }
}
```

Giải thích:

- State Management: sensorStates là một ví dụ về quản lý trạng thái phía server. Nó lưu giữ các thông tin như giá trị trước đó (previousValue), số lần vượt ngưỡng liên tiếp (breachCounter), v.v.
- Logic có điều kiện: Hàm checkThresholdBreach minh họa cách áp dụng các quy tắc phức tạp. Nó không chỉ kiểm tra giá trị hiện tại mà còn dựa vào các giá trị đã lưu trong state (như breachCounter) và các quy tắc về thời gian (COOLDOWNS) để đưa ra quyết định gửi cảnh báo, giúp hệ thống trở nên "thông minh" và tránh spam thông báo. Các hàm checkSuddenChange và checkFluctuation cũng hoạt động dựa trên nguyên tắc tương tự.

Chương 4: Kết luận

1. Kết luận

Hệ thống đã được triển khai và kiểm thử, cho thấy khả năng hoạt động ổn định và hiệu quả. Các chức năng chính như giám sát dữ liệu thời gian thực, điều khiển thiết bị từ xa, lưu trữ và truy vấn dữ liệu lịch sử, cùng với hệ thống cảnh báo thông minh, đều hoạt động đúng như thiết kế.

- Về mặt chức năng: Dự án đã tạo ra một nền tảng IoT "end-to-end", cho phép thu thập, truyền tải, xử lý, lưu trữ và trực quan hóa dữ liệu một cách liền mạch.
- Về mặt kỹ thuật: Việc áp dụng kiến trúc module hóa ở cả backend và frontend, kết hợp với các giao thức và công nghệ phù hợp (MQTT cho

IoT, WebSocket cho real-time web, RESTful API cho các tác vụ khác), đã tạo nên một hệ thống vững chắc và có hiệu năng tốt.

- Về mặt trải nghiệm người dùng: Giao diện web trực quan, đáp ứng nhanh và cung cấp đầy đủ các công cụ cần thiết để người dùng có thể tương tác và quản lý hệ thống một cách dễ dàng.

2. Đánh giá Dự án

a) (Strengths)

- Kiến trúc Toàn diện và Linh hoạt: Hệ thống được xây dựng theo kiến trúc 3 lớp và mô hình Pub/Sub, giúp các thành phần có tính độc lập cao, dễ dàng thay thế, nâng cấp hoặc mở rộng từng phần mà không ảnh hưởng đến toàn bộ hệ thống.
- Hiệu năng Thời gian thực: Việc sử dụng MQTT và WebSocket cho các tác vụ giao tiếp real-time đảm bảo độ trễ thấp, mang lại trải nghiệm giám sát và điều khiển tức thì cho người dùng.
- Khả năng Phân tích Dữ liệu Nâng cao: Backend được trang bị các API mạnh mẽ, cho phép thực hiện các thao tác tìm kiếm, sắp xếp và lọc phức tạp trực tiếp trên cơ sở dữ liệu, tối ưu hóa hiệu năng và giảm tải cho phía client.
- Hệ thống Cảnh báo Thông minh: Logic cảnh báo phía server không chỉ dựa trên ngưỡng đơn giản mà còn phân tích được các mẫu dữ liệu theo thời gian (thay đổi đột ngột, dao động liên tục), giúp cảnh báo trở nên chính xác và hữu ích hơn, đồng thời tránh được tình trạng "spam" thông báo.
- Tính Sẵn sàng cho Mở rộng: Việc sử dụng các công nghệ phổ biến và tuân thủ các nguyên tắc thiết kế phần mềm tốt (ví dụ: quản lý cấu hình bằng biến môi trường, chia nhỏ code thành các module) tạo ra một nền tảng vững chắc cho các phát triển trong tương lai.

b) Hạn chế và Tồn tại

- Bảo mật: Mặc dù giao tiếp MQTT đã được bảo vệ bằng username/password, hệ thống tổng thể vẫn còn thiếu các lớp bảo mật quan trọng như:
 - ❖ Xác thực người dùng (Authentication): Chưa có hệ thống đăng nhập/đăng xuất cho người dùng cuối trên giao diện web.

- ❖ Phân quyền (Authorization): Chưa có cơ chế phân quyền, ví dụ vai trò Admin và User.
- ❖ Mã hóa Giao tiếp: Giao tiếp HTTP và WebSocket hiện chưa sử dụng mã hóa SSL/TLS (HTTPS/WSS).
- Khả năng Mở rộng Quy mô (Scalability): Việc sử dụng một MQTT Broker đơn lẻ và một instance backend duy nhất có thể trở thành điểm nghẽn (bottleneck) khi số lượng thiết bị và người dùng tăng lên đáng kể.
- Triển khai: Hệ thống hiện tại đang được triển khai trên môi trường local (máy tính cá nhân), chưa được đóng gói và triển khai lên một môi trường cloud để có thể hoạt động 24/7 và truy cập được từ xa.
- Tính Năng Người dùng: Các tính năng như cho phép người dùng tự tùy chỉnh ngưỡng cảnh báo hay xem biểu đồ dữ liệu lịch sử theo khoảng thời gian tùy chọn vẫn chưa được triển khai.

3. Những bài học Kinh nghiệm

Qua quá trình thực hiện đồ án, bản thân đã rút ra được nhiều bài học quý báu:

- Tầm quan trọng của Thiết kế: Việc đầu tư thời gian vào giai đoạn thiết kế (kiến trúc, giao diện, API, CSDL) giúp quá trình xây dựng diễn ra suôn sẻ và giảm thiểu các thay đổi lớn về sau.
- Lựa chọn Công nghệ Phù hợp: Hiểu rõ ưu và nhược điểm của từng công nghệ để lựa chọn giải pháp tối ưu cho từng bài toán cụ thể (ví dụ: MQTT cho IoT vs. WebSocket cho Web).
- Tư duy Full-stack: Nắm vững được luồng dữ liệu "end-to-end", từ cách vi điều khiển hoạt động, cách dữ liệu được truyền đi, xử lý ở backend, lưu trữ và cuối cùng là hiển thị trên frontend.
- Kỹ năng Gỡ lỗi (Debugging): Quá trình xây dựng một hệ thống phức tạp với nhiều thành phần đòi hỏi kỹ năng gỡ lỗi một cách có hệ thống, từ việc kiểm tra log ở từng thành phần (ESP32, Broker, Backend, Frontend Console) đến việc sử dụng các công cụ như Postman.

4. Định hướng Phát triển

Dựa trên các hạn chế đã phân tích, dự án có thể được phát triển theo các hướng sau:

- Hoàn thiện Hệ thống Bảo mật:

- ❖ Tích hợp hệ thống xác thực người dùng sử dụng JWT (JSON Web Tokens).
- ❖ Xây dựng cơ chế phân quyền dựa trên vai trò (Role-Based Access Control).
- ❖ Cấu hình SSL/TLS cho tất cả các kênh giao tiếp để mã hóa dữ liệu.
- Nâng cao Trải nghiệm Người dùng:
 - ❖ Xây dựng giao diện cho phép người dùng tự cấu hình các quy tắc và ngưỡng cảnh báo.
 - ❖ Phát triển các công cụ trực quan hóa dữ liệu lịch sử mạnh mẽ hơn (chọn khoảng thời gian, so sánh dữ liệu).
- Xây dựng Ứng dụng Di động:
 - ❖ Phát triển một ứng dụng di động (sử dụng React Native) kết nối tới cùng hệ thống backend để mang lại sự tiện lợi tối đa cho người dùng.
- Tối ưu hóa và Triển khai lên Cloud:
 - ❖ Đóng gói ứng dụng backend và database bằng Docker để dễ dàng quản lý và triển khai.
 - ❖ Triển khai toàn bộ hệ thống (MQTT Broker, Backend, Frontend) lên một nền tảng đám mây (như AWS, Google Cloud) để đảm bảo tính sẵn sàng cao và khả năng truy cập toàn cầu.
- Ứng dụng Trí tuệ Nhân tạo (AI/ML):
 - ❖ Sử dụng dữ liệu lịch sử đã thu thập để huấn luyện các mô hình học máy, nhằm dự đoán các sự kiện (ví dụ: dự đoán mức tiêu thụ điện) hoặc phát hiện các mẫu bất thường mà các quy tắc thủ công không thể nhận ra.