

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN  
THÔNG

KHOA CÔNG NGHỆ THÔNG TIN 1



**BÁO CÁO BÀI TẬP IOT VÀ ỨNG DỤNG**

**ĐỀ TÀI: XÂY DỰNG HỆ THỐNG CẢM BIẾN IOT**

**Họ và tên: Đỗ Duy Đông**

**Mã sinh viên: B22DCCN215**

**Nhóm lớp học: 05**

**Giảng viên giảng dạy: Nguyễn Quốc Uy**

**HÀ NỘI – 2025**

# Mục lục

<b>Chương 1: Giới thiệu</b>	<b>3</b>
<b>I. Tổng quan dự án</b>	<b>3</b>
1. Mục đích của dự án IoT Dashboard	3
2. Các tính năng chính của hệ thống	4
<b>II. Công nghệ sử dụng</b>	<b>4</b>
1. Backend (Node.js, Express.js)	5
2. Frontend (React.js)	5
3. Cơ sở dữ liệu (MySQL với Prisma ORM)	5
4. Giao thức truyền thông MQTT	5
5. Phần cứng ESP32, DHT11	5
<b>Chương 2: Giao diện</b>	<b>6</b>
<b>I. Trang Dashboard</b>	<b>6</b>
<b>II. Trang Data Sensor</b>	<b>7</b>
<b>III. Trang Action History</b>	<b>8</b>
<b>IV. Trang Profile</b>	<b>9</b>
<b>Chương 3: Thiết kế tổng thể và chi tiết</b>	<b>10</b>
<b>I. Kiến trúc hệ thống</b>	<b>10</b>
1. Sơ đồ tổng quan về kiến trúc hệ thống	10
2. Mô tả luồng dữ liệu và tương tác giữa các thành phần	10
<b>II. Thiết kế Backend</b>	<b>13</b>
1. Cấu trúc thư mục và file	13
2. API endpoints và chức năng	13
3. Xử lý dữ liệu cảm biến và điều khiển thiết bị	13
<b>III. Kết nối MQTT</b>	<b>14</b>
1. Cấu hình MQTT broker	14
2. Xử lý tin nhắn MQTT từ ESP32	
3. Gửi lệnh điều khiển qua MQTT	
<b>IV. Mô hình dữ liệu</b>	<b>22</b>
1. Cấu trúc bảng dữ liệu cảm biến	22
2. Cấu trúc bảng lịch sử hành động	22
<b>V. Lập trình ESP32</b>	
1. Cấu hình kết nối WiFi và MQTT	

2. Đọc dữ liệu từ cảm biến và gửi qua MQTT .....	
3. Nhận lệnh điều khiển và thực thi .....	
<b>Chương 4: Kết quả .....</b>	<b>22</b>
<b>I. Chức năng đã hoàn thành .....</b>	<b>22</b>
1. Liệt kê các tính năng đã triển khai thành công .....	22
<b>II. Hiệu suất hệ thống .....</b>	<b>23</b>
1. Đánh giá về tốc độ phản hồi .....	23
2. Độ chính xác của dữ liệu cảm biến .....	24
<b>III. Cải tiến trong tương lai.....</b>	<b>24</b>
1. Tăng cường bảo mật.....	24
2. Cải thiện khả năng mở rộng.....	24
3. Nâng cao trải nghiệm người dùng.....	24
4. Mở rộng khả năng tương thích.....	24
5. Tối ưu hóa năng lượng.....	24
6. Cải thiện khả năng phân tích.....	24
7. Tăng cường độ tin cậy.....	25
<b>Lời cảm ơn .....</b>	<b>26</b>

## Chương 1: Giới thiệu

### I. Tổng quan dự án

#### 1. Mục đích của dự án IoT Dashboard

Dự án IoT Dashboard được phát triển nhằm mục đích tạo ra một hệ thống giám sát và điều khiển thông minh cho môi trường trong nhà. Hệ thống này kết hợp công nghệ Internet of Things (IoT) với giao diện người dùng trực quan, cho phép người dùng theo dõi các thông số môi trường và điều khiển các thiết bị từ xa một cách dễ dàng và hiệu quả.

Các mục tiêu chính của dự án bao gồm:

1. Giám sát môi trường: Thu thập và hiển thị dữ liệu về nhiệt độ, độ ẩm và ánh sáng trong thời gian thực.
2. Điều khiển thiết bị: Cung cấp khả năng điều khiển từ xa các thiết bị như đèn LED, quạt và điều hòa không khí.

3. Phân tích dữ liệu: Lưu trữ và phân tích dữ liệu lịch sử để đưa ra các xu hướng và thông tin chi tiết về môi trường.
  4. Tự động hóa: Thiết lập các quy tắc tự động để điều chỉnh thiết bị dựa trên các điều kiện môi trường.
  5. Giao diện thân thiện: Cung cấp một dashboard trực quan và dễ sử dụng cho người dùng.
2. Các tính năng chính của hệ thống
- **Hiển thị dữ liệu thời gian thực:**
    - Biểu đồ động hiển thị nhiệt độ, độ ẩm và cường độ ánh sáng.
    - Cập nhật liên tục từ các cảm biến được kết nối.
  - **Điều khiển bật tắt các thiết bị khác từ xa**
  - **Bảng điều khiển tương tác**
    - Giao diện trực quan cho phép người dùng dễ dàng xem và điều khiển thiết bị.
    - Hiển thị trạng thái hiện tại của tất cả các thiết bị được kết nối.
  - **Lưu trữ và truy xuất dữ liệu lịch sử**
    - Lưu trữ tất cả dữ liệu cảm biến và hành động điều khiển.
    - Cung cấp khả năng xem và phân tích dữ liệu lịch sử. ○ **Bảo mật**
    - Xác thực người dùng để đảm bảo chỉ những người được ủy quyền mới có thể truy cập và điều khiển hệ thống.
    - Mã hóa dữ liệu truyền tải giữa các thành phần của hệ thống.
  - **API RESTful**
    - Cung cấp API cho phép tích hợp với các hệ thống và ứng dụng khác.
    - Hỗ trợ truy vấn dữ liệu và điều khiển thiết bị thông qua các yêu cầu HTTP.
  - **Khả năng mở rộng**
    - Thiết kế module cho phép dễ dàng thêm các loại cảm biến và thiết bị mới.
    - Hỗ trợ nhiều phòng hoặc khu vực khác nhau trong một hệ thống.

Dự án IoT Dashboard này không chỉ cung cấp một giải pháp toàn diện cho việc giám sát và điều khiển môi trường trong nhà, mà còn đặt nền tảng cho việc phát triển và mở rộng trong tương lai, hướng tới một hệ sinh thái nhà thông minh hoàn chỉnh.

## II. Công nghệ sử dụng

Dự án IoT Dashboard sử dụng một loạt các công nghệ hiện đại để xây dựng một hệ thống giám sát và điều khiển toàn diện. Dưới đây là chi tiết về các công nghệ được sử dụng trong từng phần của dự án

## 1. Backend (Node.js, Express.js)

- Node.js: Nền tảng chạy JavaScript phía máy chủ, được sử dụng để xây dựng server.
- Express.js: Framework web cho Node.js, giúp xây dựng API RESTful một cách nhanh chóng và hiệu quả.
- Prisma ORM: ORM (Object-Relational Mapping) hiện đại cho Node.js và TypeScript, giúp tương tác với cơ sở dữ liệu một cách dễ dàng và type-safe.

## 2. Frontend (React.js)

- React.js: Thư viện JavaScript để xây dựng giao diện người dùng, giúp tạo ra các ứng dụng web động và hiệu quả.
- React Router: Thư viện định tuyến cho React, giúp xây dựng ứng dụng một trang (SPA) với nhiều route.
- Chart.js: Thư viện JavaScript để vẽ biểu đồ, được sử dụng để hiển thị dữ liệu cảm biến dưới dạng đồ thị.
- Axios: Thư viện HTTP client dựa trên Promise, được sử dụng để gửi yêu cầu API từ frontend đến backend.
- WebSocket: Công nghệ cho phép giao tiếp hai chiều giữa client và server, được sử dụng để cập nhật dữ liệu thời gian thực.

## 3. Cơ sở dữ liệu (MongoDB)

MongoDB - Hệ quản trị cơ sở dữ liệu quan hệ mã nguồn mở, được sử dụng để

- Lưu trữ dữ liệu cảm biến (nhiệt độ, độ ẩm, ánh sáng) theo thời gian thực.
- Ghi lại lịch sử hoạt động của các thiết bị (bật/tắt đèn, quạt, điều hòa).
- Quản lý thông tin người dùng và quyền truy cập.

## 4. Giao thức truyền thông MQTT

- MQTT (Message Queuing Telemetry Transport): Giao thức truyền thông nhẹ, được sử dụng để truyền dữ liệu giữa ESP32 và server.
- HTTP/HTTPS: Giao thức truyền tải siêu văn bản, được sử dụng trong API RESTful để giao tiếp giữa frontend và backend.

## 5. Phần cứng ESP32, DHT11

- Arduino IDE: Môi trường phát triển tích hợp được sử dụng để lập trình cho ESP32.
- ESP32: Vi điều khiển có khả năng kết nối WiFi và Bluetooth, được sử dụng làm thiết bị IoT chính.
- DHT11: Cảm biến nhiệt độ và độ ẩm, được sử dụng để đo các thông số môi trường.
- LDR (Light Dependent Resistor): Cảm biến ánh sáng, được sử dụng để đo cường độ ánh sáng.

## 6. Công cụ phát triển quản lý mã nguồn

- Git: Hệ thống quản lý phiên bản phân tán, được sử dụng để quản lý mã nguồn.
- GitHub: Nền tảng lưu trữ mã nguồn trực tuyến, được sử dụng để lưu trữ và chia sẻ mã nguồn dự án.
- Postman: Công cụ phát triển API, được sử dụng để kiểm thử và tài liệu hóa API.

Việc sử dụng các công nghệ này cho phép xây dựng một hệ thống IoT Dashboard mạnh mẽ, có khả năng mở rộng và dễ bảo trì. Sự kết hợp giữa các công nghệ frontend hiện đại, backend mạnh mẽ và thiết bị IoT linh hoạt tạo nên một giải pháp toàn diện cho việc giám sát và điều khiển môi trường thông minh.

## Chương 2: Giao diện

### I. Trang Dashboard



Hình 1. Trang Dashboard

- Bảng thông số hiện tại
  - Nhiệt độ (Temperature): Hiển thị nhiệt độ hiện tại với biểu tượng nhiệt kế.
  - Độ ẩm (Humidity): Hiển thị độ ẩm hiện tại với biểu tượng giọt nước.
  - Ánh sáng (Light): Hiển thị cường độ ánh sáng hiện tại với biểu tượng mặt trời.
- Biểu đồ theo dõi
  - Biểu đồ kết hợp hiển thị dữ liệu nhiệt độ và độ ẩm theo thời gian.
    - Đường màu đỏ thể hiện nhiệt độ.

- Đường màu xanh dương thể hiện độ ẩm.
- Biểu đồ ánh sáng hiển thị cường độ ánh sáng theo thời gian bằng đường màu vàng.
- Điều khiển thiết bị:  
Phần "Device" hiển thị các thiết bị có thể điều khiển
  - Quạt (Fan): Biểu tượng quạt với công tắc bật/tắt.
  - Điều hòa (Air Conditioner): Biểu tượng điều hòa với công tắc bật/tắt.
  - Đèn (Light): Biểu tượng bóng đèn với công tắc bật/tắt. ○ Giao diện này cho phép người dùng:
    - Theo dõi các thông số môi trường theo thời gian thực.
    - Xem xu hướng thay đổi của nhiệt độ, độ ẩm và ánh sáng qua thời gian.
    - Điều khiển các thiết bị như quạt, điều hòa và đèn một cách thuận tiện.

## II. Trang Data Sensor

Trang chủ

Dữ liệu cảm biến

Lịch sử

Hồ sơ

Dữ liệu cảm biến

Tất cả thiết bị

Tìm kiếm

Tìm kiếm

ID	NHIỆT ĐỘ (°C)	ĐỘ ẨM (%)	ÁNH SÁNG (MTS)	THỜI GIAN
27.9	63.5	494	13:42:48 26/09/2025	
27.9	63.5	508	13:42:46 26/09/2025	
27.9	63.5	573	13:42:44 26/09/2025	
27.8	63.5	464	13:42:42 26/09/2025	
27.8	63.6	503	13:42:40 26/09/2025	
27.8	63.5	569	13:42:38 26/09/2025	
27.8	63.5	705	13:42:36 26/09/2025	
27.7	63.4	511	13:42:34 26/09/2025	

Hình 2. Trang Data Sensor

- Thanh tìm kiếm:
  - Ô nhập liệu cho phép tìm kiếm theo ID, Nhiệt độ, Độ ẩm, Ánh sáng, hoặc Thời gian.
  - Dropdown menu để chọn phạm vi tìm kiếm (hiện đang hiển thị "Tất cả").
  - Nút "Tìm kiếm" để thực hiện tìm kiếm.
- Bảng dữ liệu cảm biến:
  - Hiển thị dữ liệu dưới dạng bảng với các cột:
  - ID: Số định danh duy nhất cho mỗi bản ghi.
  - Nhiệt độ (°C): Nhiệt độ được đo.
  - Độ ẩm (%): Độ ẩm được đo.
  - Ánh sáng (lux): Cường độ ánh sáng được đo.

- Thời gian: Thời gian ghi nhận dữ liệu.

Mỗi cột có biểu tượng mũi tên lên/xuống, cho phép sắp xếp dữ liệu.

○ Trang Data Sensor này cho phép người dùng

- Xem dữ liệu cảm biến chi tiết theo thời gian.
- Tìm kiếm dữ liệu cụ thể dựa trên các tiêu chí khác nhau.
- Sắp xếp dữ liệu theo bất kỳ cột nào để phân tích xu hướng.
- Theo dõi sự thay đổi của các thông số môi trường theo thời gian.

Giao diện này rất hữu ích cho việc phân tích dữ liệu lịch sử, kiểm tra xu hướng, và xác định bất kỳ điểm bất thường nào trong môi trường được giám sát.

### III. Trang Action History

Trang chủ	Dữ liệu cảm biến	Lịch sử	Hồ sơ																																				
<div>Lịch sử</div> <div> <div>Tất cả</div> <div> <input type="text" value="Tìm kiếm"/> <div>Tìm kiếm</div> </div> </div> <table> <tr> <th>ID</th><th>THIẾT BỊ</th><th>HÀNH ĐỘNG</th><th>THỜI GIAN</th></tr> <tr> <td>1</td><td>led2</td><td>Tắt</td><td>13:18:28 26/09/2025</td></tr> <tr> <td>2</td><td>led2</td><td>Bật</td><td>13:18:22 26/09/2025</td></tr> <tr> <td>3</td><td>led2</td><td>Tắt</td><td>11:57:08 26/09/2025</td></tr> <tr> <td>4</td><td>led2</td><td>Bật</td><td>11:57:05 26/09/2025</td></tr> <tr> <td>5</td><td>led2</td><td>Tắt</td><td>11:56:52 26/09/2025</td></tr> <tr> <td>6</td><td>led2</td><td>Bật</td><td>11:56:49 26/09/2025</td></tr> <tr> <td>7</td><td>led2</td><td>Tắt</td><td>11:26:13 26/09/2025</td></tr> <tr> <td>8</td><td>led2</td><td>Bật</td><td>11:26:12 26/09/2025</td></tr> </table>				ID	THIẾT BỊ	HÀNH ĐỘNG	THỜI GIAN	1	led2	Tắt	13:18:28 26/09/2025	2	led2	Bật	13:18:22 26/09/2025	3	led2	Tắt	11:57:08 26/09/2025	4	led2	Bật	11:57:05 26/09/2025	5	led2	Tắt	11:56:52 26/09/2025	6	led2	Bật	11:56:49 26/09/2025	7	led2	Tắt	11:26:13 26/09/2025	8	led2	Bật	11:26:12 26/09/2025
ID	THIẾT BỊ	HÀNH ĐỘNG	THỜI GIAN																																				
1	led2	Tắt	13:18:28 26/09/2025																																				
2	led2	Bật	13:18:22 26/09/2025																																				
3	led2	Tắt	11:57:08 26/09/2025																																				
4	led2	Bật	11:57:05 26/09/2025																																				
5	led2	Tắt	11:56:52 26/09/2025																																				
6	led2	Bật	11:56:49 26/09/2025																																				
7	led2	Tắt	11:26:13 26/09/2025																																				
8	led2	Bật	11:26:12 26/09/2025																																				

Hình 3. Trang Action History

○ Thanh tìm kiếm:

- Dropdown menu để chọn loại thiết bị (hiện đang hiển thị "ALL").
- Ô nhập liệu cho phép tìm kiếm theo thời gian, với định dạng "hh:mm:ss dd/mm/yyyy".

- Nút "Search" để thực hiện tìm kiếm.
- Bảng lịch sử hành động:
- Hiển thị dữ liệu dưới dạng bảng với các cột:
- ID: Số định danh duy nhất cho mỗi hành động.
- Device: Loại thiết bị được điều khiển
- Hành động: Trạng thái của thiết bị (ON hoặc OFF).
- Time: Thời gian thực hiện hành động.

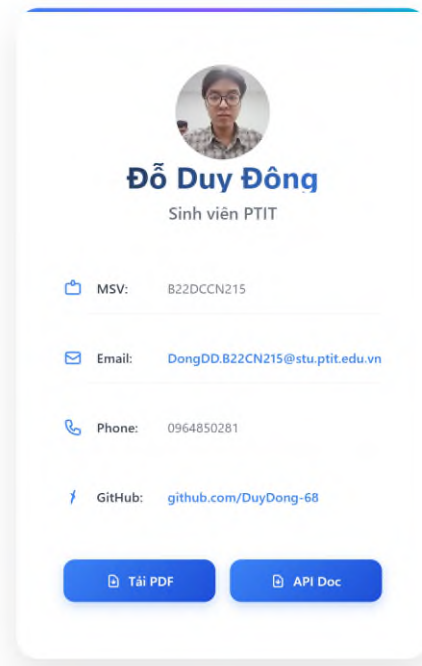
○ Trang Action History này cho phép người dùng:

- Xem lịch sử chi tiết về các hành động điều khiển thiết bị.
- Tìm kiếm các hành động cụ thể dựa trên loại thiết bị hoặc thời gian.
- Theo dõi thứ tự và tần suất của các hành động điều khiển.



- Phân tích mô hình sử dụng thiết bị theo thời gian. ● Giao diện này rất hữu ích cho việc:
- Kiểm tra lịch sử hoạt động của hệ thống.
- Xác định các mô hình sử dụng thiết bị.
- Điều tra các vấn đề hoặc sự cố liên quan đến việc điều khiển thiết bị.
- Tối ưu hóa việc sử dụng năng lượng bằng cách phân tích thói quen sử dụng thiết bị.

## IV. Trang Profile



Hình 4. Trang Profile

- Thẻ thông tin cá nhân: Được hiển thị dưới dạng một thẻ màu xanh dương nhạt ở giữa trang.
- Ảnh đại diện: Hình ảnh tròn của người dùng nằm ở phía trên của thẻ.
- Thông tin cơ bản:
  - Tên: Đỗ Duy Đông • Chức danh: Student at PTIT
  - Thông tin chi tiết:
  - MSV (Mã sinh viên): B22DCCN215
  - Email: DongDD.B22CN215@stu.ptit.edu.vn (với biểu tượng email)
  - Số điện thoại: 0949794366 (với biểu tượng điện thoại)
  - GitHub: DuyDong
  - Download PDF: Liên kết để tải xuống tài liệu PDF (với biểu tượng tài liệu)
  - API Doc: Liên kết đến tài liệu API (với biểu tượng sách) ● Trang Profile này cung cấp:
  - Thông tin cá nhân và học tập của người dùng.
  - Các phương thức liên lạc (email, số điện thoại).
  - Liên kết đến tài khoản GitHub cá nhân.

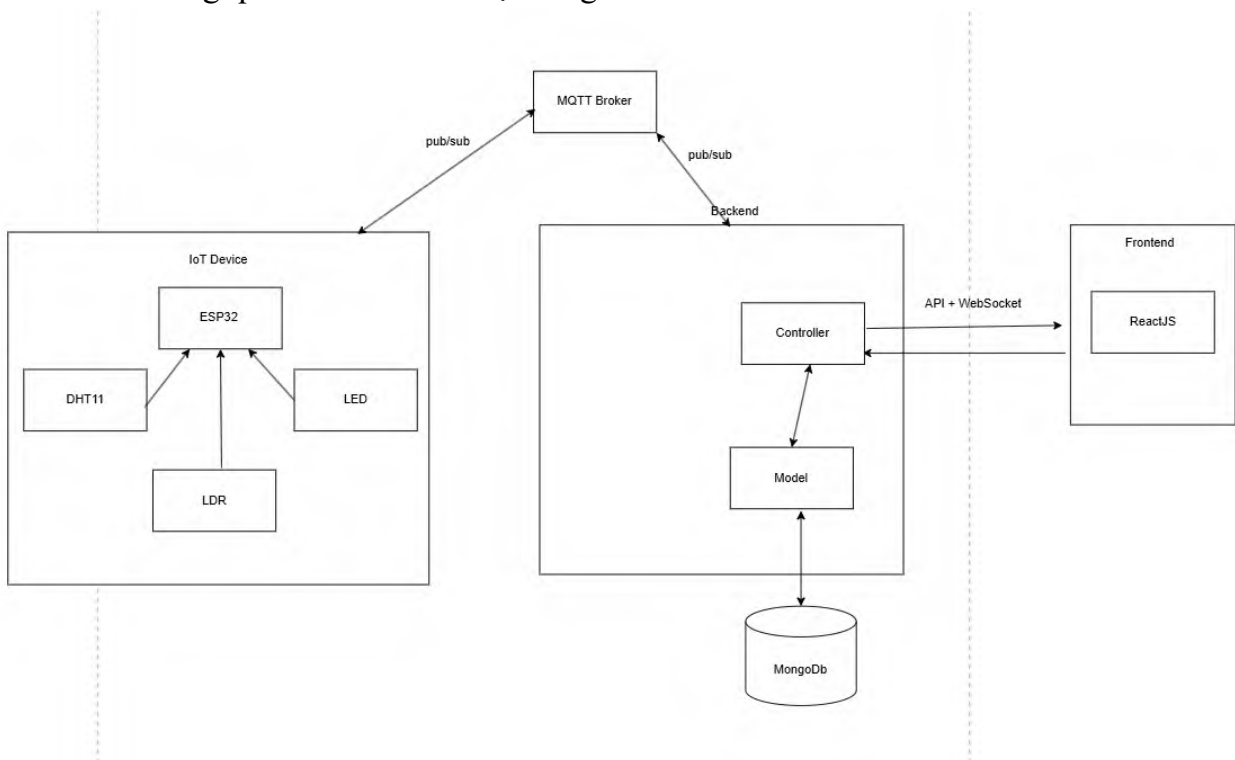
- Khả năng tải xuống tài liệu PDF của dự án.
- Truy cập nhanh đến tài liệu API của dự án. ● Thiết kế gọn gàng và trực quan này giúp:
  - Người dùng dễ dàng xem và cập nhật thông tin cá nhân.
  - Cung cấp một cách nhanh chóng để liên hệ với chủ dự án.
  - Tạo điều kiện thuận lợi cho việc chia sẻ thông tin dự án và tài liệu kỹ thuật.

Giao diện này thể hiện tính chuyên nghiệp và cung cấp một cách tiếp cận tập trung để quản lý thông tin cá nhân và tài liệu dự án trong một hệ thống IoT Dashboard.

## Chương 3: Thiết kế tổng thể và chi tiết

### I. Kiến trúc hệ thống

#### 1. Tổng quan về kiến trúc hệ thống

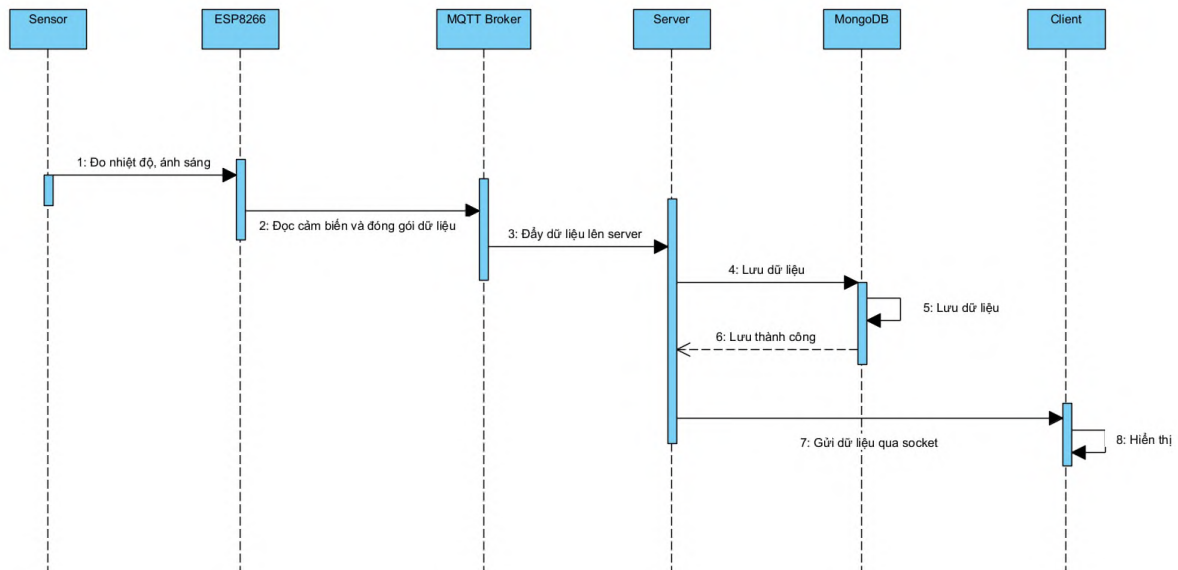


- Hệ thống của bạn bao gồm các thành phần chính sau:
- ESP32: Thiết bị phần cứng đọc dữ liệu từ cảm biến và điều khiển các thiết bị như đèn LED.
- MQTT Broker: Trung gian truyền tin giữa ESP32 và Backend.
- Backend (Node.js): Xử lý logic nghiệp vụ, lưu trữ dữ liệu, và cung cấp API cho Frontend.
- Database (MongoDb): Lưu trữ dữ liệu cảm biến và lịch sử hành động.
- Frontend (React): Giao diện người dùng để hiển thị dữ liệu và điều khiển thiết bị.

#### 2. Mô tả luồng dữ liệu và tương tác giữa các thành phần

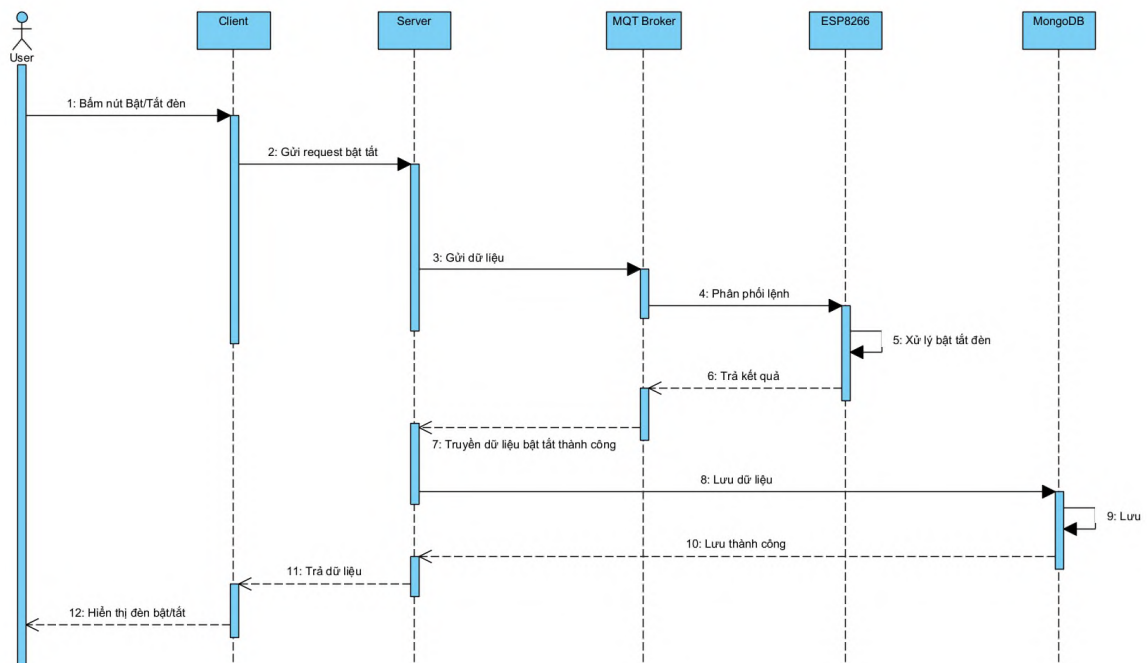
##### a. Luồng dữ liệu cảm biến

ESP32 đọc dữ liệu từ cảm biến (nhiệt độ, độ ẩm, ánh sáng)→ESP32 gửi dữ liệu qua MQTT đến MQTT Broker→Backend nhận dữ liệu từ MQTT Broker và lưu vào database→Frontend nhận dữ liệu từ Backend qua socket để lấy dữ liệu mới nhất và hiển thị.



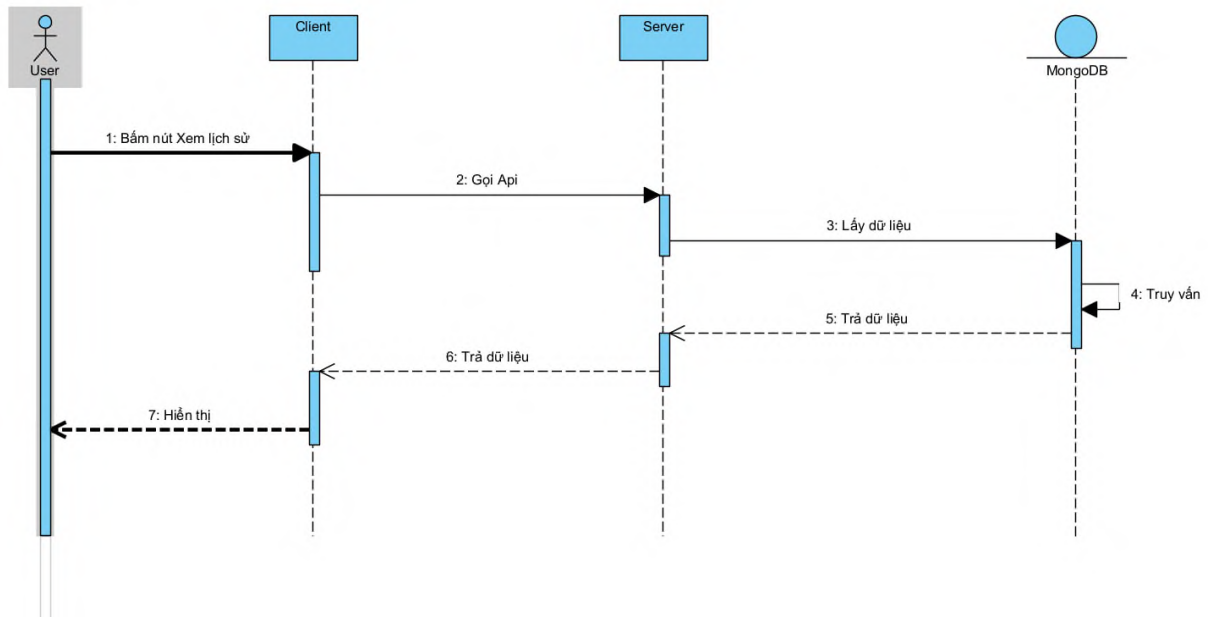
#### b. Luồng điều khiển thiết bị

Người dùng tương tác với giao diện trên Frontend để điều khiển thiết bị→Frontend gửi yêu cầu điều khiển đến Backend thông qua API→Backend xử lý yêu cầu và gửi lệnh điều khiển qua MQTT đến ESP32→ESP32 nhận lệnh và thực hiện điều khiển thiết bị tương ứng.



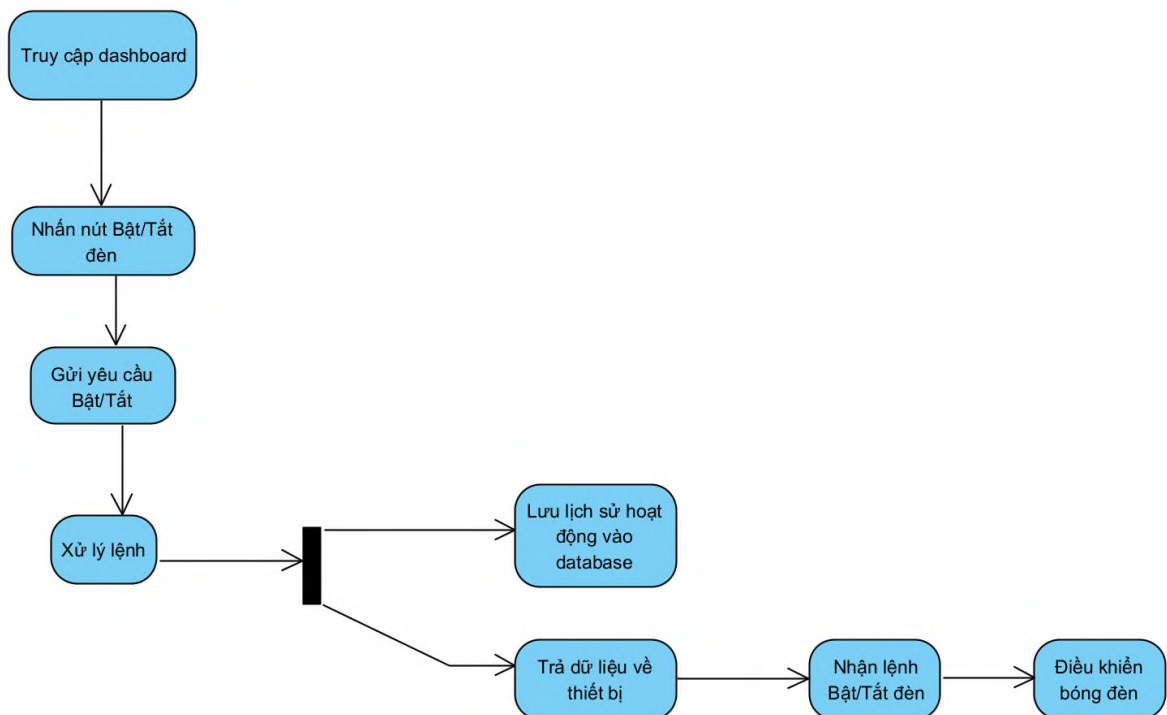
### c) Luồng xem lịch sử điều khiển đèn

Người dùng bấm nút Xem lịch sử trên giao diện. Frontend gửi yêu cầu lấy dữ liệu đến Backend thông qua API. Backend truy vấn dữ liệu từ database rồi gửi về Frontend. Frontend nhận được dữ liệu thì hiển thị lên màn hình

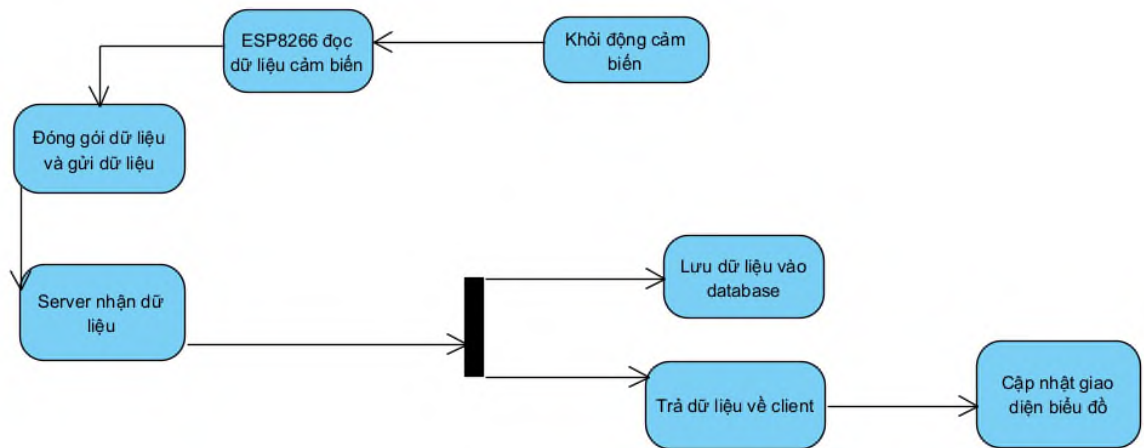


### 3. Biểu đồ activity

#### a) Biểu đồ bật tắt đèn

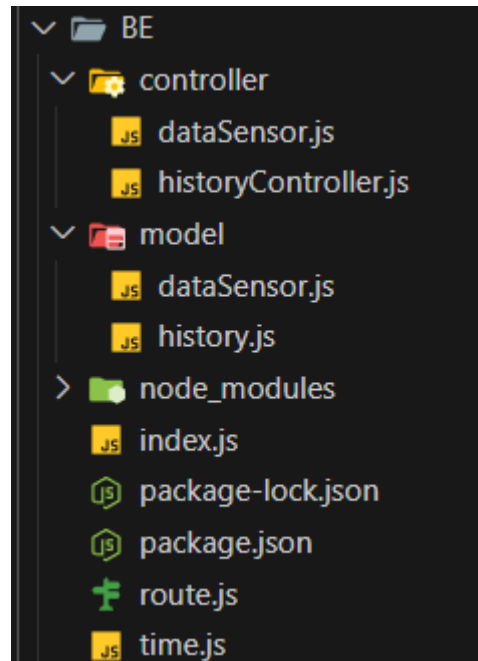


#### b) Biểu đồ lấy dữ liệu cảm biến



## II. Thiết kế Backend

### 1. Cấu trúc thư mục và file



### 2. API endpoints và chức năng

#### a) Dữ liệu cảm biến:

GET /api/getDataSensor: Lấy dữ liệu của cảm biến được lưu trong database

#### b) Lịch sử hành động:

GET /api/getHistory: Lấy lịch sử hành động với các tùy chọn lọc và phân trang

### 3. Xử lý dữ liệu cảm biến và điều khiển thiết bị

- Dữ liệu cảm biến được nhận qua MQTT và lưu vào database thông qua hàm DataSensor trong models/dataSensor.js.

- Lệnh điều khiển thiết bị được nhận qua API, xử lý trong controllers/history.controller.js, và gửi qua MQTT đến ESP32.

### III. Lập trình MQTT và ESP32

#### 1. Cấu hình MQTT broker

```
const char* ssid = "dep";  
const char* password = "06090412";  
const char* mqtt_user = "tripled";  
const char* mqtt_pass = "842004";  
const char* mqtt_server = "192.168.119.129";  
const int mqtt_port = 1883;
```

#### 2. Kết nối wifi

```
void setup_wifi() {  
  Serial.begin(115200);  
  delay(10);  
  Serial.print("Connecting to WiFi: ");  
  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());  
}
```

#### 3. Subscribe vào các topic cần thiết:

```

void reconnect() {
  while (!mqttClient.connected()) {
    Serial.print("Attempting MQTT connection...");
    if (mqttClient.connect("ESP32_DHT_Client",mqtt_user, mqtt_pass)) {
      Serial.println("connected");
      mqttClient.subscribe("home/sensor");
      mqttClient.subscribe("home/led1");
      mqttClient.subscribe("home/led2");
      mqttClient.subscribe("home/led3");
    } else {
      Serial.print("failed, rc=");
      Serial.print(mqttClient.state());
      Serial.println(" try again in 2 seconds");
      delay(2000);
    }
  }
}

```

4. Nhận tin nhắn bật tắt đèn từ Backend gửi vào topic và giả tín hiệu thành công:

```

void callback(char* topic, byte* payload, unsigned int length) {
  String message;
  for (int i = 0; i < length; i++) {
    message += (char)payload[i];
  }
  if (String(topic) == "home/led1") {
    if (message == "ON") {
      digitalWrite(LED1, HIGH);
      ledState1=HIGH;
    } else if (message == "OFF") {
      digitalWrite(LED1, LOW);
      ledState1=LOW;
    }
    if (digitalRead(LED1) == ledState1){
      char payload[100];
      snprintf(payload,sizeof(payload),"{\"led1\": %d}",ledState1);
      mqttClient.publish("home/ledStatus",payload
    );
    }
  }
}

```



## 5. Đọc dữ liệu từ cảm biến và gửi qua MQTT

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

  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;

    float temperature = dht.readTemperature();
    float humidity = dht.readHumidity();
    int digitalValue = digitalRead(D0); // 0 hoặc 1
    int analogValue = analogRead(A0); // 0 - 4095 (ESP32 ADC 12-bit)

    if (isnan(temperature)) temperature = 0;
    if (isnan(humidity)) humidity = 0;

    char payload[100];
    snprintf(payload, sizeof(payload), "{\"temperature\": %.2f, \"humidity\": %.2f, \"light\": %d}", temperature, humidity, analogValue);
    mqttClient.publish("home/sensor", (uint8_t*)payload, strlen(payload), true); // QoS 1
  }
}
```

## IV. Xử lý Backend

### 1. Kết nối MQTT

```
mqttClient.on('connect',()=>{
  console.log('Connected to MQTT broker');
  mqttClient.subscribe('home/sensor',(err)=>{
    if(!err){
      console.log('Subscribed to topic home/sensor');
    }else{
      console.error('Failed to subscribe:', err);
    }
  })
  mqttClient.subscribe('home/ledStatus',(err)=>{
    if(!err){
      console.log('Subscribed to topic home/ledStatus');
    }else{
      console.error('Failed to subscribe:', err);
    }
  })
})
})
```

### 2. Nhận dữ liệu qua topic và gửi lên socket

#### a) Dữ liệu cảm biến



```

mqttClient.on('message', async (topic, message) => {
  if (topic === "home/sensor") {
    // Lưu DB
    const tmpData = JSON.parse(message.toString());
    try {
      const record = new dataSensor({
        temperature: tmpData.temperature,
        light: tmpData.light,
        humidity: tmpData.humidity
      });
      await record.save();
    } catch (error) {
      console.log(error);
    }
    // Gửi msg đi
    io.emit('dataSensor', tmpData);
  }
}

```

b) Gửi trạng thái đèn đến esp

```

io.on('connection', (socket) => {
  console.log('Connected client');
  socket.on('ledReq', (data) => {
    let topic = `home/${data.name}`;
    let msg = data.status ? "ON" : "OFF";
    mqttClient.publish(topic, msg, err => {
      if (err) {
        console.log('Failed to publish message', err);
      }
    });
  });
});

```

c) Lưu trạng thái bật tắt đèn

```
if(topic == "home/ledStatus"){
  // Lưu DB
  try {
    const tmpLed=JSON.parse(message.toString());
    let keyLed="";
    let valueLed=0;
    for(key in tmpLed){
      keyLed= key;
      valueLed= tmpLed[key];
      break;
    }
    const record= new History({
      device: keyLed,
      action: valueLed == 1 ? "ON":"OFF"
    });
    await record.save();
    io.emit('ledStatus',{
      name: keyLed,
      msg: 1
    })
  })
}
```

d) Tắt hết đèn khi rút esp

```
if(topic=="home/status"){
  const msg= message.toString();
  if(msg=="ON") console.log("System is ON");
  else{
    let devices= ["led1","led2","led3"];
    for(let i in devices){
      const led= await History.findOne({device: devices[i]}).sort({createdAt: -1});
      if(led.action=="ON"){
        const record= new History({
          device: devices[i],
          action: "OFF"
        })
        await record.save();
      }
    }
    io.emit('ledStatus',{
      name: "all",
      msg: 0
    })
  }
}
```

### 3. API lấy dữ liệu cảm biến

```

module.exports.getData = async (req, res) => {
  try {
    // await DataSensor.deleteMany({});
    let { keyword, sortKey, sortValue, page, filter,frequent } = req.query;
    let limit= frequent ? frequent : 10;
    let find = {};
    let sort = {};
    if (sortKey && sortValue) sort[sortKey] = parseInt(sortValue)
    else sort['createdAt'] = -1
    if(filter){
      keyword = keyword ? keyword.trim() : "";
      let arr=[]
      if (filter == "all") {
        if(!isNaN(parseFloat(keyword))){
          arr=[
            { temperature: parseFloat(keyword) },
            { humidity: parseFloat(keyword) },|
            { light: parseFloat(keyword) }
          ]
        }
        if(isNaN(keyword)){
          const { start, end } = configTime(keyword);
          arr.push({
            createdAt: {
              $gte: start,
              $lte: end
            }
          })
        }
        find = {
          $or: arr
        }
      } else if (filter == "createdAt") {
        const { start, end } = configTime(keyword);
        find['createdAt'] = {
          $gte: start,
          $lte: end
        }
      }
    }
  }
}

```

```

        else {
            find = {
                [filter]: parseFloat(keyword)
            }
        }
    }
}

const totalDoc = await DataSensor.countDocuments(find);
const skipPage = page ? (page - 1) * limit : 0;

var data = await DataSensor.find(
    find
).sort(sort).skip(skipPage).limit(limit).lean();

data.map(item => {
    item['createdAt'] = moment(item.createdAt).format("HH:mm:ss DD/MM/YYYY");
})
res.status(200).json({
    doc: data,
    totalPage: Math.ceil(totalDoc / limit),
    currentPage: page ? parseInt(page) : 1
});
} catch (error) {
    console.log(error);
    res.status(500).json({ message: "Internal server error" });
}
}

```

#### 4. API lấy dữ liệu lịch sử bật tắt

```

module.exports.getHistory = async (req,res)=>{
  try {
    const {page,filter,keyword,sortKey,sortValue, frequent}= req.query;
    let find={};
    let limit= frequent ? frequent : 10;
    let sort={};
    // Sort
    if(sortKey && sortValue) sort[sortKey]= parseInt(sortValue)
    else sort['createdAt']= -1;

    // Filter
    if(filter){
      find.device=filter;
    }
    // search
    if(keyword){
      const {start,end}= configTime(keyword);
      find.createdAt={
        $gte: start,
        $lte: end
      }
    }
    // Pagination
    const totalDoc= await History.countDocuments(find);
    const skipPage= (page-1)*limit;
    const totalPage= Math.ceil(totalDoc/limit);
    // Query DB
    let history= await History.find(find).skip(skipPage).limit(limit).sort(sort).lean();
    history.forEach(item=>{
      item.createdAt= moment(item.createdAt).format('HH:mm:ss DD/MM/YYYY');
    })
    res.status(200).json({
      doc:history,
      totalPage:totalPage
    });
  } catch (error) {
    console.log(error)
  }
}

```

## 5.API lấy dữ liệu đèn






```

module.exports.getLedStatus= async (req,res)=>{
  try {
    const arr=["led1","led2","led3"];
    let arr1=[]
    for( i in arr){
      const latestData= await History.findOne({device:arr[i]}).sort({createdAt:-1}).lean();
      arr1.push({
        device: arr[i],
        status: latestData ? latestData.action : "OFF"
      })
    }
    res.status(200).json(arr1);
  } catch (error) {
    console.log(error);
    res.status(500).json({ message: "Internal server error" });
  }
}





```

## V. Mô hình dữ liệu

### 1. Cấu trúc bảng dữ liệu cảm biến

Data_Sensor			
	id	integer(10)	N
	temperature	integer(10)	N
	humidity	integer(10)	N
	light	integer(10)	N
	create_time	timestamp	N

### 2. Cấu trúc bảng lịch sử hành động

Action_history			
	id	varchar(20)	N
	name	integer(10)	N
	action	integer(10)	N
	create_time	timestamp	N

## Chương 4: Kết quả

### I. Chức năng đã hoàn thành

#### 1. Liệt kê các tính năng đã triển khai thành công

- Thu thập và hiển thị dữ liệu cảm biến: ☒ Hệ thống đã thành công trong việc thu thập dữ liệu từ các cảm biến nhiệt độ, độ ẩm và ánh sáng thông qua ESP32.
  - ☒ Dữ liệu được truyền qua MQTT và lưu trữ trong cơ sở dữ liệu MySQL.
  - ☒ Frontend hiển thị dữ liệu cảm biến dưới dạng bảng và biểu đồ thời gian thực.
- Điều khiển thiết bị từ xa:

- Người dùng có thể điều khiển quạt, đèn LED và điều hòa thông qua giao diện web.
- Lệnh điều khiển được gửi từ Frontend đến Backend, sau đó truyền qua MQTT đến ESP32.
- ESP32 thực hiện lệnh điều khiển và gửi phản hồi về trạng thái thiết bị.

c) Lưu trữ và hiển thị lịch sử hành động:

- Hệ thống ghi lại tất cả các hành động điều khiển thiết bị.
- Người dùng có thể xem lịch sử hành động với các tùy chọn lọc và phân trang.

d) Giao diện người dùng thân thiện:

- Frontend được phát triển bằng React, cung cấp giao diện trực quan và dễ sử dụng.
- Người dùng có thể dễ dàng chuyển đổi giữa các chế độ xem khác nhau: Dashboard, Data Sensor, Action History và Profile.

e) API linh hoạt:

- Backend cung cấp các API endpoint cho phép truy xuất dữ liệu cảm biến và lịch sử hành động với nhiều tùy chọn lọc, sắp xếp và phân trang.

f) Cơ chế xóa dữ liệu tự động:

- Hệ thống tự động xóa các bản ghi cũ, chỉ giữ lại 100 bản ghi mới nhất để tối ưu hóa hiệu suất và không gian lưu trữ.

## II. Hiệu suất hệ thống

### 1. Đánh giá về tốc độ phản hồi

a) Thời gian phản hồi API:

- Các API endpoint có thời gian phản hồi trung bình dưới 200ms cho các truy vấn cơ bản.
- Đối với các truy vấn phức tạp hơn (ví dụ: tìm kiếm với nhiều điều kiện), thời gian phản hồi vẫn dưới 500ms.

b) Độ trễ điều khiển thiết bị:

- Thời gian từ khi người dùng gửi lệnh điều khiển đến khi thiết bị thực sự thay đổi trạng thái trung bình khoảng 1-2 giây.
- Phần lớn độ trễ này là do thời gian truyền tin qua MQTT và xử lý trên ESP32.

c) Cập nhật dữ liệu thời gian thực:

- Dữ liệu cảm biến được cập nhật trên giao diện người dùng mỗi 2 giây, đảm bảo thông tin luôn mới nhất.

## 2. Độ chính xác của dữ liệu cảm biến

### a) Nhiệt độ:

- Độ chính xác:  $\pm 0.5^{\circ}\text{C}$
- Độ phân giải:  $0.1^{\circ}\text{C}$

### b) Độ ẩm:

- Độ chính xác:  $\pm 2\% \text{ RH}$
- Độ phân giải:  $0.1\% \text{ RH}$

### c) Ánh sáng:

- Độ chính xác:  $\pm 5\%$  của giá trị đo
- Độ phân giải: 1 lux

## III. Cải tiến trong tương

lai

### 1. Tăng cường bảo mật

- Triển khai hệ thống xác thực và phân quyền người dùng.
- Sử dụng SSL/TLS cho kết nối MQTT và API.
- Thêm cơ chế mã hóa dữ liệu nhạy cảm trong cơ sở dữ liệu.

### 2. Cải thiện khả năng mở rộng

- Triển khai cơ chế caching để giảm tải cho database.
- Xem xét sử dụng cơ sở dữ liệu NoSQL như MongoDB cho dữ liệu cảm biến để tăng hiệu suất khi lưu trữ dữ liệu lớn.
- Triển khai load balancing cho Backend khi số lượng request tăng cao.

### 3. Nâng cao trải nghiệm người dùng

- Thêm tính năng thông báo và cảnh báo khi các chỉ số vượt ngưỡng.
- Phát triển ứng dụng di động để người dùng có thể theo dõi và điều khiển từ xa dễ dàng hơn.
- Tích hợp trí tuệ nhân tạo để đưa ra các đề xuất tối ưu hóa năng lượng dựa trên dữ liệu cảm biến.

### 4. Mở rộng khả năng tương thích

- Hỗ trợ thêm các loại cảm biến và thiết bị IoT khác.
- Phát triển SDK để cho phép các nhà phát triển bên thứ ba tích hợp với hệ thống.

### 5. Tối ưu hóa năng lượng

- Triển khai chế độ ngủ sâu cho ESP32 khi không cần thu thập dữ liệu.
- Điều chỉnh tần suất gửi dữ liệu dựa trên mức độ thay đổi của các chỉ số.

### 6. Cải thiện khả năng phân tích

- Thêm các công cụ phân tích dữ liệu nâng cao để cung cấp insights về xu hướng và mẫu tiêu thụ năng lượng.
- Tích hợp machine learning để dự đoán và tối ưu hóa việc sử dụng năng lượng.



## 7. Tăng cường độ tin cậy

- Triển khai cơ chế backup và khôi phục dữ liệu tự động.
- Cải thiện logging và monitoring để phát hiện và xử lý sự cố nhanh chóng.

Những cải tiến và mở rộng này sẽ giúp hệ thống trở nên mạnh mẽ, an toàn và linh hoạt hơn, đáp ứng được nhu cầu ngày càng tăng của người dùng và thích ứng với các xu hướng công nghệ mới trong lĩnh vực IoT và quản lý năng lượng thông minh.

## ***Lời cảm ơn***

Em xin gửi lời cảm ơn sâu sắc đến thầy Nguyễn Quốc Uy, giảng viên môn học Internet of Things và Ứng dụng tại Học viện Công nghệ Bưu chính Viễn thông.

Trong suốt khóa học, thầy đã truyền đạt kiến thức một cách tận tâm và chuyên nghiệp, giúp em và các bạn sinh viên hiểu sâu sắc về lĩnh vực IoT đang phát triển nhanh chóng. Những bài giảng sinh động và các ví dụ thực tế của thầy đã truyền cảm hứng cho chúng em, khơi dậy niềm đam mê với công nghệ và mở ra nhiều cơ hội mới trong tương lai nghề nghiệp.

Thầy không chỉ là một người thầy giàu kinh nghiệm mà còn là một người cố vấn tận tụy. Sự hướng dẫn và ủng hộ của thầy trong quá trình thực hiện đồ án này đã giúp em vượt qua nhiều khó khăn, từ đó hoàn thiện dự án một cách tốt nhất.

Kiến thức và kỹ năng em học được từ môn học này chắc chắn sẽ là nền tảng vững chắc cho sự phát triển nghề nghiệp của em trong tương lai. Em xin chân thành cảm ơn thầy vì tất cả những đóng góp quý báu này.

Kính chúc thầy luôn mạnh khỏe, hạnh phúc và thành công trong sự nghiệp giảng dạy cao quý của mình. Trân trọng,