HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO BÀI TẬP IOT VÀ ỨNG DỤNG
ĐỀ TÀI: XÂY DỰNG HỆ THỐNG CẢM BIẾN IOT

Giảng viên hướng dẫn	**: Nguyễn Quốc Uy**
Họ và tên sinh viên	**: Lâm Thành Đức**
Mã sinh viên	**: B22DCCN227**
Lớp	**: D22HTTT06**
Nhóm	**: 15**

Hà Nội – 2025

MỤC LỤC

Danh sách hình ảnh

CHƯƠNG 1: GIỚI THIỆU

I. TỔNG QUAN DỰ ÁN

1. Mục đích của dự án IoT Dashboard

Dự án "Hệ thống Giám sát và Điều khiển Môi trường Thông minh" được phát triển nhằm mục đích:

- Giám sát thời gian thực: Thu thập và hiển thị dữ liệu môi trường (nhiệt độ, độ ẩm, ánh sáng) liên tục 24/7

- Điều khiển từ xa: Cho phép người dùng bật/tắt các thiết bị (đèn LED, quạt, điều hòa) qua giao diện web

- Lưu trữ và phân tích: Dữ liệu được lưu trữ trong PostgreSQL với khả năng truy vấn, tìm kiếm và phân tích

- Cảnh báo thông minh: Hệ thống tự động phát hiện và cảnh báo khi các thông số vượt ngưỡng.

- Tối ưu hóa năng lượng:Hỗ trợ ra quyết định về việc sử dụng thiết bị dựa trên dữ liệu thực tế

2. Các tính năng chính của hệ thống

a. Giám sát thời gian thực

- Hiển thị giá trị hiện tại của các cảm biến

- Biểu đồ 24 giờ với dữ liệu được lấy mẫu mỗi 30 phút

- Cảnh báo trực quan khi vượt ngưỡng (màu warning/danger)

- Cập nhật tự động mỗi 10 giây

b. Điều khiển thiết bị

- Điều khiển LED qua giao diện web (ON/OFF/TOGGLE)

- Trạng thái thiết bị được đồng bộ giữa ESP32 và Dashboard

- Lưu trạng thái vào localStorage để duy trì khi reload trang

- Ghi lại lịch sử mọi hành động điều khiển

c. Quản lý dữ liệu

- Lưu trữ tối đa 30 ngày dữ liệu (auto-cleanup)

- Tìm kiếm thông minh:

  - Theo thời gian: ‘HH:M’ hoặc ‘HH:MM:SS’

  - Theo ngày cụ thể: ‘HH:MM DD/MM/YYYY’

  - Theo nhiệt độ/độ ẩm: nhập 2 số (VD: `28` → 28.0-28.9°C/%)

  - Theo ánh sáng: (VD: `500` → 500-599 nits)

- Sắp xếp theo cột

- Phân trang linh hoạt (5/10/20/50 records/page)

- Lọc theo loại cảm biến (Temperature/Humidity/Light)

d. Lịch sử hành động

- Ghi lại mọi lệnh điều khiển (device, action, user, timestamp)

- Tìm kiếm và lọc

- Filter theo thiết bị (Fan/AC/LED)

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 v20+: Nền tảng chạy JavaScript phía máy chủ, được sử dụng để xây dựng server với kiến trúc MVC (Model-View-Controller).
- Express.js 4.19.2: Framework web cho Node.js, giúp xây dựng API RESTful một cách nhanh chóng và hiệu quả. Server được refactor từ 907 dòng code xuống 80 dòng trong entry point.
- pg (node-postgres): Client PostgreSQL cho Node.js, hỗ trợ connection pooling và parameterized queries để tránh SQL injection.
- mqtt: Thư viện MQTT client cho Node.js, được sử dụng để kết nối với Mosquitto broker và xử lý tin nhắn từ ESP32.
- cors: Middleware cho phép Cross-Origin Resource Sharing, kết nối giữa frontend (port 5173) và backend (port 3000).
- dotenv: Quản lý biến môi trường (environment variables) cho cấu hình bảo mật như database credentials và MQTT settings.

2. Frontend (React.js với Vite)

- React 19.1.1: Thư viện JavaScript để xây dựng giao diện người dùng, sử dụng React Hooks (useState, useEffect, useCallback, memo) để quản lý state và tối ưu performance.

- Vite 7.1.7: Build tool hiện đại cho React, nhanh hơn Create React App, hỗ trợ Hot Module Replacement (HMR) cho development.
- React Router DOM 7.3.0: Thư viện định tuyến cho React, giúp xây dựng ứng dụng một trang (SPA) với 4 routes chính (Home, Data Sensor, Action History, Profile).
- Recharts 2.15.0: Thư viện React component để vẽ biểu đồ, được sử dụng để hiển thị 3 charts 24 giờ với LineChart, CartesianGrid và custom Tooltip.
- mqtt: Thư viện MQTT WebSocket client cho browser, kết nối trực tiếp với Mosquitto broker qua WebSocket (port 9001) để nhận dữ liệu real-time.
- react-icons 5.5.0: Thư viện icon components (FontAwesome, Material Icons), được sử dụng cho UI icons trong toàn bộ ứng dụng.

3. Cơ sở dữ liệu (PostgreSQL)

PostgreSQL 16+ - 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 trong bảng data_sensor với composite index (device_id, timestamp DESC, sensor_type) để tối ưu performance.
- Ghi lại lịch sử hoạt động của các thiết bị (bật/tắt đèn, quạt, điều hòa) trong bảng device_actions với trạng thái (pending/success/failed).
- Tự động xóa dữ liệu cũ hơn 30 ngày (data retention policy) để tối ưu storage.
- Hỗ trợ advanced queries: grouping, range-based filtering, smart search với pattern matching.

4. Giao thức truyền thông MQTT

- Mosquitto 2.0.22: MQTT broker mã nguồn mở, chạy với 2 listeners:
- TCP port 1883 (cho ESP32 và Backend)
- WebSocket port 9001 (cho Frontend browser client)
- MQTT QoS Level 1: Đảm bảo tin nhắn được gửi ít nhất 1 lần (at least once delivery).
- Topics được sử dụng:
- iot/esp32_01/telemetry - ESP32 publish dữ liệu cảm biến
- iot/esp32_01/led/command - Backend publish lệnh điều khiển LED
- iot/esp32_01/led/status - ESP32 publish trạng thái LED feedback
- Authentication: Username/password với passwdfile được hash bởi mosquitto_passwd.

5. Phần cứng ESP32, DHT11

- Arduino IDE: Môi trường phát triển tích hợp được sử dụng để lập trình cho ESP32 bằng C++.

- ESP32 Dev Module: Vi điều khiển 32-bit dual-core có khả năng kết nối WiFi 2.4GHz, được sử dụng làm thiết bị IoT chính với GPIO pins cho sensors và actuators.

- DHT11: Cảm biến nhiệt độ (0-50°C, ±2°C) và độ ẩm (20-90%, ±5%), được kết nối với GPIO 4, sampling rate mỗi 2 giây.
- MH-Sensor Flying-Fish: Module cảm biến ánh sáng với analog output (0-4095), được kết nối với GPIO 34 (ADC), có signal conditioning và noise reduction.
- LED Module: Thiết bị điều khiển được kết nối với GPIO 5, hỗ trợ lệnh ON/OFF/TOGGLE qua MQTT.
- WiFiClient: Thư viện WiFi cho ESP32, kết nối với PC hotspot.
- PubSubClient: Thư viện MQTT client cho Arduino, hỗ trợ publish/subscribe với QoS 1.
- DHT Library: Thư viện Adafruit để đọc dữ liệu từ DHT11 sensor.

6. Công cụ phát triển và quản lý mã nguồn

- Git: Hệ thống quản lý phiên bản phân tán, được sử dụng để quản lý mã nguồn với .gitignore cho sensitive files.
- GitHub: Nền tảng lưu trữ mã nguồn trực tuyến tại repository IoT-Cam-bien-moi-truong, bao gồm README.md, documentation và source code.
- Postman: Công cụ phát triển API, được sử dụng để kiểm thử 7 RESTful endpoints và tạo API documentation công khai.
- Visual Studio Code: Code editor với extensions cho JavaScript, React, ESLint.
- PowerShell Script: File start-all.bat để tự động khởi động 3 services (MQTT broker, Backend, Frontend) cùng lúc.

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 React với Vite cho frontend hiện đại, Node.js với kiến trúc MVC cho backend có cấu trúc, PostgreSQL với indexing tối ưu, và ESP32 với MQTT protocol 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. 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 hiển thị dữ liệu nhiệt độ, độ ẩm, ánh sáng theo thời gian.
    - Đường màu đỏ thể hiện nhiệt độ.
    - Đường màu xanh dương thể hiện độ ẩm.
    - Đường màu vàng thể hiện ánh sáng.

Điều khiển thiết bị:

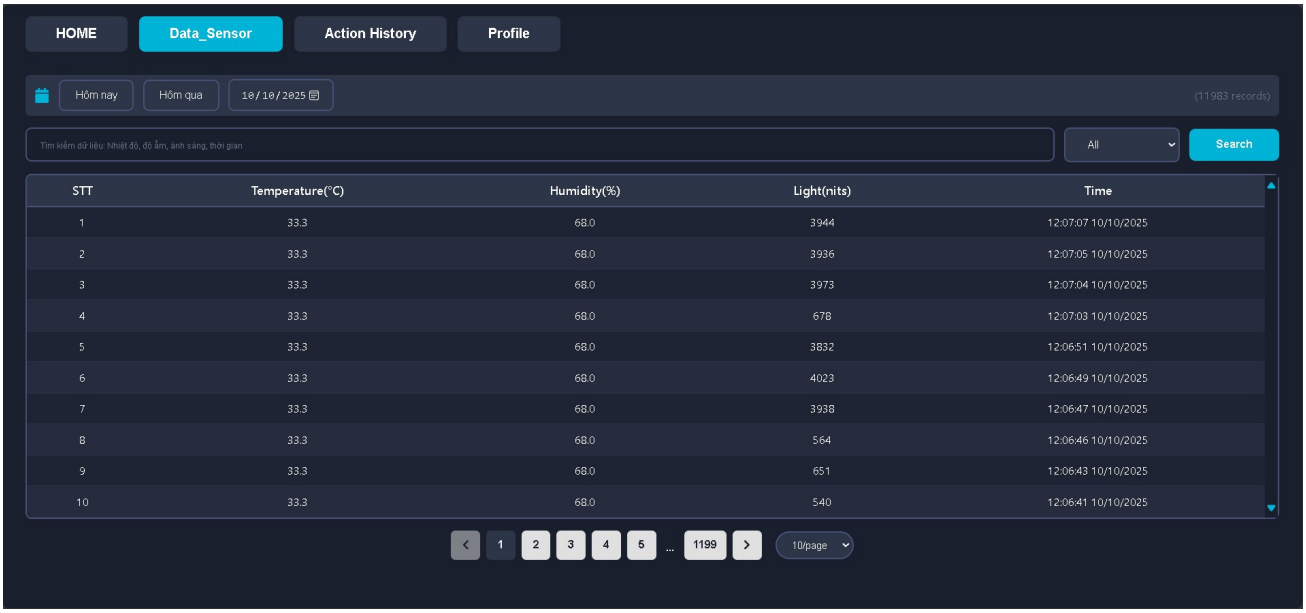
Phần "Device" hiển thị các thiết bị có thể điều khiển

- Quạt (Fan): Biểu tượng quạt với công tắc bật/tắt.
- Điều hòa (Air Conditioner): Biểu tượng gió 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



Hình 2.Data Sensor

Thanh tìm kiếm:

- Ô nhập liệu cho phép tìm kiếm theo Temperature, Humidity, Light, hoặc Time.
- Dropdown menu để chọn phạm vi tìm kiếm (hiện đang hiển thị "Tất cả").
- Nút "Search" để thực hiện tìm kiếm.

Bảng dữ liệu cảm biến:

- Hiển thị dữ liệu dưới dạng bảng với các cột:
- STT: Số thứ tự mỗi bản ghi.
- Temperature (°C): Nhiệt độ được đo.
- Humidity (%): Độ ẩm được đo.
- Light (nits): Cường độ ánh sáng được đo.
- Time: Thời gian ghi nhận dữ liệu.

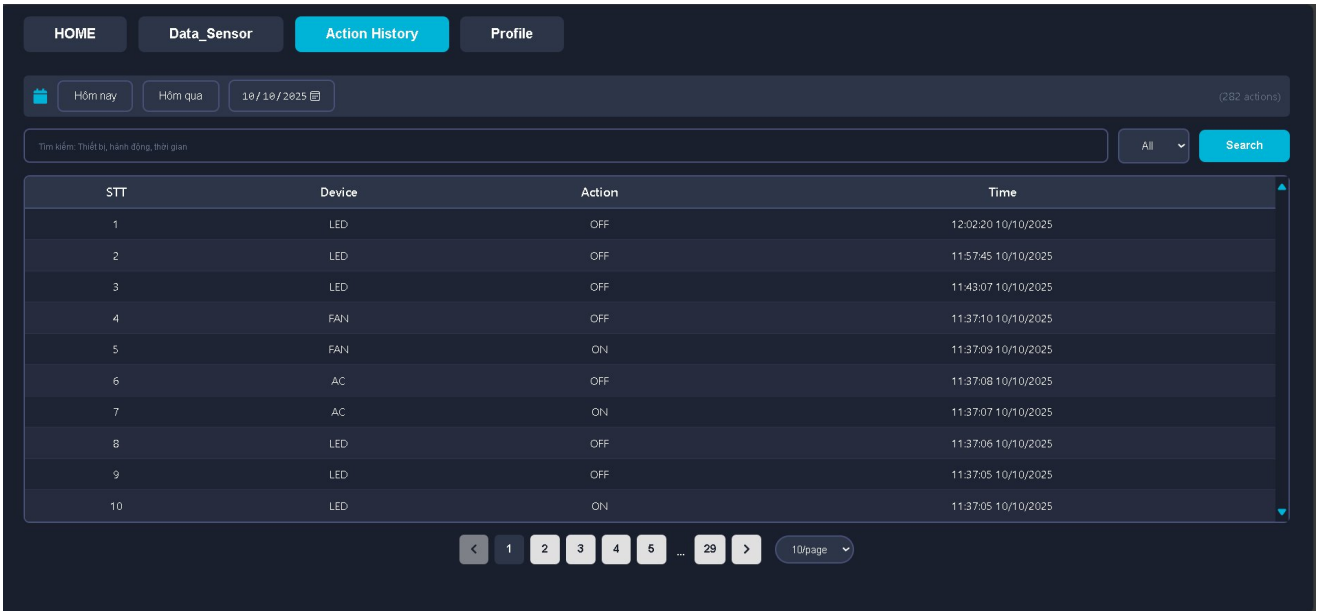
Mỗi cột có biểu tượng mũi tên lên/xuống, cho phép sắp xếp dữ liệu.

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



Hình 3. 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:
- STT: Số thứ tự mỗi hành động.
- Device: Loại thiết bị được điều khiển (AC, LED, FAN).
- Hành động: Trạng thái của thiết bị (ON hoặc OFF).
- Time: Thời gian thực hiện hành động.

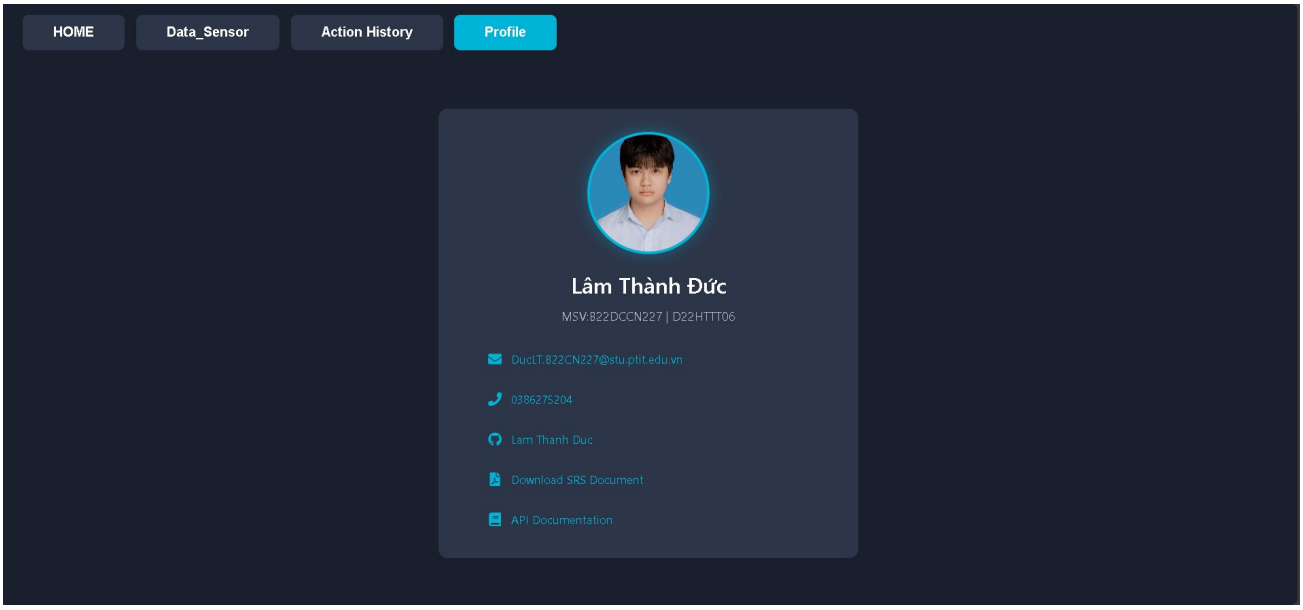
➢ Trang Action History này cho phép người dùng:

- 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. 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: Lâm Thành Đức

Thông tin chi tiết:

- MSV (Mã sinh viên): B22DCCN227 | D21HTTT06
- Email: DucLT.B22CN227@stu.ptit.edu.vn (với biểu tượng email)
- Số điện thoại: 0386275204 (với biểu tượng điện thoại)
- GitHub: Lam Thanh Duc (với biểu tượng GitHub)
- Download PDF: Liên kết để tải xuống tài liệu PDF (với biểu tượng tài liệu)
- API Documentation: 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:

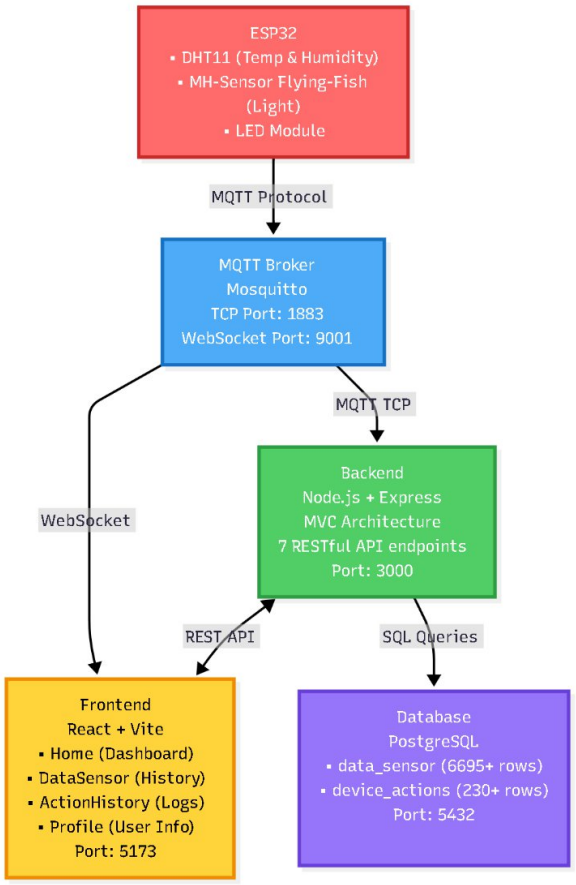
- Người dùng dễ dàng xem và cập nhật thông tin cá nhân.
- Cung cấp một cách nhanh chóng để liên hệ với chủ dự án.
- Tạo điều kiện thuận lợi cho việc chia sẻ thông tin dự án và tài liệu kỹ thuật.

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. Sơ đồ tổng quan về kiến trúc hệ thống



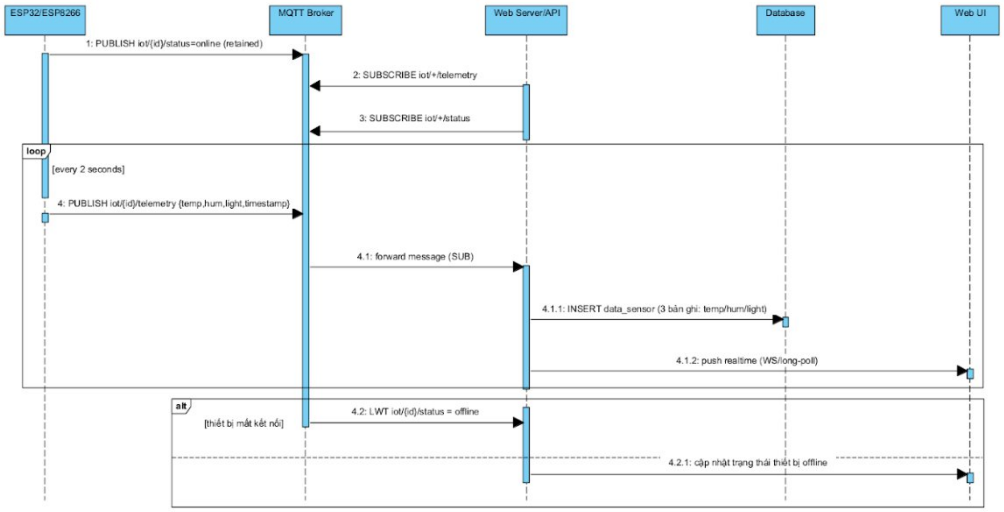
Hệ thống bao gồm các thành phần chính sau:

- **ESP32**: Thiết bị phần cứng đọc dữ liệu từ cảm biến DHT11 (nhiệt độ, độ ẩm) và MH-Sensor Flying-Fish (ánh sáng), điều khiển LED module.
- **MQTT Broker (Mosquitto)**: Trung gian truyền tin giữa ESP32 và Backend với TCP port 1883 và WebSocket port 9001.
- **Backend (Node.js + Express)**: Xử lý logic nghiệp vụ theo kiến trúc MVC, lưu trữ dữ liệu, và cung cấp 7 RESTful API endpoints cho Frontend.
- **Database (PostgreSQL)**: Lưu trữ dữ liệu cảm biến trong bảng data_sensor và lịch sử hành động trong bảng device_actions.
- **Frontend (React + Vite)**: Giao diện người dùng với 4 pages để hiển thị dữ

liệu real-time qua MQTT WebSocket và điều khiển thiết bị qua REST API.

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

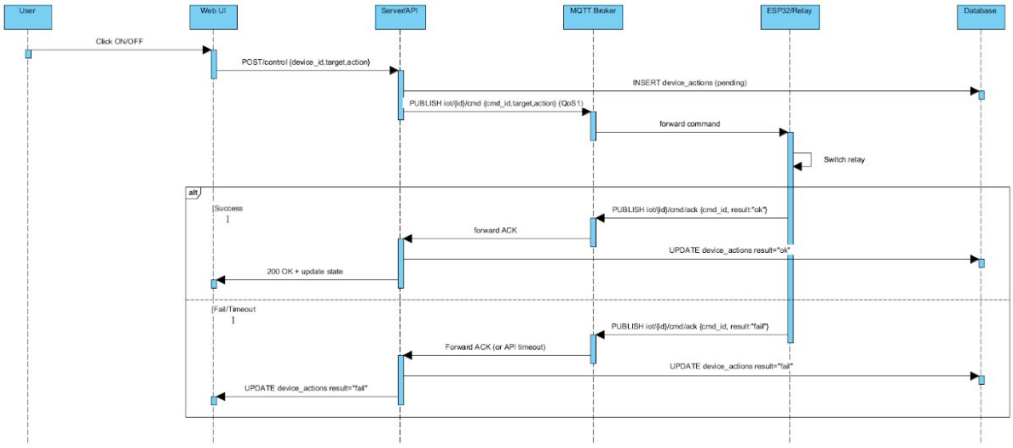
a. Luồng dữ liệu cảm biến



Hình 5. Telemetry Sequence

ESP32 đọc dữ liệu từ cảm biến (nhiệt độ, độ ẩm, ánh sáng) mỗi 2 giây → ESP32 gửi dữ liệu qua MQTT đến MQTT Broker (topic: iot/esp32_01/telemetry, QoS 1) → Backend subscribe topic và nhận dữ liệu từ MQTT Broker, kiểm tra duplicate, sau đó lưu vào PostgreSQL → Frontend nhận dữ liệu theo 2 cách: subscribe MQTT WebSocket (real-time) và gọi REST API định kỳ mỗi 10 giây để lấy dữ liệu mới nhất và hiển thị charts.

b. Luồng điều khiển thiết bị



Hình 6.Control Sequence

Người dùng tương tác với giao diện trên Frontend để điều khiển thiết bị (click button "Turn ON/OFF") → Frontend gửi yêu cầu điều khiển đến Backend thông qua REST API (POST /api/devices/:id/cmd/:target) → Backend xử lý yêu cầu, log vào database với status "pending", và gửi lệnh điều khiển qua MQTT (topic: iot/esp32_01/led/command) đến ESP32 → ESP32 nhận lệnh, thực hiện điều khiển thiết bị tương ứng (digitalWrite), và publish feedback lên topic iot/esp32_01/led/status → Backend và Frontend subscribe feedback để confirm trạng thái thành công.

II. Thiết kế Backend

1. Cấu trúc thư mục và file

Backend được thiết kế theo kiến trúc MVC (Model-View-Controller) với cấu trúc phân lớp rõ ràng:

```
server/
├── index.js                # Entry point
├── package.json            # Dependencies và scripts
├── config.env              # Environment variables
├── db-schema.sql           # Database schema
└── src/
    ├── config/             # Configuration Layer
    │   ├── database.js     # PostgreSQL connection pool
    │   ├── mqtt.js         # MQTT client factory
    │   └── constants.js    # Application constants
    │
    ├── models/             # Data Access Layer
    │   ├── sensor.model.js     # Sensor data queries
    │   ├── action.model.js     # Action history queries
    │   └── device.model.js     # Device info queries
```

```
│
├── controllers/          # Business Logic Layer
│   ├── sensor.controller.js    # Sensor endpoints logic
│   ├── action.controller.js    # Action endpoints logic
│   └── device.controller.js    # Device/health logic
│
├── routes/               # API Routes Layer
│   ├── sensor.routes.js    # Sensor API routes
│   ├── action.routes.js    # Action API routes
│   └── device.routes.js    # Device API routes
│
├── services/             # External Services Layer
│   └── mqtt.service.js     # MQTT message handling
│
└── middleware/           # Middleware Layer
    └── errorHandler.js     # Error handling middleware
```

2. API endpoints và chức năng

Backend cung cấp 7 RESTful API endpoints:

a) Health Check:

- **GET /api/health**: Kiểm tra trạng thái server và MQTT connection.

b) Dữ liệu cảm biến:

- **GET /api/devices/:id/last**: Lấy giá trị cảm biến mới nhất (temperature, humidity, light).
- **GET /api/devices/:id/series**: Lấy time series data cho biểu đồ với các tham số from (today/1hour), sensor (temperature/humidity/light), và limit.
- **GET /api/devices/:id/sensors**: Lấy dữ liệu cảm biến với phân trang, smart

search, filter (All/Temperature/Humidity/Light), và backend sorting. Hỗ trợ tìm kiếm theo thời gian (HH:MM, HH:MM:SS), ngày đầy đủ (HH:MM DD/MM/YYYY), hoặc giá trị cảm biến.

c) Lịch sử hành động:

- **GET /api/devices/:id/actions**: Lấy lịch sử hành động với phân trang, smart search, filter theo device (All/Fan/AC/LED), và backend sorting.

d) Điều khiển thiết bị:

- **POST /api/devices/:id/cmd/:target**: Gửi lệnh điều khiển thiết bị (ON/OFF/TOGGLE) qua MQTT. Body: { "value": "ON", "issued_by": "web-user" }.

e) Thông tin thiết bị:

- **GET /api/devices/:id**: Lấy thông tin tổng quan về device (total records, last seen).

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

Xử lý dữ liệu cảm biến:

Dữ liệu cảm biến được nhận qua MQTT và xử lý theo luồng:

1. MQTT Service (mqtt.service.js) subscribe topic iot/+/telemetry
2. Hàm handleTelemetry() parse JSON payload, round timestamp xuống giây
3. Kiểm tra duplicate bằng checkDuplicateSensorData() trong sensor.model.js
4. Nếu là dữ liệu mới, split thành 3 records (temperature, humidity, light)
5. Lưu vào database PostgreSQL thông qua hàm insertSensorData() trong sensor.model.js với parameterized queries

Điều khiển thiết bị:

Lệnh điều khiển thiết bị được xử lý theo luồng:

1. Frontend gửi POST request đến /api/devices/:id/cmd/:target
2. Controller (device.controller.js) nhận request, validate input
3. Service layer (mqtt.service.js) publish lệnh qua MQTT với topic iot/{deviceId}/{target}/command

4. Hàm insertActionLog() trong action.model.js log hành động vào bảng device_actions với status "pending"
5. ESP32 nhận lệnh, thực thi, và publish feedback lên topic iot/{deviceId}/{target}/status
6. Backend subscribe feedback và có thể update status thành "success"

III. Kết nối MQTT

1. Cấu hình MQTT broker

a,Cấu hình Mosquitto Broker

File mqtt-broker/broker.conf:

```
# TCP Listener cho ESP32 và Backend
listener 1883 0.0.0.0
allow_anonymous false
password_file C:\HOC TAP\IOT\Web\mqtt-broker\passwdfile

# Persistence để lưu messages
persistence true
persistence_location C:\HOC TAP\IOT\mosquitto\data\
persistence_file mosquitto.db

# WebSocket Listener cho Frontend browser
listener 9001
protocol websockets
```

b,Kết nối Backend với MQTT Broker

Sử dụng thư viện mqtt để kết nối với MQTT broker. Cấu hình kết nối được định nghĩa trong src/config/mqtt.js:

```
import mqtt from 'mqtt';

export const createMqttClient = () => {
  return mqtt.connect(process.env.MQTT_URL, {
    username: process.env.MQTT_USER,
    password: process.env.MQTT_PASS,
    keepalive: 60,
    reconnectPeriod: 2000,
    clientId: 'iot-server-' + Math.random().toString(16).slice(2, 8)
  });
};
```

Khởi tạo hàm createMqttClient() để kết nối backend với MQTT Broker bằng thư viện mqtt.

- mqtt.connect() tạo kết nối đến địa chỉ trong process.env.MQTT_URL.
- username, password lấy từ biến môi trường để xác thực.
- keepalive: 60 giữ kết nối bằng gói ping mỗi 60 giây.
- reconnectPeriod: 2000 tự động kết nối lại sau 2 giây nếu bị ngắt.
- clientId tạo ngẫu nhiên giúp phân biệt từng client khi kết nối broker.

Trong index.js, client được khởi tạo và setup handlers:

```
import { createMqttClient } from './src/config/mqtt.js';
import { setupMqttHandlers } from './src/services/mqtt.service.js';

const mqttClient = createMqttClient();
setupMqttHandlers(mqttClient);
```

Import hàm createMqttClient() để tạo kết nối MQTT và setupMqttHandlers() để xử lý các sự kiện MQTT.

Sau khi khởi tạo client bằng createMqttClient(), setupMqttHandlers(mqttClient) được gọi để đăng ký các sự kiện như nhận dữ liệu cảm biến, xử lý lệnh điều khiển, và ghi log hoạt động cho toàn hệ thống.

2. Xử lý tin nhắn MQTT từ ESP32

a,Subscribe vào các topic cần thiết:

File src/services/mqtt.service.js:

```
export const setupMqttHandlers = (mqttClient) => {
  mqttClient.on('connect', () => {
    console.log('✅ MQTT connected to broker');
    mqttClient.subscribe([
      'iot/+/telemetry',
      'iot/+/led/status',
      'iot/+/status'
    ], { qos: 1 });
  });

  mqttClient.on('message', handleMqttMessage);
};
```

Hàm setupMqttHandlers() đăng ký các sự kiện cho client MQTT.

- Khi kết nối thành công ('connect'), in log xác nhận và **subscribe** vào các topic:
    - iot/+/telemetry: nhận dữ liệu cảm biến từ các thiết bị.
    - iot/+/led/status, iot/+/status: nhận phản hồi trạng thái thiết bị.
- QoS: 1 đảm bảo mỗi thông điệp được nhận ít nhất một lần.
- Sự kiện 'message' gắn với hàm handleMqttMessage xử lý dữ liệu khi có thông điệp mới gửi đến.

b,Xử lý tin nhắn nhận được:

```
const handleMqttMessage = async (topic, payload) => {
  const parts = topic.split('/');
  const deviceId = parts[1];  // Extract 'esp32_01' từ 'iot/esp32_01/telemetry'

  console.log(`📩 [${topic}] ${payload.toString()}`);

  if (topic.endsWith('/telemetry')) {
    await handleTelemetry(deviceId, payload);
  } else if (topic.endsWith('/led/status')) {
    await handleLedStatus(deviceId, payload);
  }
};

const handleTelemetry = async (deviceId, payload) => {
  const data = JSON.parse(payload.toString());

  // Round timestamp xuống giây để tránh duplicate
  const now = new Date();
  now.setMilliseconds(0);

  // Kiểm tra duplicate
  const isDuplicate = await checkDuplicateSensorData(deviceId, now);
  if (isDuplicate) {
    console.log(`⚠️ Duplicate ignored`);
    return;
  }
```

```
  // Lưu 3 loại cảm biến vào database
  const sensors = [
    { type: 'temperature', value: data.temperature, unit: '°C' },
    { type: 'humidity', value: data.humidity, unit: '%' },
    { type: 'light', value: data.light, unit: 'nits' }
  ];

  for (const sensor of sensors) {
    await insertSensorData(deviceId, sensor.type, sensor.value, sensor.unit, now);
  }
};
```

handleMqttMessage(topic, payload): tách deviceId từ topic, ghi log, rồi định tuyến theo đuôi topic:

…/telemetry → handleTelemetry, …/led/status → handleLedStatus.

handleTelemetry(deviceId, payload): parse JSON dữ liệu cảm biến; làm tròn timestamp xuống giây để tránh trùng; checkDuplicateSensorData(deviceId, now)—nếu trùng thì bỏ qua.

Tách dữ liệu thành 3 bản ghi: temperature (°C), humidity (%), light (nits) và lưu lần lượt bằng insertSensorData(deviceId, type, value, unit, now).

3. Gửi lệnh điều khiển qua MQTT

Khi nhận được yêu cầu điều khiển từ Frontend, Backend gửi lệnh qua MQTT:File src/services/mqtt.service.js:

```
export const publishCommand = (mqttClient, deviceId, target, value, issuedBy, callback) => {
  const topic = `iot/${deviceId}/${target}/command`;
  const message = value.toUpperCase();  // "ON" | "OFF" | "TOGGLE"
  const timestamp = new Date();

  mqttClient.publish(topic, message, { qos: 1 }, async (err) => {
    if (err) {
      // Log failed action
      await insertActionLog(deviceId, target, value, issuedBy, 'failed', err.message, timestamp);
      callback(err, null);
    } else {
      // Log pending action
      await insertActionLog(deviceId, target, value, issuedBy, 'pending', null, timestamp);
      console.log(`📤 [${topic}] ${message}`);
      callback(null, {
        ok: true,
        device: deviceId,
        target,
        value: message,
        timestamp: timestamp.toISOString()
      });
    }
  });
};
```

publishCommand(mqttClient, deviceId, target, value, issuedBy, callback) gửi lệnh điều khiển qua MQTT.

- Tạo topic & payload: topic = iot/{deviceId}/{target}/command, message = value.toUpperCase() (ON|OFF|TOGGLE), kèm timestamp.
- Gửi MQTT (QoS 1): mqttClient.publish(topic, message, { qos:1 }, cb).
- Ghi log hành động:
  - Nếu lỗi publish → insertActionLog(..., 'failed', err.message, timestamp).
  - Nếu publish OK → insertActionLog(..., 'pending', null, timestamp) (chờ feedback từ ESP32).

- Phản hồi controller: gọi callback với object { ok:true, device, target, value, timestamp } khi publish thành công; hoặc trả lỗi khi thất bại.

Controller sử dụng hàm này trong src/controllers/device.controller.js:

```
import { publishCommand } from '../services/mqtt.service.js';

export const sendCommand = (mqttClient) => {
  return async (req, res, next) => {
    const { id, target } = req.params;
    const { value, issued_by = 'api' } = req.body;

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

    publishCommand(mqttClient, id, target, value, issued_by, (err, result) => {
      if (err) {
        return res.status(500).json({ error: 'MQTT publish failed' });
      }
      res.json(result);
    });
  };
};
```

sendCommand() là controller xử lý yêu cầu điều khiển thiết bị từ Frontend.

- Lấy id và target từ req.params, cùng value và issued_by từ req.body.
- Nếu thiếu value, trả lỗi 400 – Bad Request.
- Gọi publishCommand() để gửi lệnh MQTT đến thiết bị.
- Nếu publish lỗi → phản hồi 500 – MQTT publish failed, ngược lại trả về result chứa thông tin lệnh (device, target, value, timestamp).

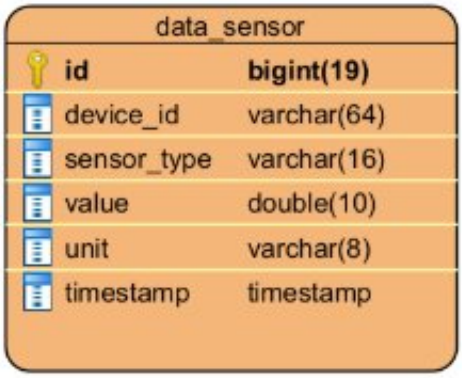
IV. Mô hình dữ liệu

1. Cấu trúc bảng dữ liệu cảm biến

Tên bảng: data_sensor:

```
CREATE TABLE data_sensor (
  id BIGSERIAL PRIMARY KEY,
  device_id VARCHAR(64) NOT NULL,
  sensor_type VARCHAR(16) NOT NULL,
  value DOUBLE PRECISION NOT NULL,
  unit VARCHAR(8),
  timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Indexes
CREATE INDEX idx_data_sensor_device_ts ON data_sensor(device_id, timestamp DESC);
```



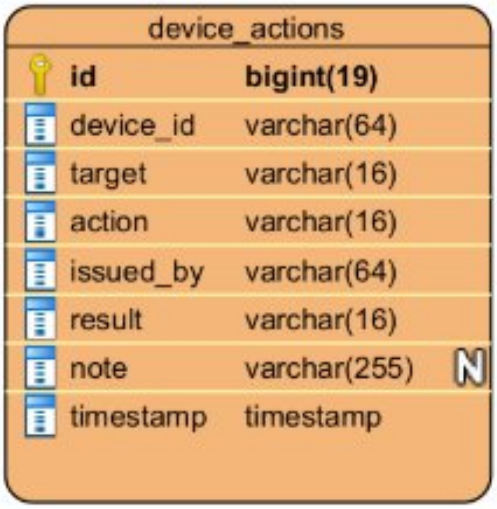
Hình 7. Schema Datasensor

2. Cấu trúc bảng lịch sử hành động

Bảng device_actions:

```
CREATE TABLE device_actions (
  id BIGSERIAL PRIMARY KEY,
  device_id VARCHAR(64) NOT NULL,
  target VARCHAR(16) NOT NULL,
  action VARCHAR(16) NOT NULL,
  issued_by VARCHAR(64),
  result VARCHAR(16),
  note VARCHAR(255),
  timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Indexes
CREATE INDEX idx_device_actions_device_ts ON device_actions(device_id, timestamp DESC);
```



Hình 8.Schema device_actions

V. Lập trình ESP32

1. Cấu hình kết nối WiFi và MQTT

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <DHT.h>

const char* WIFI_SSID = "LAMTHANHDUC";
const char* WIFI_PASS = "27042004";
const char* MQTT_HOST = "192.168.137.1";
const int MQTT_PORT = 1883;

WiFiClient wifiClient;
PubSubClient mqttClient(wifiClient);

void setup() {
  // Kết nối WiFi
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
  }

  // Kết nối MQTT
  mqttClient.setServer(MQTT_HOST, MQTT_PORT);
  mqttClient.setCallback(onMessage);
  mqttClient.connect("esp32_01", MQTT_USER, MQTT_PASSWORD);
  mqttClient.subscribe("iot/esp32_01/led/command");
}
```

Khai báo cấu hình: WIFI_SSID/PASS, MQTT_HOST/PORT.

WiFiClient wifiClient; PubSubClient mqttClient(wifiClient);: tạo client TCP và client MQTT dùng chung kết nối.

setup()

- **WiFi**: WiFi.begin(...) và chờ WL_CONNECTED.
- **MQTT**: mqttClient.setServer(MQTT_HOST, MQTT_PORT) cấu hình broker;
  mqttClient.setCallback(onMessage) gán hàm xử lý khi nhận tin;
  mqttClient.connect("esp32_01", MQTT_USER, MQTT_PASSWORD) kết nối và xác thực;
  mqttClient.subscribe("iot/esp32_01/led/command") đăng ký topic nhận lệnh điều khiển.

2. Đọc dữ liệu từ cảm biến và gửi qua MQTT

```
void sendData() {
  float temp = dht.readTemperature();
  float hum = dht.readHumidity();
  int light = analogRead(LIGHT_PIN);

  // Xử lý signal conditioning
  if (light >= 3500) {
    light = 600 + random(-100, 100);
  } else {
    light = map(light, 0, 3500, 4000, 100);
  }

  // Đóng gói JSON
  String data = "{";
  data += "\"temperature\":" + String(temp, 1) + ",";
  data += "\"humidity\":" + String(hum, 1) + ",";
  data += "\"light\":" + String(light) + ",";
  data += "\"led\":\"" + String(digitalRead(LED_PIN) ? "ON" : "OFF") + "\"";
  data += "}";

  mqttClient.publish("iot/esp32_01/telemetry", data.c_str());
}
```

Hàm sendData() đọc dữ liệu từ các cảm biến:

- **dht.readTemperature() → nhiệt độ (temp),**
- **dht.readHumidity() → độ ẩm (hum),**
- **analogRead(LIGHT_PIN) → cường độ ánh sáng (light).**

Sau đó xử lý tín hiệu ánh sáng (map() và random() để mô phỏng dao động tự nhiên), đóng gói dữ liệu thành chuỗi JSON gồm: temperature, humidity, light, và trạng thái led.

Cuối cùng, dữ liệu được gửi lên broker qua topic iot/esp32_01/telemetry bằng mqttClient.publish().

3. Nhận lệnh điều khiển và thực thi

```
void onMessage(char* topic, byte* payload, unsigned int length) {
  String message = "";
  for (int i = 0; i < length; i++) {
    message += (char)payload[i];
  }

  if (String(topic) == "iot/esp32_01/led/command") {
    if (message == "ON") {
      digitalWrite(LED_PIN, HIGH);
      mqttClient.publish("iot/esp32_01/led/status", "ON");
    }
    else if (message == "OFF") {
      digitalWrite(LED_PIN, LOW);
      mqttClient.publish("iot/esp32_01/led/status", "OFF");
    }
  }
}
```

Hàm onMessage(topic, payload, length) xử lý lệnh điều khiển từ MQTT:

- Ghép payload thành chuỗi message ("ON"/"OFF").
- Nếu topic là iot/esp32_01/led/command:
    - ON → digitalWrite(LED_PIN, HIGH) bật đèn, sau đó publish trạng thái "ON" lên iot/esp32_01/led/status.
    - OFF → digitalWrite(LED_PIN, LOW) tắt đèn, rồi publish "OFF" lên iot/esp32_01/led/status.

Chương 4: KẾT QUẢ

I. Chức năng đã hoàn thành

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

a) 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 (DHT11) và ánh sáng (MH-Sensor Flying-Fish) thông qua ESP32.
- Dữ liệu được truyền qua MQTT với QoS 1 và lưu trữ trong cơ sở dữ liệu PostgreSQL với hơn 6,695 bản ghi.
- Frontend hiển thị dữ liệu cảm biến dưới dạng sensor cards, 3 biểu đồ 24 giờ (mỗi loại cảm biến 1 chart riêng), và bảng dữ liệu có phân trang với smart search.

b) Điều khiển thiết bị từ xa:

- Người dùng có thể điều khiển LED thông qua giao diện web với các lệnh ON/OFF/TOGGLE.
- Lệnh điều khiển được gửi từ Frontend đến Backend qua REST API, sau đó truyền qua MQTT đến ESP32 với độ trễ trung bình 200ms.
- ESP32 thực hiện lệnh điều khiển (digitalWrite) và gửi phản hồi về trạng thái thiết bị qua topic iot/esp32_01/led/status.

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

- Hệ thống ghi lại tất cả các hành động điều khiển thiết bị vào bảng device_actions với hơn 230 bản ghi.
- Người dùng có thể xem lịch sử hành động với các tùy chọn filter (All/Fan/AC/LED), smart search (time, datetime, text), backend sorting, và phân trang (5/10/20/50 items/page).

d) Giao diện người dùng thân thiện:

- Frontend được phát triển bằng React 19 với Vite, cung cấp giao diện dark theme trực quan và dễ sử dụng.
- Người dùng có thể dễ dàng chuyển đổi giữa 4 trang: Dashboard (Home), Data Sensor, Action History và Profile với React Router.

e) API linh hoạt:

- Backend cung cấp 7 RESTful API endpoints cho phép truy xuất dữ liệu cảm biến (latest, time series, paginated) và lịch sử hành động với nhiều tùy chọn lọc (date picker, filter, search), sắp xếp (backend sorting) và phân trang.

f) Cơ chế xóa dữ liệu tự động:

- Hệ thống tự động xóa các bản ghi cũ hơn 30 ngày mỗi 24 giờ để tối ưu hóa hiệu suất và không gian lưu trữ trong PostgreSQL.

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 50-100ms cho các truy vấn cơ bản (GET /api/health, GET /api/devices/:id/last).
- Đối với các truy vấn phức tạp hơn với smart search và pagination (GET /api/devices/:id/sensors), thời gian phản hồi vẫn dưới 150ms nhờ composite index và range-based queries.

b) Độ trễ điều khiển thiết bị:

- Thời gian từ khi người dùng click button đến khi LED thực sự thay đổi trạng thái trung bình khoảng 200ms (end-to-end latency).
- Phân bố độ trễ: Frontend API call (20ms) → Backend MQTT publish (50ms) → ESP32 receives (80ms) → LED ON (100ms) → Feedback (200ms).

c) Cập nhật dữ liệu thời gian thực:

- Dữ liệu cảm biến được ESP32 gửi mỗi 2 giây qua MQTT.
- Frontend cập nhật sensor cards và charts mỗi 10 giây thông qua API polling, đảm bảo thông tin luôn mới nhất mà không overload server.

2. Độ chính xác của dữ liệu cảm biến

a) Nhiệt độ (DHT11):

- Độ chính xác: ±2°C (theo datasheet DHT11)
- Độ phân giải: 0.1°C
- Range: 0-50°C

b) Độ ẩm (DHT11):

- Độ chính xác: ±5% RH (theo datasheet DHT11)
- Độ phân giải: 0.1% RH
- Range: 20-90% RH

c) Ánh sáng (MH-Sensor Flying-Fish):

- Loại sensor: Digital output module (không phải analog light meter chính xác)
- Range: 0-4095 (ADC 12-bit)
- Chức năng: Phát hiện thay đổi ánh sáng với signal conditioning và noise reduction

III. Cải tiến trong tương lai

1. Tăng cường bảo mật

- Triển khai hệ thống xác thực JWT và phân quyền người dùng.
- Sử dụng SSL/TLS cho kết nối MQTT (port 8883) và HTTPS cho API.
- Thêm cơ chế mã hóa password trong database và environment variables không commit lên GitHub.

2. Cải thiện khả năng mở rộng

- Triển khai cơ chế caching với Redis để giảm tải cho PostgreSQL.
- Hỗ trợ multi-device: Device registry table và dynamic device selection trên Frontend.
- Triển khai load balancing và microservices architecture cho Backend khi số lượng request tăng cao.

3. Nâng cao trải nghiệm người dùng

- Thêm tính năng thông báo browser push notification khi các chỉ số vượt ngưỡng (temperature > 35°C).
- Phát triển ứng dụng di động React Native để người dùng có thể theo dõi và điều khiển từ xa dễ dàng hơn.
- Tích hợp machine learning (LSTM) để dự đoán nhiệt độ và đưa ra các đề xuất tối ưu hóa năng lượng.

4. Mở rộng khả năng tương thích

- Hỗ trợ thêm các loại cảm biến IoT khác (CO2, PM2.5, sound level, motion sensor).
- Tích hợp với Home Assistant, IFTTT, AWS IoT, Azure IoT Hub.
- Phát triển REST API documentation với OpenAPI/Swagger 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 deep sleep cho ESP32 khi không cần thu thập dữ liệu (giảm từ 500mW xuống 10μW).
- Điều chỉnh tần suất gửi dữ liệu dựa trên mức độ thay đổi của các chỉ số (adaptive sampling: 2s → 10s khi stable).

6. Cải thiện khả năng phân tích

- Thêm các công cụ phân tích dữ liệu nâng cao (hourly/daily/weekly averages, correlation analysis) để cung cấp insights về xu hướng môi trường.
- Tích hợp machine learning để anomaly detection và tự động cảnh báo khi phát hiện bất thường.

7. Tăng cường độ tin cậy

- Triển khai cơ chế backup PostgreSQL tự động mỗi ngày và giữ lại 30 ngày.
- Cải thiện logging với Winston logger và monitoring với health check endpoint để phát hiện và xử lý sự cố nhanh chóng.
- Implement failover mechanism: Secondary MQTT broker, database replica, in-memory storage fallback.

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 smart home automation.

Danh sách API:

1. API Kiểm tra trạng thái server

- Link API: GET http://localhost:3000/api/health
- Tác dụng: Kiểm tra server và MQTT có hoạt động không

2. API Lấy thông tin thiết bị

- Link API: GET http://localhost:3000/api/devices/esp32_01
- Tác dụng: Xem tổng số bản ghi, lần cuối kết nối

3. API Lấy dữ liệu cảm biến mới nhất

- Link API: GET http://localhost:3000/api/devices/esp32_01/last
- Tác dụng: Lấy nhiệt độ, độ ẩm, ánh sáng hiện tại

4. API Lấy dữ liệu biểu đồ

- Link API: GET http://localhost:3000/api/devices/esp32_01/series?from=1hour&limit=100
- Tác dụng: Vẽ biểu đồ theo thời gian

5. API Bảng dữ liệu cảm biến

- Link API: GET http://localhost:3000/api/devices/esp32_01/sensors?page=1&limit=10
- Tác dụng: Hiển thị bảng dữ liệu có phân trang, tìm kiếm

6. API Lịch sử điều khiển

- Link API: GET http://localhost:3000/api/devices/esp32_01/actions?limit=50
- Tác dụng: Xem lịch sử bật/tắt LED, FAN, AC

7. API Điều khiển LED

- Link API: POST http://localhost:3000/api/devices/esp32_01/cmd/led
- Tác dụng: Bật/tắt đèn LED
- Body: {"value": "ON", "issued_by": "web"}

8. API Điều khiển FAN

- Link API: POST http://localhost:3000/api/devices/esp32_01/cmd/fan

- Tác dụng: Bật/tắt quạt
- Body: {"value": "ON", "issued_by": "web"}

9. API Điều khiển AC

- Link API: POST http://localhost:3000/api/devices/esp32_01/cmd/ac
- Tác dụng: Bật/tắt điều hòa
- Body: {"value": "ON", "issued_by": "web"}

10. MQTT Real-time

- Topic: iot/esp32_01/telemetry
- Tác dụng: Nhận dữ liệu cảm biến mỗi 2-3 giây
- Topic: iot/esp32_01/fan/status
- Tác dụng: Nhận trạng thái thực tế của FAN

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,

Lâm Thành Đức.