HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA HỆ THỐNG THÔNG TIN

BÁO CÁO MÔN IOT VÀ ỨNG DỤNG

Đề tài: Quản lý và điều khiển thiết bị trong nhà thông minh

Sinh viên thực hiện: Hoàng Minh Tuấn B22DCCN753
Môn học: IOT và Ứng dụng
Nhóm môn học: 16
Giảng viên hướng dẫn: Nguyễn Quốc Uy
Hà Nội, Ngày 22 Tháng 8 Năm 2025

MỤC LỤC

I. GIỚI THIỆU TỔNG QUAN

1. Giới thiệu đề tài

Trong bối cảnh cuộc sống hiện đại, công nghệ thông tin đã và đang len lỏi vào từng ngóc ngách của cuộc sống. Cùng với đó thì Internet of Thing (IoT) trở thành một phần không thể thiếu trong cuộc sống. Nó được sử dụng trong nhiều lĩnh vực công nghệ, nông nghiệp, các hoạt động sống hàng ngày,... Đề tài quản lý và điều khiển thiết bị trong nhà thông minh sẽ tập 3 chung xây dựng một hệ thống IoT đơn giản có chức năng là thu thập thông tin từ các cảm biến nhiệt độ, độ ẩm, ánh sáng đồng thời điều khiển các thiết bị quạt, điều hòa, bóng điện trong nhà. Hệ thống này nhằm giúp cho mọi người có thể theo dõi ngôi nhà của mình mọi lúc mọi nơi, chỉ cần có internet.

2. Mục tiêu đề tài

- Thu thập dữ liệu môi trường từ các cảm biến: Hệ thống cần có khả năng thu thập dữ liệu từ các cảm biến nhiệt độ, độ ẩm và ánh sáng. Dữ liệu này sẽ được ghi nhận liên tục và cập nhật lên hệ thống để người dùng có thể theo dõi từ xa.
- Điều khiển các thiết bị điện trong nhà: Hệ thống có khả năng điều khiển các thiết bị như quạt, điều hòa và bóng điện dựa trên các điều kiện môi trường hoặc theo yêu cầu của người dùng. Người dùng có thể điều khiển thiết bị thông qua giao diện từ xa bằng kết nối internet.
- Tích hợp với mạng không dây và kết nối internet: Hệ thống sử dụng kết nối WiFi để truyền tải dữ liệu từ cảm biến và gửi lệnh điều khiển đến các thiết bị. Điều này giúp người dùng dễ dàng theo dõi và điều khiển thiết bị trong nhà từ xa thông qua điện thoại hoặc máy tính.
- Lưu trữ và phân tích dữ liệu: Dữ liệu từ cảm biến sẽ được lưu trữ vào cơ sở dữ liệu, cho phép người dùng theo dõi các thay đổi theo thời gian, từ đó đưa ra các quyết định tự động dựa trên điều kiện môi trường (ví dụ, tự động bật điều hòa khi nhiệt độ vượt quá mức nhất định).

3. Mô tả hệ thống

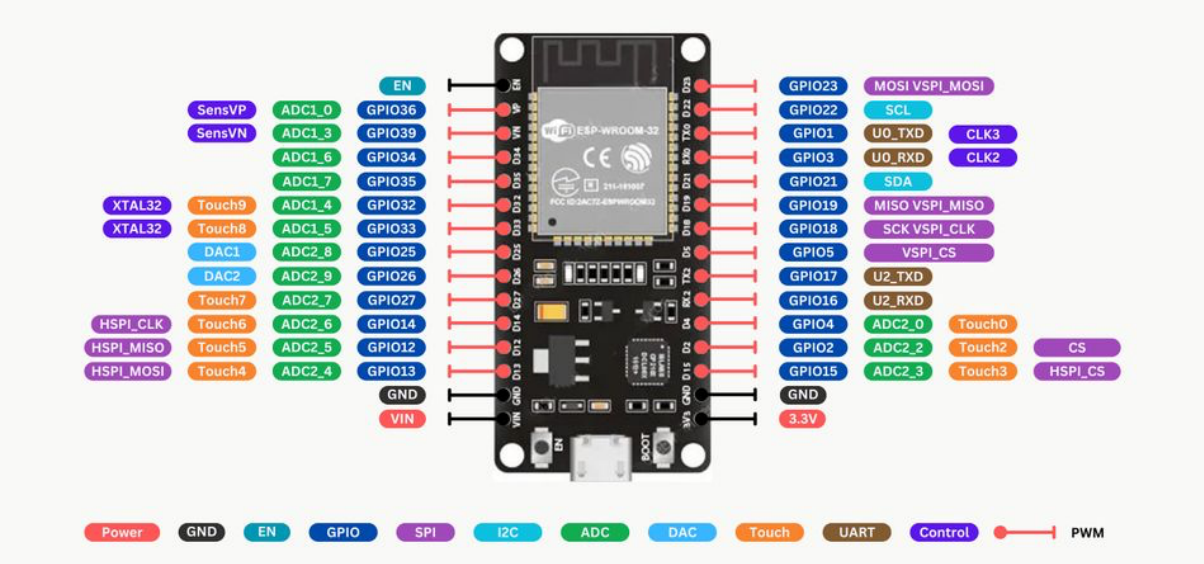
Hệ thống giám sát nhiệt độ, độ ẩm, ánh sáng:

- Cảm biến DHT22: Sử dụng cảm biến DHT11 để đo lường nhiệt độ (từ 0 đến 50°C) và độ ẩm (từ 20 đến 100%).
- Quang trở: Sử dụng quang trở giúp dễ dàng theo dõi sự thay đổi của cường độ

ánh sáng trong nhà.
-Dữ liệu được cập nhật theo thời gian thực để có thể giúp người dùng cập nhật nhiệt độ, độ ẩm ngay lập tức. Ngoài ra, người dùng cũng có thể theo dõi lịch sử của nhiệt độ, độ ẩm để biết khi nào nhiệt độ cao, khi nào nhiệt độ thấp.
-Hệ thống điều khiển (bật/tắt) các thiết bị trong nhà theo mong muốn mà không cần thao tác trực tiếp với thiết bị. Chỉ cần có internet, người dùng có thể thao tác trực tiếp bật/tắt các thiết bị trên website.

4. Các thiết bị sử dụng cho hệ thống

a. ESP32 WROM Wifi



Thông số kỹ thuật chính:

- **CPU:** Xtensa Dual-Core 32-bit LX6, xung nhịp 160 – 240 MHz
- **RAM:** 520 KB SRAM
- **Flash:** thường 4 MB (SPI Flash)
- **Kết nối không dây:**
  - Wi-Fi 802.11 b/g/n (2.4 GHz)
  - Bluetooth v4.2 + BLE

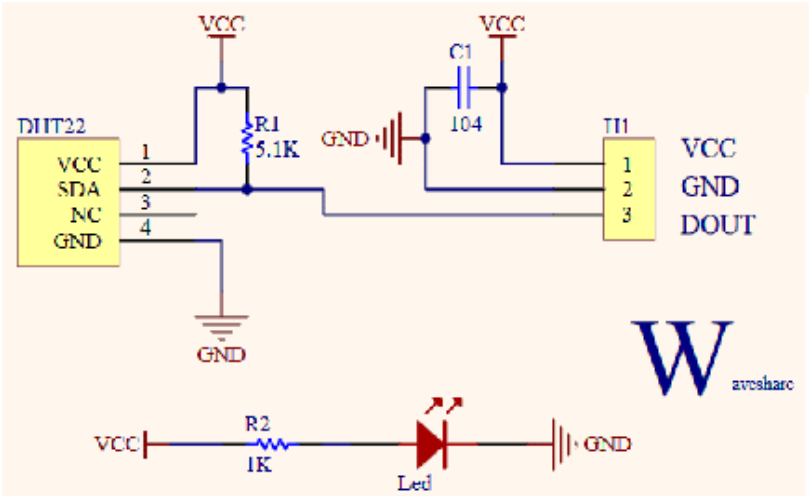
GPIO & I/O:

- **Tổng GPIO:** 34 chân khả dụng (nhiều chân đa chức năng)
- **ADC:** 18 kênh 12-bit (ADC1 & ADC2)
- **DAC:** 2 kênh 8-bit (DAC1: GPIO25, DAC2: GPIO26)
- **Touch Sensor:** 10 kênh cảm ứng điện dung
- **PWM:** gần như trên tất cả GPIO
- **SPI:** 4 SPI (VSPI, HSPI, v.v.)
- **I2C:** 2 I2C
- **UART:** 3 UART (UART0, UART1, UART2)
- **I2S:** 2 bộ I2S (âm thanh kỹ thuật số)
- **CAN:** 1 bộ CAN bus

Điện áp & năng lượng:

- **Điện áp hoạt động:** 3.0V – 3.6V (thường 3.3V)
- **Dòng tiêu thụ:**
  - ~240 mA (Wi-Fi TX full power)
  - <10 µA (deep sleep)

b, Cảm biến DHT22 (Cảm biến nhiệt độ, độ ẩm)



Thông số kỹ thuật DHT22

- **Điện áp hoạt động:** 3.3V – 6V (thường dùng 3.3V hoặc 5V)
- **Dòng tiêu thụ:**
  - Đo lường: ~1.5 mA
  - Chế độ chờ: ~0.1 mA

Đo độ ẩm:

- **Dải đo:** 0 – 100 %RH
- **Độ chính xác:** ±2 – 5 %RH
- **Độ phân giải:** 0.1 %RH

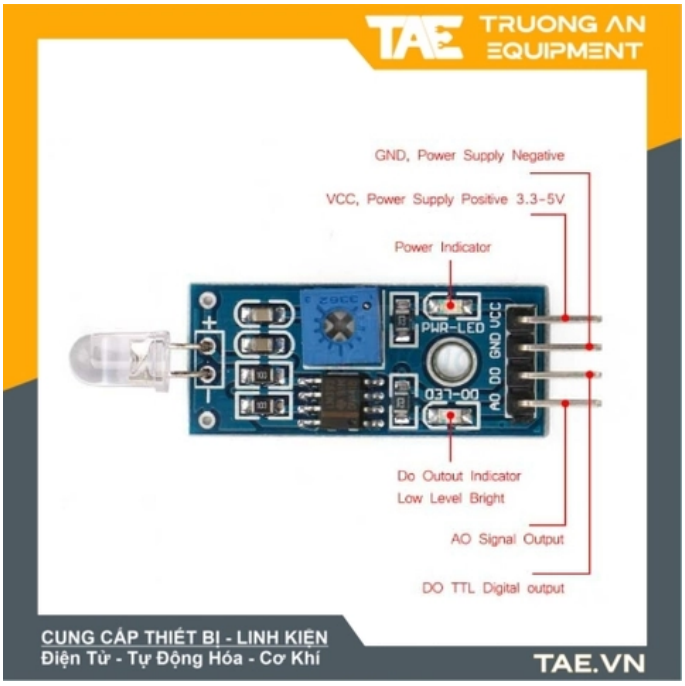
Đo nhiệt độ:

- **Dải đo:** -40 °C đến +80 °C
- **Độ chính xác:** ±0.5 °C (ở -10 °C đến +80 °C)
- **Độ phân giải:** 0.1 °C

Tín hiệu & giao tiếp:

- **Ngõ ra:** Digital (1 dây, giao tiếp dạng Single-bus)
- **Chu kỳ đo:** ~2 giây (tần suất lấy mẫu)
- **Khoảng cách truyền tín hiệu:** tối đa ~20m (cần điện trở kéo lên 5–10kΩ)

c, Cảm Biến Ánh Sáng Light Sensitive Photodiode



Thông số kỹ thuật cơ bản (phổ biến):

- **Điện áp hoạt động (Reverse bias):** 2V – 5V (có loại lên đến 30V)
- **Dòng tối (dark current):** vài nA (dòng rò khi không có ánh sáng)
- **Dòng bão hòa ánh sáng (photocurrent):** vài µA đến mA (tùy cường độ ánh sáng)
- **Bước sóng nhạy sáng:** 400 nm – 1100 nm (tốt nhất trong vùng ánh sáng nhìn thấy và cận hồng ngoại)
- **Tốc độ đáp ứng:** rất nhanh (µs đến ns) → dùng trong điều khiển từ xa, thu tín hiệu hồng ngoại.
- **Kích thước cảm biến:** nhỏ gọn, thường dạng diode 3mm, 5mm hoặc SMD.

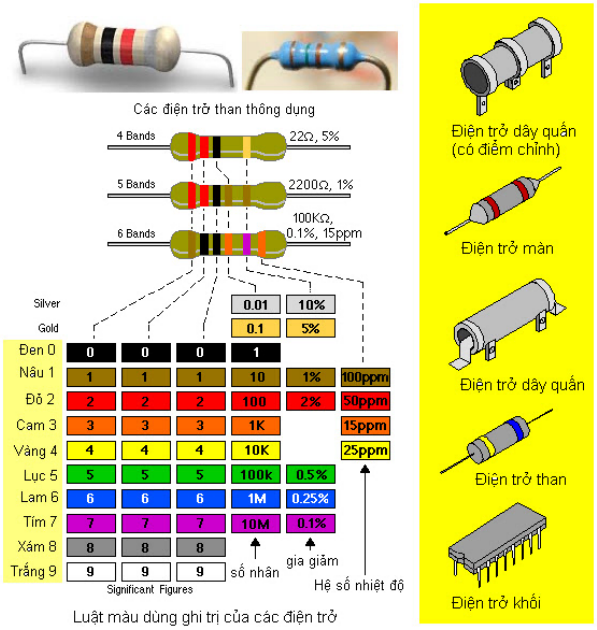
Nguyên lý hoạt động:

1. **Không có ánh sáng:** gần như không có dòng chảy (chỉ có dòng tối rất nhỏ).
2. **Có ánh sáng:** photon làm giải phóng electron trong diode → tạo dòng điện tỉ lệ thuận với cường độ ánh sáng.

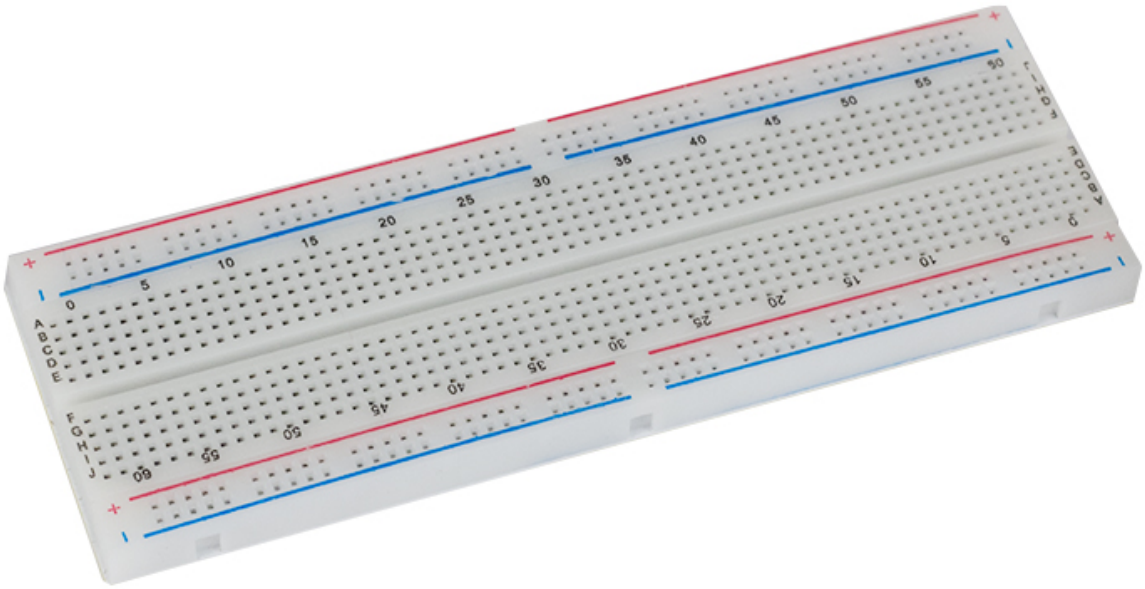
Ứng dụng:

- Cảm biến ánh sáng môi trường (bật/tắt đèn tự động).
- Remote hồng ngoại (IR receiver).
- Hệ thống an ninh (phát hiện tia sáng bị chặn).
- Thiết bị y tế (máy đo nhịp tim, SpO_2).
- Đo cường độ ánh sáng trong IoT, nông nghiệp thông minh.

d, Điện trở



e, BoardTest



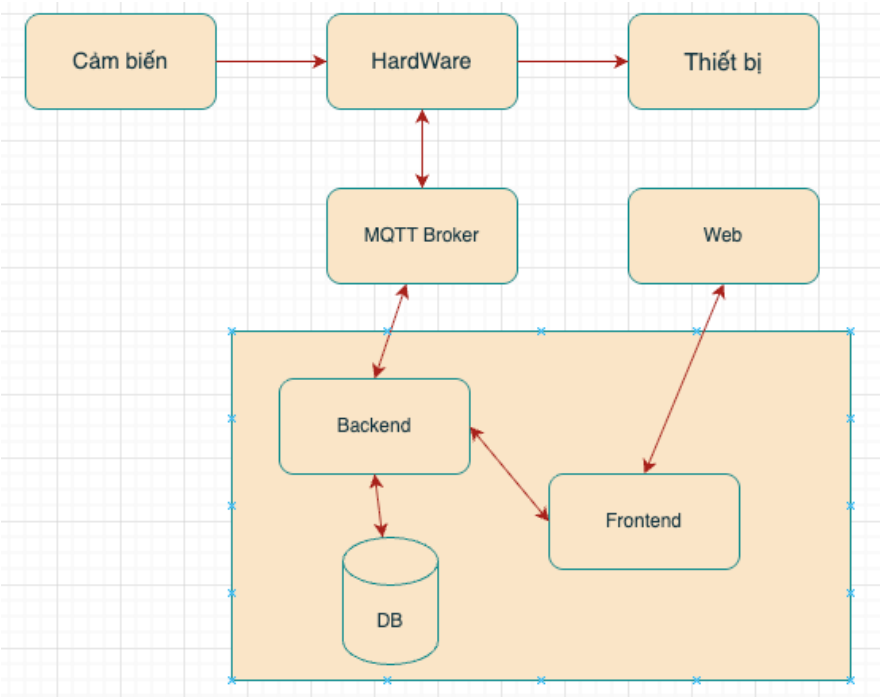
f, Dây cáp

g, các đèn led

III. THIẾT KẾ HỆ THỐNG

1. Thiết kế kiến trúc

* Sơ đồ thiết kế hệ thống:



Kiến trúc Hệ thống

* Thiết bị IoT (ESP32)

- **Publish** dữ liệu từ các cảm biến (nhiệt độ, độ ẩm, ánh sáng) lên MQTT topic **sensor/data**.
- **Subscribe** vào topic **control/led** để nhận lệnh điều khiển các thiết bị ngoại vi (quạt, đèn, điều hòa) từ server.

* Trung tâm Giao tiếp (MQTT Broker)

- Là cầu nối trung gian, nhận và chuyển tiếp tin nhắn giữa ESP32 và Server Web.
- **Các topic chính:**
  - **sensor/data**: Chứa dữ liệu cảm biến gửi từ ESP32.
  - **control/led**: Chứa lệnh điều khiển gửi từ Server Web đến ESP32.

* Server Web (ASP.NET Core & PostgreSQL)

- **Backend Service (MqttClientService.cs):**
  - Là một dịch vụ chạy nền, **subscribe** vào topic **sensor/data**.
  - Khi nhận được dữ liệu, nó sẽ xử lý và lưu vào bảng **sensordata** trong cơ sở dữ liệu PostgreSQL.
- **API Controller (IotApiController.cs):**
  - Cung cấp các API endpoint để giao tiếp với giao diện người dùng (Frontend).
  - **GET /api/IotApi/sensordata/...**: Trả về dữ liệu JSON cho các yêu cầu của web (lấy bản ghi mới nhất, lấy dữ liệu lịch sử cho biểu đồ, tìm kiếm & phân trang).
  - **GET /api/IotApi/devicestates**: Đọc bảng **actionhistories_new** để lấy trạng thái mới nhất của các thiết bị.
  - **POST /api/IotApi/devices/{deviceName}/toggle**:

1. Nhận yêu cầu bật/tắt từ người dùng.
2. Lưu hành động vào bảng **actionhistories_new**.
3. **Publish** một lệnh điều khiển (ví dụ: fan_on) vào topic **control/led** trên MQTT Broker.

***** Giao diện Người dùng (Web Dashboard)

- **Trang Home (Index.cshtml):**
  - Sử dụng JavaScript (fetch) để gọi API .../sensordata/latest và .../devicestates để hiển thị các thông số và trạng thái thiết bị.
  - Vẽ biểu đồ bằng **Chart.js** bằng cách gọi API .../sensordata/history.
  - Khi người dùng gạt công tắc, gửi yêu cầu đến API .../devices/{deviceName}/toggle.
- **Các trang khác (SensorData.cshtml, History.cshtml):**
  - Sử dụng JavaScript (fetch) để gọi các API tương ứng để lấy dữ liệu, hỗ trợ tìm kiếm và phân trang.
  - Sử dụng **SignalR** để nhận và hiển thị các cập nhật (dữ liệu cảm biến, lịch sử hành động) theo thời gian thực mà không cần tải lại trang.

2. Thiết kế Giao diện

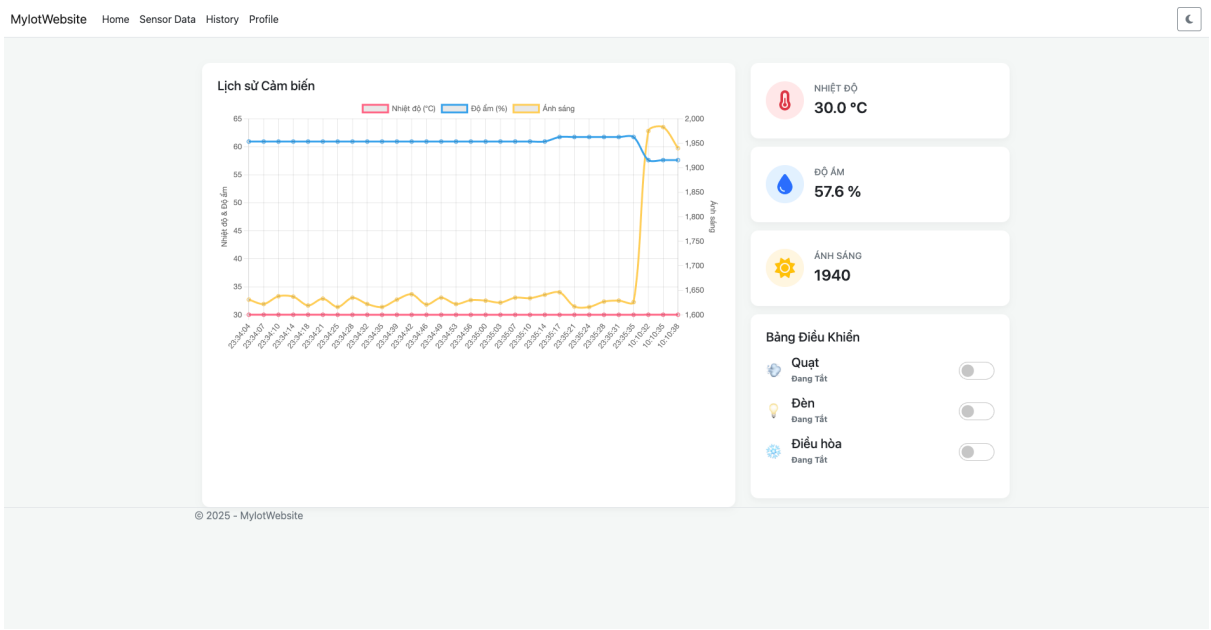
Giao diện hệ thống được thiết kế theo phong cách hiện đại, sạch sẽ, và đáp ứng (responsive), lấy người dùng làm trung tâm. Toàn bộ website hỗ trợ hai chế độ giao diện **Sáng/Tối (Light/Dark Mode)** và bao gồm 4 trang chính: Trang chủ (Dashboard), trang Dữ liệu Cảm biến (Sensor Data), trang Lịch sử (History), và trang Thông tin (Profile).

Công nghệ sử dụng:

- **Frontend:** HTML, CSS, Bootstrap 5, JavaScript.
- **Thư viện Frontend:** Chart.js (vẽ biểu đồ), SignalR Client (cho cập nhật thời gian thực).
- **Backend:** ASP.NET Core (C#).

---

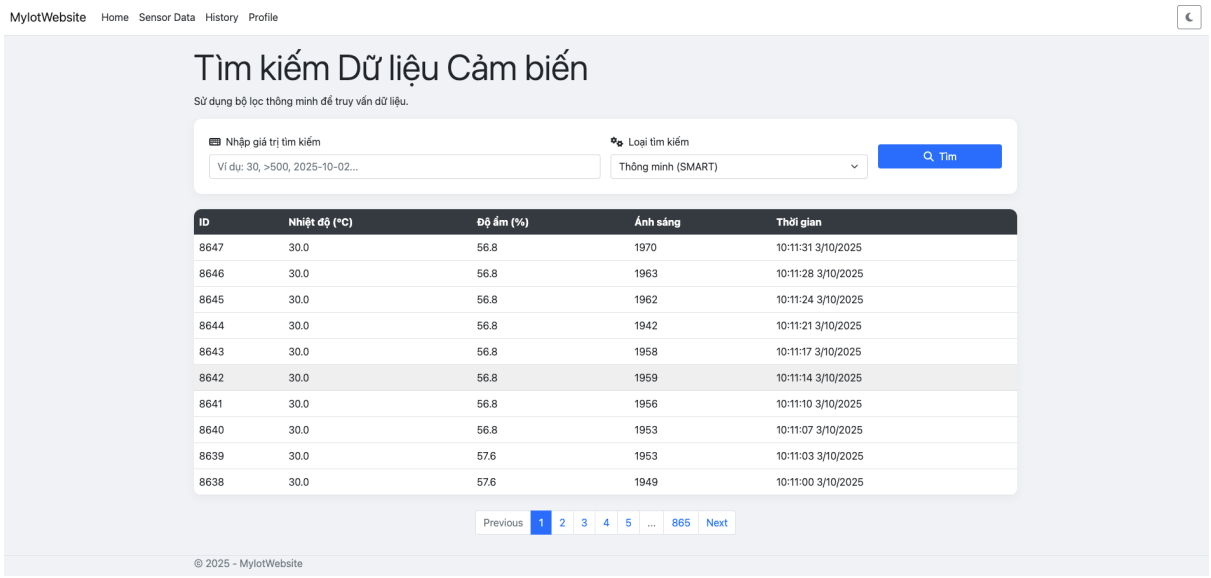
1. Trang chủ (Dashboard)



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

- **Bố cục:** Sử dụng layout 2 cột linh hoạt, tối ưu cho màn hình lớn và tự động điều chỉnh trên thiết bị di động.
- **Cột chính:** Hiển thị **Biểu đồ Lịch sử Cảm biến** sử dụng Chart.js. Biểu đồ được thiết kế với 2 trục tung (Y-axis) riêng biệt để hiển thị rõ ràng các dữ liệu có thang đo khác nhau (Nhiệt độ/Độ ẩm và Ánh sáng).
- **Cột phụ:**
    - **Thẻ Trạng thái:** Ba thẻ thông số (Nhiệt độ, Độ ẩm, Ánh sáng) được thiết kế tinh tế với icon và màu nhấn, hiển thị giá trị mới nhất nhận được.
    - **Bảng Điều Khiển:** Cho phép người dùng tương tác với các thiết bị (Quạt, Đèn, Điều hòa) thông qua các **công tắc gạt (toggle switch)** hiện đại, trực quan. Trạng thái của thiết bị được cập nhật theo thời gian thực.

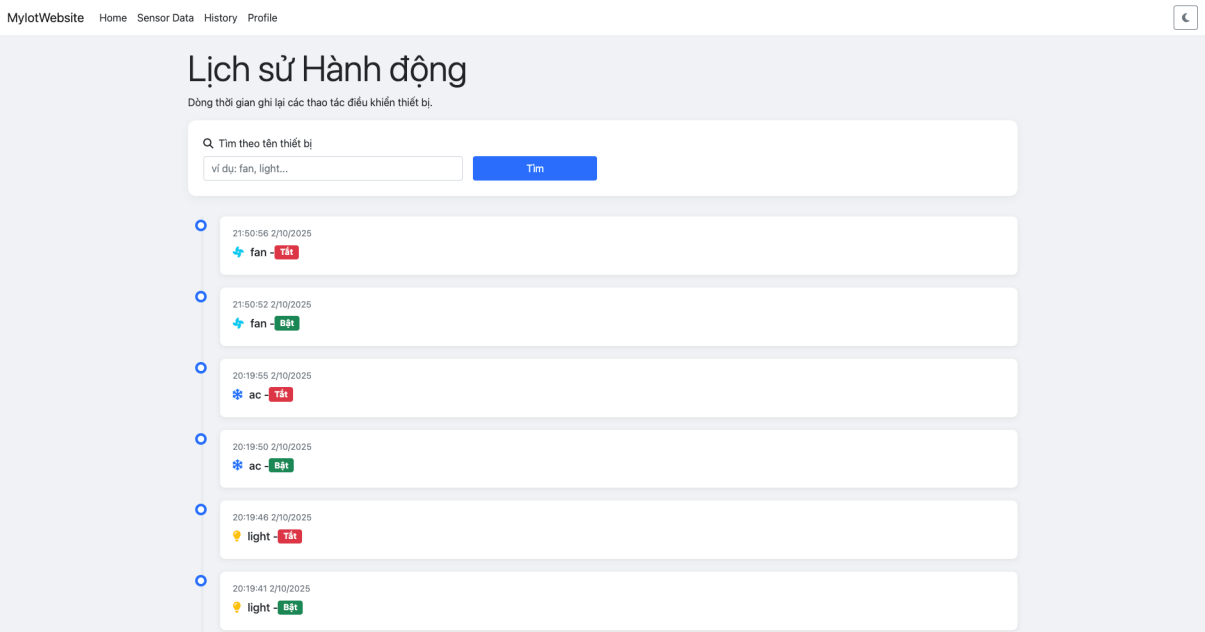
2. Trang Dữ liệu Cảm biến (Sensor Data)



Trang này là một công cụ mạnh mẽ để truy vấn và phân tích dữ liệu.

- **Giao diện:** Dữ liệu được trình bày trong một bảng hiện đại, được đặt trong thẻ "card" với hiệu ứng đổ bóng.
- **Bộ lọc "SMART":** Cung cấp một giao diện tìm kiếm hợp nhất và thông minh, cho phép người dùng:
    - Nhập một giá trị số để tìm kiếm theo ID hoặc các giá trị cảm biến lân cận.
    - Nhập một chuỗi ngày tháng (`YYYY-MM-DD`) để tự động lọc theo ngày.
    - Nhập các toán tử so sánh (>, <, =) để lọc theo ngưỡng giá trị.
- **Phân trang:** Tự động chia nhỏ dữ liệu thành các trang để đảm bảo hiệu suất và trải nghiệm người dùng.
- **Thời gian thực:** Tích hợp SignalR để các bản ghi dữ liệu mới tự động xuất hiện ở đầu bảng với hiệu ứng highlight mà không cần tải lại trang.

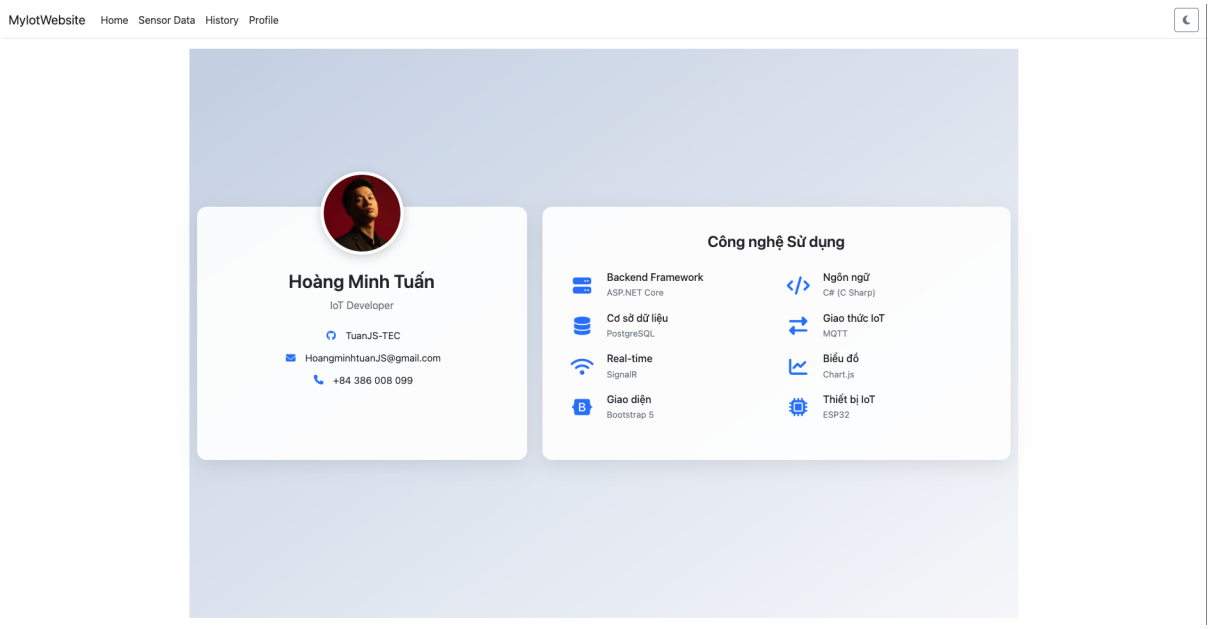
3. Trang Lịch sử (History)



Trang này hiển thị nhật ký các hành động điều khiển thiết bị của người dùng.

- **Giao diện Dòng thời gian (Timeline):** Thay vì bảng, lịch sử được trình bày dưới dạng timeline thẳng đứng, giúp người dùng dễ dàng theo dõi các sự kiện theo trình tự thời gian.
- **Trực quan hóa:** Mỗi hành động được biểu diễn bằng **icon** (quạt, đèn...) và **huy hiệu màu** (xanh cho "Bật", đỏ cho "Tắt"), giúp quét thông tin cực kỳ nhanh chóng.
- **Chức năng:** Hỗ trợ tìm kiếm theo tên thiết bị và phân trang đầy đủ.
- **Thời gian thực:** Các hành động mới nhất cũng được đẩy xuống qua SignalR và xuất hiện ngay lập tức ở đầu dòng thời gian.

4. Trang Thông tin (Profile)



Đây là trang giới thiệu về tác giả và chính dự án.

- **Thiết kế:** Sử dụng layout 2 cột.
- **Thẻ Thông tin cá nhân:** Hiển thị thông tin liên hệ (Email, SĐT, GitHub) và ảnh đại diện được lấy tự động từ dịch vụ **Gravatar**.
- **Thẻ "Showcase" Công nghệ:** Trình bày một cách chuyên nghiệp tất cả các công nghệ đã được sử dụng để xây dựng website, từ backend (ASP.NET Core, C#) đến frontend (Bootstrap, Chart.js) và IoT (ESP32, MQTT).

3. Luồng hoạt động

1. Thu thập và Hiển thị Dữ liệu Cảm biến theo Thời gian thực

1. **ESP32** đọc dữ liệu từ các cảm biến (DHT22, cảm biến ánh sáng).
2. **ESP32 Publish** một chuỗi dữ liệu (ví dụ: 26.5,60.2,1500) lên topic sensor/data của **MQTT Broker**.
3. **Backend** (MqttClientService chạy nền) đang **Subscribe** topic sensor/data và nhận được tin nhắn.
4. **Backend** xử lý chuỗi dữ liệu, tạo một đối tượng SensorData và lưu vào bảng sensordata trong **Database PostgreSQL**.
5. Ngay sau khi lưu thành công, **Backend** sử dụng **SignalR** để "đẩy" (push) đối tượng SensorData mới tới tất cả các **Frontend** đang kết nối.
6. **Frontend** (JavaScript trên trang Sensor Data và History) nhận được dữ liệu qua SignalR và tự động thêm một dòng mới vào đầu bảng/timeline với hiệu ứng highlight mà không cần tải lại trang.

2. Người dùng Truy vấn Dữ liệu Cảm biến (Tìm kiếm & Phân trang)

1. Người dùng nhập điều kiện vào bộ lọc "SMART" trên trang **Sensor Data** và nhấn "Tìm".

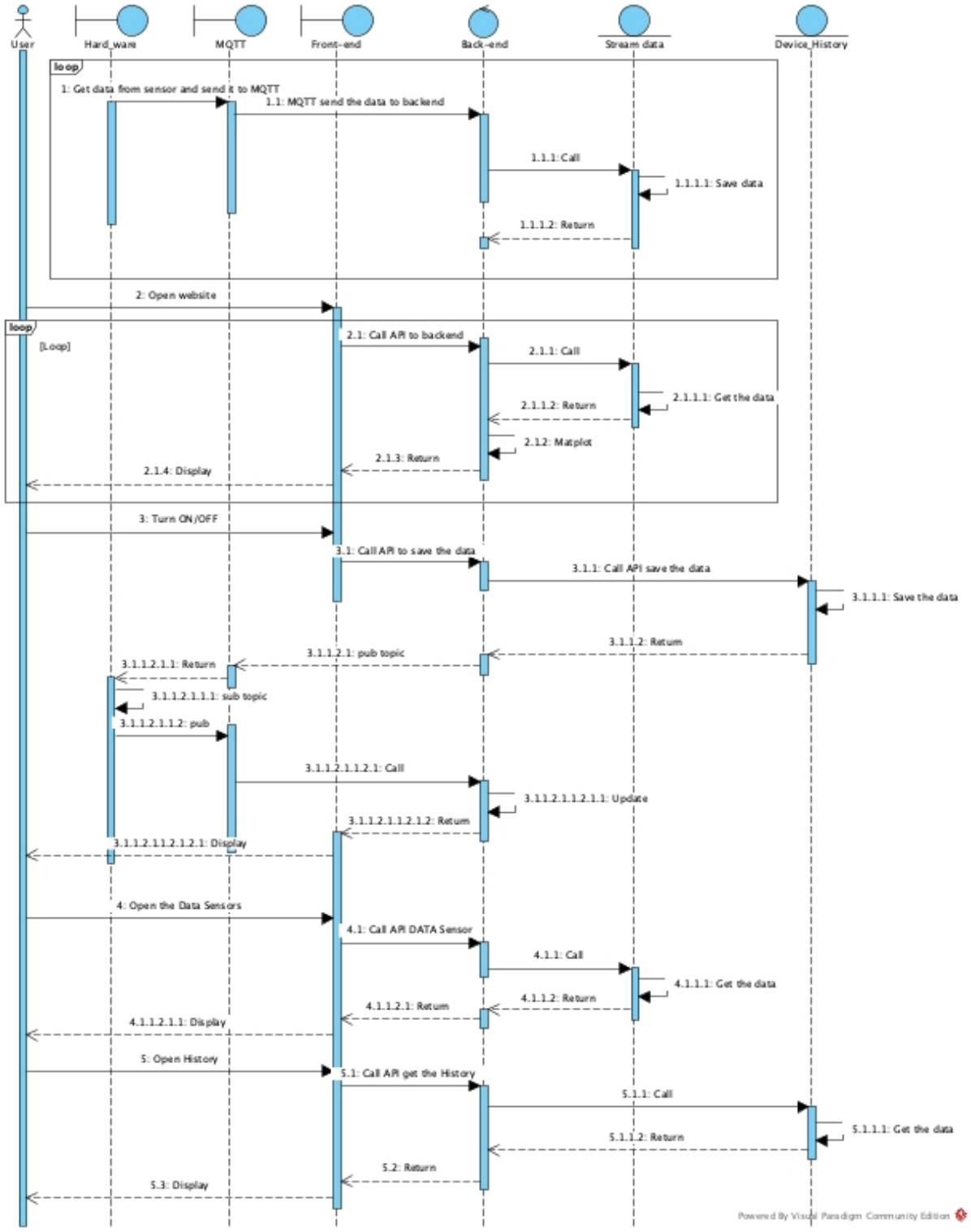
2. **Frontend** (JavaScript) gửi một yêu cầu GET đến API api/IotApi/sensordata/search của **Backend**, đính kèm các tham số tìm kiếm (searchTerm, searchType, pageNumber...).
3. **Backend** (IotApiController) nhận yêu cầu, phân tích các tham số và xây dựng một câu truy vấn LINQ phức tạp để lọc dữ liệu trong **Database PostgreSQL**.
4. **Database** trả về một trang dữ liệu đã được lọc và sắp xếp.
5. **Backend** đóng gói dữ liệu và thông tin phân trang (tổng số trang) vào một đối tượng JSON và gửi về cho **Frontend**.
6. **Frontend** nhận dữ liệu JSON, xóa bảng cũ và vẽ lại bảng dữ liệu mới cùng với các nút bấm phân trang.

3. Điều khiển Thiết bị từ xa

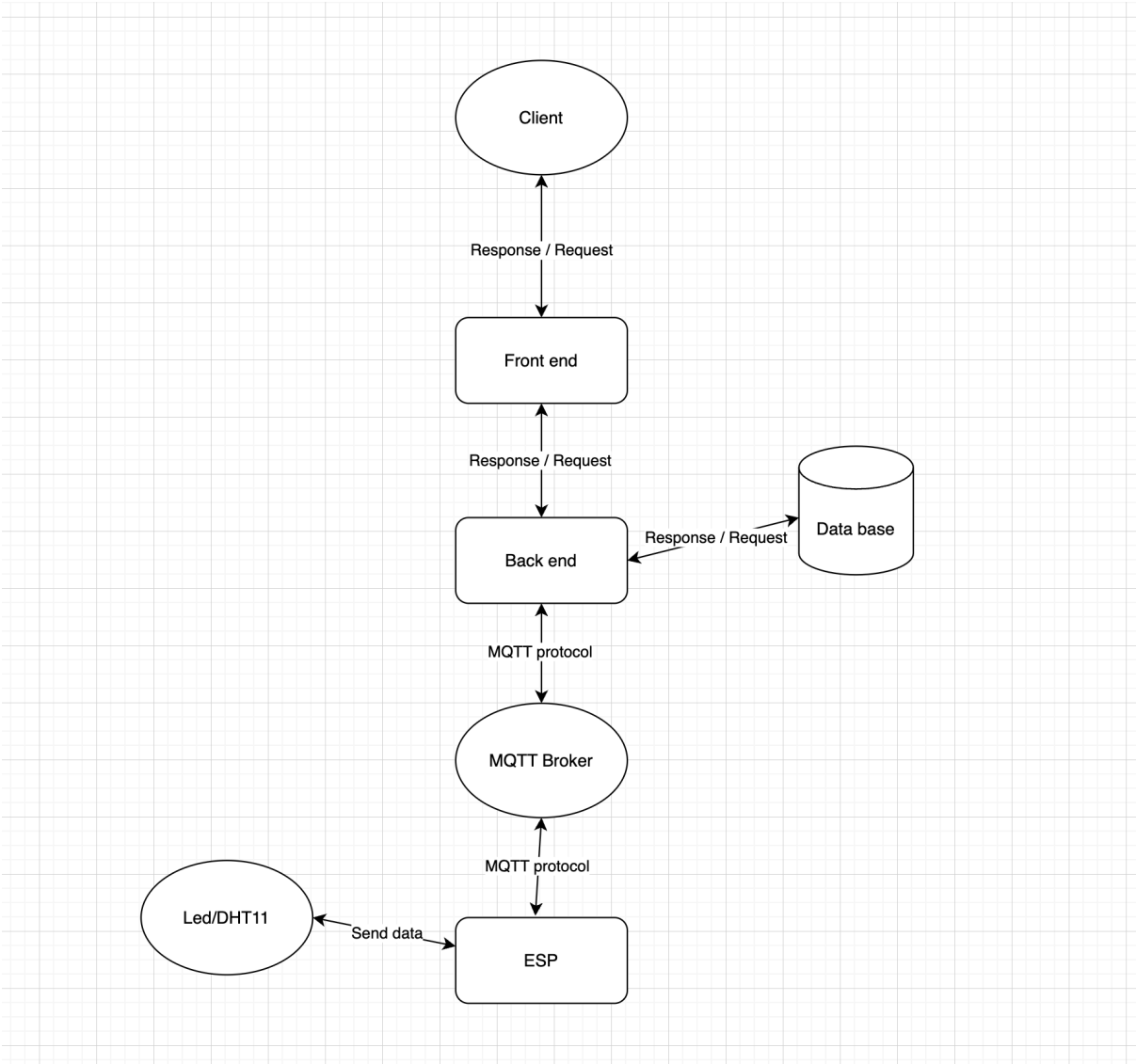
1. Người dùng gạt một công tắc (ví dụ: "Bật Quạt") trên trang **Home**.
2. **Frontend** (JavaScript) gửi một yêu cầu POST đến API api/IotApi/devices/fan/toggle của **Backend**.
3. **Backend** (IotApiController) nhận yêu cầu và truy vấn **Database** để tìm trạng thái gần nhất của "fan".
4. **Backend** tạo một bản ghi ActionHistory mới với trạng thái bị đảo ngược (IsOn = true) và lưu vào bảng actionhistories_new trong **Database**.
5. **Backend** sử dụng **SignalR** để "đẩy" thông tin hành động mới này đến các client, giúp trang **History** tự động cập nhật.
6. **Backend** tạo một tin nhắn MQTT (payload) là fan_on và **Publish** nó lên topic control/led của **MQTT Broker**.
7. **MQTT Broker** chuyển tiếp tin nhắn fan_on đến **ESP32** (đang subscribe topic control/led).
8. **ESP32** nhận được lệnh, thực thi hành động vật lý (bật quạt) và in log ra Serial Monitor.

4. Truy xuất Lịch sử Hành động

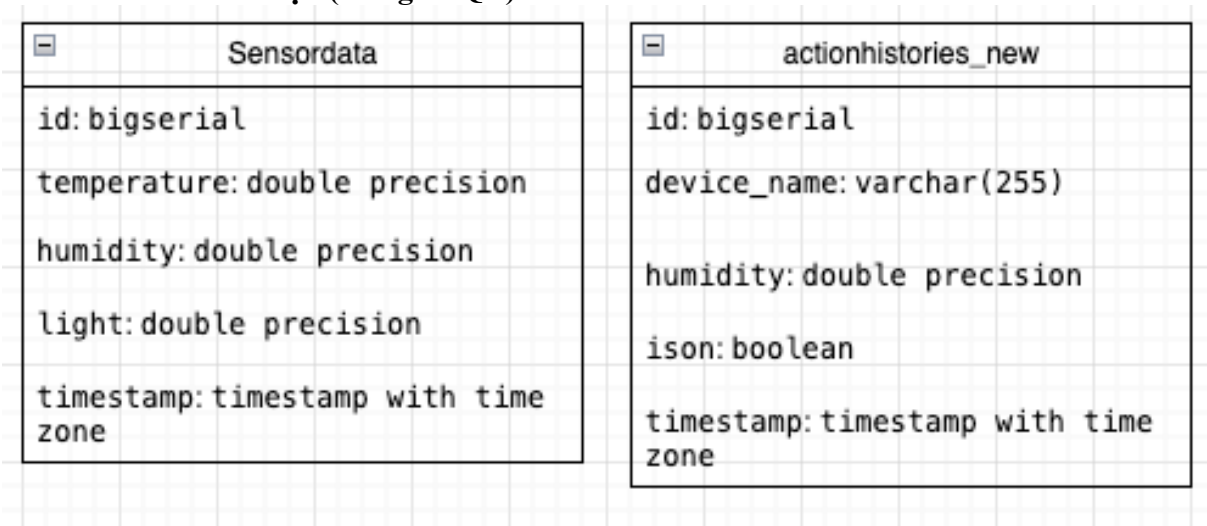
1. Người dùng mở trang **History**.
2. **Frontend** gọi API api/IotApi/actionhistory để lấy trang dữ liệu đầu tiên.
3. **Backend** truy xuất dữ liệu từ bảng actionhistories_new trong **Database**, hỗ trợ tìm kiếm theo tên thiết bị và phân trang.
4. **Backend** trả kết quả về **Frontend**.
5. **Frontend** hiển thị dữ liệu dưới dạng một **dòng thời gian (timeline)** trực quan thay vì bảng.


User
Hard_ware
MQTT
Front-end
Back-end
Stream data
Device_History
loop
1: Get data from sensor and send it to MQTT
1.1: MQTT send the data to backend
1.1.1: Call
1.1.1.1: Save data
1.1.1.2: Return
2: Open website
loop
[Loop]
2.1: Call API to backend
2.1.1: Call
2.1.1.1: Get the data
2.1.1.2: Return
2.1.2: Matplot
2.1.3: Return
2.1.4: Display
3: Turn ON/OFF
3.1: Call API to save the data
3.1.1: Call API save the data
3.1.1.1: Save the data
3.1.1.2: Return
3.1.1.2.1: pub topic
3.1.1.2.1.1: Return
3.1.1.2.1.1.1: sub topic
3.1.1.2.1.1.2: pub
3.1.1.2.1.1.2.1: Call
3.1.1.2.1.1.2.1.1: Update
3.1.1.2.1.1.2.1.2: Return
3.1.1.2.1.1.2.1.2.1: Display
4: Open the Data Sensors
4.1: Call API DATA Sensor
4.1.1: Call
4.1.1.1: Get the data
4.1.1.2: Return
4.1.1.2.1: Return
4.1.1.2.1.1: Display
5: Open History
5.1: Call API get the History
5.1.1: Call
5.1.1.1: Get the data
5.1.1.2: Return
5.2: Return
5.3: Display
Powered By Visual Paradigm Community Edition

4. Data Flow



5.Thiết kế cơ sở dữ liệu (PostgreSQL)



III. XÂY DỰNG HỆ THỐNG

1.API Docs

Nhóm	Tên API	URL	Method	Chức năng
Sensor Data	GetLatestSensorData	/api/IotApi/sensordata/latest	GET	Lấy bản ghi dữ liệu cảm biến mới nhất để hiển thị trên các thẻ thông số.
Sensor Data	GetSensorDataHistory	/api/IotApi/sensordata/history	GET	Lấy 30 bản ghi dữ liệu cảm biến gần nhất để vẽ biểu đồ trên trang chủ.
Sensor Data	SearchAllSensors	/api/IotApi/sensordata/search	GET	Tìm kiếm, lọc và phân trang dữ liệu cảm biến cho trang "Sensor Data".
Device Control	GetDeviceStates	/api/IotApi/devicestates	GET	Lấy trạng thái hiện tại của tất cả các thiết bị (Quạt, Đèn, Điều hòa).
Device Control	ToggleDevice	/api/IotApi/devices/{deviceName}/toggle	POST	Gửi lệnh bật/tắt một thiết bị. Đồng thời lưu hành động và publish lệnh qua MQTT.
History	GetActionHistory	/api/IotApi/actionhistory	GET	Lấy lịch sử các hành động bật/tắt thiết bị, hỗ trợ tìm kiếm và phân trang.

V. KẾT LUẬN

Trong đề tài này, em đã xây dựng thành công một hệ thống IoT giám sát và điều khiển thiết bị từ xa. Hệ thống dựa trên vi điều khiển **ESP32**, giao thức **MQTT**, cơ sở dữ liệu **PostgreSQL** và một ứng dụng web được xây dựng bằng **ASP.NET Core (C#)**. Kiến trúc hệ thống được thiết kế phân tầng rõ ràng, bao gồm: lớp thiết bị (ESP32 và các cảm biến), lớp truyền thông (MQTT Broker), lớp xử lý (Backend ASP.NET Core) và lớp giao diện (Web Dashboard).

Về mặt giám sát, hệ thống thu thập dữ liệu cảm biến (nhiệt độ, độ ẩm, ánh sáng) từ **ESP32**, truyền về **MQTT Broker**. Một dịch vụ chạy nền (`MqttClientService`) trên server sẽ lắng nghe và lưu trữ dữ liệu này vào **PostgreSQL**. Người dùng có thể theo dõi dữ liệu trên giao diện web thông qua các thẻ thông tin trực quan, biểu đồ biến thiên theo thời gian (sử dụng **Chart.js**), và xem lại lịch sử dữ liệu với chức năng tìm kiếm "SMART" và phân trang. Đặc biệt, hệ thống sử dụng **SignalR** để đẩy dữ liệu mới từ server đến trình duyệt ngay lập tức, mang lại trải nghiệm thời gian thực thực thụ.

Về mặt điều khiển, hệ thống cho phép người dùng bật/tắt các thiết bị (đèn, quạt, điều hòa) trực tiếp từ giao diện web. Lệnh điều khiển được API xử lý, lưu lại vào lịch sử hành động trong database, đồng thời publish một tin nhắn đến **MQTT Broker** để **ESP32** nhận và thực thi. Luồng hoạt động này đảm bảo khả năng phản hồi nhanh chóng và trạng thái hệ thống luôn được đồng bộ.

Hệ thống được xây dựng với ưu điểm:

- Kiến trúc được phân tách rõ ràng giữa phần cứng (ESP32), lớp giao tiếp (MQTT) và ứng dụng web (ASP.NET Core), giúp dễ dàng bảo trì và mở rộng trong tương lai.

- Cập nhật dữ liệu theo thời gian thực hiệu quả nhờ **SignalR**, giúp giảm tải cho server so với phương pháp hỏi liên tục (polling).
- Giao diện người dùng hiện đại, đáp ứng (responsive), hỗ trợ giao diện Sáng/Tối (Light/Dark mode) và có tính ứng dụng thực tế cao trong các mô hình nhà thông minh.

Tuy nhiên, hệ thống vẫn còn một số hạn chế:

- Mới dừng ở mức thử nghiệm trên môi trường phát triển cục bộ (local development), chưa triển khai trên một server thực tế.
- Chưa có cơ chế bảo mật nâng cao cho MQTT (SSL/TLS) và API (xác thực người dùng).
- Chưa có chức năng phân quyền người dùng (ví dụ: admin, user).
- Chưa có ứng dụng di động để hỗ trợ điều khiển từ xa thuận tiện hơn.

Trong tương lai, em định hướng sẽ mở rộng và hoàn thiện hệ thống theo các hướng:

- Bổ sung thêm nhiều loại cảm biến (khí gas, khói, chuyển động) để nâng cao tính ứng dụng.
- Tích hợp hệ thống xác thực và phân quyền người dùng (Authentication & Authorization) cho API và giao diện web.
- Phát triển ứng dụng di động (sử dụng .NET MAUI hoặc các nền tảng khác) để giám sát và điều khiển mọi lúc, mọi nơi.
- Nghiên cứu và áp dụng Trí tuệ nhân tạo (AI) để phân tích dữ liệu lịch sử, từ đó đưa ra các dự đoán và tự động điều chỉnh thiết bị một cách thông minh.