HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO BÀI TẬP LỚN

MÔN IOT VÀ ỨNG DỤNG

Giảng viên hướng dẫn: Nguyễn Quốc Uy

Sinh viên thực hiện: Nguyễn Văn Phú

Mã sinh viên: B22DCCN623

Hà Nội, 2025

MỤC LỤC

LỜI CẢM ƠN

Em xin gửi lời cảm ơn chân thành đến thầy Nguyễn Quốc Uy - giảng viên môn IoT và Ứng dụng, người đã hướng dẫn, truyền đạt kiến thức và kinh nghiệm quý báu trong suốt quá trình học tập. Thầy đã giúp em hiểu rõ về công nghệ IoT và cách ứng dụng vào thực tế, đồng thời truyền cảm hứng để em tự tin triển khai dự án của mình.

Em cũng xin cảm ơn thầy đã hướng dẫn em trong quá trình làm báo cáo. Những kiến thức và kỹ năng học được từ môn học này sẽ là nền tảng quan trọng cho con đường học tập và nghiên cứu của em sau này.

Em xin chân thành cảm ơn thầy!

BẢNG KÍ HIỆU VÀ THUẬT NGỮ

STT	Ký hiệu/Thuật ngữ	Ý nghĩa
1	IoT	Internet of Things – Internet vạn vật, các thiết bị kết nối mạng để thu thập và trao đổi dữ liệu.
2	Sensor	Thiết bị cảm biến, đo các thông số môi trường như nhiệt độ, độ ẩm, ánh sáng.
3	MQTT	Message Queuing Telemetry Transport – giao thức truyền dữ liệu nhẹ dùng cho IoT.
4	Broker	Thành phần trung gian quản lý và phân phối dữ liệu giữa các thiết bị và hệ thống.
5	Back End	Phần server xử lý dữ liệu, quản lý cơ sở dữ liệu và cung cấp API cho Front End.
6	Front End	Giao diện web hiển thị dữ liệu cảm biến và cho phép người dùng điều khiển thiết bị.
7	Device	Thiết bị có thể điều khiển được (ON/OFF) thông qua hệ thống IoT.
8	Action	Lịch sử thao tác người dùng hoặc trạng thái thiết bị được ghi lại trong hệ thống.
9	Database	Cơ sở dữ liệu lưu trữ các thông tin về cảm biến, thiết bị và hành động.
10	ON/OFF	Trạng thái bật/tắt của thiết bị trong hệ thống.
11	API	Application Programming Interface – giao diện lập trình để Front End và Back End giao tiếp.
12	Dashboard	Bảng điều khiển hiển thị dữ liệu thời gian thực, biểu đồ và các nút điều khiển thiết bị.

DANH MỤC HÌNH ẢNH

GIỚI THIỆU ĐỀ TÀI

Ngày nay, nhu cầu xây dựng các hệ thống nhà thông minh (Smart Home) và Internet of Things (IoT) đang ngày càng phát triển mạnh mẽ, đặc biệt trong lĩnh vực giám sát và điều khiển từ xa. Người dùng có thể dễ dàng quản lý thiết bị điện trong gia đình, theo dõi thông số môi trường (nhiệt độ, độ ẩm, ánh sáng) ngay trên điện thoại hoặc máy tính mà không cần phải thao tác trực tiếp. Việc này giúp tiết kiệm thời gian, nâng cao tính tiện lợi và mang lại trải nghiệm sống hiện đại, an toàn hơn.

Từ nhu cầu đó, đề tài được thực hiện với mục tiêu xây dựng một hệ thống giám sát và điều khiển thiết bị IoT. Cụ thể, hệ thống cho phép:

- Thu thập dữ liệu cảm biến (nhiệt độ, độ ẩm, ánh sáng).

- Gửi dữ liệu thời gian thực lên server qua giao thức MQTT.

- Điều khiển thiết bị (bật/tắt LED giả lập relay) từ giao diện web.

- Quản lý dữ liệu cảm biến, thiết bị trên backend để phục vụ cho việc theo dõi

Về mặt công nghệ:

- Frontend: ReactJS để xây dựng giao diện người dùng thân thiện, hiển thị dữ liệu cảm biến và điều khiển thiết bị.

- Backend: Django để xử lý dữ liệu, cung cấp API.

- Cơ sở dữ liệu: SQLite3 để lưu trữ thông tin thiết bị, dữ liệu cảm biến.

- Thiết bị IoT: ESP8266 kết hợp cảm biến DHT11, LDR và các chân điều khiển thiết bị.

- Truyền thông: giao thức MQTT với Mosquitto Broker để truyền dữ liệu và lệnh điều khiển theo cơ chế publish/subscribe.

Nội dung đồ án gồm 4 chương chính:

- Chương 1: Trình bày cơ sở lý thuyết về công nghệ sử dụng.

- Chương 2: Phân tích và thiết kế hệ thống.

- Chương 3: Cài đặt và triển khai hệ thống.

- Chương 4: Đưa ra tổng kết, đánh giá kết quả đạt được và hướng phát triển sau này.

CHƯƠNG I. CƠ SỞ LÝ THUYẾT

1.1. Front-End

1.1.1. Giới thiệu

Giao diện người dùng (Front-End) được xây dựng bằng ReactJS – một thư viện JavaScript phổ biến dùng để xây dựng giao diện web một cách linh hoạt, hiện đại. React cho phép tạo các component độc lập, tái sử dụng được và dễ dàng mở rộng hệ thống.

1.1.2. Tại sao chọn ReactJS

- React có cộng đồng lớn, tài liệu phong phú và dễ học.

- Cập nhật giao diện theo thời gian thực nhờ cơ chế Virtual DOM.

- Phù hợp với mô hình SPA (Single Page Application), giúp trải nghiệm người dùng mượt mà hơn.

- Dễ tích hợp với backend qua các API RESTful.

1.1.3. Ứng dụng trong hệ thống

ReactJS được sử dụng để xây dựng:

- Trang chủ hiển thị thông tin dữ liệu cảm biến (nhiệt độ, độ ẩm, ánh sáng).

- Giao diện trực quan để bật/tắt thiết bị IoT từ xa.

- Biểu đồ theo dõi dữ liệu cảm biến theo thời gian.

1.2. Back-End

1.2.1. Giới thiệu

Phần back-end sử dụng Django, một framework web mạnh mẽ viết bằng Python. Django giúp tạo API nhanh chóng, xử lý logic nghiệp vụ, xác thực người dùng và tương tác với cơ sở dữ liệu.

1.2.2. Tại sao chọn Django

- Django có sẵn hệ thống quản trị mạnh mẽ.

- Hỗ trợ ORM giúp thao tác cơ sở dữ liệu dễ dàng.

- Phân chia rõ ràng giữa các thành phần: views, models, serializers, urls.

- Dễ dàng tích hợp với Django REST Framework để xây dựng API phục vụ cho frontend.

1.2.3. Ứng dụng trong hệ thống

Django được sử dụng để:

- Cung cấp API nhận dữ liệu từ MQTT broker và lưu vào cơ sở dữ liệu.

- Trả về dữ liệu cảm biến cho frontend hiển thị.

- Quản lý dữ liệu cảm biến, thiết bị.

1.3. Database

1.3.1. Giới thiệu

Dự án sử dụng SQLite3 – hệ quản trị cơ sở dữ liệu mặc định của Django. SQLite là dạng cơ sở dữ liệu nhẹ, lưu toàn bộ dữ liệu trong một file duy nhất.

1.3.2. Tại sao chọn SQLite3

- Cài đặt đơn giản, không cần cấu hình phức tạp.

- Phù hợp với dự án cá nhân, demo hoặc quy mô nhỏ.

- Tích hợp sẵn trong Django, tiết kiệm thời gian phát triển ban đầu.

1.3.3. Ứng dụng trong hệ thống

SQLite3 được sử dụng để lưu trữ:

- Dữ liệu cảm biến (nhiệt độ, độ ẩm, ánh sáng) theo thời gian.

- Trạng thái các thiết bị IoT (ON/OFF).

1.4. Giao thức MQTT

1.4.1. Giới thiệu

MQTT (Message Queuing Telemetry Transport) là một giao thức nhắn tin nhẹ, được thiết kế tối ưu cho các hệ thống IoT với băng thông thấp, độ trễ thấp. Giao thức hoạt động dựa trên mô hình publish/subscribe.

1.4.2. Tại sao chọn MQTT

- Tiết kiệm băng thông, phù hợp cho mạng IoT.

- Dễ triển khai với Mosquitto broker.

- Hỗ trợ nhiều client kết nối đồng thời.

- Hoạt động ổn định trên các vi điều khiển như ESP8266.

1.4.3. Ứng dụng trong hệ thống

MQTT được sử dụng để:

- Thiết bị ESP8266 publish dữ liệu cảm biến lên topic sensor/data.

- Frontend/Backend subscribe để nhận dữ liệu thời gian thực.

- Người dùng gửi lệnh điều khiển thiết bị qua topic device/control/[id].

1.5. ESP8266

1.5.1. Giới thiệu

ESP8266 là một vi điều khiển WiFi giá rẻ, phổ biến trong các dự án IoT nhờ khả năng kết nối mạng mạnh mẽ và hỗ trợ nhiều thư viện.

1.5.2. Ứng dụng trong hệ thống

ESP8266 được sử dụng để:

- Kết nối WiFi và giao tiếp với MQTT broker.

- Đọc dữ liệu cảm biến DHT11 và LDR.

- Điều khiển thiết bị (LED giả lập relay) dựa trên tín hiệu nhận được từ server.

1.6. Cảm biến

1.6.1. DHT11

- Đo nhiệt độ và độ ẩm.

- Được sử dụng trong hệ thống để giám sát môi trường.

1.6.2. LDR (Light Dependent Resistor)

- Đo cường độ ánh sáng môi trường.

- Dữ liệu dùng để tự động bật/tắt thiết bị chiếu sáng hoặc giám sát tình trạng ánh sáng.

1.7. Tổng kết chương I

Chương 1 đã cung cấp cái nhìn tổng quan về các công nghệ và nền tảng lý thuyết phục vụ cho việc xây dựng hệ thống IoT:

- ReactJS: được lựa chọn để phát triển giao diện người dùng trực quan, dễ sử dụng, hỗ trợ cập nhật dữ liệu theo thời gian thực.

- Django: đóng vai trò xử lý backend, quản lý dữ liệu và cung cấp API phục vụ frontend.

- SQLite3: cơ sở dữ liệu nhẹ, phù hợp cho quy mô hệ thống demo và nghiên cứu.

- MQTT: giao thức truyền thông tin nhẹ, hiệu quả, giúp các thiết bị IoT và server trao đổi dữ liệu nhanh chóng theo mô hình publish/subscribe.

- ESP8266: vi điều khiển WiFi đóng vai trò thiết bị IoT chính, chịu trách nhiệm thu thập dữ liệu cảm biến và nhận lệnh điều khiển.

- Cảm biến DHT11 và LDR: cung cấp dữ liệu nhiệt độ, độ ẩm và ánh sáng cho hệ thống.

Những cơ sở lý thuyết này là nền tảng để triển khai các bước tiếp theo trong đồ án, bao gồm phân tích – thiết kế hệ thống, cài đặt và triển khai, cũng như đánh giá và kết luận.

CHƯƠNG II. PHÂN TÍCH, THIẾT KẾ HỆ THỐNG

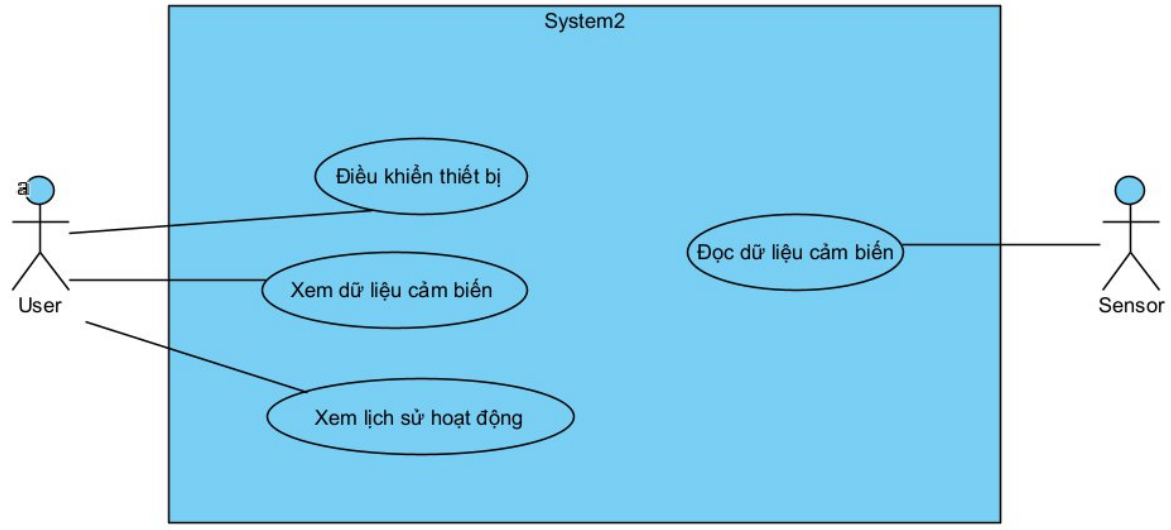
2.1. Tổng quan về nghiệp vụ

Hệ thống có 3 tác nhân chính là thiết bị cảm biến (Sensor), người dùng (User):

- Thiết bị cảm biến (Sensor) là các thiết bị IoT được lắp đặt trong môi trường để đo các thông số như nhiệt độ, độ ẩm, ánh sáng. Các thiết bị này tự động gửi dữ liệu định kỳ về hệ thống để hiển thị trên dashboard và phục vụ các phân tích thời gian thực. Sensor chỉ thực hiện đo và gửi dữ liệu, không có quyền truy cập trực tiếp vào giao diện người dùng.

- Người dùng (User) là người truy cập hệ thống thông qua giao diện. Người dùng có thể xem dữ liệu cảm biến theo thời gian thực, xem biểu đồ lịch sử, và thực hiện các hành động điều khiển thiết bị (ON/OFF) thông qua dashboard.

2.2. Biểu đồ usecase tổng quát



Hình 1. Usecase tổng quát

2.3. Sơ đồ lớp thực thể

2.3.1. Các lớp thực thể

- DataSensor: dữ liệu cảm biến

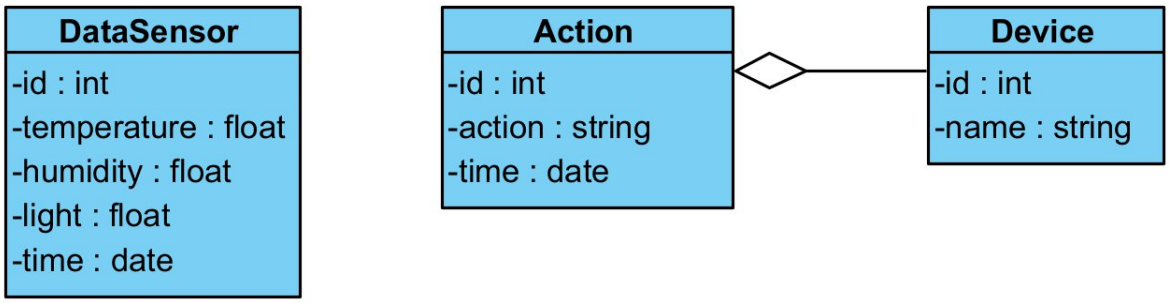
- Device: thiết bị

- Action: lịch sử hành động của thiết bị

2.3.2. Mối quan hệ giữa các thực thể

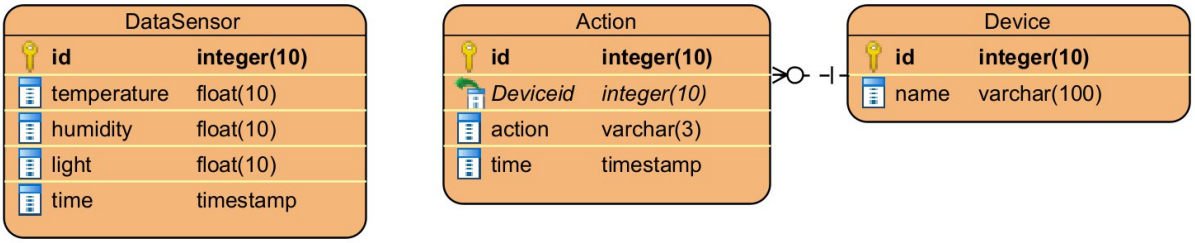
- Mỗi Action gắn với 1 Device nhất định, 1 Device có nhiều Action

2.3.3. Vẽ sơ đồ lớp thực thể



Hình 2. Sơ đồ lớp thực thể

2.4. Thiết kế cơ sở dữ liệu



Hình 3. Cơ sở dữ liệu

* Mô tả:

- Bảng Data Sensor:

STT	Tên cột	Kiểu dữ liệu	Mô tả
1	id	interger	Mã định danh (FK)
2	temperature	interger	Nhiệt độ (°C)
3	humidity	interger	Độ ẩm (%)
4	light	interger	Cường độ ánh sáng (lux)
5	time	datetime	Thời gian nhận dữ liệu

- Bảng Action:

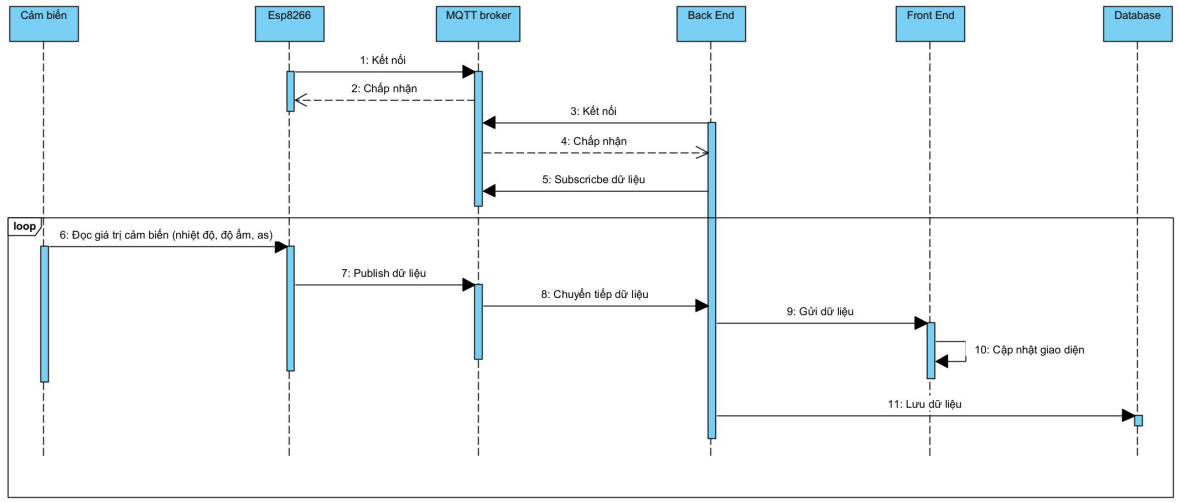
STT	Tên cột	Kiểu dữ liệu	Mô tả
1	id	interger	Mã định danh (FK)
2	deviceId	varchar	Mã thiết bị
3	action	varchar	Trạng thái thiết bị
4	time	datetime	Thời gian thay đổi trạng thái

- Bảng Device:

STT	Tên cột	Kiểu dữ liệu	Mô tả
1	id	interger	Mã định danh (FK)
2	name	varchar	Tên thiết bị

2.5. Thiết kế sơ đồ tuần tự

2.5.1. Đọc dữ liệu cảm biến



Hình 4. Sơ đồ tuần tự đọc dữ liệu cảm biến

1. ESP8266 kết nối với MQTT Broker

2. MQTT Broker chấp nhận kết nối từ ESP8266

3. Back End kết nối với MQTT Broker

4. MQTT Broker chấp nhận kết nối từ Back End

5. Back End đăng ký (subscribe) đến topic chứa dữ liệu cảm biến mà ESP8266 sẽ gửi lên.

6. ESP8266 liên tục đọc các giá trị từ cảm biến như nhiệt độ, độ ẩm, và ánh sáng.

7. ESP8266 gửi dữ liệu (Publish) lên MQTT Broker

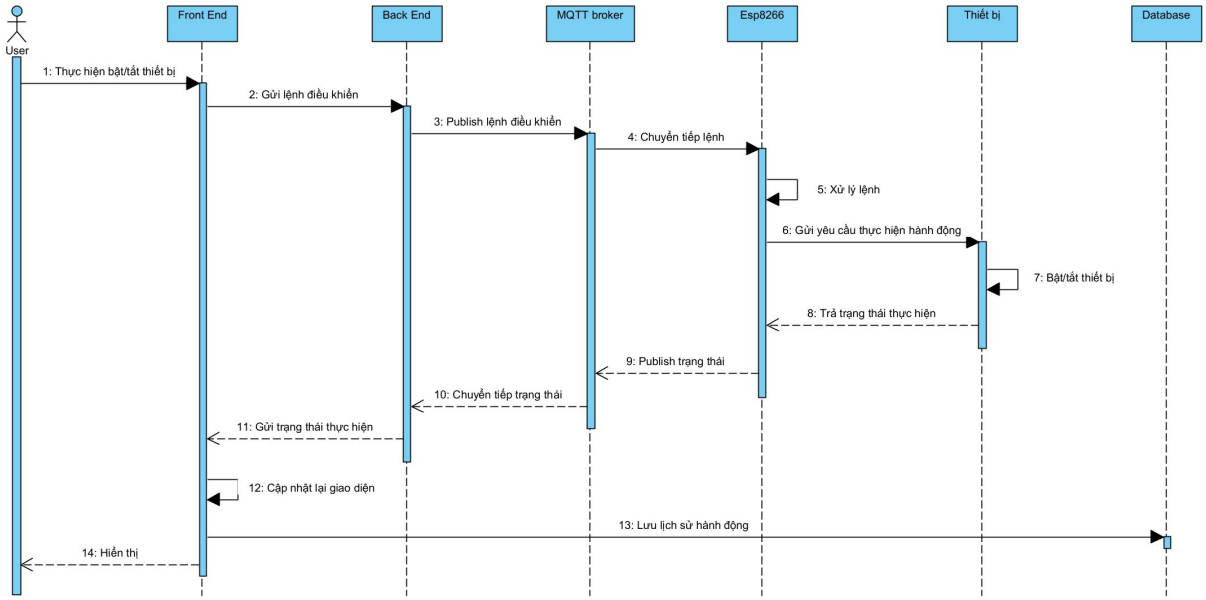
8. MQTT Broker chuyển tiếp dữ liệu đến Back End đã subscribe topic tương ứng.

9. Back End gửi dữ liệu đến Front End

10. Front End cập nhật giao diện theo dữ liệu mới

11. Back End lưu dữ liệu vào Database

2.52. Thao tác với thiết bị



Hình 5. Sơ đồ tuần tự thao tác với thiết bị

1. User thực hiện thao tác bật/tắt thiết bị trên giao diện.

2. Front End gửi lệnh điều khiển đến Back End.

3.Back End publish lệnh điều khiển lên MQTT Broker.

4. MQTT Broker chuyển tiếp lệnh đến ESP8266.

5. ESP8266 xử lý lệnh nhận được.

6. ESP8266 gửi yêu cầu thực hiện hành động đến thiết bị.

7. Thiết bị thực hiện bật/tắt theo yêu cầu.

8. Thiết bị trả về trạng thái thực hiện cho ESP8266.

9. ESP8266 publish trạng thái mới lên MQTT Broker.

10. MQTT Broker chuyển tiếp trạng thái đến Back End.

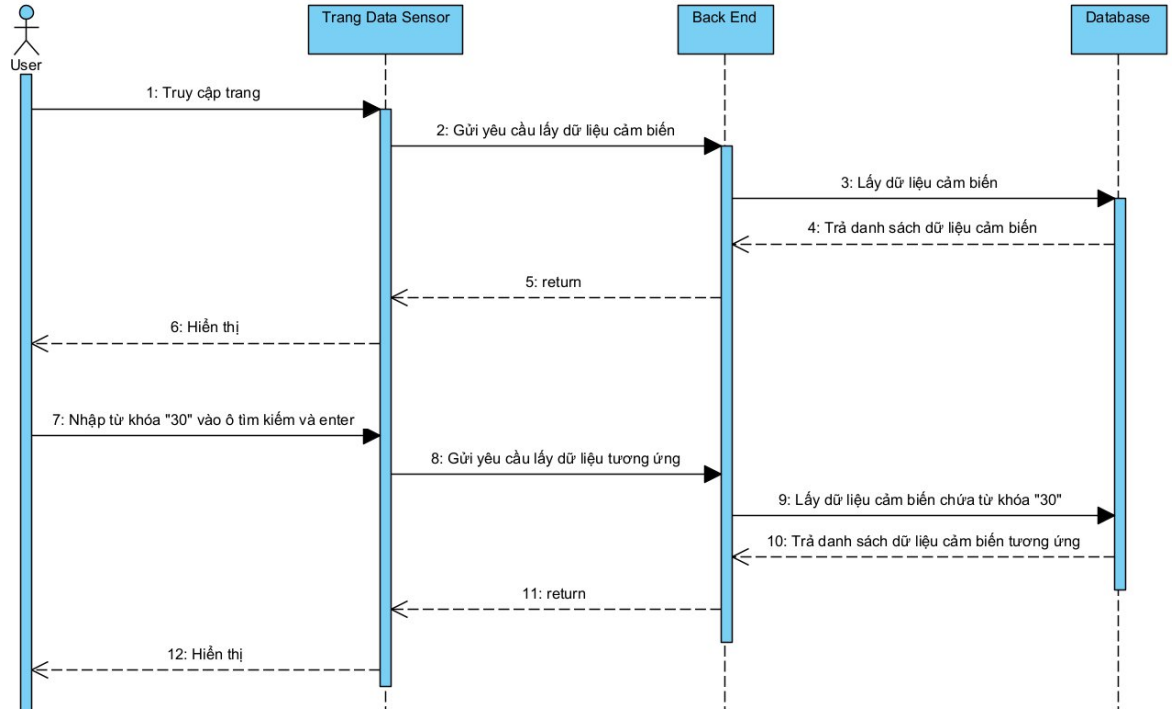
11. Back End gửi trạng thái thực hiện về Front End.

12. Front End cập nhật lại giao diện theo trạng thái mới.

13. Back End lưu lịch sử hành động vào Database.

14. User nhìn thấy kết quả hiển thị cập nhật trên giao diện.

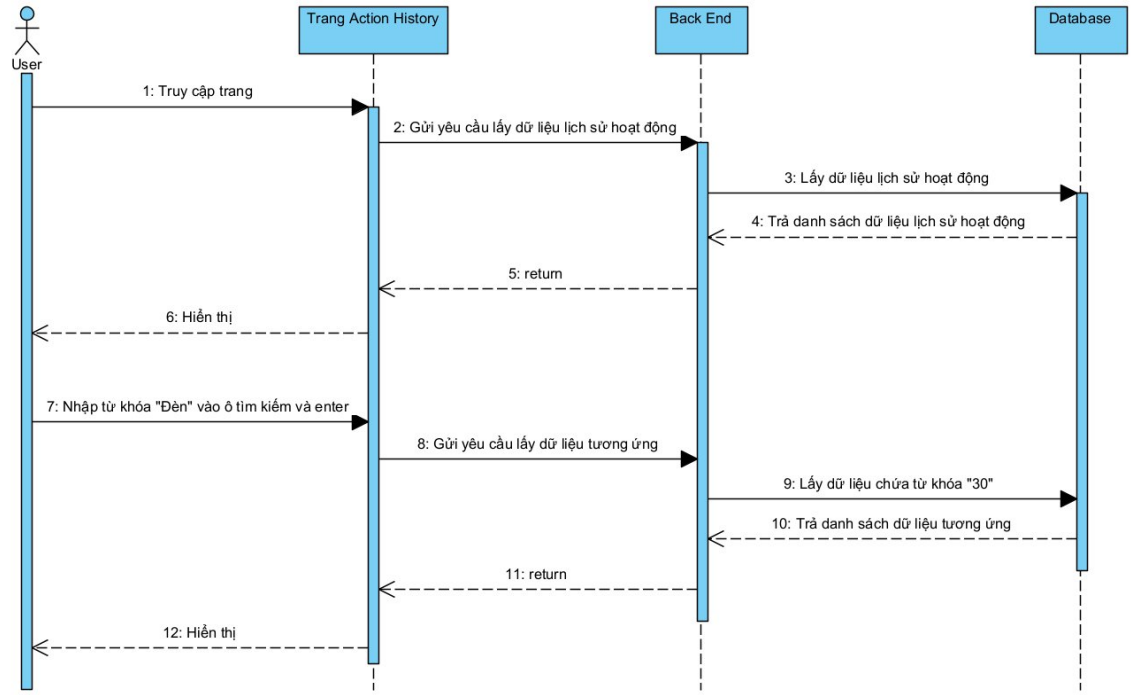
2.5.3. Xem dữ liệu cảm biến



Hình 6. Sơ đồ tuần tự xem dữ liệu cảm biến

1. User truy cập vào Trang Data Sensor.
2. Trang Data Sensor gửi yêu cầu đến Back End để lấy dữ liệu cảm biến.
3. Back End truy vấn và lấy dữ liệu cảm biến từ Database.
4. Database trả danh sách dữ liệu cảm biến về cho Back End.
5. Back End gửi dữ liệu cảm biến trả lại cho Trang Data Sensor.
6. Trang Data Sensor hiển thị danh sách dữ liệu cảm biến cho User.
7. User nhập từ khóa “30” vào ô tìm kiếm và nhấn Enter.
8. Trang Data Sensor gửi yêu cầu đến Back End để lấy dữ liệu cảm biến tương ứng với từ khóa.
9. Back End truy vấn Database để lấy dữ liệu cảm biến có chứa từ khóa “30”.
10. Database trả về danh sách dữ liệu cảm biến tương ứng cho Back End.
11. Back End gửi kết quả trả về Trang Data Sensor.
12. Trang Data Sensor hiển thị danh sách dữ liệu cảm biến đã lọc cho User.

2.5.4. Xem lịch sử hoạt động

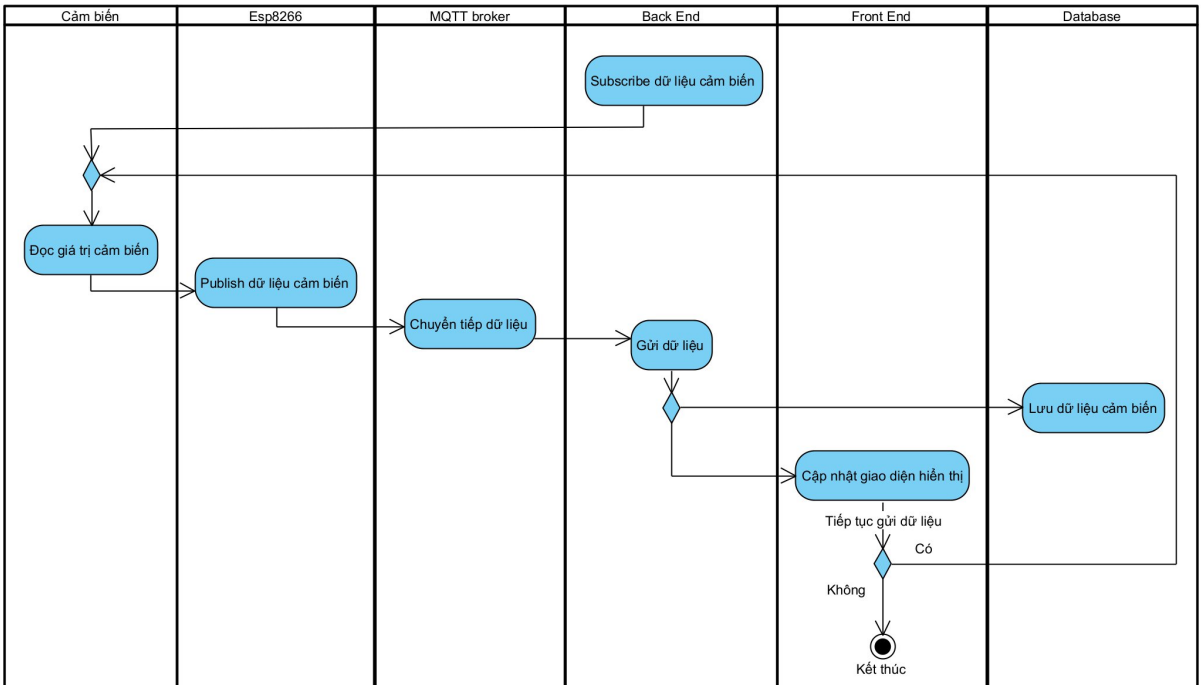


Hình 7. Sơ đồ tuần tự xem lịch sử hoạt động

1. User truy cập vào Trang Action History.
2. Trang Action History gửi yêu cầu đến Back End để lấy dữ liệu lịch sử hoạt động.
3. Back End truy vấn và lấy dữ liệu lịch sử hoạt động từ Database.
4. Database trả danh sách dữ liệu lịch sử hoạt động về cho Back End.
5. Back End gửi dữ liệu lịch sử hoạt động trả lại cho Trang Action History.
6. Trang Action History hiển thị danh sách lịch sử hoạt động cho User.
7. User nhập từ khóa “Đèn” vào ô tìm kiếm và nhấn Enter.
8. Trang Action History gửi yêu cầu đến Back End để lấy dữ liệu lịch sử tương ứng với từ khóa.
9. Back End truy vấn Database để lấy dữ liệu có chứa từ khóa “Đèn”.
10. Database trả danh sách dữ liệu lịch sử tương ứng cho Back End.
11. Back End gửi kết quả trả về Trang Action History.
12. Trang Action History hiển thị danh sách dữ liệu đã lọc cho User.

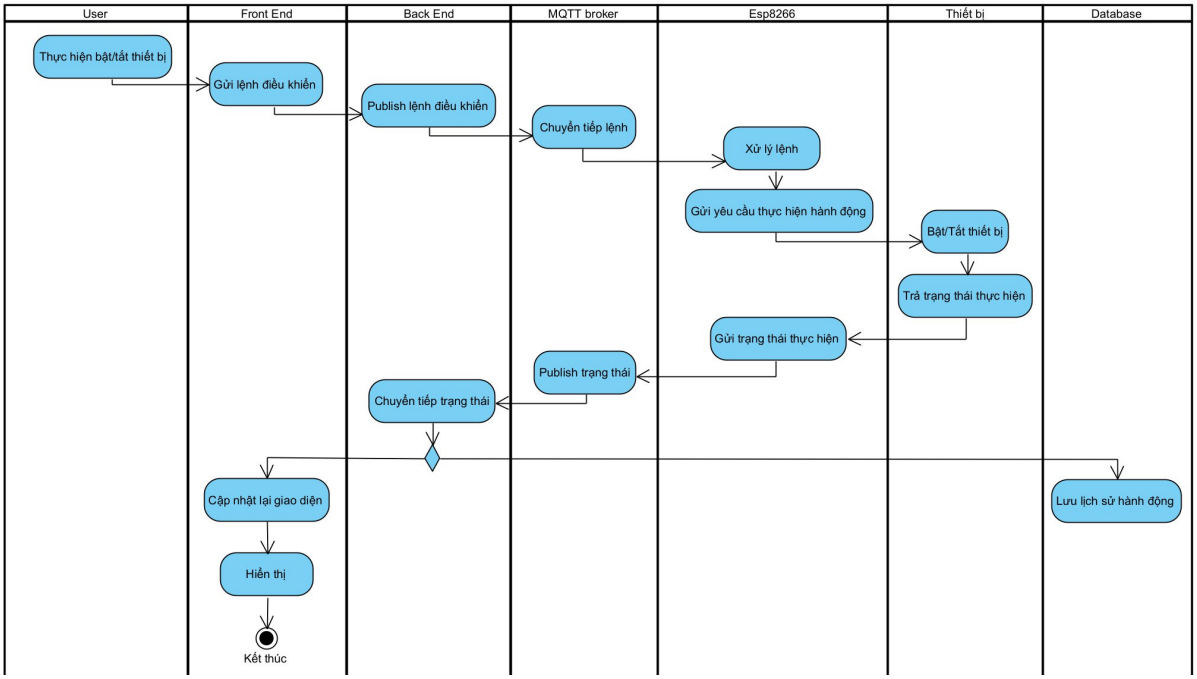
2.6. Thiết kế sơ đồ luồng hoạt động

2.6.1. Đọc dữ liệu cảm biến



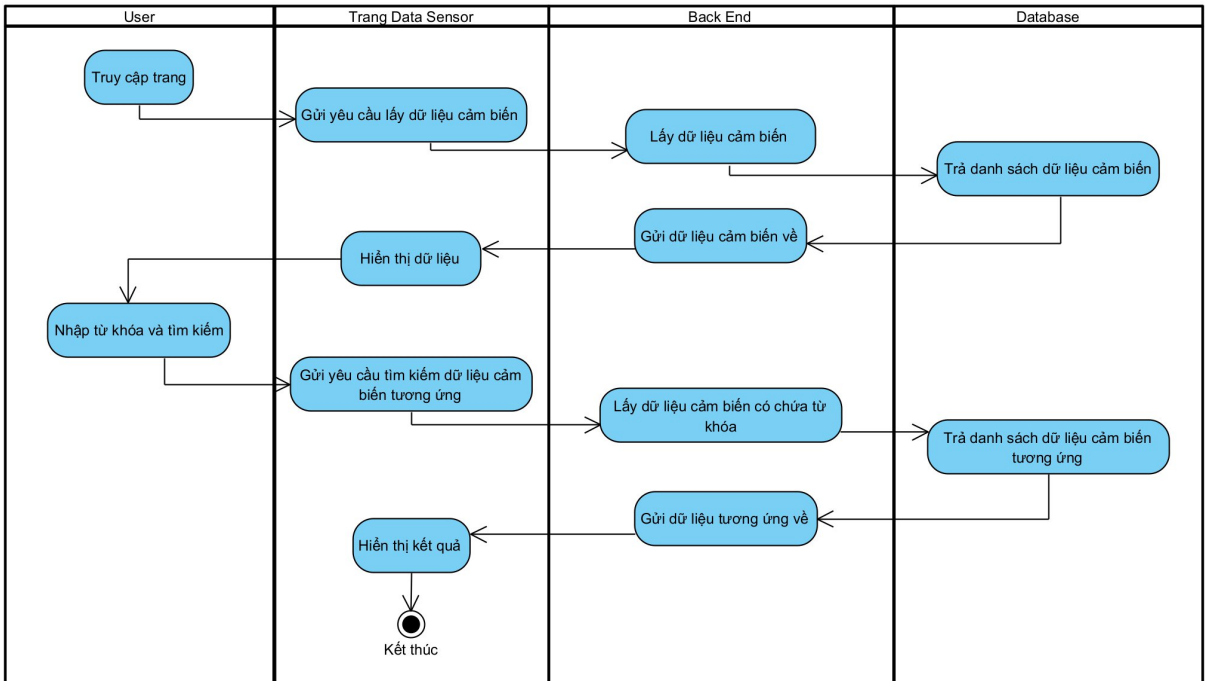
Hình 8. Sơ đồ hoạt động đọc dữ liệu cảm biến

2.6.2. Thao tác với thiết bị



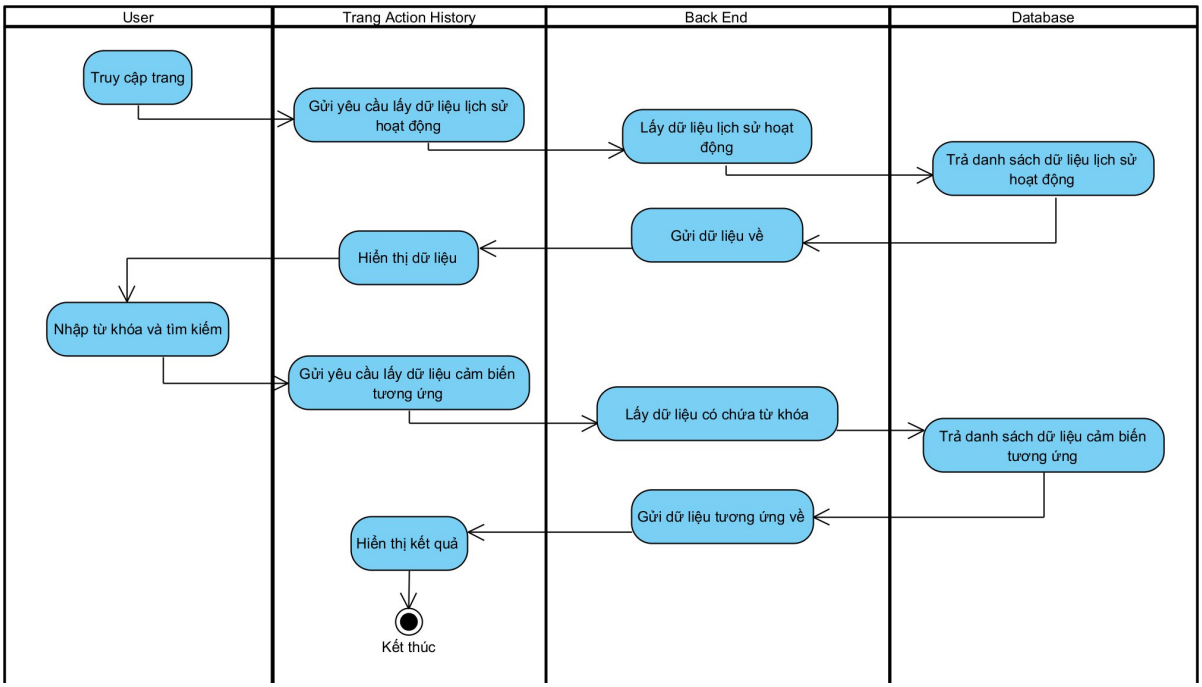
Hình 9. Sơ đồ hoạt động thao tác với thiết bị

2.6.3. Xem dữ liệu cảm biến



Hình 10. Sơ đồ hoạt động xem dữ liệu cảm biến

2.6.4. Xem lịch sử hoạt động



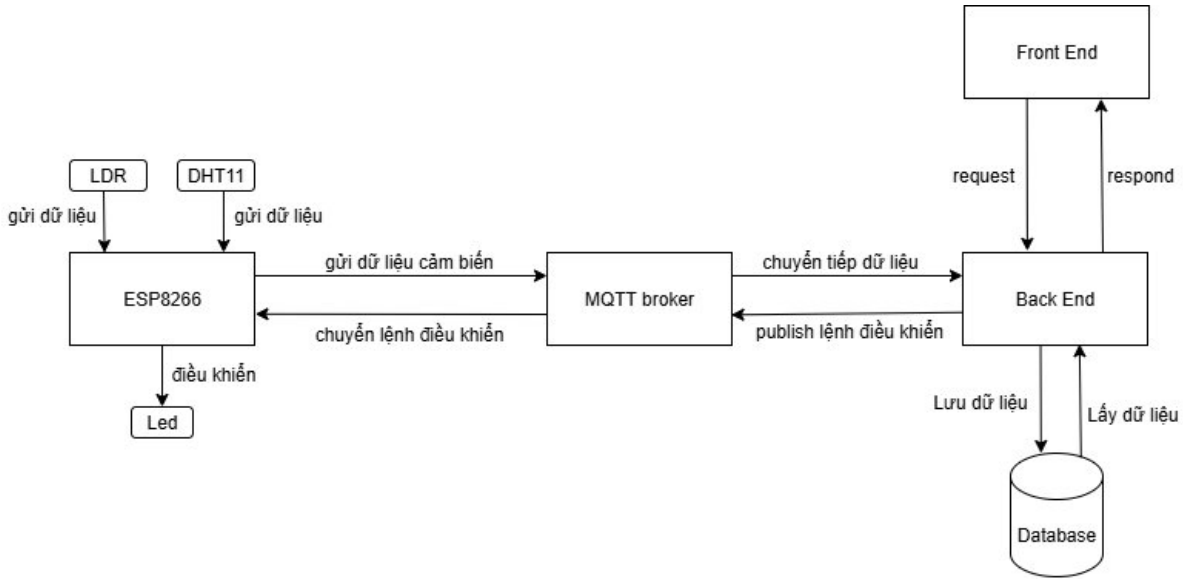
Hình 11. Sơ đồ hoạt động xem lịch sử hoạt động

2.7. Tổng kết chương 2

Chương 2 đã trình bày tổng quan về use case của hệ thống. Chương này cũng mô tả các chức năng chính của hệ thống theo từng vai trò, đồng thời thể hiện các biểu đồ tuần tự, sơ đồ lớp phân tích và thiết kế cơ sở dữ liệu, tạo cơ sở vững chắc cho việc triển khai và phát triển hệ thống.

CHƯƠNG III. CÀI ĐẶT VÀ TRIỂN KHAI

3.1. Kiến trúc tổng hệ thống



Hình 12. Kiến trúc tổng hệ thống

3.2. Môi trường triển khai

- Ngôn ngữ lập trình: React, Python

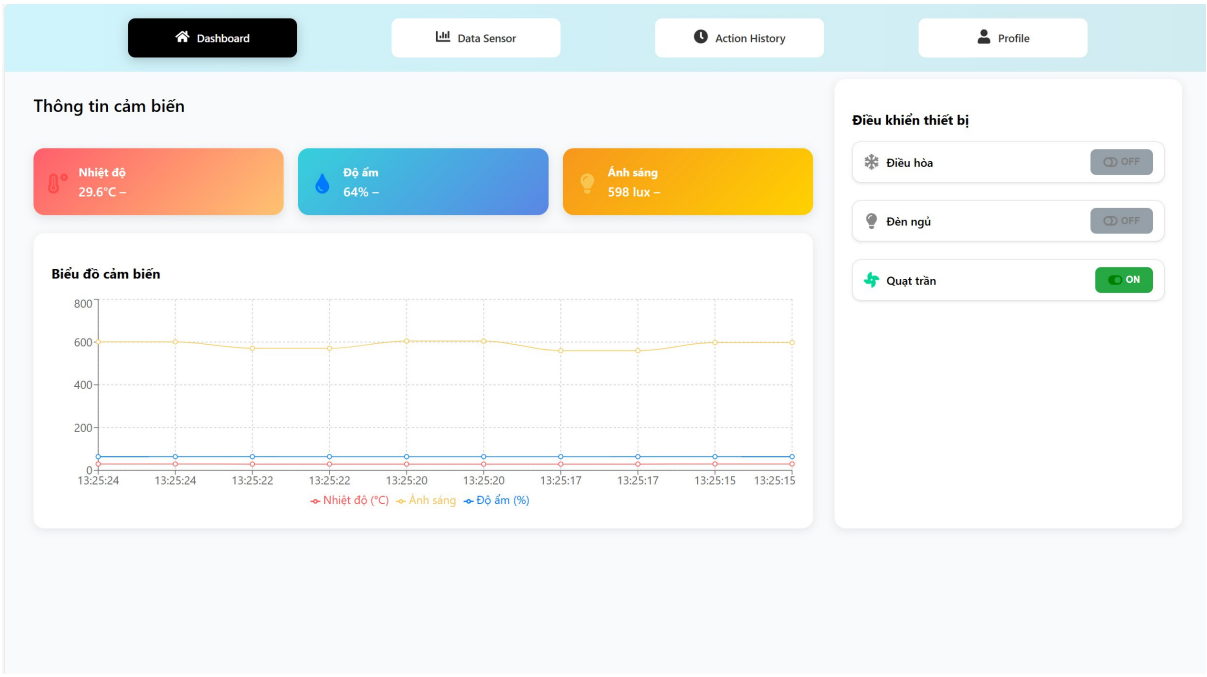
- Công cụ lập trình: Visual Studio Code

- Cơ sở dữ liệu: SQLite3

3.3. Cài đặt và triển khai

3.3.1. Giao diện người dùng

*** Trang Dashboard:**



Hình 13. Trang Dashboard

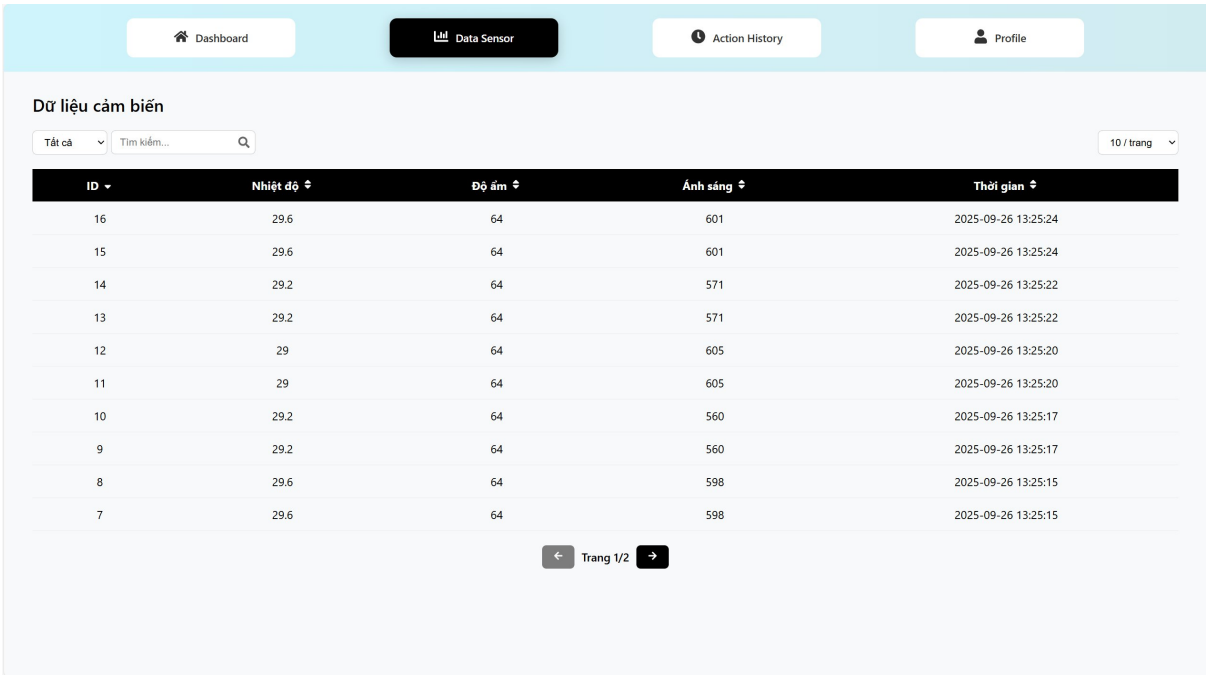
Trang Dashboard hiển thị thông tin cảm biến theo thời gian thực, bao gồm ba thông số chính: Nhiệt độ (°C), Độ ẩm (%), Ánh sáng (lux).

Các giá trị này được cập nhật liên tục thông qua kết nối MQTT giữa thiết bị IoT và máy chủ Django.

Phần biểu đồ cảm biến thể hiện dữ liệu nhiệt độ, độ ẩm và ánh sáng theo thời gian giúp người dùng dễ dàng quan sát biến động môi trường.

Bên phải là phần "Điều khiển thiết bị", cho phép người dùng bật/tắt các thiết bị như điều hòa, đèn ngủ, quạt trần.
Khi người dùng thay đổi trạng thái, ứng dụng gửi lệnh qua MQTT topic điều khiển, giúp phản hồi tức thời đến thiết bị thật.

*** Trang DataSensor:**



Hình 14. Trang Data Sensor

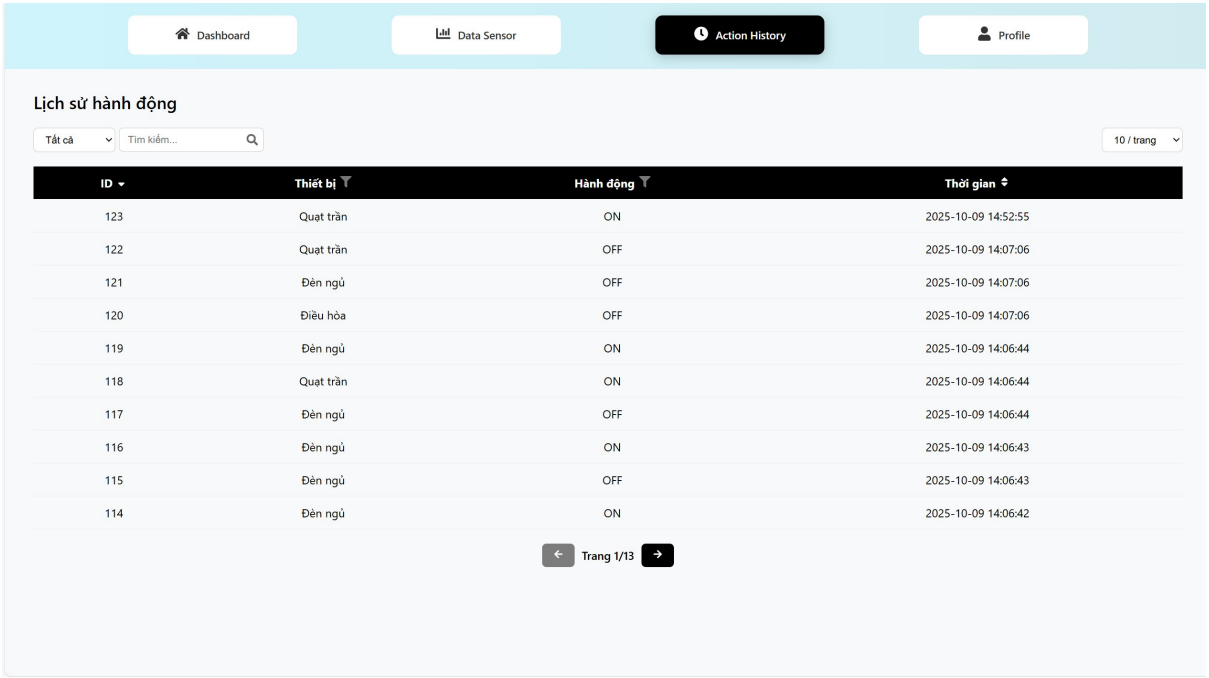
Trang Data Sensor hiển thị toàn bộ dữ liệu cảm biến được ghi nhận từ hệ thống. Các thông tin bao gồm: ID, nhiệt độ, độ ẩm, ánh sáng, và thời gian đo.

Người dùng có thể:

- Lọc, tìm kiếm, phân trang dữ liệu.

- Theo dõi chi tiết từng bản ghi.

*** Trang ActionHistory:**



Hình 15. Trang Action History

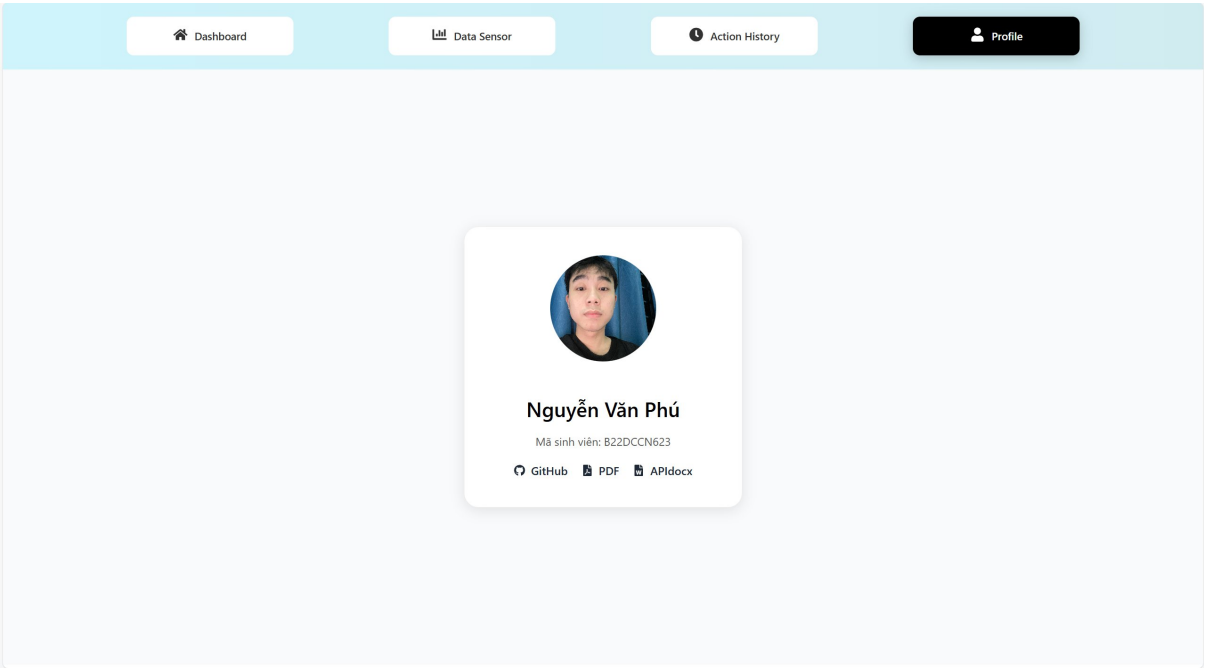
Trang Action History ghi lại toàn bộ các hành động điều khiển thiết bị của người dùng. Các thông tin bao gồm: ID hành động, Tên thiết bị, Trạng thái (ON/OFF),Thời gian thực hiện.

Người dùng có thể:

- Lọc, tìm kiếm, phân trang dữ liệu.

- Theo dõi chi tiết từng bản ghi.

*** Trang Profile:**

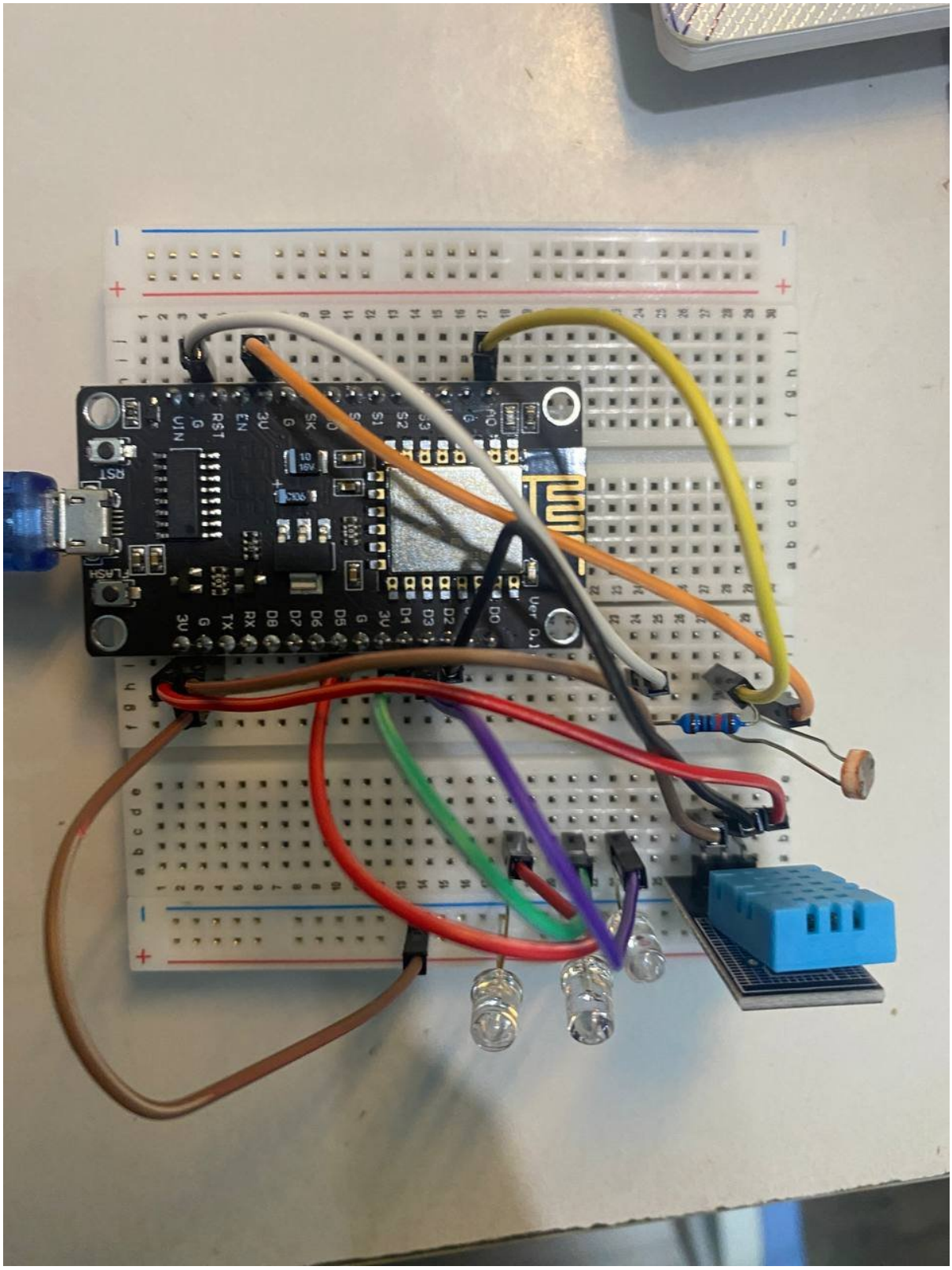


Hình 16. Trang Profile

Trang Profile hiển thị thông tin cá nhân của người dùng hiện tại, bao gồm:

- Họ tên và mã sinh viên
- Link GitHub
- File PDF báo cáo và API documentation.

3.3.2. Thiết bị



Hình 17. Thiết bị

3.3.3. API

1. Lấy dữ liệu cảm biến

Mô tả: API này trả về danh sách tất cả dữ liệu cảm biến đã ghi nhận từ các thiết bị IoT.

Phương thức: GET

Endpoint: /api/sensors/

Tham số truy vấn:

Tên	Kiểu	Bắt buộc	Mô tả
ordering	String	Không	Sắp xếp dữ liệu theo một trường cụ thể (mặc định: -id)
page	Integer	Không	Trang cần lấy (mặc định: 1)
page_size	Integer	Không	Số bản ghi trên mỗi trang (mặc định: 10)

Ví dụ phản hồi (Response 200):


```
{
    "count": 4955,
    "page": 1,
    "pages": 248,
    "page_size": 20,
    "next": "http://127.0.0.1:8000/api/sensors/?ordering=-id&page=2&page_size=20",
    "previous": null,
    "results": [
        {
            "id": 4955,
            "temperature": 30.2,
            "humidity": 77,
            "light": 567,
            "time": "2025-10-10T13:30:34.551511+07:00"
        },
        {
            "id": 4954,
            "temperature": 30.2,
            "humidity": 77,
            "light": 567,
            "time": "2025-10-10T13:30:34.551055+07:00"
        },
        {
            "id": 4953,
            "temperature": 30.2,
            "humidity": 77,
            "light": 567,
            "time": "2025-10-10T13:30:34.552028+07:00"
        },
        {
            "id": 4952,
            "temperature": 30.2,
            "humidity": 77,
            "light": 567,
            "time": "2025-10-10T13:30:34.551395+07:00"
        },
        {
            "id": 4951,
            "temperature": 30.2,
            "humidity": 77,
            "light": 567,
            "time": "2025-10-10T13:30:34.551230+07:00"
        },
```

2. Lấy lịch sử hoạt động

Mô tả: API này trả về danh sách tất cả lịch sử hoạt động bật/tắt thiết bị.

Phương thức: GET

Endpoint: /api/actions/

Tham số truy vấn hoặc dữ liệu gửi:

Tên	Kiểu	Bắt buộc	Mô tả
ordering	String	Không	Sắp xếp dữ liệu theo trường (mặc định: -id)
page	Integer	Không	Trang dữ liệu cần lấy
page_size	Integer	Không	Số bản ghi trên mỗi trang

Ví dụ phản hồi (Response 200):


```
{
    "count": 190,
    "page": 1,
    "pages": 10,
    "page_size": 20,
    "next": "http://127.0.0.1:8000/api/actions/?ordering=id&page=2&page_size=20",
    "previous": null,
    "results": [
        {
            "id": 1,
            "device": 1,
            "device_name": "Điều hòa",
            "action": "ON",
            "time": "2025-09-25T14:46:36.436188+07:00"
        },
        {
            "id": 2,
            "device": 2,
            "device_name": "Đèn ngủ",
            "action": "ON",
            "time": "2025-09-25T14:46:39.902068+07:00"
        },
        {
            "id": 3,
            "device": 3,
            "device_name": "Quạt trần",
            "action": "ON",
            "time": "2025-09-25T14:46:43.787291+07:00"
        },
        {
            "id": 4,
            "device": 1,
            "device_name": "Điều hòa",
            "action": "OFF",
            "time": "2025-09-25T14:46:52.132662+07:00"
        },
        {
            "id": 5,
            "device": 1,
            "device_name": "Điều hòa",
            "action": "ON",
            "time": "2025-09-25T14:49:27.055100+07:00"
        },
```

3. Lấy danh sách thiết bị

Mô tả: API này trả về danh sách tất cả các thiết bị IoT trong hệ thống.

Phương thức: GET

Endpoint: /api/devices/

Tham số truy vấn hoặc dữ liệu gửi:

Tên	Kiểu	Bắt buộc	Mô tả
ordering	String	Không	Sắp xếp danh sách thiết bị
page	Integer	Không	Trang dữ liệu
page_size	Integer	Không	Số bản ghi trên mỗi trang

Ví dụ phản hồi (Response 200):


```
{
    "count": 3,
    "next": null,
    "previous": null,
    "results": [
        {
            "id": 1,
            "name": "Điều hòa"
        },
        {
            "id": 2,
            "name": "Đèn ngủ"
        },
        {
            "id": 3,
            "name": "Quạt trần"
        }
    ]
}
```

4. Gửi yêu cầu bật/tắt thiết bị

Mô tả: API này gửi yêu cầu điều khiển thiết bị IoT thông qua MQTT (bật hoặc tắt).

Phương thức: POST

Endpoint: /api/devices/{id}/control/

Dữ liệu gửi:

Tên	Kiểu	Bắt buộc	Mô tả
action	String	Có	Giá trị 'ON' hoặc 'OFF' để bật/tắt thiết bị

Ví dụ phản hồi (Response 200):


```
{
    "status": "pending",
    "message": "Đã gửi lệnh ON đến thiết bị Điều hòa, chờ phản hồi..."
}
```


5. Kiểm tra trạng thái thiết bị

Mô tả: API dùng để kiểm tra trạng thái hiện tại của một thiết bị cụ thể (ON/OFF).

Phương thức: GET

Endpoint: /api/devices/{id}/status/

Ví dụ phản hồi (Response 200):


```
{
    "device": "Điều hòa",
    "status": "OFF",
    "message": "Chưa có phản hồi từ thiết bị."
}
```

6. Tìm kiếm dữ liệu cảm biến

Mô tả: API cho phép tìm kiếm dữ liệu cảm biến theo từ khóa, ngày giờ hoặc giá trị đo.

Phương thức: GET

Endpoint: /api/sensors/

Tham số truy vấn:

Tên	Kiểu	Bắt buộc	Mô tả
search	String	Không	Từ khóa tìm kiếm (ID, giá trị đo, thời gian)
ordering	String	Không	Sắp xếp dữ liệu (mặc định: -id)
page	Integer	Không	Trang dữ liệu
page_size	Integer	Không	Số bản ghi trên mỗi trang

Ví dụ phản hồi (Response 200):


```
{
    "count": 14,
    "page": 1,
    "pages": 1,
    "page_size": 20,
    "next": null,
    "previous": null,
    "results": [
        {
            "id": 16,
            "temperature": 29.6,
            "humidity": 64,
            "light": 601,
            "time": "2025-09-26T13:25:24.258779+07:00"
        },
        {
            "id": 15,
            "temperature": 29.6,
            "humidity": 64,
            "light": 601,
            "time": "2025-09-26T13:25:24.258759+07:00"
        },
        {
            "id": 14,
            "temperature": 29.2,
            "humidity": 64,
            "light": 571,
            "time": "2025-09-26T13:25:22.038950+07:00"
        },
        {
            "id": 13,
            "temperature": 29.2,
            "humidity": 64,
            "light": 571,
            "time": "2025-09-26T13:25:22.038917+07:00"
        },
        {
            "id": 12,
            "temperature": 29,
            "humidity": 64,
            "light": 605,
            "time": "2025-09-26T13:25:20.016650+07:00"
        },
```

7. Tìm kiếm lịch sử hoạt động

Mô tả: API này dùng để tìm kiếm hoặc lọc lịch sử bật/tắt thiết bị IoT theo từ khóa, tên thiết bị hoặc thời gian.

Phương thức: GET

Endpoint: /api/actions/

Tham số truy vấn hoặc dữ liệu gửi:

Tên	Kiểu	Bắt buộc	Mô tả
search	String	Không	Từ khóa tìm kiếm (VD: 'Đèn', 'ON')
ordering	String	Không	Sắp xếp dữ liệu (mặc định: -id)
page	Integer	Không	Trang dữ liệu
page_size	Integer	Không	Số bản ghi trên mỗi trang

Ví dụ phản hồi (Response 200):


```
{
    "count": 43,
    "page": 1,
    "pages": 3,
    "page_size": 20,
    "next": "http://127.0.0.1:8000/api/actions/?ordering=-id&page=2&page_size=20&search=%22%C4%90%C3%A8n%22
    "previous": null,
    "results": [
        {
            "id": 201,
            "device": 2,
            "device_name": "Đèn ngủ",
            "action": "OFF",
            "time": "2025-10-09T21:31:24.197707+07:00"
        },
        {
            "id": 198,
            "device": 2,
            "device_name": "Đèn ngủ",
            "action": "ON",
            "time": "2025-10-09T21:31:16.067540+07:00"
        },
        {
            "id": 164,
            "device": 2,
            "device_name": "Đèn ngủ",
            "action": "OFF",
            "time": "2025-10-09T20:47:19.133558+07:00"
        },
        {
            "id": 163,
            "device": 2,
            "device_name": "Đèn ngủ",
            "action": "ON",
            "time": "2025-10-09T20:47:14.710383+07:00"
        },
        {
            "id": 136,
            "device": 2,
            "device_name": "Đèn ngủ",
            "action": "OFF",
            "time": "2025-10-09T20:25:56.830330+07:00"
        },
```


3.3.4. Code

1. BackEnd

***** models.py:

- Định nghĩa các bảng dữ liệu (Model) trong cơ sở dữ liệu của hệ thống. Mỗi lớp trong file đại diện cho một bảng (table) trong SQLite.

```python
class DataSensor(models.Model):
    temperature = models.FloatField()
    humidity = models.FloatField()
    light = models.FloatField()
    time = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"Nhiệt độ: {self.temperature}, Độ ẩm: {self.humidity}, Ánh sáng:{self.light} tại {self.time}"

class Device(models.Model):
    name = models.CharField(max_length=100)

    def __str__(self):
        return self.name
class Action(models.Model):
    ACTION_CHOICES = [
        ("ON", "Bật"),
        ("OFF", "Tắt"),
    ]

    device = models.ForeignKey(Device, on_delete=models.CASCADE, related_name="actions")
    action = models.CharField(max_length=3, choices=ACTION_CHOICES)
    time = models.DateTimeField(auto_now_add=True)
        You, last week • new …
    def __str__(self):
        return f"{self.device.name} → {self.action} tại {self.time}"
```

* views.py:

- Định nghĩa các API endpoint được client gọi tới. Các class hoặc hàm trong file này thực hiện:

+ Nhận request từ frontend (GET, POST, PUT, DELETE).

+ Xử lý logic nghiệp vụ, như truy vấn dữ liệu, gửi lệnh điều khiển MQTT.

+ Trả response dạng JSON về cho frontend.

```python
class DataSensorViewSet(viewsets.ModelViewSet):
    queryset = DataSensor.objects.all().order_by("-time")
    serializer_class = DataSensorSerializer
    filter_backends = [DjangoFilterBackend, filters.SearchFilter, filters.OrderingFilter]
    filterset_class = DataSensorFilter
    search_fields = ['id', 'temperature', 'humidity', 'light', 'time']
    ordering_fields = ['id', 'temperature', 'humidity', 'light', 'time']
    pagination_class = CustomPageNumberPagination

class DeviceViewSet(viewsets.ModelViewSet):
    queryset = Device.objects.all()
    serializer_class = DeviceSerializer

    @action(detail=True, methods=["post"])
    def control(self, request, pk=None):
        try:
            device = Device.objects.get(pk=pk)
        except Device.DoesNotExist:
            return Response({"error": "Device not found"}, status=status.HTTP_404_NOT_FOUND)

        action_type = request.data.get("action", "").upper()
        if action_type not in ["ON", "OFF"]:
            return Response({"error": "Invalid action"}, status=status.HTTP_400_BAD_REQUEST)

        send_device_command(device.id, action_type)

        return Response({
            "status": "pending",
            "message": f"Đã gửi lệnh {action_type} đến thiết bị {device.name}, chờ phản hồi..."
        }, status=status.HTTP_200_OK)

    @action(detail=True, methods=["get"])
    def status(self, request, pk=None):
        try:
            device = Device.objects.get(pk=pk)
        except Device.DoesNotExist:
            return Response({"error": "Device not found"}, status=status.HTTP_404_NOT_FOUND)

        state = device_states.get(int(pk))

        if state and state["updated"]:
            return Response({
                "device": device.name,
                "status": state["status"],
                "updated": state["updated"],
            })
        else:
            return Response({
                "device": device.name,
                "status": "OFF",
                "message": "Chưa có phản hồi từ thiết bị."
            })
```

```python
class ActionViewSet(viewsets.ModelViewSet):
    queryset = Action.objects.all().order_by("-time")
    serializer_class = ActionSerializer
    filter_backends = [DjangoFilterBackend, filters.SearchFilter, filters.OrderingFilter]
    filterset_class = ActionFilter
    search_fields = ['id', 'device__name', 'action', 'time']
    ordering_fields = ['id', 'device', 'action', 'time']
    pagination_class = CustomPageNumberPagination
```

* serialzer.py:

- Định nghĩa các lớp Serializer, có nhiệm vụ:

+ Chuyển đổi dữ liệu từ Model sang JSON để trả về frontend.

+ Nhận dữ liệu từ client (POST/PUT) và xác thực trước khi lưu vào database.

```python
class DataSensorSerializer(serializers.ModelSerializer):
    You, last week | 1 author (You)
    class Meta:
        model = DataSensor
        fields = '__all__'


You, last week | 1 author (You)
class DeviceSerializer(serializers.ModelSerializer):
    You, last week | 1 author (You)
    class Meta:
        model = Device
        fields = '__all__'


You, last week | 1 author (You)
class ActionSerializer(serializers.ModelSerializer):
    device_name = serializers.CharField(source="device.name", read_only=True)

    You, last week | 1 author (You)
    class Meta:
        model = Action
        fields = ['id', 'device', 'device_name', 'action', 'time']
```

* urls.py:

- Định nghĩa các đường dẫn (URL) cho toàn bộ API trong hệ thống Django.

```
router = DefaultRouter()
router.register(r"sensors", DataSensorViewSet, basename="sensor")
router.register(r"actions", ActionViewSet, basename="action")
router.register(r"devices", DeviceViewSet, basename="device")

urlpatterns = [
    path("", include(router.urls)),
]
```

* mqtt_client.py:

- Kết nối giữa server Django và broker MQTT (ví dụ: Mosquitto).
Nó sử dụng thư viện paho-mqtt để:

+ Kết nối đến broker (client.connect())

+ Đăng ký (subscribe) topic nhận dữ liệu cảm biến từ thiết bị.

+ Gửi lệnh điều khiển (publish) tới thiết bị IoT khi người dùng bật/tắt trên giao diện.

+ Xử lý sự kiện nhận tin (on_message) và lưu dữ liệu vào database.

Nhận dữ liệu cảm biến:

```
def on_message(client, userdata, msg):

    from .models import DataSensor, Action, Device

    global last_sensor_time

    topic = msg.topic
    payload = msg.payload.decode(errors="ignore").strip()
    print(f"[MQTT] {topic} → {payload}")

    # DỮ LIỆU CẢM BIẾN
    if topic == "sensor/data":
        try:
            data = json.loads(payload)
            now = time.time()

            # Lưu mỗi 2 giây 1 lần
            if now - last_sensor_time >= SENSOR_SAVE_INTERVAL:
                DataSensor.objects.create(
                    temperature=data.get("temperature"),
                    humidity=data.get("humidity"),
                    light=data.get("light"),
                )
                last_sensor_time = now
                print(f"Saved sensor: {data}")
            else:
                print("Skipped sensor (debounce)")

        except Exception as e:
            print("Error saving sensor:", e)
        return
```

Trả trạng thái thiết bị:

```
    # XÁC NHẬN THIẾT BỊ
    if topic.startswith("device/confirm/"):
        try:
            device_id = int(topic.split("/")[2])
            current_time = time.time()

            # Chống spam confirm
            if current_time - last_confirm_time[device_id] < DEBOUNCE_SECONDS:
                print(f"Ignored duplicate confirm for device {device_id}")
                return
            last_confirm_time[device_id] = current_time

            device = Device.objects.filter(pk=device_id).first()
            if not device:
                print(f"Device id={device_id} not found")
                return

            payload_upper = payload.upper()

            # Nếu ESP trả "OK" → dùng trạng thái từ lệnh gần nhất
            if payload_upper == "OK":
                state = last_command.get(device_id, "UNKNOWN")
            elif payload_upper in ["ON", "OFF"]:
                state = payload_upper
            else:
                print(f"{device.name} phản hồi không hợp lệ: {payload}")
                return

            # Cập nhật cache trạng thái
            device_states[device_id] = {
                "status": state,
                "updated": time.strftime("%Y-%m-%d %H:%M:%S"),
            }

            # Ghi vào DB
            Action.objects.create(device=device, action=state)
            print(f"{device.name} đã {state} (MQTT confirm)")
```

Gửi lệnh bật/tắt thiết bị:

```
# HÀM GỬI LỆNH
def send_device_command(device_id, action):
    global mqtt_client
    if not mqtt_client:
        mqtt_client = start_mqtt()

    topic = f"device/control/{device_id}"
    mqtt_client.publish(topic, action.upper())
    last_command[device_id] = action.upper()
    print(f"Gửi lệnh {action.upper()} → {topic}")
```

2. FrontEnd

* App.js:

Là thành phần gốc của ứng dụng React, định nghĩa các tuyến đường (route) giữa các trang.

```
export default function App() {
    return (
        <BrowserRouter>
        <Header />
        <Routes>
            <Route path="/" element={<Navigate to="/dashboard" replace />} />
            <Route path="/dashboard" element={<Dashboard />} />
            <Route path="/datasensor" element={<DataSensor />} />
            <Route path="/action" element={<Action />} />
            <Route path="/profile" element={<Profile />} />
        </Routes>
        </BrowserRouter>
    );
}
```

* Header.jsx:

Hiển thị thanh menu trên đầu trang, giúp người dùng chuyển đổi giữa các chức năng: Dashboard, Data Sensor, Action History, và Profile.

```
export default function Header() {
    const navigate = useNavigate();
    const location = useLocation();

    const tabs = [
        { key: "dashboard", label: "Dashboard", icon: <FaHome />, path: "/dashboard" },
        { key: "datasensor", label: "Data Sensor", icon: <FaChartBar />, path: "/datasensor" },
        { key: "action", label: "Action History", icon: <FaClock />, path: "/action" },
        { key: "profile", label: "Profile", icon: <FaUser />, path: "/profile" },
    ];

    // Xác định tab hiện tại dựa vào URL
    const currentTab = location.pathname.split("/")[1] || "dashboard";

    return (
        <div className="header">
        <div className="header-container">
            {tabs.map((tab) => (
            <div
                key={tab.key}
                className={`tab-button ${
                currentTab === tab.key ? "active" : ""
                }`}
                onClick={() => navigate(tab.path)}
            >
                <div className="tab-icon">{tab.icon}</div>
                <span>{tab.label}</span>
            </div>
            ))}
        </div>
        </div>
    );
}
```

3.4. Tổng kết chương 3

Chương 3 đã cài đặt và phát triển hệ thống website IoT dùng để giám sát dữ liệu cảm biến và điều khiển thiết bị. Hệ thống đã có chức năng cơ bản như hiển thị dữ liệu cảm biến, lưu trữ lịch sử hoạt động, điều khiển thiết bị bật/tắt.Tuy nhiên vẫn còn một vài chức năng chưa được tích hợp, đây cũng là tiền đề để phát triển trong tương lai.

CHƯƠNG IV. KẾT LUẬN

4.1. Kết luận

Hệ thống IoT Dashboard đã được xây dựng thành công, cho phép theo dõi dữ liệu cảm biến và giả lập điều khiển thiết bị thông qua giao diện web. Hệ thống đã thực hiện được các chức năng chính gồm:

- Thu thập dữ liệu từ các cảm biến (nhiệt độ, độ ẩm, ánh sáng) và hiển thị theo thời gian thực.

- Hiển thị biểu đồ lịch sử các thông số.

- Điều khiển thiết bị ON/OFF thông qua dashboard.

- Lưu trữ lịch sử hành động thiết bị để tra cứu.

Giao diện người dùng được thiết kế trực quan, thân thiện, dễ sử dụng, đồng thời hệ thống có khả năng mở rộng và bảo trì nhờ kiến trúc Back End – Front End – MQTT Broker – Database rõ ràng.

4.2. Hạn chế cần khắc phục

Mặc dù hệ thống đã hoàn thiện các chức năng cơ bản, vẫn còn một số hạn chế cần cải thiện trong tương lai:

- Chưa có cơ chế cảnh báo tự động khi dữ liệu cảm biến vượt ngưỡng an toàn.

- Chưa có giao diện cho thiết bị di động.

4.3. Hướng phát triển

Trong thời gian tới, hệ thống IoT Dashboard có thể được mở rộng và phát triển theo các hướng sau:

- Tích hợp cảnh báo thông minh, gửi thông báo tới người dùng khi thông số cảm biến vượt ngưỡng.

- Cải tiến giao diện responsive, hỗ trợ tốt trên thiết bị di động và tablet.

- Phát triển ứng dụng di động (mobile app) để theo dõi và điều khiển thiết bị mọi lúc mọi nơi.

- Mở rộng tích hợp với các thiết bị IoT khác và các cảm biến mới để tăng khả năng giám sát môi trường.

TÀI LIỆU THAM KHẢO

- React: https://vi.legacy.reactjs.org/docs/getting-started.html

- Django Rest Framework: https://www.django-rest-framework.org/