

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA CÔNG NGHỆ THÔNG TIN 1



BÁO CÁO BÀI TẬP LỚN

Môn học: IoT và ứng dụng

Giảng viên hướng dẫn: Nguyễn Quốc Uy

Họ và tên sinh viên: Lê Trung Đức

Mã sinh viên: B21DCCN243

Số điện thoại: 0936068537

Hà Nội, 2024

MỤC LỤC

I. Giới thiệu	4
1.1. IoT (Internet of Things - Mạng lưới vạn vật kết nối)	4
1.1.1. Khái niệm	4
1.1.2. Các thành phần của IoT	4
1.1.3. Ứng dụng	4
1.2. Mô tả và mục đích của đề tài	4
1.2.1. Mô tả đề tài	4
1.2.2. Mục đích của đề tài	5
1.3. Thiết bị sử dụng	5
1.3.1. Node MCU ESP 8266	5
1.3.2. Cảm biến nhiệt độ - độ ẩm	7
1.3.3. Cảm biến cường độ ánh sáng quang trở	8
1.3.4. Board Test MB-102 16.5x5.5	8
1.3.5. Điện trở vạch 1/4W sai số 5% 250V 1R-10M	9
1.3.6. Dây nối chân các thiết bị	9
1.3.7. Led 5mm, 7 màu 2 chân (1,5-3V)	10
1.4. Công nghệ sử dụng	10
1.4.1. Trình biên dịch Arduino IDE	10
1.4.2. Frontend: Angular	10
1.4.3. Backend: Spring Boot	11
1.4.4. Database: MySQL	11
1.4.5. Mosquitto	12
II. Giao diện	13
2.1. Giao diện website	13
2.2. Giao diện API, API Docs	15

2.3. Giao diện thiết bị.....	16
III. Thiết kế chi tiết	17
3.1. Thiết kế hệ thống.....	17
3.2. Sequence Diagram	17
IV. Code.....	18
4.1. Arduino code	18
4.2. Backend code	24
4.3. Frontend Code.....	26
V. Kết quả thu được.....	29
5.1. Tổng quan	29
5.2. Từ DHT11 và cảm biến ánh sáng	29
5.3. Khi người dùng điều khiển thiết bị	29
VI. Tài liệu tham khảo	30

I. Giới thiệu

1.1. IoT (Internet of Things - Mạng lưới vạn vật kết nối)

1.1.1. Khái niệm

Mạng lưới vạn vật kết nối (IoT) là một hệ thống mạng lưới trong đó các thiết bị và đối tượng thông minh có khả năng giao tiếp và trao đổi dữ liệu với nhau thông qua internet. IoT cho phép các thiết bị không chỉ kết nối với mạng internet, mà còn có khả năng truyền nhận thông tin và tương tác với nhau mà không cần sự can thiệp của con người.

1.1.2. Các thành phần của IoT

- Thiết bị IoT: Đây là các thiết bị thông minh được trang bị cảm biến, các công nghệ kết nối và khả năng thu thập, gửi và nhận dữ liệu.
- Mạng: Đảm bảo việc truyền dữ liệu giữa các thiết bị IoT và hệ thống xử lý dữ liệu.
- Hệ thống xử lý dữ liệu: Bao gồm các máy chủ và hệ thống phần mềm để lưu trữ, xử lý và phân tích dữ liệu từ các thiết bị IoT.
- Ứng dụng và dịch vụ: Cung cấp các ứng dụng và dịch vụ dựa trên dữ liệu từ các thiết bị IoT, nhằm giúp cải thiện cuộc sống và công việc của con người.

1.1.3. Ứng dụng

IoT có thể được áp dụng trong nhiều lĩnh vực khác nhau, bao gồm:

- Nhà thông minh: Điều khiển ánh sáng, nhiệt độ, an ninh, thiết bị gia đình thông qua điện thoại di động.
- Công nghiệp: Giám sát và quản lý các quy trình sản xuất, dự báo bảo trì thiết bị, tăng cường an toàn lao động.
- Giao thông: Điều khiển giao thông thông minh, quản lý đỗ xe, theo dõi vận chuyển hàng hóa.
- Y tế: Theo dõi sức khỏe, quản lý thuốc, giám sát bệnh nhân từ xa.
- Năng lượng: Theo dõi và quản lý tiêu thụ năng lượng, tối ưu hóa sử dụng tài nguyên.

1.2. Mô tả và mục đích của đề tài

1.2.1. Mô tả đề tài

Đề tài này tập trung vào việc xây dựng một hệ thống IoT để giám sát và điều khiển các yếu tố môi trường như nhiệt độ, độ ẩm và các thiết bị điện (điều hòa, đèn, quạt) thông qua một ứng dụng website. Hệ thống được phát triển với backend sử dụng Java Spring Boot và cơ sở dữ liệu MySQL để lưu trữ và quản lý dữ liệu. Các cảm biến sẽ thu thập dữ

liệu nhiệt độ và độ ẩm trong thời gian thực, và người dùng có thể điều khiển các thiết bị điện từ xa thông qua giao diện website.

1.2.2. Mục đích của đề tài

Mục đích của đề tài là xây dựng một hệ thống quản lý và điều khiển thông minh với các mục tiêu chính như sau:

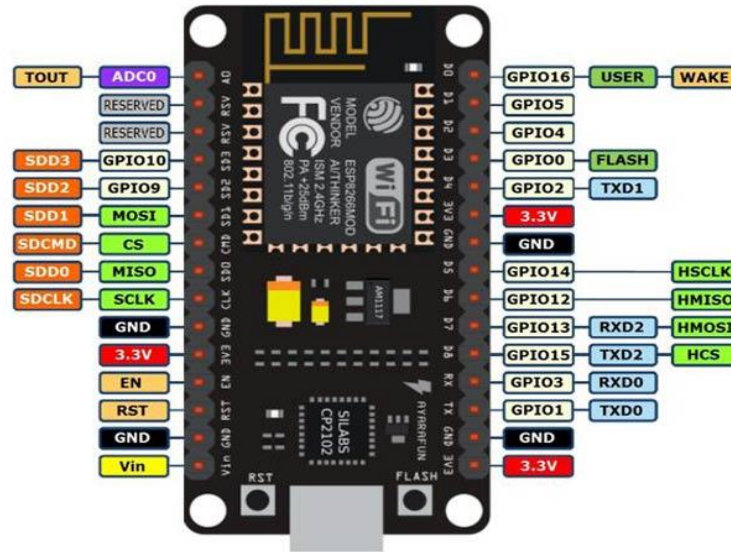
- Giám sát và cảnh báo:
 - Thu thập và hiển thị dữ liệu nhiệt độ, độ ẩm từ các cảm biến một cách trực quan trên website.
 - Cảnh báo người dùng khi các chỉ số môi trường vượt quá ngưỡng an toàn (ví dụ: nhiệt độ quá cao hoặc độ ẩm quá thấp).
- Điều khiển thiết bị từ xa:
 - Cho phép người dùng bật/tắt và điều chỉnh các thiết bị như điều hòa, đèn, quạt thông qua giao diện website.
- Quản lý và lưu trữ dữ liệu:
 - Lưu trữ dữ liệu giám sát (nhiệt độ, độ ẩm) và lịch sử hoạt động của các thiết bị vào cơ sở dữ liệu MySQL.
 - Cung cấp các báo cáo thống kê, biểu đồ phân tích dữ liệu giúp người dùng nắm bắt tình trạng hoạt động của hệ thống.

1.3. Thiết bị sử dụng

1.3.1. Node MCU ESP 8266

- Thông số kỹ thuật:
 - Hỗ trợ chuẩn 802.11 b/g/n.
 - Wi-Fi 2.4 GHz, hỗ trợ WPA/WPA2.
 - Chuẩn điện áp hoạt động: 3.3V.
 - Chuẩn giao tiếp nối tiếp UART với tốc độ Baud lên đến 115200.
 - Có 3 chế độ hoạt động: Client, Access Point, Both Client and Access Point.
 - Hỗ trợ các chuẩn bảo mật như: OPEN, WEP, WPA_PSK, WPA2_PSK, WPA WPA2 PSK.

- Hỗ trợ cả 2 giao tiếp TCP và UDP.



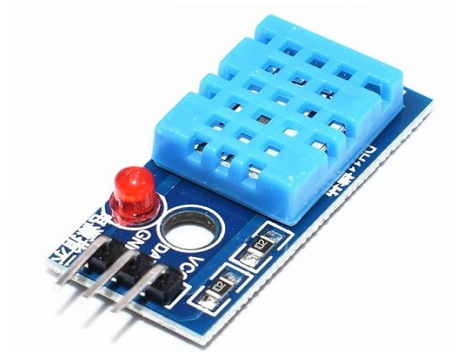
Hình 1: MCU ESP8266

- Các chân ESP 8266:

- VIN: Chân nguồn cung cấp cho board, có thể được kết nối với nguồn cung cấp từ 5V đến 12V.
- GND: Chân mẫu điện âm.
- 3V3: Chân nguồn cung cấp 3.3V.
- RST: Chân khởi động lại board.
- AO: Chân đo lường tín hiệu Analog-to-Digital Converter (ADC), được sử dụng để đo các tín hiệu điện áp tương tự.
- DO - D8: Các chân kết nối đầu vào/số GPIO từ 0 đến 8, được sử dụng để điều khiển các thiết bị hoặc đọc các tín hiệu từ các cảm biến.
- TXD: Chân truyền UART, được sử dụng để truyền dữ liệu từ board đến một thiết bị khác.
- RXD: Chân nhận UART, được sử dụng để nhận dữ liệu từ một thiết bị khác.
- CH_PD: Chân đưa board ra khỏi chế độ nghỉ, và bắt đầu chạy.
- USB: Cổng USB được sử dụng để kết nối board với máy tính để lập trình và cung cấp nguồn.

1.3.2. Cảm biến nhiệt độ - độ ẩm

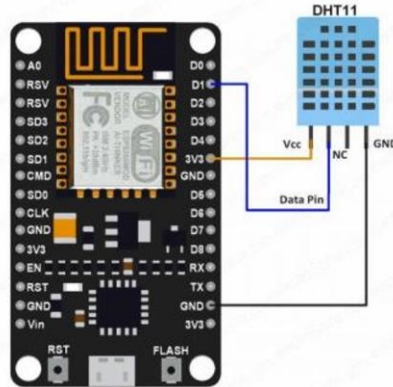
- Thông số kỹ thuật:
 - Điện áp hoạt động: 3V - 5V DC
 - Dòng điện tiêu thụ: 2.5mA
 - Phạm vi cảm biến độ ẩm: 20% - 90% RH, sai số $\pm 5\%RH$
 - Phạm vi cảm biến nhiệt độ: $0^{\circ}C \sim 50^{\circ}C$, sai số $\pm 2^{\circ}C$
 - Tần số lấy mẫu tối đa: 1Hz (1 giây 1 lần)
 - Kích thước: 23 * 12 * 5 mm



Hình 2: Cảm biến DHT11

- Sơ đồ chân DHT11:

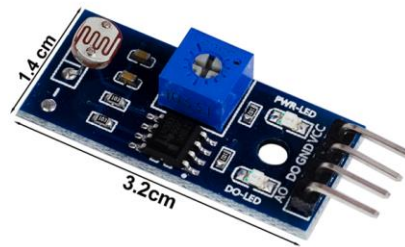
Số chân	Tên chân	Mô tả
1	VCC	Nguồn 3.3V đến 5V
2	Data	Đầu ra nhiệt độ độ ẩm thông qua dữ liệu nối tiếp
3	NC	Không có kết nối và do đó không sử dụng
4	Ground	Nối đất



Hình 3: Sơ đồ chân nối DHT11

1.3.3. Cảm biến cường độ ánh sáng quang trở

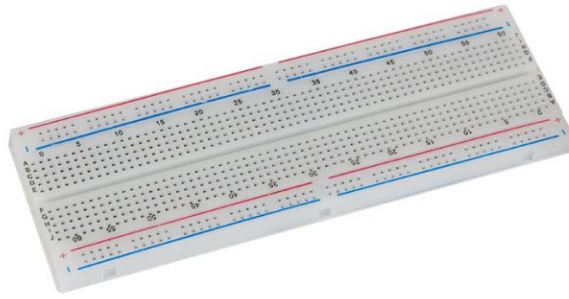
- Thông số kỹ thuật
 - Điện áp hoạt động: 3.3V – 5V
 - Kết nối 4 chân với 2 chân cấp nguồn (VCC và GND) và 2 chân tín hiệu ngõ ra (AO và DO).
 - Hỗ trợ cả 2 dạng tín hiệu ra Analog và TTL. Ngõ ra Analog 0– 5V tỷ lệ thuận với cường độ ánh sáng, ngõ TTL tích cực mức thấp.
 - Độ nhạy cao với ánh sáng được tùy chỉnh bằng biến trở.
 - Kích thước: 32 x 14mm



Hình 4: Cảm biến ánh sáng

1.3.4. Board Test MB-102 16.5x5.5

Dùng để test mạch trước khi làm mạch in hoàn chỉnh. Giúp bạn dễ dàng thực hiện các mạch điện thực tế trước khi hàn trực tiếp linh kiện lên board mạch đồng



Hình 5: Board test

1.3.5. Điện trở vạch 1/4W sai số 5% 250V 1R-10M



Hình 6: Điện trở

1.3.6. Dây nối chân các thiết bị



Hình 7: Dây nối

1.3.7. Led 5mm, 7 màu 2 chân (1,5-3V)



Hình 8: Led

1.4. Công nghệ sử dụng

1.4.1. Trình biên dịch Arduino IDE

- Arduino IDE là một môi trường phát triển tích hợp được sử dụng để viết và nạp mã cho các vi điều khiển Arduino. Đây là công cụ chính để lập trình các thiết bị phần cứng, cho phép giao tiếp với các cảm biến (như nhiệt độ, độ ẩm) và thiết bị điều khiển (như quạt, đèn, điều hòa).
- Tính năng chính:
 - Hỗ trợ các thư viện và hàm đặc biệt dành riêng cho lập trình IoT.
 - Dễ dàng nạp mã cho các bo mạch Arduino thông qua cổng USB.
 - Giao diện đơn giản, dễ sử dụng cho việc lập trình và kiểm tra mã nguồn.

1.4.2. Frontend: Angular

- Angular là một framework mạnh mẽ của Google để phát triển các ứng dụng web động. Angular giúp xây dựng giao diện người dùng trực quan và phản hồi nhanh với các chức năng giám sát và điều khiển thiết bị.
- Tính năng chính:
 - Component-based: Tách biệt các phần giao diện thành các component riêng biệt, dễ bảo trì và mở rộng.
 - Reactive Forms: Hỗ trợ tạo các biểu mẫu linh hoạt, dễ dàng quản lý dữ liệu người dùng.
 - Two-way Data Binding: Đồng bộ hóa dữ liệu giữa giao diện và logic ứng dụng một cách tự động.

- HTTP Client: Tích hợp các phương thức giao tiếp với backend qua RESTful API, giúp lấy và cập nhật dữ liệu từ server.

1.4.3. Backend: Spring Boot

- Spring Boot là một framework mạnh mẽ để phát triển các ứng dụng Java, đặc biệt là các ứng dụng web và RESTful API. Spring Boot cung cấp môi trường phát triển nhanh chóng và đơn giản với các tính năng bảo mật và quản lý dữ liệu hiệu quả.
- Tính năng chính:
 - RESTful API: Xây dựng các dịch vụ API để giao tiếp với frontend, cung cấp dữ liệu từ cơ sở dữ liệu và thực hiện các chức năng điều khiển thiết bị.
 - Spring Security: Cung cấp bảo mật cho ứng dụng với các chức năng xác thực và phân quyền người dùng.
 - Spring Data JPA: Tương tác với cơ sở dữ liệu một cách dễ dàng, hỗ trợ các thao tác CRUD (Create, Read, Update, Delete).
 - Dependency Injection: Quản lý các thành phần và phụ thuộc trong ứng dụng một cách hiệu quả.

1.4.4. Database: MySQL

- MySQL là một hệ quản trị cơ sở dữ liệu quan hệ phổ biến, được sử dụng để lưu trữ và quản lý thông tin về nhiệt độ, độ ẩm và trạng thái của các thiết bị. MySQL cung cấp khả năng truy vấn và quản lý dữ liệu mạnh mẽ, phù hợp cho các ứng dụng có quy mô vừa và lớn.
- Tính năng chính:
 - Khả năng lưu trữ: Lưu trữ thông tin lịch sử hoạt động của các thiết bị và dữ liệu cảm biến.
 - Quan hệ dữ liệu: Tổ chức dữ liệu theo mô hình quan hệ, đảm bảo tính nhất quán và toàn vẹn dữ liệu.
 - Truy vấn nhanh chóng: Sử dụng các chỉ mục và khóa ngoại để tối ưu hóa hiệu suất truy vấn dữ liệu.
 - Bảo mật dữ liệu: Cung cấp các tính năng quản lý người dùng và phân quyền truy cập để đảm bảo an toàn dữ liệu.

1.4.5. Mosquitto

- Mosquitto là một MQTT broker mã nguồn mở, cho phép giao tiếp giữa các thiết bị IoT với nhau và với server. Nó đảm bảo việc truyền tải thông tin nhanh chóng và ổn định, giúp hệ thống giám sát và điều khiển hoạt động trơn tru.
- Tính năng chính:
 - Giao tiếp MQTT: Hỗ trợ giao thức MQTT để truyền tải thông điệp giữa các thiết bị IoT và ứng dụng server.
 - Chất lượng dịch vụ (QoS): Cung cấp nhiều mức độ QoS để đảm bảo thông điệp được truyền tải một cách tin cậy.
 - Nhẹ và hiệu quả: Tiêu thụ ít tài nguyên hệ thống, phù hợp cho các thiết bị IoT có tài nguyên hạn chế.
 - Bảo mật: Hỗ trợ các cơ chế bảo mật như TLS và xác thực người dùng để đảm bảo an toàn trong việc truyền thông điệp.

II. Giao diện

2.1. Giao diện website



Hình 9: Giao diện Dashboard

- Trang Dashboard hiển thị nhiệt độ, độ ẩm, ánh sáng mà cảm biến đo được, cùng với biểu đồ thống kê 3 giá trị đó theo thời gian. Để điều khiển bật tắt các thiết bị có thể nhấn On/Off

House IoT System					Dashboard	Data Sensor	Action History	Profile
Search by date...								
Show 10 value								
ID ↑	Device	Action	Time					
1	Led-1	On	2024-09-07 16:30:12					
2	Led-2	On	2024-09-07 16:43:58					
3	Led-2	Off	2024-09-07 16:44:05					
4	Fan	On	2024-09-07 16:44:15					
5	Fan	Off	2024-09-07 16:44:21					
6	Led-1	Off	2024-09-07 16:44:26					
7	Led-1	On	2024-09-11 15:45:47					
8	Fan	On	2024-09-11 16:16:47					
9	Fan	On	2024-09-11 16:27:42					
10	Fan	Off	2024-09-11 19:37:29					

Hình 10: Giao diện màn Action History

- Trang Action History hiển thị hoạt động bật tắt của các thiết bị theo thời gian.

House IoT System

Dashboard

Data Sensor

Action History

Profile

Search by:

Date

▼

Search...

Q

Show

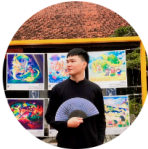
10 value

▼

ID ↑	Temperature	Humidity	Light	Time
1	25.5°C	60%	300 lux	2024-09-06 10:15:00
2	25.5°C	60%	300 lux	2024-09-04 10:15:00
3	25.5°C	60%	300 lux	2024-09-06 10:15:00
4	25.5°C	60%	300 lux	2024-09-06 10:15:00
5	25.5°C	60%	300 lux	2024-09-04 10:15:00
6	25.5°C	60%	300 lux	2022-01-01 10:15:00
7	25.5°C	60%	300 lux	2024-09-06 14:37:06
8	1°C	1%	1 lux	2024-09-07 16:29:13
9	20°C	60%	300 lux	2024-09-11 16:38:32
10	20°C	60%	200 lux	2024-09-11 16:38:47

Hình 11: Giao diện màn Data Sensor

- Trang Data Sensor hiển thị các giá trị nhiệt độ, độ ẩm, ánh sáng theo thời gian

House IoT System					Dashboard	Data Sensor	Action History	Profile
<div><p>Họ và tên: Lê Trung Đức Lớp: D21CNPM2 Mã SV: B21DCCN243 SĐT: 0936068537 Email: letrungducabcxyz@gmail.com File báo cáo Link GitHub API Documentation</p></div>								

Hình 12: Giao diện màn Profile

- Trang Profile hiển thị thông tin sinh viên và các liên kết theo yêu cầu

2.2. Giao diện API, API Docs

The screenshot displays an API documentation interface. At the top, there's a section for 'lot' with a placeholder 'Add collection description...'. Below this, the 'GET device' endpoint is highlighted in green, with a URL 'localhost:8080/api/device-action' and a link 'Open request→'. A placeholder 'Add request description...' is also present. Further down, the 'POST pub-device' endpoint is highlighted in orange, with a URL 'http://localhost:8080/api/publish' and a link 'Open request→'. On the right side, a 'JUMP TO' sidebar lists 'Introduction', 'GET device', 'POST pub-device', and 'GET enviroment'. Below the main content, a 'Body raw (json)' section shows a JSON object:

```
{  "topic": "device/action",  "message": "Fan,On"}
```

. The 'GET enviroment' endpoint is also visible at the bottom, with a URL 'localhost:8080/api/environmental-data' and a link 'Open request→'. A placeholder 'Add request description...' is at the very bottom.

lot

Add collection description...

GET device [Open request→](#)

localhost:8080/api/device-action

Add request description...

POST pub-device [Open request→](#)

http://localhost:8080/api/publish

Body raw (json)

```
{  "topic": "device/action",  "message": "Fan,On"}
```

GET enviroment [Open request→](#)

localhost:8080/api/environmental-data

Add request description...

JUMP TO

- Introduction
- [GET device](#)
- [POST pub-device](#)
- [GET enviroment](#)

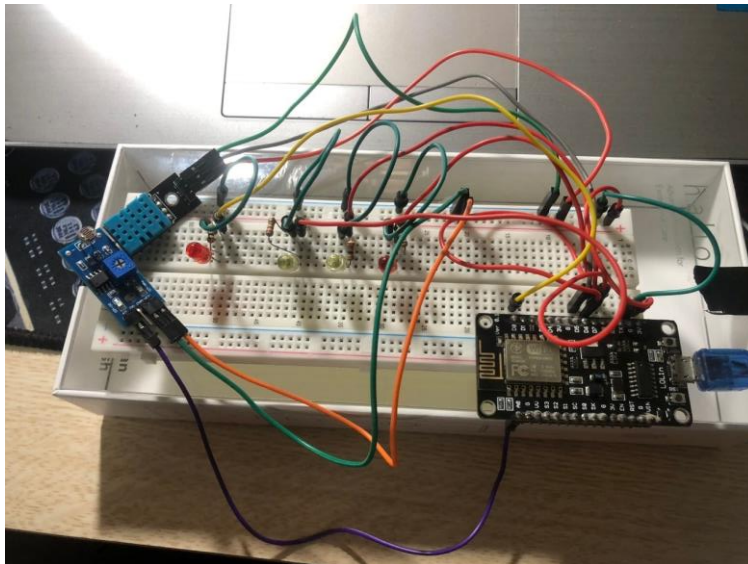
Hình 13: Giao diện API

```
← ↻ 🔒 https://api.postman.com/collections/34374853-5e54e70b-4b35-4ef1-af74-ec614566a8e1

1 {
2   "collection": {
3     "info": {
4       "postman_id": "5e54e70b-4b35-4ef1-af74-ec614566a8e1",
5       "name": "IoT",
6       "schema": "https://schema.getpostman.com/json/collection/v2.1.0/collection.json",
7       "updatedAt": "2024-09-19T03:57:22.000Z",
8       "createdAt": "2024-07-02T09:03:56.000Z",
9       "lastUpdatedBy": "34374853",
10      "uid": "34374853-5e54e70b-4b35-4ef1-af74-ec614566a8e1"
11    },
12    "item": [
13      {
14        "name": "device",
15        "id": "b6455e93-53b4-4b9c-886d-383065ffc691",
16        "protocolProfileBehavior": {
17          "disableBodyPruning": true
18        },
19        "request": {
20          "method": "GET",
21          "header": [],
22          "url": {
23            "raw": "localhost:8080/api/device-action",
24            "host": [
25              "localhost"
26            ],
27            "port": "8080",
28            "path": [
29              "api",
30              "device-action"
31            ]
32          },
33          "response": [],
34          "uid": "34374853-b6455e93-53b4-4b9c-886d-383065ffc691"
35        },
36      },
37      {
38        "name": "pub-device",
39        "id": "1efcf68d-afe7-409e-af27-c2f03466e45a",
40        "protocolProfileBehavior": {
41          "disableBodyPruning": true
42        },
43        "request": {
44          "method": "POST",
45          "header": [],
46          "body": {
47            "mode": "raw",
48            "raw": "{\n  \"topic\": \"device/action\", \n  \"message\": \"Fan,On\" \n}",
49            "options": {
50              "raw": {
51                "language": "json"
52              }
53            },
54            "url": {
55              "raw": "http://localhost:8080/api/publish",
56              "protocol": "http",
57              "host": [
58                "localhost"
59              ],
60              "port": "8080",
61              "path": [
62                "api",
63                "publish"
64              ]
65            }
66          },
67        }
68      }
69    ]
70  }
71 }
```

Hình 14: Giao diện API Docs

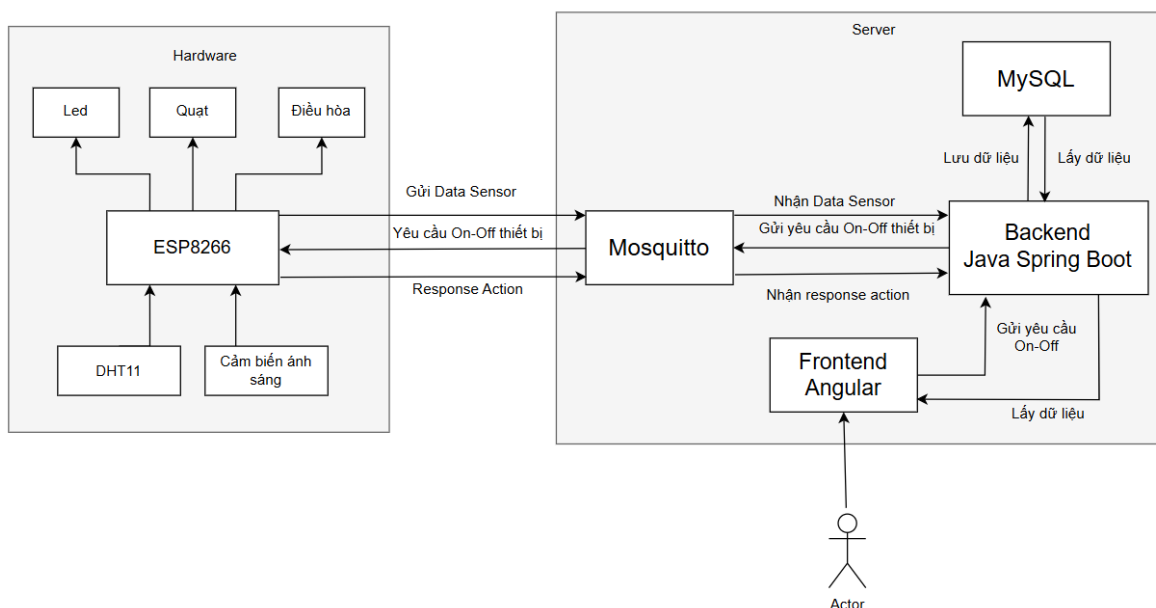
2.3. Giao diện thiết bị



Hình 15: Giao diện thiết bị

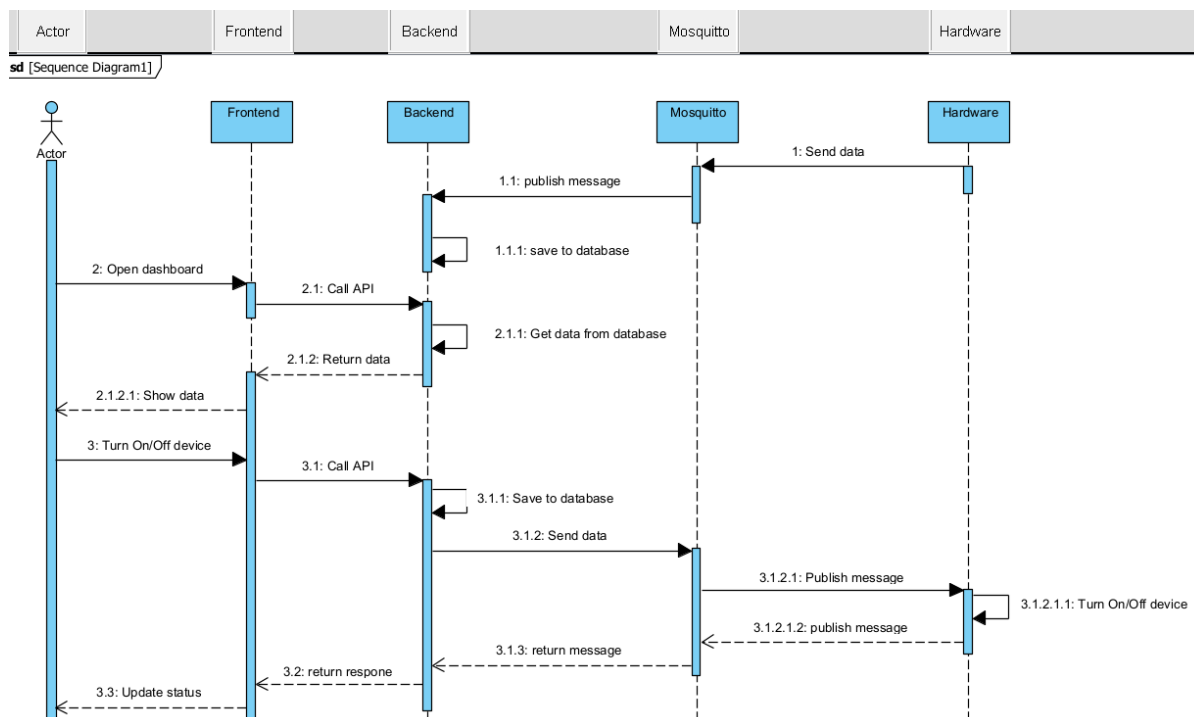
III. Thiết kế chi tiết

3.1. Thiết kế hệ thống



Hình 16: Thiết kế hệ thống

3.2. Sequence Diagram



Hình 17: Sequence Diagram của hệ thống

IV. Code

4.1. Arduino code

```
1  #include "DHT.h"
2  #include <ESP8266WiFi.h>
3  #include <Ticker.h>
4  #include <AsyncMqttClient.h>
5  #include <ArduinoJson.h>
```

- Khai báo các thư viện cần thiết gồm:
 - DHT.h: Đọc dữ liệu nhiệt độ và độ ẩm từ cảm biến DHT.
 - ESP8266WiFi.h: Kết nối ESP8266 với Wi-Fi.
 - Ticker.h: Tạo bộ định thời (timer) để thực hiện các tác vụ định kỳ.
 - AsyncMqttClient.h: Gửi/nhận dữ liệu MQTT không đồng bộ.
 - ArduinoJson.h: Xử lý dữ liệu JSON (tạo và phân tích).

```
7  #define WIFI_SSID "iPhone"
8  #define WIFI_PASSWORD "khongcomatkhou"
9
```

- Định nghĩa các thông tin như tên wifi, mật khẩu wifi mà ESP8266 sẽ kết nối

```
10 #define MQTT_HOST IPAddress(192, 168, 1,2)
11 #define MQTT_PORT 1883
12
13 #define MQTT_PUB_SENSOR "environmental/data"
14 #define MQTT_SUB_DEVICE_ACTION "device/action"
15
```

- Định nghĩa các thông tin về server của MQTT Broker (ở đây là Mosquitto), tên các topic, port

```
18 #define LIGHT_SENSOR_PIN A0
19
20 #define Fan_PIN 12
21 #define Led1_PIN 13
22 #define Led2_PIN 15
23
```

- Định nghĩa các chân (pin) mà cảm biến ánh sáng và các thiết bị (quạt và đèn LED) được kết nối trên ESP8266. Cụ thể:
 - Cảm biến ánh sáng được kết nối với chân analog A0 của ESP8266.
 - Chân GPIO số 12 (chân D6) được sử dụng để điều khiển quạt.

- Chân GPIO số 13 (chân D7) được dùng để điều khiển LED 1.
- Chân GPIO số 15 (chân D8) được dùng để điều khiển LED 2.

```
24  #define DHTPIN 14
25  #define DHTTYPE DHT11
26  DHT dht(DHTPIN, DHTTYPE);
```

- Định nghĩa chân kết nối với DHT11 là chân GPI14, chân Data của DHT11 sẽ nối với chân D5 trong mạch và định nghĩa loại DHT ở đây là DHT11

```
28  float temp;
29  float hum;
30  int light;
31
32  AsyncMqttClient mqttClient;
33  Ticker mqttReconnectTimer;
34  WiFiEventHandler wifiConnectHandler;
35  WiFiEventHandler wifiDisconnectHandler;
36  Ticker wifiReconnectTimer;
37
38  unsigned long previousMillis = 0;
39  const long interval = 10000;
```

- Đoạn mã này khai báo các biến và đối tượng cần thiết để làm việc với cảm biến, MQTT, và kết nối Wi-Fi

```
41  #define MQTT_USERNAME "trungduc"
42  #define MQTT_PASSWORD "trungduc"
--
```

- Khai báo user, password của mosquitto

```

45 void connectToWifi() {
46     Serial.println("Connecting to Wi-Fi...");
47     WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
48 }
49
50 void onWifiConnect(const WiFiEventStationModeGotIP& event) {
51     Serial.println("Connected to Wi-Fi.");
52     connectToMqtt();
53 }
54
55 void onWifiDisconnect(const WiFiEventStationModeDisconnected& event) {
56     Serial.println("Disconnected from Wi-Fi.");
57     mqttReconnectTimer.detach();
58     wifiReconnectTimer.once(2, connectToWifi);
59 }

```

- Định nghĩa các hàm để quản lý kết nối Wi-Fi cho ESP8266:

- connectToWifi(): Hàm này được gọi để kết nối ESP8266 với mạng Wi-Fi.
- onWifiConnect(): Hàm này được gọi tự động khi ESP8266 kết nối thành công với Wi-Fi và nhận được địa chỉ IP.
- onWifiDisconnect(): Hàm này được gọi khi ESP8266 bị mất kết nối Wi-Fi.

```

61 void connectToMqtt() {
62     Serial.println("Connecting to MQTT...");
63     mqttClient.connect();
64 }
65
66 void onMqttConnect(bool sessionPresent) {
67     Serial.println("Connected to MQTT.");
68     Serial.print("Session present: ");
69     Serial.println(sessionPresent);
70     mqttClient.subscribe(MQTT_SUB_DEVICE_ACTION, 1); // Subscribe
71 }
72
73 void onMqttDisconnect(AsyncMqttClientDisconnectReason reason) {
74     Serial.println("Disconnected from MQTT.");
75     if (WiFi.isConnected()) {
76         mqttReconnectTimer.once(2, connectToMqtt);
77     }
78 }

```

- Định nghĩa các hàm để quản lý kết nối MQTT cho ESP8266, bao gồm kết nối, đăng ký chủ đề (topic) và xử lý mất kết nối:

- connectToMqtt(): Kết nối với máy chủ MQTT.
- onMqttConnect(): Đã kết nối MQTT, đăng ký chủ đề nhận lệnh.

- `onMqttDisconnect()`: Mất kết nối, thử kết nối lại sau 2 giây nếu Wi-Fi còn kết nối.

```

80 void onMqttPublish(uint16_t packetId) {
81     Serial.print("Publish acknowledged.");
82     Serial.print(" packetId: ");
83     Serial.println(packetId);
84 }

```

- Hàm `onMqttPublish()` được gọi khi ESP8266 nhận được xác nhận (acknowledgment) từ máy chủ MQTT rằng một thông điệp đã được gửi đi thành công.

```

86 void onMessage(char* topic, char* payload, AsyncMqttClientMessageProperties properties, size_t length, size_t index, size_t total) {
87     Serial.println("Message arrived [" + String(topic) + "]");
88
89     String message(payload);
90
91     message.trim();
92     Serial.println("Raw Message: " + message);
93
94     int commaIndex = message.indexOf(',');
95     if (commaIndex != -1) {
96         String device = message.substring(0, commaIndex);
97         String status = message.substring(commaIndex + 1);
98
99         if (status.length() > 3) {
100             status = status.substring(0, 3);
101         }
102
103         device.trim();
104         status.trim();
105
106         Serial.println("Device: " + device);
107         Serial.println("Status: " + status);
108
109         if (device == "Fan") {

```

- Hàm `onMessage()` xử lý khi một thông điệp MQTT mới đến từ một chủ đề (topic) cụ thể. Các bước hoạt động chính của hàm:

- In thông báo về thông điệp mới:
- Xử lý thông điệp:
- Tách thiết bị và trạng thái: Tách phần trước dấu phẩy là device và phần sau là status.

```

108
109     if (device == "Fan") {
110         digitalWrite(Fan_PIN, status == "On" ? HIGH : LOW);
111         Serial.println("Fan set to " + status);
112     } else if (device == "Led-1") {
113         digitalWrite(Led1_PIN, status == "On" ? HIGH : LOW);
114         Serial.println("Led-1 set to " + status);
115     } else if (device == "Led-2") {
116         digitalWrite(Led2_PIN, status == "On" ? HIGH : LOW);
117         Serial.println("Led-2 set to " + status);
118     } else {
119         Serial.println("Unknown device: " + device);
120     }
121 } else {
122     Serial.println("Invalid message format");
123 }
124 }
125 }

```

- Kiểm tra loại thiết bị và trạng thái để bật/ tắt thiết bị trên hardware

```

127 void setup() {
128     Serial.begin(115200);
129     Serial.println();
130
131     dht.begin();
132     wifiConnectHandler = WiFi.onStationModeGotIP(onWifiConnect);
133     wifiDisconnectHandler = WiFi.onStationModeDisconnected(onWifiDisconnect);
134
135     mqttClient.onConnect(onMqttConnect);
136     mqttClient.onDisconnect(onMqttDisconnect);
137     mqttClient.onPublish(onMqttPublish);
138     mqttClient.onMessage(onMessage);
139
140     mqttClient.setServer(MQTT_HOST, MQTT_PORT);
141     mqttClient.setCredentials(MQTT_USERNAME, MQTT_PASSWORD);
142
143     pinMode(Fan_PIN, OUTPUT);
144     pinMode(Led1_PIN, OUTPUT);
145     pinMode(Led2_PIN, OUTPUT);
146     pinMode(TempCheck_LED_PIN, OUTPUT);
147
148     connectToWifi();
149 }

```

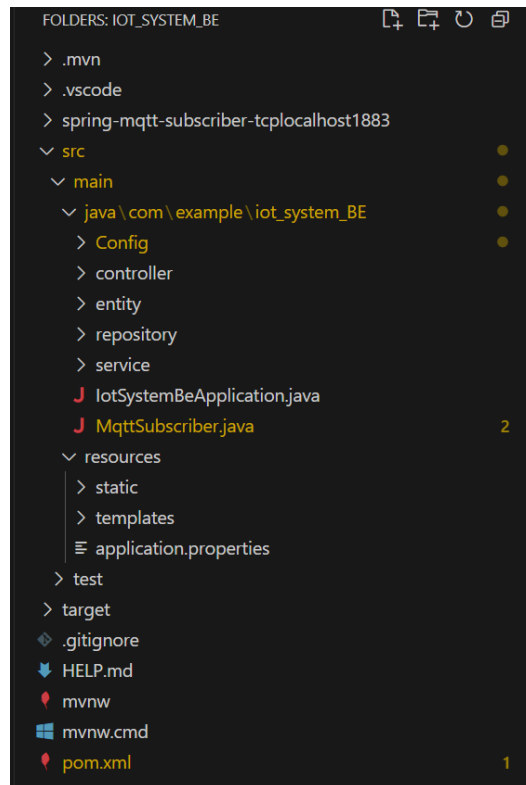
- Hàm setup() cấu hình các thiết lập cần thiết cho ESP8266 khi khởi động. Các bước thực hiện:

- Khởi tạo Serial: Bắt đầu giao tiếp serial.
- Khởi động cảm biến DHT: Khởi động cảm biến để thu thập dữ liệu.
- Thiết lập sự kiện Wi-Fi: Đăng ký hàm xử lý cho kết nối và mất kết nối Wi-Fi.
- Thiết lập sự kiện MQTT: Đăng ký hàm xử lý cho kết nối, mất kết nối, xác nhận và nhận thông điệp MQTT.
- Cấu hình MQTT: Thiết lập máy chủ và thông tin đăng nhập MQTT.
- Thiết lập chân điều khiển: Đặt các chân (pin) cho quạt và LED là đầu ra.
- Kết nối Wi-Fi: Bắt đầu kết nối với mạng Wi-Fi.

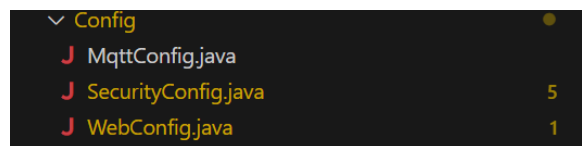
```
151 void loop() {
152     unsigned long currentMillis = millis();
153     if (currentMillis - previousMillis >= interval) {
154         previousMillis = currentMillis;
155
156         hum = dht.readHumidity();
157         temp = dht.readTemperature();
158         light = analogRead(LIGHT_SENSOR_PIN);
159
160         float roundedTemp = round(temp * 10) / 10.0;
161         float roundedHum = round(hum * 10) / 10.0;
162
163         String jsonString = String(roundedTemp) + "," + String(roundedHum) + "," + String(light) ;
164
165         Serial.print("Nhiệt độ: ");
166         Serial.print(roundedTemp);
167         Serial.print(" °C, Độ ẩm: ");
168         Serial.print(roundedHum);
169         Serial.println(" %");
170
171         uint16_t packetId = mqttClient.publish(MQTT_PUB_SENSOR, 1, true, jsonString.c_str());
172         Serial.printf("Đang xuất bản trên chủ đề %s với QoS 1, packetId %i: ", MQTT_PUB_SENSOR, packetId);
173         Serial.printf("Tin nhắn: %s \n", jsonString.c_str());
174     }
175 }
```

- Vòng lặp chính của chương trình sẽ liên tục đọc giá trị từ các cảm biến và ghép thành dạng Json rồi publish cho BE.

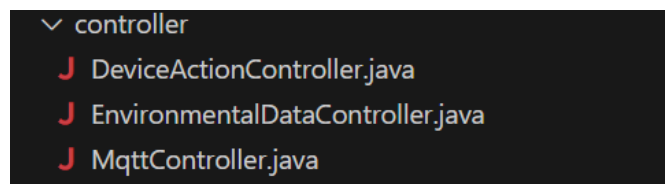
4.2. Backend code



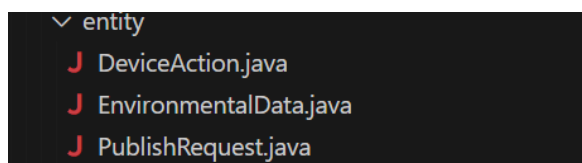
Cấu trúc thư mục của backend gồm:



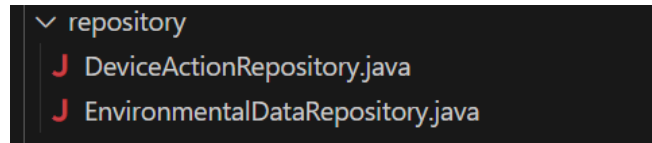
- Config: Chứa các cấu hình chung của ứng dụng, như cấu hình bảo mật, kết nối cơ sở dữ liệu, cấu hình MQTT, v.v.



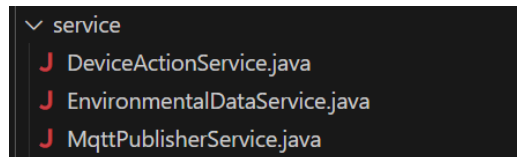
- Controller: Chứa các lớp điều khiển (controller) xử lý các yêu cầu HTTP từ phía người dùng, nhận và trả dữ liệu.



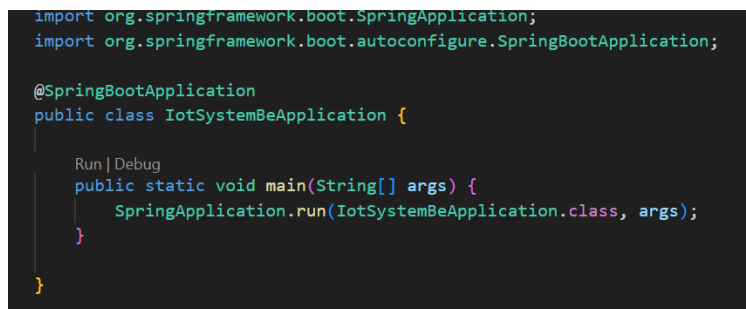
- Entity: Chứa các lớp biểu diễn các thực thể (entity) tương ứng với các bảng trong cơ sở dữ liệu.



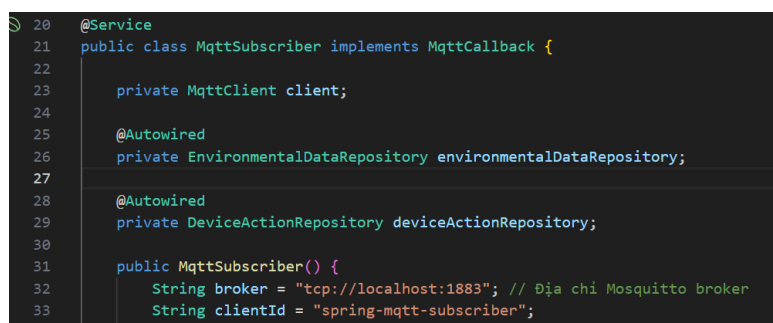
- Repository: Chứa các lớp giao tiếp với cơ sở dữ liệu, thường là các interface mở rộng JpaRepository để thực hiện các thao tác CRUD.



- Service: Chứa các lớp xử lý nghiệp vụ chính của ứng dụng, kết nối và xử lý dữ liệu từ repository và controller.

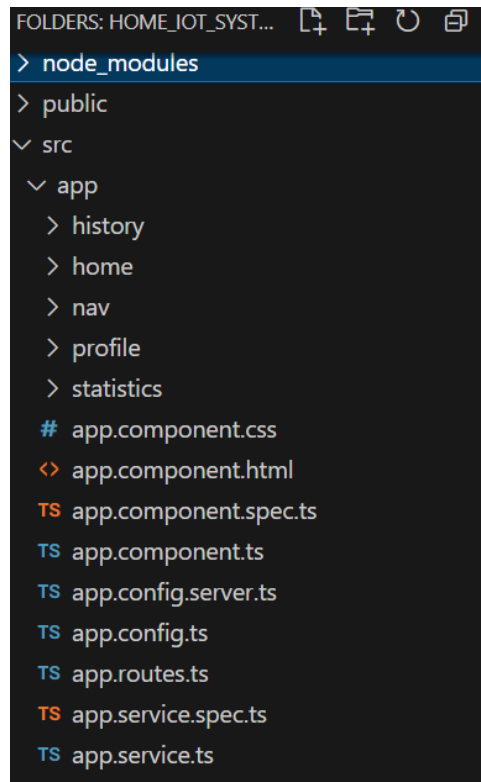


- IotSystemBeApplication.java: Lớp chính của ứng dụng Spring Boot, nơi chứa phương thức main để khởi động ứng dụng.

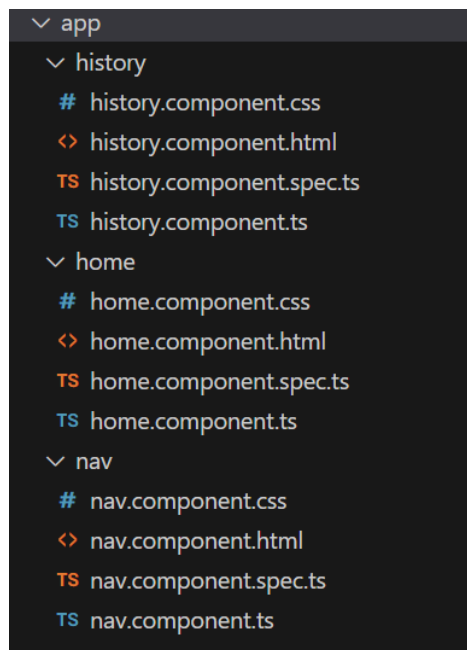


- MqttSubscriber.java: Chứa logic để xử lý các thông điệp nhận được từ máy chủ MQTT, có thể bao gồm việc đăng ký chủ đề và xử lý dữ liệu nhận được.

4.3. Frontend Code



Cấu trúc thư mục như sau:



- app: Thư mục chính chứa các thành phần (components), dịch vụ (services) và cấu hình của ứng dụng.

- history, home, nav, profile, statistics: Các thư mục này chứa các thành phần (components) tương ứng với các tính năng cụ thể của trang. Mỗi thư mục có các tệp .html, .css, và .ts để quản lý giao diện và logic của từng thành phần.

```
app.component.html X
src > app > app.component.html > router-outlet
Go to component
1 <app-nav></app-nav>
2 <router-outlet></router-outlet>
```

- app.component.html: Tệp HTML gốc chứa template giao diện chính của ứng dụng.

```
TS app.config.ts X
src > app > TS app.config.ts > ...
1 import { ApplicationConfig, provideZoneChangeDetection } from '@angular/core';
2 import { provideRouter } from '@angular/router';
3
4 import { routes } from './app.routes';
5 import { provideClientHydration } from '@angular/platform-browser';
6 import { provideHttpClient } from '@angular/common/http';
7
8 export const appConfig: ApplicationConfig = {
9   providers: [provideZoneChangeDetection({ eventCoalescing: true }),
10              provideRouter(routes),
11              provideClientHydration(),
12              provideHttpClient()],
13 };
14
15
```

- app.config.ts: Tệp cấu hình ứng dụng, chứa các hằng số và cài đặt chung (như API URL, thông tin cấu hình).

```
TS app.routes.ts X
src > app > TS app.routes.ts > ...
1 import { Routes } from '@angular/router';
2 import { ProfileComponent } from '../profile/profile.component';
3 import { HomeComponent } from '../home/home.component';
4 import { HistoryComponent } from '../history/history.component';
5 import { StatisticsComponent } from '../statistics/statistics.component';
6
7 export const routes: Routes = [
8   {path: 'profile', component: ProfileComponent},
9   {path: 'home', component: HomeComponent},
10  {path: 'history', component: HistoryComponent},
11  {path: 'statistics', component: StatisticsComponent},
12  {path: '**', component: HomeComponent},
13  {path: '', redirectTo: '/login', pathMatch: 'full'}
14 ];
15
```

- app.routes.ts: Tệp định tuyến, xác định các đường dẫn (routes) cho các thành phần trong ứng dụng.

```

9  export class AppService {
10     private apiUrl = 'http://localhost:8080'; // Đảm bảo URL chính xác
11
12     constructor(private http: HttpClient) { }
13
14     // Sử dụng Observable để lấy dữ liệu từ backend
15     getAllEnvironmentalData(): Observable<any[]> {
16         const url = `${this.apiUrl}/api/environmental-data`;
17         return this.http.get<any[]>(url).pipe(
18             catchError((error) => {
19                 console.error('Error fetching environmental data:', error);
20                 return throwError(() => error);
21             })
22         );
23     }
24     getAllDeviceAction(): Observable<any[]> {
25         const url = `${this.apiUrl}/api/device-action`;

```

- app.service.ts: Chứa các dịch vụ (services) dùng để lấy dữ liệu từ Backend và chia sẻ dữ liệu cho các component.

V. Kết quả thu được

5.1. Tổng quan

Hệ thống IoT đã thực hiện thành công các chức năng đo thông số môi trường theo thời gian thực, điều khiển thiết bị điện từ xa, và lưu trữ dữ liệu cho người dùng. Hệ thống đã được triển khai và thử nghiệm, đáp ứng tốt các yêu cầu đề ra.

5.2. Từ DHT11 và cảm biến ánh sáng

447	30.2°C	71%	973 lux	2024-09-24 23:47:29
446	30.2°C	71%	969 lux	2024-09-24 23:47:19
445	30.2°C	71%	973 lux	2024-09-24 23:47:09
444	30.2°C	71%	973 lux	2024-09-24 23:46:59
443	30.2°C	71%	973 lux	2024-09-24 23:46:49
442	30.2°C	71%	973 lux	2024-09-24 23:46:39
441	30.2°C	70%	973 lux	2024-09-24 23:46:29

- Hệ thống đã đo được các thông số của nhiệt độ, độ ẩm, ánh sáng và truyền lên website theo thời gian thực để hiển thị cho người dùng một cách tương đối trên trang Data Sensor.

5.3. Khi người dùng điều khiển thiết bị

ID ↓	Device	Action	Time
175	Fan	Off	2024-09-24 23:48:46
174	Led-1	Off	2024-09-24 23:48:46
173	Led-2	Off	2024-09-24 23:48:45
172	Led-2	On	2024-09-24 23:48:43
171	Led-1	On	2024-09-24 23:48:41
170	Fan	On	2024-09-24 23:48:41

- Hệ thống đã lưu thành công các dữ liệu khi người dùng nhấn On/Off trên website và hiển thị trên trang Action History

VI. Tài liệu tham khảo

- Tham khảo về Esp8266, mosquito, DHT11, Arduino IDE:

- [esp8266-nodemcu-mqtt-publish-dht11 -arduino](#)
- [esp8266-nodemcu-mqtt-publish-subscribe-dht22-readings/](#)