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

Sinh viên thực hiện : Nguyễn Anh Kiệt

Mã sinh viên : B21DCCN471

Số điện thoại : 0392514756

Hà Nội, 2024

LỜI CẢM ƠN

Để hoàn thành bài tập lớn này, em xin chân thành gửi lời cảm ơn đến thầy Nguyễn Quốc Uy, người đã tận tình hướng dẫn, định hướng cho em đi theo sự hiểu biết và khả năng của mình cũng như tạo điều kiện giúp đỡ em trong suốt quá trình học tập và hoàn thành tốt để tài được giao. Trong quá trình làm, kiến thức của em vẫn còn hạn chế và nhiều bờ ngỡ. Do vậy, những thiếu sót là điều chắc chắn không thể tránh khỏi, em rất mong nhận được những ý kiến đóng góp của thầy để kiến thức của em được cải thiện và hoàn thiện dự án hơn trong tương lai.

Cuối cùng, em xin kính chúc thầy nhiều sức khỏe, thành công và hạnh phúc trong công việc cũng như cuộc sống. Em hy vọng sẽ tiếp tục nhận được sự hướng dẫn và hỗ trợ từ thầy trong những lần tới.

Em xin chân thành cảm ơn!

MỤC LỤC

I. Giới thiệu

1. Mô tả đề tài và mục đích đề tài

Đề tài sẽ tập trung vào việc phát triển hệ thống giám sát và điều khiển các thông số môi trường và các thiết bị điện gia đình thông minh. Hệ thống sẽ sử dụng các cảm biến IoT để thu thập thông tin môi trường và điều khiển các thiết bị điện tử qua giao diện web. Cụ thể:

- Các cảm biến sử dụng: nhiệt độ, độ ẩm, ánh sáng, và độ bụi.
- Các thiết bị điều khiển: quạt, đèn, điều hòa.
- Giao diện web cho phép người dùng theo dõi thông số môi trường và điều khiển thiết bị từ xa.

Tuy nhiên, phạm vi của đề tài không bao gồm các khía cạnh bảo mật nâng cao cho hệ thống hoặc tối ưu hóa hiệu suất hệ thống ở quy mô lớn. Hệ thống sẽ tập trung vào các thiết bị đơn giản và việc giám sát tại một không gian giới hạn (như một căn phòng hoặc ngôi nhà nhỏ).

Các mục tiêu cụ thể của đề tài bao gồm:

- Phát triển giao diện web trực quan cho người dùng để giám sát các thông số môi trường và điều khiển thiết bị điện.
- Sử dụng các cảm biến IoT để thu thập dữ liệu về nhiệt độ, độ ẩm, độ bụi, ánh sáng và truyền tải thông tin này về hệ thống.
- Tích hợp hệ thống điều khiển thiết bị từ xa qua các API, cho phép người dùng bật/tắt quạt, đèn và điều hòa.
- Xây dựng chức năng tự động hóa, giúp thiết bị tự bật/tắt dựa trên các điều kiện môi trường định sẵn.
- Lưu trữ và quản lý dữ liệu thông qua cơ sở dữ liệu MySQL, bao gồm thông tin về môi trường, trạng thái thiết bị và lịch sử tương tác của người dùng.

2. Thiết bị phần cứng

a. ESP32-WROOM-32

- Thông số kỹ thuật

+ CPU: Xtensa® Dual-Core 32-bit LX6

+ RAM: 520 KB

+ Flash: 4 MB

+ Xung nhịp tối đa: 240 MHz

+ Wi-Fi:Chuẩn 802.11 b/g/n, 2.4 GHz (tốc độ tối đa 150 Mbps)

+ Chế độ hoạt động: STA (Station), AP (Access Point), STA+AP (Station + Access Point)

+ Bluetooth: BLE 4.2 (Bluetooth Low Energy)

+ Số chân GPIO: 30 chân

+ Các chức năng GPIO: PWM, I2C, SPI, UART, ADC

+ Điện áp hoạt động: 2.2V đến 3.6V

+ Tốc độ truyền dữ liệu (baud rate): 110 - 460800 bps

+ Hỗ trợ nâng cấp firmware từ xa (OTA): Có

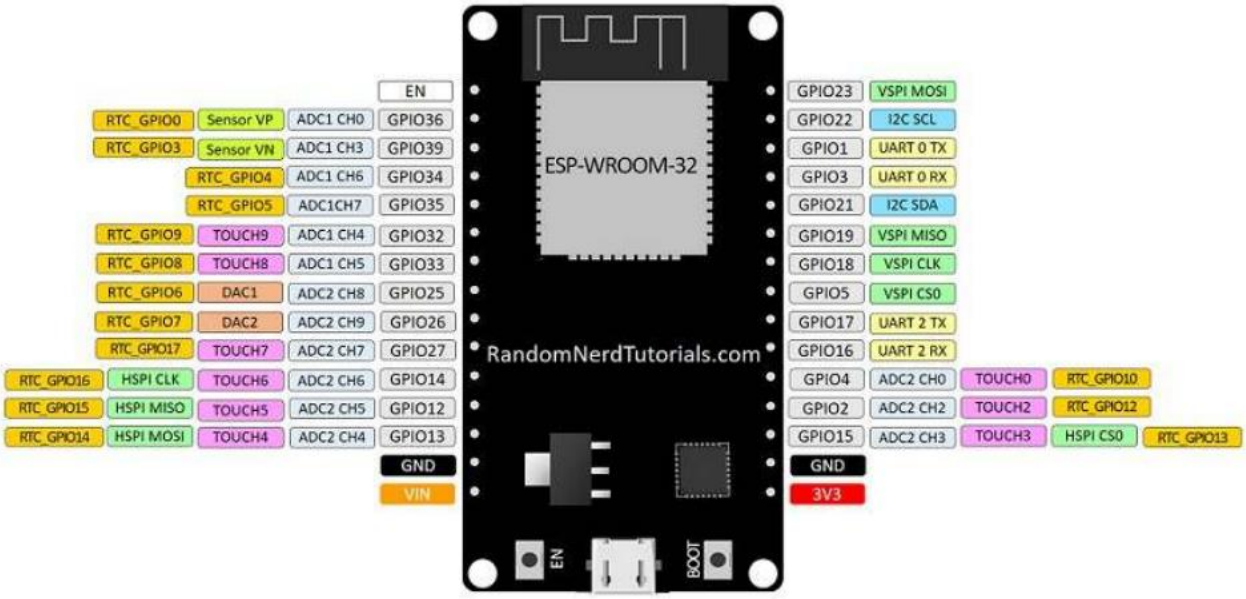
+ Chuyển đổi tín hiệu: Hỗ trợ ADC (Analog to Digital Converter)

+ Kích thước: 5.5 cm x 2.8 cm x 1.3 cm (bao gồm cả chân kết nối)

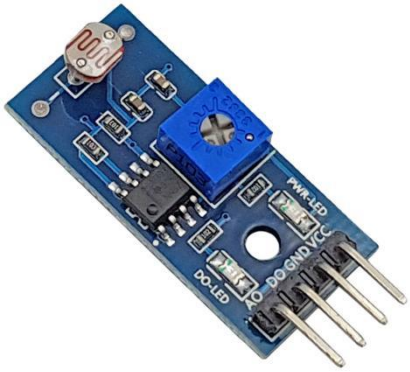
+ Chế độ tiết kiệm năng lượng: Hỗ trợ chế độ ngủ sâu (Deep Sleep)

+ Cảm biến tích hợp: Cảm biến điện dung, cảm biến nhiệt độ

- Sơ đồ chân ESP32



b. Cảm biến cường độ ánh sáng quang trở MS-CDS05



- Thông số kỹ thuật:

+ Điện áp hoạt động 3.3 – 5 V

+ 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.

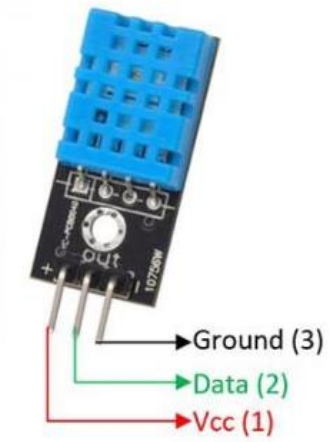
+ Độ nhạy cao với ánh sáng được tùy chỉnh bằng biến trở .

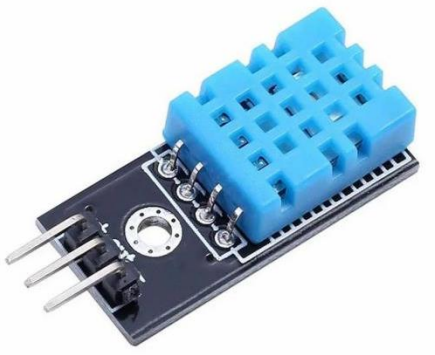
+ Kích thước 32 x 14mm

- Sơ đồ chân kết nối MS-CDS05

Tên chân	Mô tả
VCC	Chân cấp nguồn (điện áp đầu vào, thông thường từ 3.3V đến 5V).
GND	Chân nối đất (Ground).
AO	Chân tín hiệu Analog Output (giá trị điện áp tỷ lệ với cường độ ánh sáng).
DO	Chân tín hiệu Digital Output (mức cao/thấp, có thể điều chỉnh ngưỡng độ sáng bằng chiết áp tích hợp).

c. *Cảm biến nhiệt độ, độ ẩm DHT11*





- Thông số kỹ thuật

+ Điện áp hoạt động: 3V - 5V DC

+ Chuẩn giao tiếp: TTL, 1 wire.

+ Khoảng đo độ ẩm: 20%-90%RH, sai số ± 5%RH

+ Khoảng đo nhiệt độ: 0-50°C, sai số ± 2°C

+ Tần số lấy mẫu tối đa 1Hz (1 giây / lần tùy vào lập trình)

+ Kích thước: 23 x 12 x 5 mm

- Sơ đồ chân kết nối DHT11:

Tên chân	Mô tả
VCC	Chân cấp nguồn (điện áp đầu vào, thông thường từ 3.3V đến 5V).
GND	Chân nối đất (Ground).
DATA	Chân dữ liệu, truyền tín hiệu nhiệt độ và độ ẩm (kết nối với vi điều khiển).

d. Board test MB-102

Dùng để test mạch trước khi làm mạch in hoàn chỉnh. Thường được sử dụng trong các dự án điện tử để kết nối các linh kiện mà không cần hàn

e. Dây nối

f. Led màu 5mm chân dài

3. Công nghệ sử dụng

a. Visual Studio Code

- Visual Studio Code là một môi trường phát triển mã nguồn mở, hỗ trợ nhiều ngôn ngữ lập trình và công cụ khác nhau.

- Chức năng: Được sử dụng như một IDE để phát triển các ứng dụng NodeJS, ReactJS, đồng thời tích hợp với các công cụ như Prettier, và Git để quản lý mã nguồn, định dạng và kiểm tra lỗi mã nguồn.

- Ứng dụng: Phát triển và quản lý toàn bộ hệ thống backend và frontend của dự án.

b. ReactJS

- ReactJS là một thư viện JavaScript được sử dụng để xây dựng giao diện người dùng (UI), đặc biệt là các ứng dụng web đơn trang (Single Page Applications - SPA).
- Chức năng: Cung cấp cách thức xây dựng giao diện động, cập nhật và hiển thị dữ liệu theo thời gian thực mà không cần phải tải lại trang.
- Ứng dụng: Xây dựng giao diện web cho hệ thống giám sát và điều khiển từ xa, cho phép người dùng theo dõi nhiệt độ, độ ẩm, ánh sáng, độ bụi và điều khiển các thiết bị điện tử như quạt, đèn, điều hòa.

c. NodeJS

- NodeJS là một môi trường thực thi JavaScript phía server, được xây dựng trên nền tảng V8 engine của Google Chrome.
- Chức năng: Được sử dụng để xây dựng các API, giao tiếp giữa server và client, xử lý các yêu cầu từ frontend và giao tiếp với cơ sở dữ liệu.
- Ứng dụng: NodeJS đảm nhiệm vai trò Backend của hệ thống, cung cấp các API để truyền tải dữ liệu giữa các cảm biến, cơ sở dữ liệu, và giao diện người dùng. Đồng thời, NodeJS xử lý các yêu cầu điều khiển thiết bị và quản lý cơ sở dữ liệu MySQL.

d. Mosquitto

- Mosquitto là một môi trường triển khai giao thức MQTT (Message Queuing Telemetry Transport) – một giao thức nhắn tin nhẹ dành cho các ứng dụng IoT.
- Chức năng: Được sử dụng để trao đổi dữ liệu giữa các thiết bị IoT (như cảm biến, ESP32) và server. MQTT giúp truyền tải dữ liệu với độ trễ thấp, tiết kiệm băng thông, phù hợp với các thiết bị có tài nguyên hạn chế.
- Ứng dụng: Mosquitto đảm nhiệm vai trò trung gian giữa các thiết bị IoT và hệ thống, đảm bảo các cảm biến có thể gửi dữ liệu thời gian thực tới server,

đồng thời nhận các lệnh điều khiển từ hệ thống để điều khiển các thiết bị như quạt, đèn, điều hòa.

e. Trình biên dịch Arduno IDE

- Arduino IDE là một môi trường phát triển tích hợp (IDE) dùng để lập trình các vi điều khiển như ESP32, Arduino Uno, và các vi điều khiển khác.
- Chức năng: Biên dịch và nạp mã lập trình (sketch) lên vi điều khiển, đồng thời cung cấp các thư viện cần thiết để giao tiếp với các linh kiện ngoại vi như cảm biến, động cơ, đèn LED.
- Ứng dụng: Được sử dụng để lập trình, biên dịch và tải mã điều khiển cho các cảm biến, thiết bị ngoại vi trong dự án.

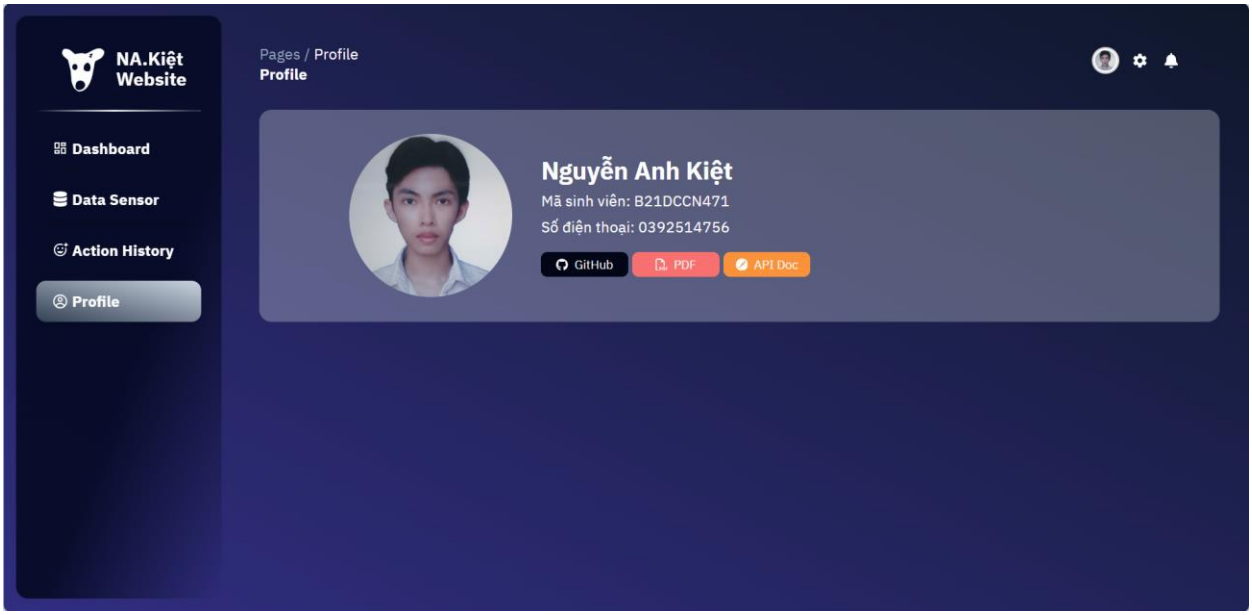
II. Giao diện và thiết kế tổng thể

1. Giao diện Web

Giao diện Dashboard

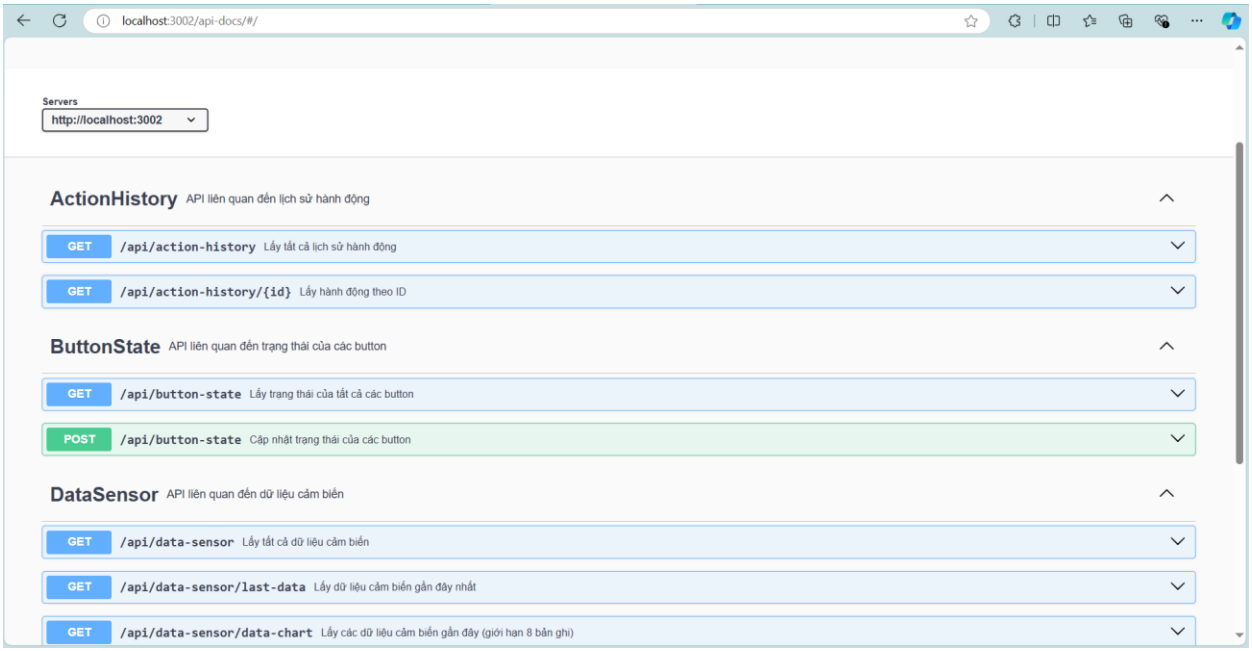
Giao diện Data Sensor

Giao diện Action History

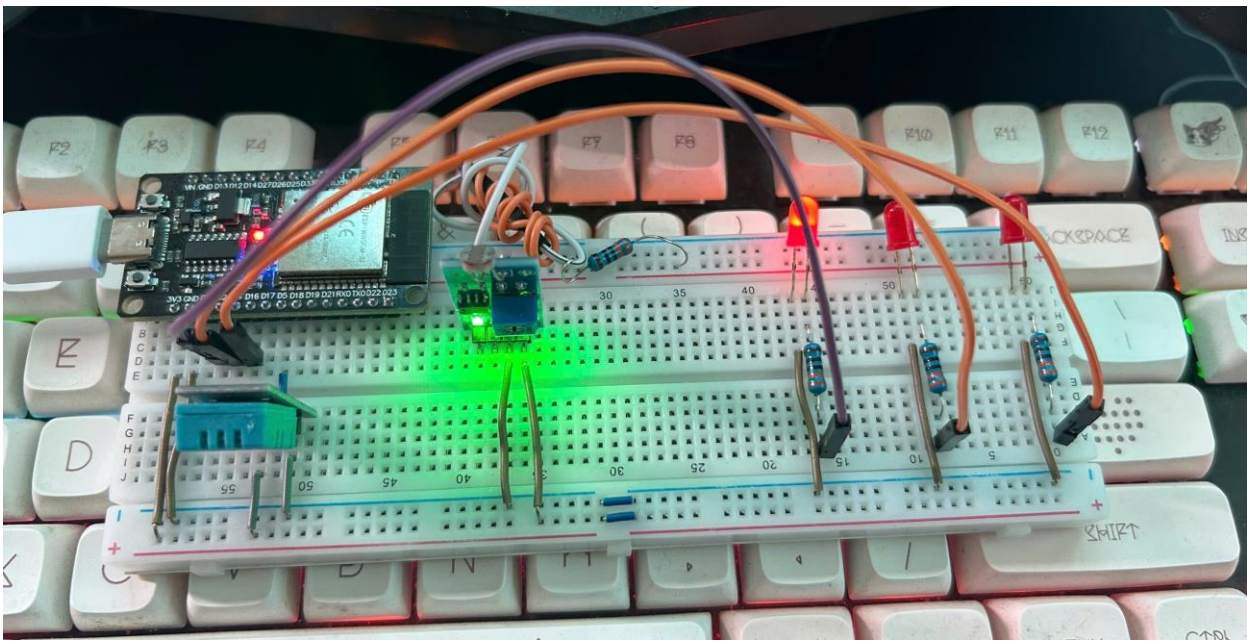


Giao diện Profile

2. Giao diện API Doc


localhost:3002/api-docs/#/
Servers
http://localhost:3002
ActionHistory API liên quan đến lịch sử hành động
GET /api/action-history Lấy tất cả lịch sử hành động
GET /api/action-history/{id} Lấy hành động theo ID
ButtonState API liên quan đến trạng thái của các button
GET /api/button-state Lấy trạng thái của tất cả các button
POST /api/button-state Cập nhật trạng thái của các button
DataSensor API liên quan đến dữ liệu cảm biến
GET /api/data-sensor Lấy tất cả dữ liệu cảm biến
GET /api/data-sensor/last-data Lấy dữ liệu cảm biến gần đây nhất
GET /api/data-sensor/data-chart Lấy các dữ liệu cảm biến gần đây (giới hạn 8 bản ghi)

3. Bảng mạch



III. Code

1. Embedded Code

```
#include <WiFi.h>
#include <AsyncMqttClient.h>
#include <DHT.h>
#include <Ticker.h>
```

- Khai báo thư viện:

+ WiFi.h: Thư viện này cung cấp các chức năng để kết nối ESP32 với mạng Wi-Fi. Thư viện giúp quản lý quá trình kết nối và kiểm tra trạng thái kết nối Wi-Fi.

+ AsyncMqttClient.h: Đây là thư viện để làm việc với giao thức MQTT một cách bất đồng bộ (không chặn luồng chính). MQTT là một giao thức nhẹ cho phép truyền và nhận dữ liệu giữa client và server qua các "chủ đề" (topics).

+ DHT.h: Thư viện này dùng để làm việc với cảm biến DHT11, cho phép đọc nhiệt độ và độ ẩm từ cảm biến.

+ Ticker.h: Thư viện Ticker cho phép gọi các hàm theo chu kỳ thời gian mà không cần dùng vòng lặp.

```
const char* ssid = "503-Trùm PTIT";
const char* password = "khongcomatkhau";
const char* mqtt_server = "192.168.1.101";

const int mqtt_port = 2003;
const char* mqtt_username = "nguyenanhkiet";
const char* mqtt_password = "b21dccn471";
const char* topic = "sensor/data";
const char* ledControlTopic = "led/control";
const char* ledStatusTopic = "led/status";
```

- Định nghĩa thông tin Wifi (tên, mật khẩu)

- Cấu hình MQTT (host, port, username, password, topic)

```
#define DHTPIN 5
#define DHTTYPE DHT11
#define LDR_AO 34
#define LED1_PIN 2
#define LED2_PIN 4
#define LED3_PIN 16
```

- Khai báo chân kết nối cảm biến và LED

+ DHTPIN và DHTTYPE: Khai báo chân kết nối của cảm biến DHT11. Cảm biến DHT11 được kết nối với chân GPIO 5 trên ESP32.

+ LDR_AO: Đây là chân GPIO 34, được sử dụng để kết nối cảm biến ánh sáng (Light Dependent Resistor - LDR).

+ LED1_PIN, LED2_PIN, LED3_PIN: Các chân điều khiển LED, được nối lần lượt với các chân GPIO 2, GPIO 4, và GPIO 16 trên ESP32. Các chân này sẽ được dùng để bật/tắt các LED.

```
DHT dht(DHTPIN, DHTTYPE);
AsyncMqttClient mqttClient;
Ticker mqttTicker;
```

- Khởi tạo các đối tượng cho cảm biến, MQTT và Ticker

+ DHT: Khởi tạo đối tượng cảm biến DHT11 với chân kết nối là DHTPIN (GPIO 5) và kiểu cảm biến là DHT11.

+ AsyncMqttClient: Tạo đối tượng MQTT để quản lý kết nối với broker, xử lý các sự kiện như gửi và nhận dữ liệu.

+ Ticker: Khởi tạo một đối tượng Ticker để thực hiện các công việc theo chu kỳ

```
void connectToWiFi() {
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
  }
}
```

- Hàm kết nối WiFi

+ WiFi.begin: Hàm này bắt đầu kết nối ESP32 với mạng Wi-Fi bằng cách sử dụng ssid và password.

+ WiFi.status: Hàm này kiểm tra trạng thái kết nối Wi-Fi.

WL_CONNECTED có nghĩa là đã kết nối thành công. Trong khi chưa kết nối, ESP32 sẽ đợi 500ms rồi kiểm tra lại.

```
void connectToMQTT() {
  Serial.println("Đang kết nối tới MQTT broker...");
  mqttClient.setCredentials(mqtt_username, mqtt_password);
  mqttClient.connect();
}
```

- Hàm kết nối MQTT

```
float convertToLux(int lightLevel) {
  float voltage = (lightLevel / 4095.0) * 3.3;
  float lux;
  if (voltage < 0.1) {
    lux = 1000;
  } else {
    lux = 1000 / voltage;
  }
  return lux;
}
```

- Hàm quy đổi giá trị ánh sáng từ tín hiệu Analog

```
void sendMessage() {
  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();
  int lightLevel = analogRead(LDR_AO);
  float lux = convertToLux(lightLevel);

  float do_bui = (lux > 0) ? 100000 / lux : 500;

  if (isnan(temperature) || isnan(humidity)) {
    return;
  }

  String payload = "{\"nhietdo\": ";
  payload += temperature;
  payload += ", \"doam\": ";
  payload += humidity;
  payload += ", \"anhsang\": ";
  payload += lux;
  payload += ", \"dobui\": ";
  payload += do_bui;
  payload += "}";

  mqttClient.publish(topic, 0, false, payload.c_str());
}
```

- Gửi dữ liệu cảm biến tới MQTT

+ dht.readTemperature và dht.readHumidity: Đọc giá trị nhiệt độ và độ ẩm từ cảm biến DHT11.

+ analogRead(LDR_AO): Đọc giá trị ánh sáng từ chân LDR_AO.

+ payload: Xây dựng một chuỗi JSON chứa các thông số đo được như nhiệt độ, độ ẩm, ánh sáng.

+ mqttClient.publish: Gửi dữ liệu cảm biến qua chủ đề sensor/data trên MQTT.

```
void onMqttMessage(char* topic, char* payload, AsyncMqttClientMessageProperties properti
  String message;
  for (int i = 0; i < length; i++) {
    message += payload[i];
  }

  if (String(topic) == ledControlTopic) {
    if (message == "LED1_ON") {
      digitalWrite(LED1_PIN, HIGH);
      sendLedStatus("LED1_ON");
    } else if (message == "LED1_OFF") {
      digitalWrite(LED1_PIN, LOW);
      sendLedStatus("LED1_OFF");
    } else if (message == "LED2_ON") {
      digitalWrite(LED2_PIN, HIGH);
      sendLedStatus("LED2_ON");
    } else if (message == "LED2_OFF") {
      digitalWrite(LED2_PIN, LOW);
      sendLedStatus("LED2_OFF");
    } else if (message == "LED3_ON") {
      digitalWrite(LED3_PIN, HIGH);
      sendLedStatus("LED3_ON");
    } else if (message == "LED3_OFF") {
      digitalWrite(LED3_PIN, LOW);
```

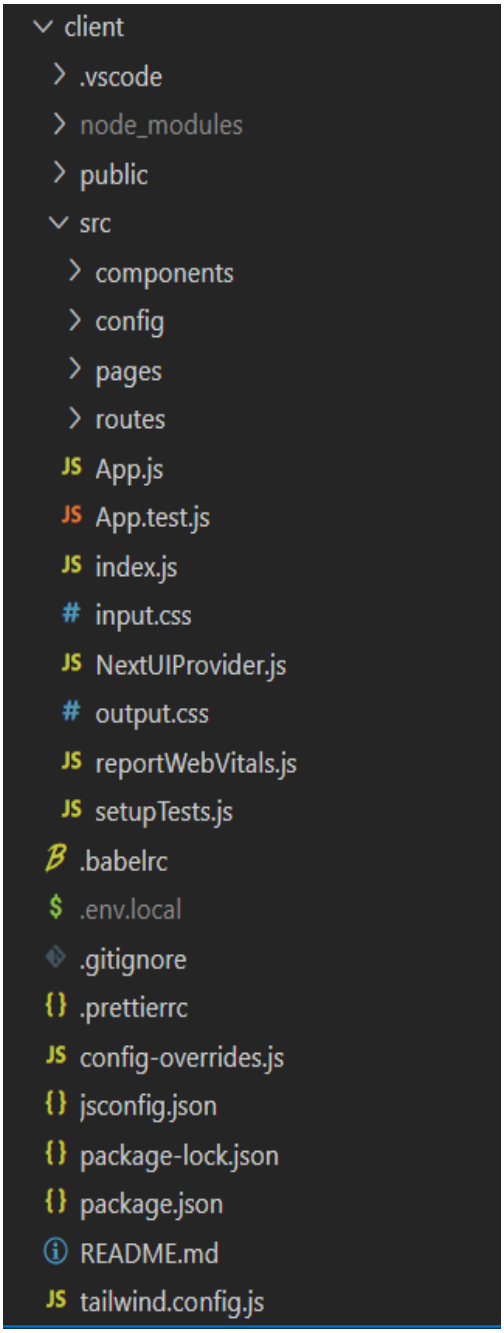
- onMqttMessage: Xử lý các lệnh điều khiển nhận được từ chủ đề led/control. Dựa trên nội dung của tin nhắn, nó sẽ bật hoặc tắt các đèn LED tương ứng, và gửi lại trạng thái bật/tắt lên chủ đề led/status.

```
mqttClient.onConnect([](bool sessionPresent) {
    Serial.println("Kết nối MQTT thành công!");
    mqttTicker.attach(5, sendMessage);
    mqttClient.subscribe(ledControlTopic, 0);
});
```

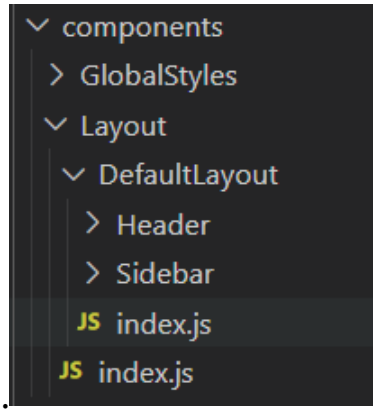
- Sau khi kết nối thành công tới MQTT, nó sẽ in ra thông báo, bắt đầu gửi dữ liệu cảm biến theo chu kỳ 5 giây một lần, và đăng ký để nhận các lệnh điều khiển đèn LED từ broker thông qua ledControlTopic.

2. Code client

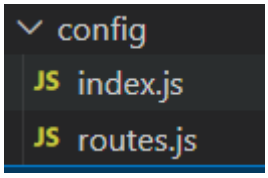
- Cấu trúc thư mục:


client
.vscode
node_modules
public
src
components
config
pages
routes
App.js
App.test.js
index.js
input.css
NextUIProvider.js
output.css
reportWebVitals.js
setupTests.js
.babelrc
.env.local
.gitignore
.prettierrc
config-overrides.js
jsconfig.json
package-lock.json
package.json
README.md
tailwind.config.js

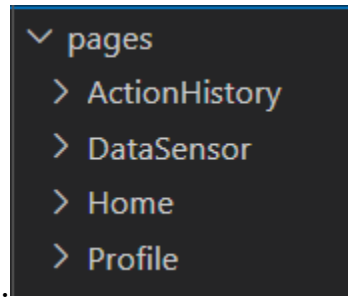
- Components: chứa các thành phần giao diện (UI components) chính của ứng dụng React. Đây là các thành phần được sử dụng lại trong nhiều phần của ứng dụng để giữ cho mã ngắn gọn và dễ bảo trì.



- config: Chứa file quản lý các đường dẫn trong ứng dụng

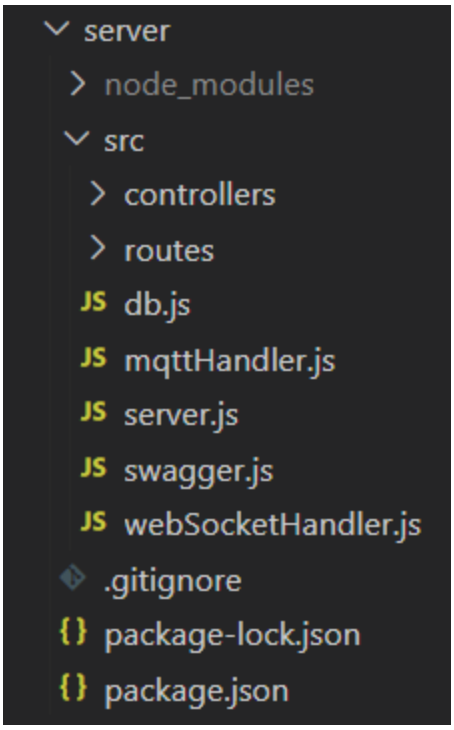


- pages: chứa các page components, đại diện cho từng trang (page) cụ thể của ứng dụng. Mỗi thư mục con tương ứng với một trang riêng, và trang đó sẽ chứa các logic, UI cụ thể cho chức năng tương ứng.



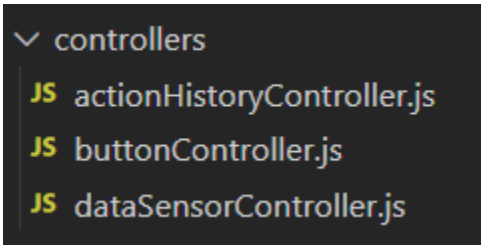
3. Code server

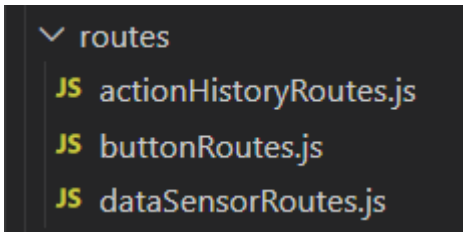
- Cấu trúc thư mục:

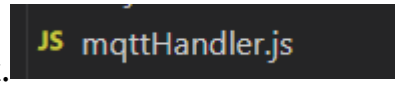


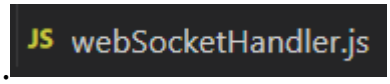
- db.js: file thiết lập kết nối với cơ sở dữ liệu MySQL 

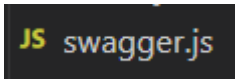
- controllers: chứa các file xử lý các yêu cầu HTTP và trả về cho người dùng



- routes: chứa các file định nghĩa đường dẫn cho các API . Các route này sẽ chỉ định controller nào được gọi để xử lý dựa trên đường dẫn (URL) và phương thức HTTP 

- mqttHandler.js: đảm nhận việc giao tiếp giữa thiết bị cảm biến, MQTT broker, và cơ sở dữ liệu MySQL. Nó cũng chịu trách nhiệm gửi dữ liệu thời gian thực tới client thông qua WebSocket. 

- webSocketHandler.js: đóng vai trò cầu nối giữa server và client, cho phép giao tiếp thời gian thực thông qua WebSocket, đảm bảo rằng các dữ liệu cảm biến được cập nhật liên tục đến client và các lệnh điều khiển được gửi đúng đến thiết bị thông qua MQTT.. 

- swagger.js: 

- server.js: 

IV. Tổng kết và hướng phát triển

1. Tổng kết

Sau quá trình gần 2 tháng nghiên cứu, tìm hiểu và phát triển hệ thống IoT, em đã triển khai, thử nghiệm và thu được những kết quả:

- Theo dõi, đo đạc các thông số theo thời gian thực
- Điều khiển tốt trạng thái thiết bị
- Lưu trữ dữ liệu cho người dùng

a. Theo dõi thông tin thời gian thực

- Hệ thống đo đạc các chỉ số môi trường và truyền lên ứng dụng web để hiển thị cho người dùng.
- Các chỉ số môi trường, biểu đồ được cập nhật theo thời gian thực.
- Khi bật tắt các thiết bị, nút điều khiển chờ phản hồi từ phần cứng, rồi mới cập nhật trạng thái điều kiển (bật, tắt).

b. Thông tin về các hoạt động bật tắt thiết bị

Hệ thống đã lưu thành công lịch sử bật tắt của các thiết bị. Người dùng có thể dùng lọc trạng thái bật hoặc tắt, tìm kiếm lịch sử theo thời gian trên thanh tìm kiếm.

c. Thông tin về dữ liệu được đọc từ phần cứng

Hệ thống đã lưu được tất cả dữ liệu được đọc từ phần cứng. Người dùng có thể tìm kiếm theo thời gian, chọn hiển thị các cột dữ liệu, sắp xếp.

2. Hướng phát triển

- Mở rộng số lượng các thiết bị cần điều khiển
- Mở rộng chức năng bật tắt tự động sử dụng quét mặt, vân tay...
- Tự động tắt khi chủ nhà quên tắt
- Phát triển thêm hệ thống báo cháy, dập lửa tự động
- Thêm camera giám sát
- Phát triển hệ thống đọc mức tiêu thụ điện