TRƯỜNG ĐẠI HỌC KHOA HỌC HUẾ

KHOA CÔNG NGHỆ THÔNG TIN

BÀI TIỂU LUẬN

MÔN: PHÁT TRIỂN ỨNG DỤNG IOT

NHÓM: 4

***Đề tài*: Điều khiển thiết bị gia dụng qua Wi-Fi với ESP32**

Học phần: Phát triển ứng dụng IoT - Nhóm 4

Giảng viên hướng dẫn: Võ Việt Dũng

HUẾ, 04/2025

MỤC LỤC

LỜI CẢM ƠN

Trước tiên, em xin gửi lời cảm ơn chân thành đến quý thầy cô trong khoa Công nghệ Thông tin, đặc biệt là giảng viên bộ môn Phát triển ứng dụng IoT, người đã tận tình hướng dẫn và truyền đạt những kiến thức quý báu trong suốt quá trình học tập. Sự hỗ trợ và định hướng của thầy/cô không chỉ giúp em hoàn thành tiểu luận này mà còn tạo động lực để nhóm khám phá sâu hơn về lĩnh vực Internet of Things.

Em cũng xin bày tỏ lòng biết ơn đến các bạn học cùng lớp, những người đã đóng góp ý kiến, chia sẻ kinh nghiệm và hỗ trợ nhóm trong quá trình thực hiện đề tài. Sự động viên và hợp tác từ các bạn là nguồn cảm hứng lớn để chúng tôi nỗ lực hoàn thiện sản phẩm.

Mặc dù đã cố gắng hết sức, tiểu luận này chắc chắn vẫn còn những thiếu sót. Em rất mong nhận được sự thông cảm và những ý kiến đóng góp quý giá từ thầy cô cũng như các bạn để đề tài được hoàn thiện hơn.

Trân trọng,

Em xin chân thành cảm ơn!

I. TỔNG QUAN VỀ ĐỀ TÀI NGHIÊN CỨU.

1. Đặt vấn đề.

Trong cuộc sống hiện đại, nhu cầu sử dụng các thiết bị gia dụng thông minh ngày càng gia tăng nhằm mang lại sự tiện lợi, tiết kiệm năng lượng và nâng cao chất lượng cuộc sống. Các thiết bị như đèn, quạt, máy lạnh, và các thiết bị điện khác cần được điều khiển từ xa một cách linh hoạt để tối ưu hóa hiệu suất sử dụng và tiết kiệm năng lượng. Việc điều khiển thủ công thường gây bất tiện, đặc biệt khi người dùng vắng nhà hoặc quên tắt các thiết bị.

Công nghệ Internet of Things (IoT) đã mở ra một hướng đi mới, cho phép tích hợp các cảm biến và vi điều khiển vào thiết bị gia dụng, từ đó tạo ra các hệ thống thông minh có khả năng tự động điều chỉnh theo nhu cầu thực tế. ESP32, với ưu điểm là một vi điều khiển mạnh mẽ, chi phí thấp, tích hợp Wi-Fi và khả năng xử lý tín hiệu nhanh, đã trở thành lựa chọn lý tưởng cho các ứng dụng IoT. Kết hợp với relay module, ESP32 có thể được sử dụng để xây dựng một hệ thống điều khiển thiết bị gia dụng từ xa thông qua Wi-Fi, giúp người dùng dễ dàng bật/tắt hoặc tự động hóa các thiết bị từ ứng dụng web hoặc di động.

Xuất phát từ thực tế trên, đề tài "Điều khiển thiết bị gia dụng qua Wi-Fi với ESP32" được lựa chọn nhằm nghiên cứu và phát triển một hệ thống IoT đơn giản nhưng thiết thực. Hệ thống này sử dụng ESP32 để nhận lệnh từ ứng dụng web hoặc mobile thông qua giao thức HTTP hoặc MQTT, từ đó điều khiển thiết bị gia dụng theo yêu cầu của người dùng. Đề tài không chỉ có ý nghĩa trong việc ứng dụng kiến thức lý thuyết vào thực tiễn mà còn góp phần hướng tới các giải pháp tiết kiệm năng lượng và thân thiện với môi trường.

2. Mục tiêu đề tài.

- Điều khiển thiết bị gia dụng (đèn, quạt, máy lạnh…) thông qua Wi-Fi bằng ESP32.
- Tích hợp ứng dụng web hoặc mobile để giám sát và điều khiển từ xa.
- Ứng dụng giao thức HTTP hoặc MQTT để truyền tải dữ liệu hiệu quả.
- Đảm bảo hệ thống hoạt động ổn định, tiết kiệm năng lượng và nâng cao tiện ích cho người dùng.

3. Ứng dụng thực tế.

Hệ thống có thể được ứng dụng trong:

- Nhà thông minh: Tự động điều chỉnh quạt khi nhiệt độ thay đổi.
- Quản lý nhiệt độ phòng server: Giúp duy trì nhiệt độ phù hợp.
- Nông nghiệp: Điều khiển quạt trong nhà kính để bảo vệ cây trồng.

4. Đối tượng và phạm vi nghiên cứu.

Hệ thống này cho phép người dùng điều khiển các thiết bị gia dụng như đèn, quạt từ xa thông qua ứng dụng web hoặc mobile. Có hai giao thức phổ biến để giao tiếp với ESP32: **HTTP** và **MQTT**.

II. CƠ SỞ LÝ THUYẾT.

1. ESP32 Node MCU và Thành phần chính của ESP32 Node MCU.

1.1. Giới thiệu về ESP32 Node MCU.

ESP32 là một vi điều khiển (MCU) tích hợp Wi-Fi và Bluetooth, được phát triển bởi **Espressif Systems**. Đây là phiên bản nâng cấp từ ESP8266, cung cấp hiệu suất cao hơn, nhiều tính năng hơn và khả năng tiết kiệm năng lượng tốt hơn. ESP32 thường được sử dụng trong các ứng dụng IoT (Internet of Things), nhà thông minh, điều khiển tự động và các thiết bị nhúng.

NodeMCU ESP32 là một bo mạch phát triển dựa trên vi điều khiển **ESP32** của **Espressif Systems**, được thiết kế để dễ dàng lập trình và kết nối với các thiết bị ngoại vi.

1.2. Thành phần chính của NodeMCU ESP32.

Thành phần	**Chức năng**
Vi điều khiển ESP32	Bộ xử lý trung tâm, có Wi-Fi & Bluetooth.

Thành phần	Chức năng
Cổng Micro-USB	Kết nối với máy tính để lập trình và cấp nguồn.
Chip USB-TTL (CP2102/CH340G)	Chuyển đổi tín hiệu từ USB sang UART để giao tiếp với máy tính.
Mạch nguồn (Voltage Regulator)	Chuyển đổi 5V từ USB xuống 3.3V để cấp nguồn cho ESP32.
Nút nhấn (BOOT & RESET)	Dùng để nạp chương trình và khởi động lại ESP32.
Đèn LED tích hợp	LED hiển thị trạng thái hoạt động.
Các chân GPIO (General Purpose Input/Output)	Giao tiếp với cảm biến, module, thiết bị ngoại vi.
Bộ ADC & DAC	Chuyển đổi tín hiệu tương tự sang số và ngược lại.
Module Wi-Fi & Bluetooth	Kết nối mạng và giao tiếp không dây.

---

2. Chức năng của các chân trên NodeMCU ESP32.

ESP32 có nhiều chân **GPIO (General Purpose Input/Output)** hỗ trợ nhiều chức năng khác nhau.

2.1. Các chân nguồn và GND.

Chân	Chức năng
VIN	Cấp nguồn từ ngoài (5V).

Chân	Chức năng
3V3	Cấp nguồn 3.3V cho module, cảm biến.
GND	Mass (chung nguồn) của mạch.

2.2. Các chân giao tiếp Digital (GPIO)

ESP32 có **34 chân GPIO**, có thể dùng làm Input/Output, PWM, SPI, I2C, UART,...

GPIO	Chức năng chính
GPIO 0	Boot mode, dùng khi nạp firmware.
GPIO 2	Đèn LED tích hợp trên board.
GPIO 5	SPI (CS), có thể dùng làm Output.
GPIO 12, 13, 14, 15	SPI (MISO, MOSI, SCK, CS).
GPIO 16, 17	Dùng làm UART hoặc Output.

⚠ **Lưu ý**:

- **GPIO 6 → GPIO 11** là các chân kết nối với bộ nhớ Flash, không nên sử dụng.
- **GPIO 34 → GPIO 39** chỉ hỗ trợ Input (không thể làm Output).

2.3. Các chân Analog (ADC & DAC)

ESP32 có **18 kênh ADC (độ phân giải 12-bit)** và **2 kênh DAC (độ phân giải 8-bit)**.

Chân	Chức năng
ADC (GPIO 32 - GPIO 39)	Đọc tín hiệu Analog từ cảm biến.
DAC (GPIO 25, GPIO 26)	Xuất tín hiệu Analog.

2.4. Các chân giao tiếp truyền thông

UART (Giao tiếp Serial)

Chân	Chức năng
GPIO 1 (TX0), GPIO 3 (RX0)	UART0 – Giao tiếp Serial mặc định.
GPIO 9, GPIO 10	UART1 (ít dùng).
GPIO 16, GPIO 17	UART2.

I2C (Giao tiếp với cảm biến, LCD,...)

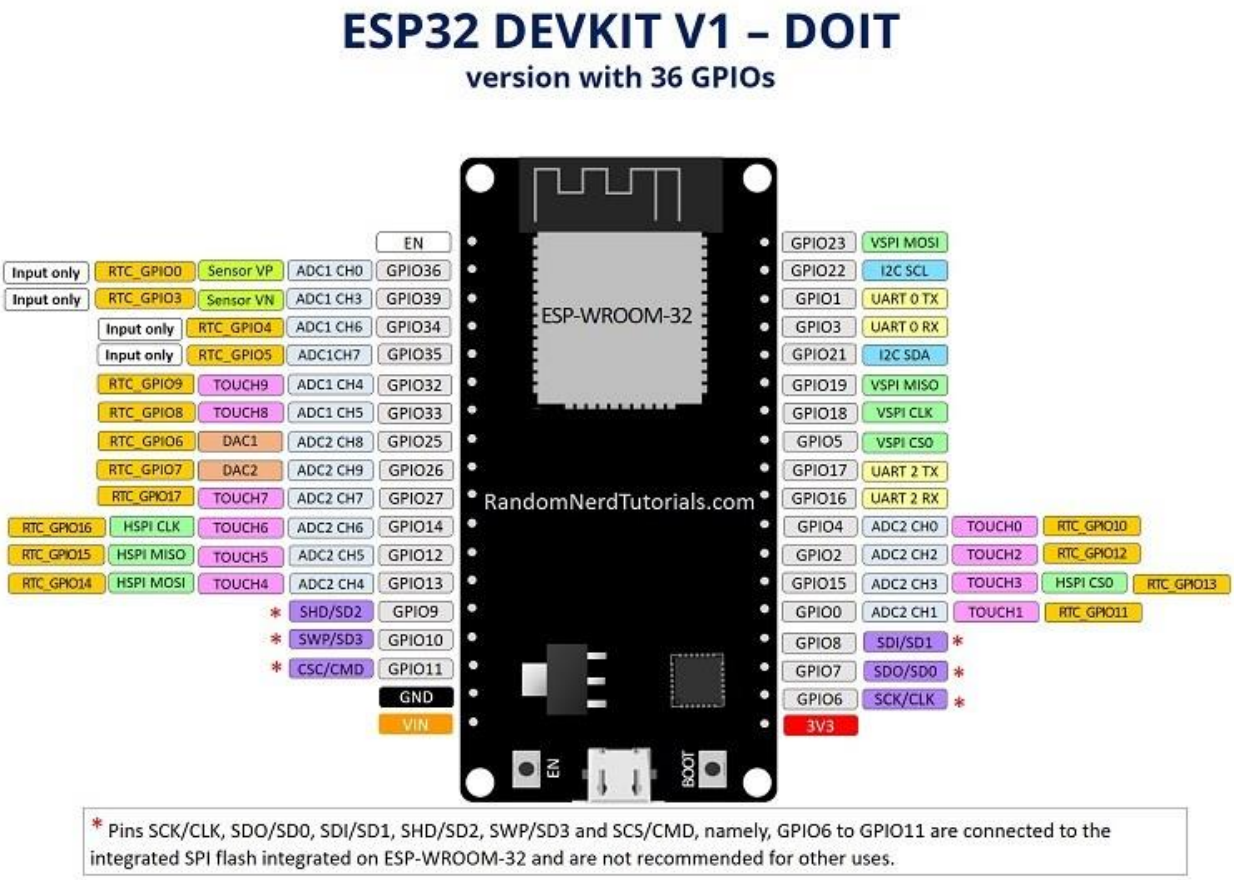
Chân	Chức năng
GPIO 21	SDA (Data).
GPIO 22	SCL (Clock).

SPI (Giao tiếp với module RF, thẻ nhớ,...)

Chân	Chức năng
GPIO 23	MOSI (Master Out Slave In).
GPIO 19	MISO (Master In Slave Out).
GPIO 18	SCK (Clock).
GPIO 5	CS (Chip Select).

PWM (Điều chế độ rộng xung, dùng để điều khiển quạt, LED, động cơ servo,...)

Tất cả các chân **GPIO** trên ESP32 đều có thể xuất tín hiệu **PWM** với tần số và độ phân giải tùy chỉnh.



Hình ảnh sơ đồ chân kết nối ESP32.

3. Công nghệ sử dụng.

3.1. Giới thiệu về React Native.

React Native là một **framework mã nguồn mở** do Facebook phát triển, cho phép lập trình viên xây dựng ứng dụng di động **đa nền tảng (Android & iOS)** chỉ với **một codebase** bằng JavaScript.

3.2. Đặc điểm của React Native.

3.2.1. Đa nền tảng(Cross-platform).

- Viết một lần, chạy trên cả **Android & iOS**, không cần tạo hai ứng dụng riêng biệt.
- Giảm thời gian và chi phí phát triển.

3.2.2. Hiệu suất cao gần như Native.

- React Native sử dụng các thành phần giao diện gốc (native components) thay vì WebView.

- Nhờ đó, hiệu suất gần như ứng dụng native viết bằng Swift (iOS) hay Kotlin (Android).

3.2.3. Hỗ trợ Hot Reload & Fast Refresh.

- Hot Reload giúp cập nhật giao diện ngay lập tức mà không cần chạy lại ứng dụng.

- Tiết kiệm thời gian trong quá trình lập trình.

3.2.4. Hỗ trợ thư viện phong phú.

- Có thể sử dụng thư viện JavaScript, cũng như tích hợp với các thư viện gốc của Android & iOS.

- Dễ dàng mở rộng với các module native.

3.2.5. Được hỗ trợ mạnh mẽ bởi cộng đồng.

- React Native có một cộng đồng lớn và nhiều tài liệu hỗ trợ.

- Facebook và nhiều công ty lớn (Instagram, Shopify, Tesla, v.v.) đang sử dụng React Native.

4. Giới thiệu phần mềm lập trình.

4.1. PlatformIO.

PlatformIO là một nền tảng phát triển nhúng (IoT) hỗ trợ nhiều loại vi điều khiển (ESP32, Arduino, STM32, AVR,...). PlatformIO hoạt động như một extension trong VSCode, giúp lập trình dễ dàng hơn so với Arduino IDE truyền thống.

+ Mở **VSCode** → Click vào biểu tượng **PlatformIO.**

+ Chọn **New Project.**

+ Đặt tên dự án, chọn **Board: ESP32**, Framework: **Arduino.**

+ Click **Finish** để tạo dự án.

4.2. MQTT.

***MQTT** hoạt động theo mô hình **publish/subscribe**, thay vì mô hình **client/server** truyền thống:

- **Broker (Máy chủ trung gian)**

  - MQTT sử dụng một thành phần trung gian gọi là **broker** để tiếp nhận và phân phối tin nhắn giữa các thiết bị.
  - Các client không giao tiếp trực tiếp với nhau mà thông qua broker.

- **Client (Thiết bị hoặc ứng dụng MQTT)**

  - **Publisher**: Gửi dữ liệu lên broker theo một **chủ đề (topic)**.
  - **Subscriber**: Đăng ký (subscribe) để nhận dữ liệu từ một hoặc nhiều **chủ đề** trên broker.

- **Mô hình Publish/Subscribe**

  - **Publisher** gửi tin nhắn đến một **topic** (ví dụ: home/livingroom/light).
  - **Broker** nhận tin nhắn và gửi nó đến tất cả các **subscriber** đã đăng ký topic đó.

- **Quality of Service (QoS) - Mức độ đảm bảo tin nhắn**

MQTT hỗ trợ 3 mức QoS để đảm bảo dữ liệu được truyền chính xác:

  - **QoS 0 (At most once)**: Gửi một lần, không đảm bảo nhận được (nhanh nhưng có thể mất tin nhắn).
  - **QoS 1 (At least once)**: Gửi lại nhiều lần cho đến khi nhận được xác nhận (có thể trùng lặp).
  - **QoS 2 (Exactly once)**: Đảm bảo nhận chính xác một lần (chậm nhưng đáng tin cậy).

- **Cơ chế Retained Message và Last Will**

  - **Retained Message**: Lưu tin nhắn cuối cùng trên một topic để client mới có thể nhận ngay.
  - **Last Will and Testament (LWT)**: Khi một client ngắt kết nối bất thường, broker sẽ gửi thông báo đến các subscriber đã đăng ký.

Trong dự án **"Điều khiển thiết bị gia dụng qua Wi-Fi với ESP32"**, **MQTT Broker** đóng vai trò trung gian, nhận dữ liệu cảm biến từ ESP32 và gửi lệnh bật/tắt thiết bị dựa trên ngưỡng cài đặt trong ứng dụng.

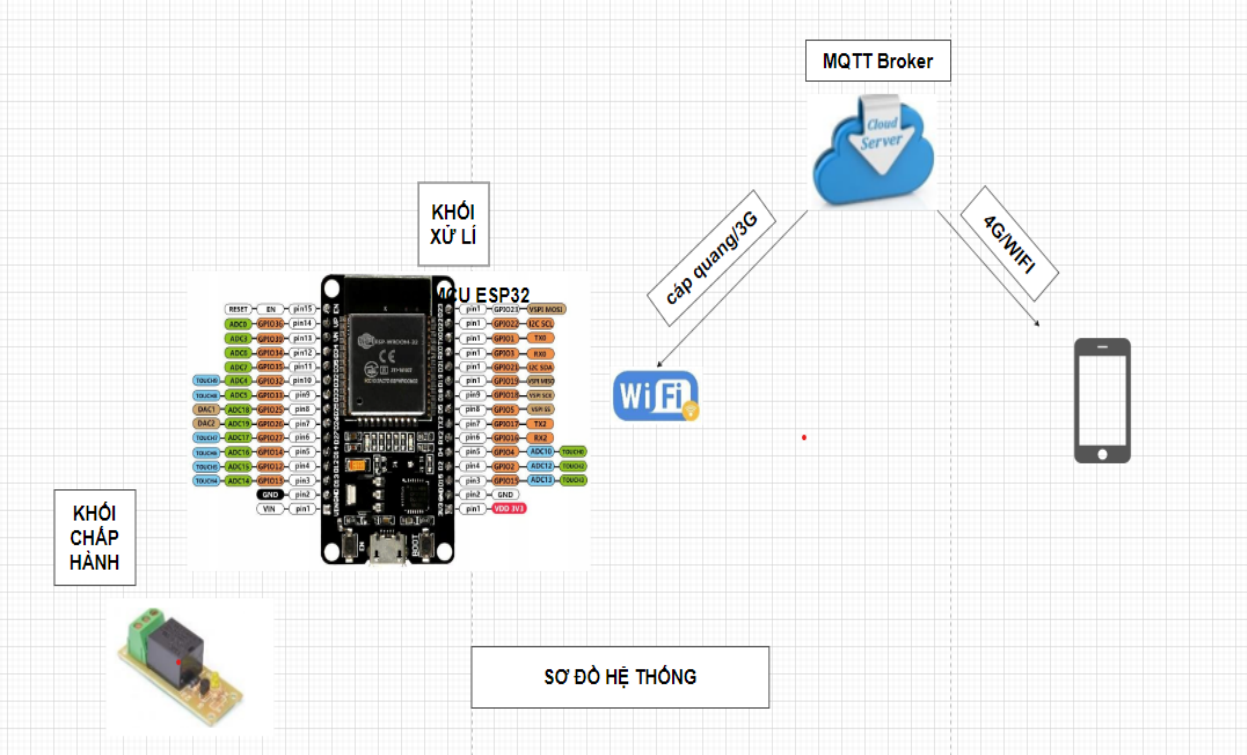
III. SƠ ĐỒ KHỐI & NGUYÊN LÝ HOẠT ĐỘNG.

3.1. Tổng quan hệ thống.

Hệ thống điều khiển thiết bị gia dụng qua Wi-Fi sử dụng ESP32 có thể chia thành ba thành phần chính:

1. Thiết bị điều khiển (ESP32)
2. Ứng dụng điều khiển (Web hoặc Mobile)
3. Giao thức truyền thông (HTTP hoặc MQTT)

3.2. Sơ đồ khối hệ thống.



Hình 1.1 Sơ đồ khối

Hệ thống bao gồm các thành phần chính sau:

- KHỐI XỬ LÝ (ESP32).
- MQTT Broker (Cloud Server).
- KHỐI CHẤP HÀNH (relay và thiết bị gia dụng).
- Thiết bị người dùng (smartphone).

3.3. Nguyên lý hoạt động.

3.3.1. Khởi tạo kết nối.

- ESP32 khởi động và kết nối với mạng Wi-Fi.
- ESP32 thiết lập kết nối với MQTT Broker thông qua internet (cáp quang/3G).
- Ứng dụng điện thoại của người dùng cũng kết nối đến MQTT Broker (qua 4G/WiFi).

3.3.2. Điều khiển thiết bị.

Khi người dùng gửi lệnh từ ứng dụng:

1. Ứng dụng điện thoại gửi lệnh điều khiển đến MQTT Broker.
2. MQTT Broker nhận và phân phối lệnh đến ESP32 (theo cơ chế publish/subscribe).
3. ESP32 nhận lệnh, xử lý và gửi tín hiệu điều khiển đến relay tương ứng.
4. Relay đóng/ngắt mạch điện để bật/tắt thiết bị (đèn, quạt).

3.3.3. Cập nhật trạng thái.

- ESP32 kiểm tra trạng thái của thiết bị sau khi thực hiện lệnh.
- ESP32 gửi thông tin trạng thái mới về MQTT Broker.
- MQTT Broker chuyển tiếp thông tin đến ứng dụng điện thoại.
- Ứng dụng cập nhật giao diện người dùng để hiển thị trạng thái mới.

3.4 Mô tả hoạt động theo sơ đồ trạng thái

Trạng thái	Điều kiện	Hành động của ESP32	Trạng thái quạt
Chờ lệnh	ESP32 kết nối Wi-Fi và MQTT Broker	Lắng nghe tín hiệu từ ứng dụng	-
Bật thiết bị	Nhận lệnh **ON** từ ứng dụng hoặc cảm biến	Gửi tín hiệu đến relay để đóng mạch điện	Thiết bị **ON**
Tắt thiết bị	Nhận lệnh **OFF** từ ứng	Gửi tín hiệu đến relay để ngắt mạch điện	Thiết bị **OFF**

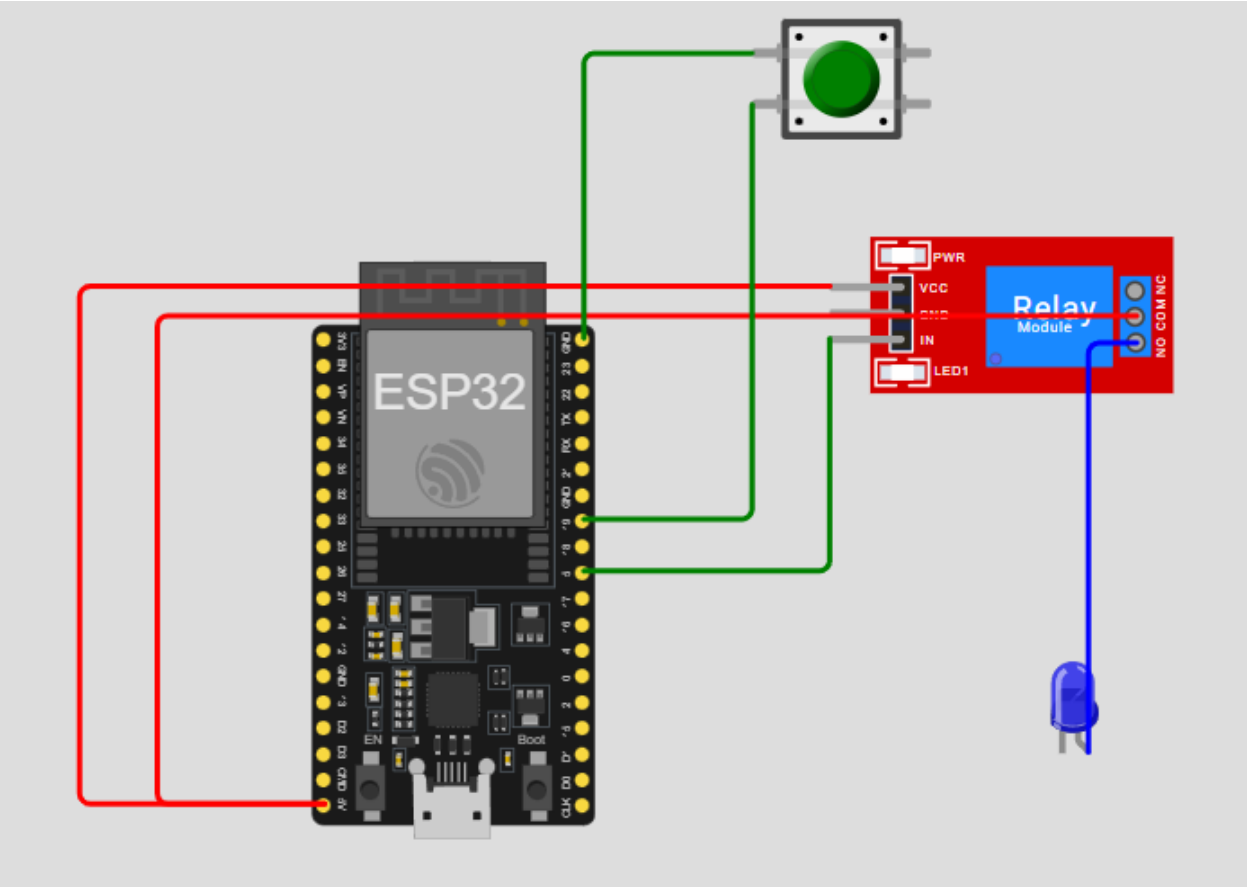
	dụng hoặc cảm biến		
Cập nhật trạng thái	Sau khi thực hiện lệnh, kiểm tra trạng thái thực tế	Gửi trạng thái hiện tại lên MQTT Broker	Ứng dụng cập nhật trạng thái
Mất kết nối	Mất kết nối với Wi-Fi hoặc MQTT Broker	Thử kết nối lại, nếu thất bại sẽ bật chế độ điều khiển thủ công	Không thay đổi (giữ trạng thái trước đó)

IV. THIẾT KẾ PHẦN CỨNG

4.1 Thành phần

STT	Linh kiện	Số lượng	Chức năng
1	ESP32	1	Vi điều khiển trung tâm, giao tiếp Wi-Fi và xử lý lệnh điều khiển thiết bị.
2	Module relay 5V	1	Điều khiển bật/tắt thiết bị điện (quạt, đèn,...) bằng tín hiệu từ ESP32.
3	Nguồn 5V	1	Cung cấp điện cho ESP32 và các linh kiện.
4	Nút nhấn	1	Điều khiển thiết bằng nút thủ công
5	LED báo trạng thái	1	Hiển thị trạng thái của hệ thống.
7	Quạt,đèn	1	Thiết bị cần điều khiển.

4.2 Sơ đồ mạch



4.2.1. Chi tiết kết nối phần cứng:

- Nguồn 5V:

- Cấp nguồn cho ESP32 (VCC và GND).
- Cấp nguồn cho module relay 5V (VCC và GND).
- Cấp nguồn cho LED báo trạng thái (thông qua điện trở hạn dòng).

- ESP32:

- **Kết nối với Relay 5V Module:** Một chân GPIO (General Purpose Input/Output) của ESP32 sẽ được kết nối với chân điều khiển của module relay. Tín hiệu từ ESP32 sẽ bật/tắt relay.
- **Kết nối với Nút Nhấn:** Một chân GPIO khác của ESP32 sẽ được kết nối với nút nhấn. Thông thường, một điện trở kéo lên (pull-up) hoặc kéo xuống (pull-down) sẽ được sử dụng để đảm bảo tín hiệu đầu vào ổn định khi nút nhấn không được nhấn.

- **Kết nối với LED Báo Trạng Thái:** Một chân GPIO của ESP32 sẽ được kết nối với LED báo trạng thái thông qua một điện trở hạn dòng (ví dụ: 220 Ohm hoặc 330 Ohm) để bảo vệ LED khỏi dòng điện quá lớn.

- **Relay 5V Module:**

  - **Kết nối với ESP32:** Như đã nói ở trên, chân điều khiển của relay được kết nối với một chân GPIO của ESP32.
  - **Kết nối với Quạt, Đèn:** Các chân COM (Common), NO (Normally Open), hoặc NC (Normally Closed) của relay sẽ được kết nối với quạt/đèn để điều khiển bật/tắt thiết bị.

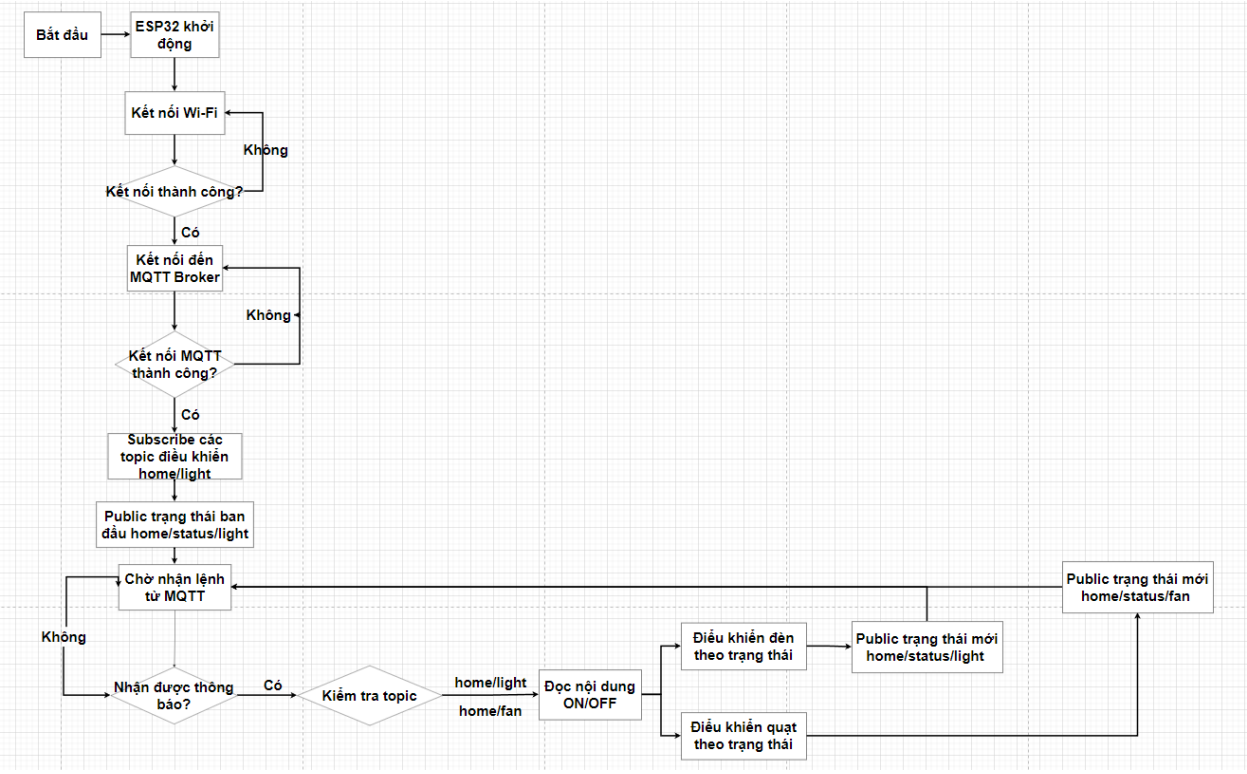
- **Nút Nhấn:**

  - **Kết nối với ESP32:** Nút nhấn được kết nối với một chân GPIO của ESP32 và một điện trở kéo lên/kéo xuống.

- **LED Báo Trạng Thái:**

  - **Kết nối với ESP32:** LED được kết nối với một chân GPIO của ESP32 và một điện trở hạn dòng.

4.2.2. Lưu đồ thuật toán(Flowchart):



V. LẬP TRÌNH ESP32

5.1. Mô tả chương trình:

Chương trình được viết trên Visual Studio Code.

- Sử dụng thư viện hỗ trợ:

    + WiFi.h – kết nối Wi-Fi

    + PubSubClient.h – giao tiếp MQTT

    + Relay – điều khiển thiết bị điện

- Chức năng hệ thống:

- **Kết nối Wi-Fi và MQTT Broker**
    - ESP32 tự động kết nối Wi-Fi khi khởi động.
    - Kết nối đến MQTT Broker (có thể đặt tại máy chủ nội bộ hoặc sử dụng dịch vụ công cộng như HiveMQ, Mosquitto...).
- **Giao tiếp MQTT để điều khiển thiết bị**
    - ESP32 subscribe các topic điều khiển như:
        - home/light (điều khiển đèn)
    - Khi nhận được lệnh "ON" hoặc "OFF", ESP32 bật/tắt relay tương ứng.
- **Bật/tắt relay để điều khiển thiết bị gia dụng**
    - Các thiết bị như đèn, quạt được nối với các chân GPIO của ESP32 thông qua relay.
    - Relay đóng/mở dựa trên lệnh điều khiển từ MQTT.
- **Cập nhật trạng thái thiết bị về MQTT**
    - Sau mỗi thao tác điều khiển, ESP32 gửi trạng thái mới lên topic như:
        - home/status/light (trạng thái đèn)
    - Ứng dụng mobile/web có thể subscribe các topic này để cập nhật UI theo thời gian thực.

- Ứng dụng Web/Mobile (Frontend):

- Giao diện trực quan với nút bật/tắt thiết bị.
- Khi người dùng nhấn nút:
    - Gửi lệnh "ON" hoặc "OFF" tới topic tương ứng trên MQTT.
- App cũng nhận phản hồi từ các topic trạng thái để cập nhật giao diện theo đúng trạng thái thiết bị.

5.2. Đoạn code cho ESP32:

```
/*
  ESP32 MQTT Relay Control with Button
  - Controls a relay via MQTT
  - Syncs state between physical button and MQTT
  - Publishes status updates to MQTT topic
*/

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

// WiFi credentials
const char* ssid = "Wokwi-GUEST";
const char* password = "";

// MQTT broker settings
const char* mqtt_server = "broker.emqx.io";  // Public MQTT broker (change to your broker)
const int mqtt_port = 1883;
const char* mqtt_username = "";  // Leave empty if no authentication
const char* mqtt_password = "";  // Leave empty if no authentication

// MQTT topics
```

```
const char* mqtt_topic_sub = "esp32/relay/cmd";     // For receiving commands
const char* mqtt_topic_pub = "esp32/relay/status";  // For publishing status
const char* mqtt_client_id = "ESP32_Relay_Controller";  // Should be unique

// GPIO Pins
const int RELAY_PIN = 5;    // Connected to IN pin of relay module
const int BUTTON_PIN = 19;  // Connected to push button

// Device state variables
bool relayState = false;
bool lastButtonState = HIGH;  // Assumes pull-up configuration
bool currentButtonState;
unsigned long lastDebounceTime = 0;
unsigned long debounceDelay = 50;  // Debounce time in milliseconds

// MQTT client
WiFiClient espClient;
PubSubClient client(espClient);

// Function prototypes
void setup_wifi();
void reconnect();
```

```
void callback(char* topic, byte* payload, unsigned int length);
void checkButton();
void controlRelay(bool state);
void publishState();

void setup() {
  // Initialize serial communication
  Serial.begin(115200);

  // Configure GPIO pins
  pinMode(RELAY_PIN, OUTPUT);
  pinMode(BUTTON_PIN, INPUT_PULLUP);

  // Initialize relay to OFF state
  digitalWrite(RELAY_PIN, LOW);

  // Setup WiFi connection
  setup_wifi();

  // Configure MQTT client
  client.setServer(mqtt_server, mqtt_port);
  client.setCallback(callback);
```

```
  // Print basic info
  Serial.println("ESP32 MQTT Relay Controller");
  Serial.println("---------------------------");
}

void loop() {
  // Handle MQTT connection
  if (!client.connected()) {
    reconnect();
  }
  client.loop();

  // Check physical button state
  checkButton();
}

void setup_wifi() {
  delay(10);
  Serial.println();
  Serial.print("Kết nối tới WiFi: ");
  Serial.println(ssid);
```

```
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi đã kết nối thành công");
  Serial.print("Địa chỉ IP: ");
  Serial.println(WiFi.localIP());
}

void callback(char* topic, byte* payload, unsigned int length) {
  // Convert payload to string
  String message;
  for (int i = 0; i < length; i++) {
    message += (char)payload[i];
  }

  Serial.print("Nhận tin nhắn [");
```

```
  Serial.print(topic);
  Serial.print("]: ");
  Serial.println(message);

  // Process MQTT command
  if (message.equals("ON")) {
    controlRelay(true);
  } else if (message.equals("OFF")) {
    controlRelay(false);
  } else if (message.equals("TOGGLE")) {
    controlRelay(!relayState);
  } else {
    Serial.println("Lệnh không hợp lệ");
  }
}

void reconnect() {
  // Loop until connected to MQTT broker
  while (!client.connected()) {
    Serial.print("Đang kết nối tới MQTT broker...");

    // Attempt to connect with authentication if provided
```

```
  bool connected = false;
  if (strlen(mqtt_username) > 0) {
    connected = client.connect(mqtt_client_id, mqtt_username, mqtt_password);
  } else {
    connected = client.connect(mqtt_client_id);
  }

  if (connected) {
    Serial.println("đã kết nối");

    // Subscribe to command topic
    client.subscribe(mqtt_topic_sub);

    // Publish current status
    publishState();
  } else {
    Serial.print("kết nối thất bại, lỗi = ");
    Serial.print(client.state());
    Serial.println(", thử lại sau 5 giây");
    delay(5000);
  }
}
```

```
}

void checkButton() {
  // Read button state with debouncing
  int reading = digitalRead(BUTTON_PIN);

  // Check if button state changed
  if (reading != lastButtonState) {
    lastDebounceTime = millis();
  }

  // Wait for debounce period
  if ((millis() - lastDebounceTime) > debounceDelay) {
    // If button state has truly changed
    if (reading != currentButtonState) {
      currentButtonState = reading;

      // If button is pressed (LOW when using INPUT_PULLUP)
      if (currentButtonState == LOW) {
        // Toggle relay state
        controlRelay(!relayState);
        publishState();
```

```
      }
    }
  }

  lastButtonState = reading;
}

void controlRelay(bool state) {
  // Update relay state
  relayState = state;

  // Control relay hardware
  digitalWrite(RELAY_PIN, state ? HIGH : LOW);

  // Log state change
  Serial.print("Relay: ");
  Serial.println(state ? "BẬT" : "TẮT");
}

void publishState() {
  // Publish current state to MQTT
  if (client.connected()) {
```

```
    client.publish(mqtt_topic_pub, relayState ? "ON" : "OFF", true);
    Serial.println("Đã đăng trạng thái lên MQTT");
  }
}
```

5.3. Đoạn code cho giao diện mobile (React Native):

```
// App.js
import React, { useEffect, useState } from 'react';
import { View, Button, Text, StyleSheet } from 'react-native';
import { Client } from 'react-native-paho-mqtt';

// MQTT WebSocket URL (broker hỗ trợ WS)
const mqttHost = 'ws://broker.emqx.io:8083/mqtt';
const topic = 'esp32/relay/cmd';

const mqttClient = new Client({
  uri: mqttHost,
  clientId: 'ReactNative_MQTT_Client_' + Math.random().toString(16).substr(2, 8),
  storage: {
    setItem: (key, item) => Promise.resolve(localStorage.setItem(key, item)),
    getItem: key => Promise.resolve(localStorage.getItem(key)),
    removeItem: key => Promise.resolve(localStorage.removeItem(key)),
  }
```

```
});

export default function App() {
  const [isConnected, setIsConnected] = useState(false);

  useEffect(() => {
    mqttClient.connect()
      .then(() => {
        setIsConnected(true);
        console.log('✅ Kết nối MQTT thành công');
      })
      .catch(err => {
        console.log('❌ Kết nối MQTT thất bại:', err);
      });

    return () => {
      mqttClient.disconnect();
    };
  }, []);

  const sendCommand = (command) => {
    if (isConnected) {
```

```
    mqttClient.publish(topic, command);
    console.log('📤 Đã gửi:', command);
  } else {
    console.log('⚠️ MQTT chưa kết nối');
  }
 };

 return (
  <View style={styles.container}>
    <Text style={styles.title}>Điều khiển Relay</Text>
    <Button title="BẬT" onPress={() => sendCommand('ON')} />
    <Button title="TẮT" onPress={() => sendCommand('OFF')} />
    <Button title="CHUYỂN TRẠNG THÁI" onPress={() =>
sendCommand('TOGGLE')} />
    <Text style={{ marginTop: 20 }}>{isConnected ? '✏️ MQTT Connected' : '✏️ Đang
kết nối...'}</Text>
  </View>
 );
}

const styles = StyleSheet.create({
 container: {
  flex: 1, justifyContent: 'center', alignItems: 'center', backgroundColor: '#f4f4f4'
```

```
  },
  title: {
    fontSize: 24, fontWeight: 'bold', marginBottom: 20
  }
});
```

VI. TÀI LIỆU KHAM KHẢO

[1] TS. Nguyễn Chí Nhân, “Bài giảng: Phát triển ứng dụng Internet vạn vật - Internet of Things (IoT)”, Bộ môn Vật lý Điện tử, Khoa Vật lý-Vật lý Kỹ thuật, Trường Đại học Khoa học Tự nhiên, ĐHQG TP.HCM (lưu hành nội bộ).

[2] https://www.hivemq.com/mqtt/

[3] https://randomnerdtutorials.com/

[4] https://pubsubclient.knolleary.net/

[5] https://github.com/me-no-dev/ESPAsyncWebServer