**TRƯỜNG ĐẠI HỌC BÌNH DƯƠNG**

**KHOA CÔNG NGHỆ THÔNG TIN, ROBOT VÀ TRÍ TUỆ NHÂN TẠO**

**A gold and blue emblem with a book and a helicopter

AI-generated content may be incorrect.**

**TIỂU LUẬN CUỐI KÌ**

***Tên đề tài:***

**MÔ HÌNH THÙNG RÁC THÔNG MINH**

Giảng Viên Hướng Dẫn: **LÊ DUY HÙNG**

Sinh viên thực hiện và mã số sinh viên:

**LÊ TUẤN HẢI - 23050116**

**NGUYỄN MINH CANG – 23050120**

**Bình Dương, tháng 5 năm 2025**

**LỜI CAM ĐOAN**

Sinh viên (hoặc nhóm sinh viên) ký tên dưới đây xin cam đoan: bài tiểu luận này là kết quả nghiên cứu, phát triển của cá nhân (hoặc nhóm), không phải là sản phẩm được sao chép hoặc đã được tác giả khác công bố.

|  |  |
| --- | --- |
|  | *Bình Dương, ngày 6 tháng 5 năm 2025*  Chữ ký từng thành viên  Lê Tuấn Hải  Nguyễn Minh Cang |

**MỤC LỤC**

[CHƯƠNG 1 : PHẦN GIỚI THIỆU 1](#_Toc196982674)

[1.1 Thông Tin Chung 1](#_Toc196982675)

[1.2 Kế Hoạch Thực Hiện 2](#_Toc196982676)

[CHƯƠNG 2 : TỔNG QUÁT VỀ HỆ THỐNG 3](#_Toc196982677)

[2.1 Giới Thiệu Hệ Thống 3](#_Toc196982678)

[2.2 Mô Hình Kiến Trúc Và Chức Năng 3](#_Toc196982679)

[2.2.1 Tầng Cảm Biến (Sensor Layer) 3](#_Toc196982680)

[2.2.2 Tầng Xử Lý (Processing Layer) 3](#_Toc196982681)

[2.2.3 Tầng Truyền Thông (Communication Layer) 3](#_Toc196982682)

[2.2.4 Tầng Hiển Thị Và Điều Khiển (UI/Control Layer) 3](#_Toc196982683)

[2.3 Sơ Đồ Vị Trí 4](#_Toc196982684)

[2.4 Sơ Đồ Kết Nối Phần Cứng & Phần Mềm 4](#_Toc196982685)

[CHƯƠNG 3 : KẾT QUẢ TRIỂN KHAI 6](#_Toc196982686)

[CHƯƠNG 4 : KẾT LUẬN – KIẾN NGHỊ 7](#_Toc196982687)

[4.1 Đánh Giá Kết Quả Đạt Được 7](#_Toc196982688)

[4.1.1 Ưu Điểm Nổi Bật 8](#_Toc196982689)

[4.1.2 Hạn Chế Khắc Phục 8](#_Toc196982690)

[4.1.3 So Sánh Với Các Giải Pháp Khác 8](#_Toc196982691)

[4.1.4 Đề Xuất Cải Tiến 9](#_Toc196982692)

[PHỤ LỤC 10](#_Toc196982693)

[Code Trong File Docker Compose 10](#_Toc196982694)

[Code trong file ino trong Arduino 10](#_Toc196982695)

# PHẦN GIỚI THIỆU

## Thông Tin Chung

Tên đề tài: Thùng Rác Thông Minh

Danh sách thành viên của nhóm: Lê Tuấn Hải, Nguyễn Minh Cang

Ngày hoàn thành: 25/4/2025

Hệ thống triển khai:

Thùng rác thông minh được triển khai dựa trên nền tảng **ESP8266**, giao tiếp với các cảm biến (nhiệt độ độ ẩm, cảm biến siêu âm) và cảm biến dạng Actuator (còi,LED vàng, LED 3 màu, servo 9g) thông qua các **chân GPIO**. Ngoài ra, hệ thống còn sử dụng thêm 2 board Arduino Uno để cấp nguồn 5V ổn định cho động cơ servo và cảm biến siêu âm.

Dữ liệu từ các cảm biến được gửi về **MQTTX** thông qua kết nối WiFi và được giám sát, quản lý, điều khiển trên nền tảng **FUXA**. Hệ thống hỗ trợ điều khiển linh hoạt từ xa thông qua ứng dụng **MQTTX**, giao diện **FUXA** trên trình duyệt, hoặc trực tiếp từ điện thoại thông minh – giúp người dùng dễ dàng giám sát và tương tác với thiết bị mọi lúc, mọi nơi.

Link github: <https://github.com/LeTuanHai-SVDHBD/CONG_NGHE_IOT-THUNG-RAC>

|  |  |  |  |
| --- | --- | --- | --- |
| **Tên Tiếng Việt** | **Tên Tiếng Anh** | **Chân GPIO** | **Chức Năng** |
| Cảm biến nhiệt độ độ ẩm | DHT11 | D3 | Đo nhiệt độ và độ ẩm trong thùng rác |
| Cảm biến siêu âm | HC-RS04 | D2,D1 | Phát hiện khoảng cách đến vật thể |
| Còi | Buzzer | D7 | Phát tiếng kêu khi mở nắp thùng rác |
| Đèn LED vàng | Yellow LED | D6 | Phát sáng khi mở nắp thùng rác |
| Động cơ điều khiển vị trí | Micro Servo Digital 9g | D5 | Đóng mở nắp thùng rác |
| Đèn LED 3 màu | Module LED RGB | D0,D4,D8 | Phát sáng 3 màu tạo hiệu ứng ánh sáng trang trí |
| Vi điều khiển Arduino Uno | Arduino Uno | 5V, GND | Cấp nguồn 5V ổn định cho động cơ servo và cảm biến siêu âm |

Bảng . Các Thiết Bị Sẽ Sử Dụng

## Kế Hoạch Thực Hiện

Bảng phân công các thành viên của nhóm và tiến độ hoàn thành từng công việc

|  |  |  |
| --- | --- | --- |
| Họ & Tên | Công Việc Cần Làm | Hoàn Thành |
| Lê Tuấn Hải | Lên Kế Hoạch, Xây Dựng Mô Hình, Lắp Ráp Mô Hình, Đi Đường Điện, Tìm Kiếm Code, Kết Nối MQTTX và FUXa | 100% |
| Nguyễn Minh Cang | Hỗ Trợ: Xây Dựng Mô Hình, Kết Nối MQTTX và FUXA | 100% |

Bảng . Bảng Phân Công Và Tiến Độ Hoàn Thành

# TỔNG QUÁT VỀ HỆ THỐNG

## Giới Thiệu Hệ Thống

Hệ thống thùng rác thông minh thiết kế để giám sát và cảnh báo các sự cố như:

- Phát hiện khoảng cách vật thể

- Nhiệt độ độ ẩm

Hệ thống thùng rác thông minh của bạn sử dụng vi điều khiển ESP8266 kết nối WiFi để giao tiếp với MQTT Broker và tích hợp với PlatForm FUXA nhằm hiển thị trạng thái và điều khiển các thiết bị từ xa. Hệ thống cho phép điều khiển MQTTX, FUXA hoặc tự động dựa trên các sự kiện cảm biến.

## Mô Hình Kiến Trúc Và Chức Năng

### Tầng Cảm Biến (Sensor Layer)

* **Cảm biến siêu âm:** Phát hiện khoảng cách vật cản (HC-RS04)
* **Cảm biến nhiệt độ độ ẩm:** Đo nhiệt độ độ ẩm trong môi trường thùng rác và nếu nhiệt độ trên 40 độ C tự độ mở thùng rác (DHT11)

### Tầng Xử Lý (Processing Layer)

* **ESP8266:** Trung tâm điều khiển, xử lý các sự kiện cảm biến, đèn, còi,…

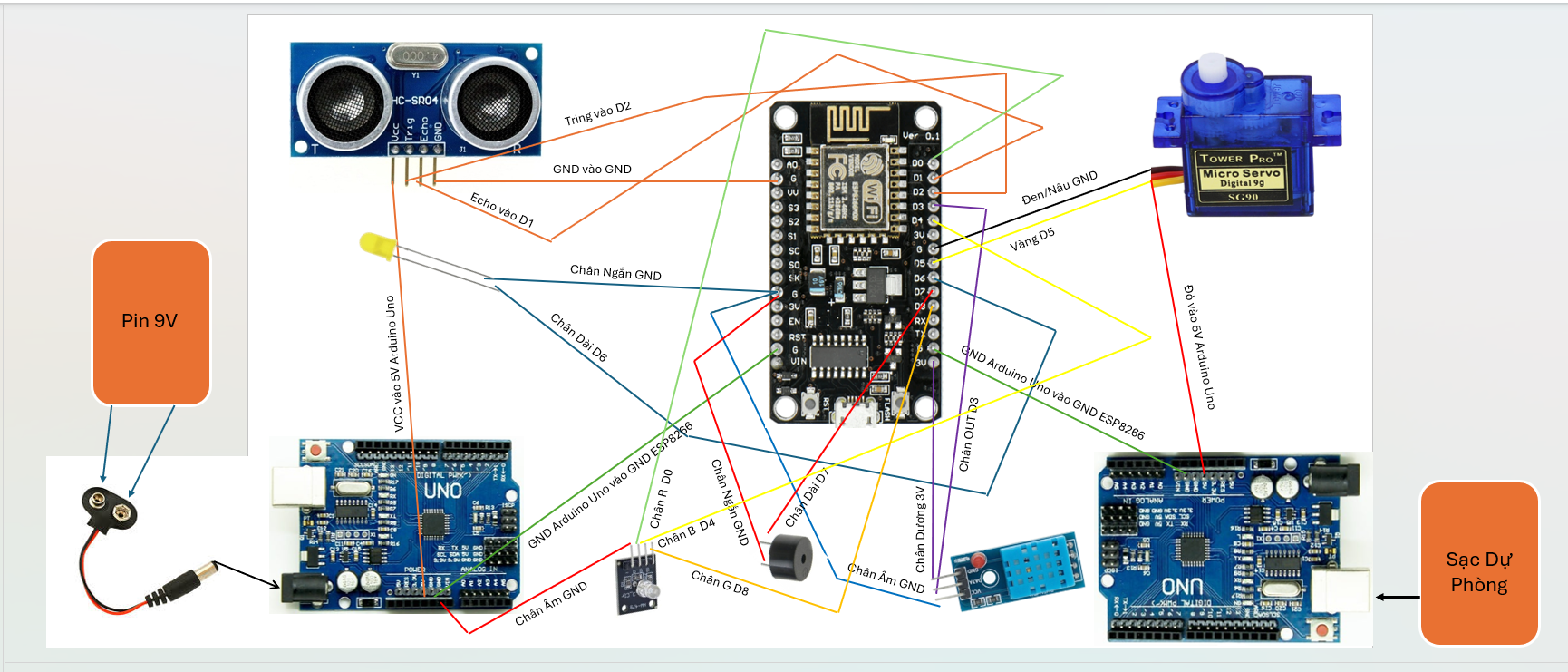
### Tầng Truyền Thông (Communication Layer)

* **Wifi:** Kết nối internet
* **MQTTX:** Kết nối với **ESP8266** thông qua wifi và phải cùng cổng IP: 192.168.x.y

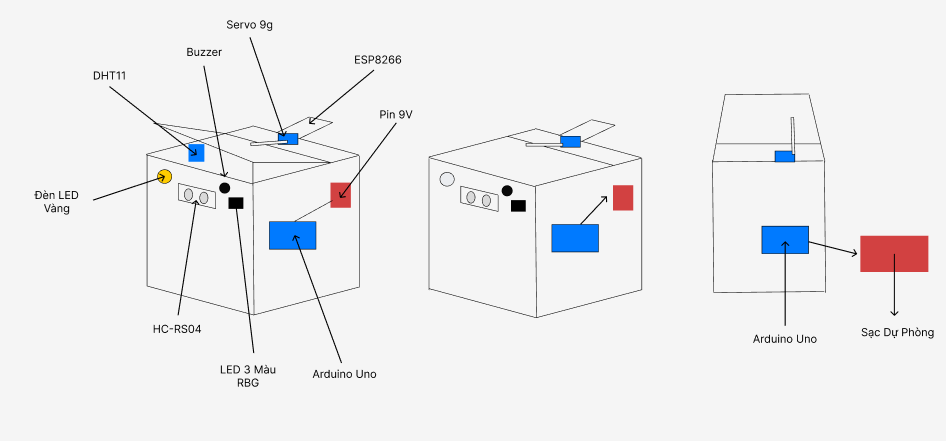
### Tầng Hiển Thị Và Điều Khiển (UI/Control Layer)

* **FlatForm FUXA:** Thùng rác thông minh sử dụng FUXA để hiển thị dữ liệu và điều khiển từ xa. Hệ thống điều khiển qua MQTT, FUXA và điện thoại – cùng WIFI (http://192.168.x.y:1881).
* Sơ Đồ Kết Nối

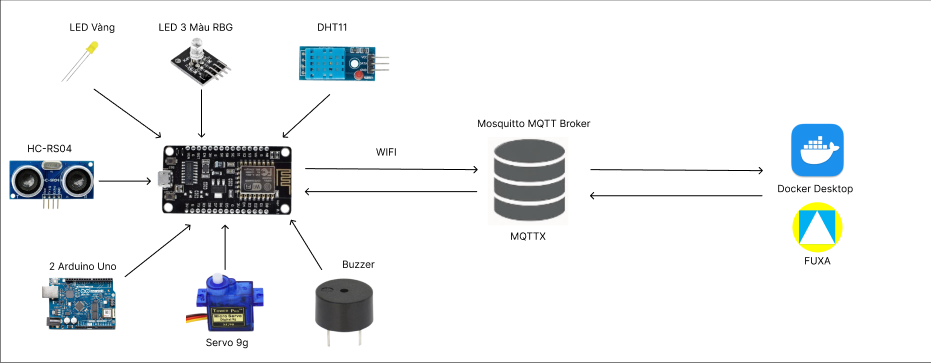
Vẽ sơ đồ kết nối từ thiết bị đã lựa chọn



## Sơ Đồ Vị Trí



## Sơ Đồ Kết Nối Phần Cứng & Phần Mềm



# KẾT QUẢ TRIỂN KHAI

Demo mô hình thực tế nhóm đã lắp đặt và triển khai



# KẾT LUẬN – KIẾN NGHỊ

## Đánh Giá Kết Quả Đạt Được

So sánh hệ thống đã triển khai so với hệ thống khác :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tiêu chí** | **Hệ thống của nhóm** | **Serverless (AWS IOT)** | **Edge Computing Thuần Túy** | **Hybrid (Edge + Cloud)** |
| **Chi phí** | Rất thấp (tự host, mã nguồn mở) | Cao (trả phí theo request/data) | Không phát sinh (offline hoàn toàn) | Trung bình (cloud + edge) |
| **Hiệu năng** | Ổn định (phụ thuộc vào WiFi và server local) | Cao (AWS dùng hệ thống toàn cầu) | Real-time (xử lý tại thiết bị) | Cao (kết hợp ưu điểm cả hai) |
| **Khả năng tự mở rộng** | Hạn chế (phụ thuộc vào cấu hình server host) | Vô hạn (auto-scaling) | Cứng (phụ thuộc phần cứng ESP) | Tốt (cloud hỗ trợ mở rộng) |
| **Bảo mật** | Tùy chỉnh (cần cấu hình TLS, MQTT auth) | Mạnh (tích hợp sẵn IAM, KMS) | Yếu (phụ thuộc WiFi) | Mạnh (sử dụng cloud security) |
| **Offline Support** | Có (chạy local, không cần internet) | Không hỗ trợ offline | Toàn bộ (hoạt động độc lập) | Một phần (edge xử lý offline) |
| **Tích hợp AI/ML** | Khó (phải tự cấu hình server chạy AI) | Dễ (AWS SageMaker, Greengrass) | Hạn chế (chạy TinyML đơn giản) | Tốt (AI chia tải cloud + thiết bị) |
| **Quản lý từ xa** | Có (FUXA Dashboard tự host) | Có (AWS IoT Core Dashboard) | Không có | Dashboard kết hợp |
| **Độ phức tạp** | Trung bình (cần biết Docker, cấu hình MQTT, FUXA) | Cao (học curve cloud) | Đơn giản (chỉ ESP8266) | Rất cao (quản lý đa nền tảng, đồng bộ) |

Bảng ‑ Đánh giá kết quả đạt được

### Ưu Điểm Nổi Bật

• **Kiểm soát dữ liệu tuyệt đối**: Không phụ thuộc nhà cung cấp dịch vụ bên thứ ba. Dữ liệu được lưu trữ cục bộ → Phù hợp với các ứng dụng yêu cầu mức độ riêng tư cao như thùng rác thông minh trong gia đình.  
• **Chi phí tối ưu**: Sử dụng nguồn điện nhỏ từ máy tính để vận hành. Phần mềm mã nguồn mở như Mosquitto và FUXA.  
• **Tích hợp linh hoạt**: FUXA hỗ trợ giao diện trực quan, dễ dàng thêm cảm biến mới và giám sát thiết bị qua MQTT.  
• **Phù hợp cho nghiên cứu học thuật**: Minh họa rõ ràng kiến trúc IoT 4 tầng:  
Cảm biến → Xử lý (ESP8266) → Truyền thông (MQTT) → Điều khiển (FUXA).

### Hạn Chế Khắc Phục

• **Khả năng mở rộng hạn chế**: Vì sử dụng FUXA chạy cục bộ, khi số lượng thiết bị và dữ liệu tăng cao, hệ thống có thể bị chậm hoặc quá tải do giới hạn tài nguyên.  
• **Phụ thuộc vào hạ tầng mạng**: Vì FUXA hoạt động qua mạng, nếu mất kết nối WiFi, hệ thống có thể ngừng hoạt động (trừ khi Mosquitto chạy ở chế độ offline).  
• **Thiếu tính năng AI nâng cao**: Vì FUXA chủ yếu dùng để giám sát và điều khiển, không hỗ trợ xử lý các mô hình AI hoặc học máy phức tạp.

### So Sánh Với Các Giải Pháp Khác

**FUXA Local (Server tại chỗ)**

* **Ưu điểm**: Hoạt động ổn định không cần Internet, dữ liệu được lưu trữ cục bộ, phù hợp ứng dụng cá nhân hoặc nghiên cứu.
* **Nhược điểm**: Khó mở rộng quy mô lớn, hạn chế tích hợp AI/ML.
* **Phù hợp**: Hệ thống đơn giản như thùng rác thông minh, giám sát tại chỗ.

**Edge Computing (ESP8266 tự xử lý)**

* **Ưu điểm**: Phản hồi nhanh, có thể hoạt động offline nếu MQTT chạy local.
* **Nhược điểm**: Không có giao diện điều khiển nếu không dùng FUXA, khó kiểm soát và cấu hình từ xa.
* **Phù hợp**: Tình huống yêu cầu phản hồi tức thì (mở nắp, cảnh báo còi, đèn...).

**Hybrid (ESP8266 + FUXA + MQTT)**

* **Ưu điểm**: Vừa xử lý nhanh tại chỗ, vừa có giao diện trực quan qua FUXA, dữ liệu có thể giám sát từ xa.
* **Nhược điểm**: Cấu hình phức tạp hơn, cần đồng bộ nhiều thành phần.
* **Phù hợp**: Các ứng dụng IoT học thuật hoặc demo minh họa mô hình IoT đầy đủ.

### Đề Xuất Cải Tiến

• **Kết hợp xử lý tại chỗ và hiển thị từ xa**: ESP8266 xử lý các tác vụ đơn giản như mở nắp, bật đèn, còi báo tại chỗ để đảm bảo phản hồi real-time. Dữ liệu như nhiệt độ, độ ẩm được gửi lên FUXA để hiển thị và giám sát từ xa qua MQTT.

• **Nâng cấp bảo mật**: Có thể triển khai MQTT over TLS để mã hóa dữ liệu và bảo vệ kết nối. Thiết lập mạng riêng (VLAN) cho các thiết bị IoT để tách biệt với hệ thống nội bộ.

• **Hạn chế AI tại biên**: Do tài nguyên hạn chế, ESP8266 không thể chạy TinyML, tuy nhiên có thể tích hợp thuật toán đơn giản như phát hiện khoảng cách hoặc trạng thái đóng/mở để xử lý tại chỗ.

• **Tối ưu năng lượng**: ESP8266 có thể sử dụng chế độ Deep Sleep khi không hoạt động để tiết kiệm điện năng, phù hợp với các thiết bị chạy pin hoặc năng lượng hạn chế.

**🡪 Kết luận**: Hệ thống thùng rác thông minh sử dụng ESP8266 và FUXA là giải pháp phù hợp cho mục đích học thuật hoặc triển khai quy mô nhỏ nhờ chi phí thấp, dễ kiểm soát và không phụ thuộc nền tảng bên thứ ba. Để mở rộng trong thực tế, cần xem xét thêm các yếu tố về bảo mật, tối ưu năng lượng và tích hợp xử lý nâng cao tại biên.

**PHỤ LỤC**

**Code Trong File Docker Compose**

version: '3.8'

services:

  fuxa:

    image: frangoteam/fuxa:latest

    container\_name: fuxa

    ports:

      - "1881:1881"

    volumes:

      - fuxa\_data:/app/server/db

    restart: unless-stopped

  mosquitto:

    image: eclipse-mosquitto:latest

    container\_name: mqtt

    ports:

      - "1883:1883"      # MQTT broker port

      - "9001:9001"      # WebSocket (optional)

    volumes:

      - ./mosquitto/config:/mosquitto/config

      - ./mosquitto/data:/mosquitto/data

      - ./mosquitto/log:/mosquitto/log

    restart: unless-stopped

volumes:

  fuxa\_data:

**Code trong file ino trong Arduino**

#include <Servo.h>

#include <DHT.h>

#include <ESP8266WiFi.h>

#include <PubSubClient.h>

// ====== Servo & Cảm biến ======

Servo myservo;

#define GOC\_DONG 0

#define GOC\_MO 130

int trigPin = D2;

int echoPin = D1;

int servoPin = D5;

#define BUZZER\_PIN D7

#define LED\_PIN D6

#define DHTPIN D3

#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);

// ====== RGB LED Pins ======

#define RED\_PIN D0

#define GREEN\_PIN D8

#define BLUE\_PIN D4

int openCount = 0;  // Thêm dòng này

unsigned long previousMillisLED = 0;  // Lưu thời gian thay đổi LED

int ledState = 0;  // Trạng thái LED, sẽ thay đổi từ 0 -> 1 -> 2 -> 0 ->...

bool overheatTriggered = false;

unsigned long previousMillis = 0;

unsigned char autoTrigger = 0;

unsigned long autoMillis = 0;

// ====== WiFi & MQTT ======

const char\* ssid = "KTX MINH CHAU 25-29";

const char\* password = "Mc270579@";

const char\* mqtt\_server = "192.168.100.121";

WiFiClient espClient;

PubSubClient client(espClient);

// ====== Kết nối WiFi ======

void setup\_wifi() {

  delay(10);

  Serial.println("Đang kết nối WiFi...");

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL\_CONNECTED) {

    Serial.println("WiFi bị mất kết nối. Đang kết nối lại...");

    delay(500);

    Serial.print(".");

  }

  Serial.println("");

  Serial.println("Đã kết nối WiFi!");

  Serial.print("IP của ESP8266: ");

  Serial.println(WiFi.localIP());

}

// ====== Kết nối lại MQTT nếu bị mất ======

void reconnect() {

  while (!client.connected()) {

    Serial.print("Đang kết nối MQTT...");

    if (client.connect("ESP8266Client")) {

      Serial.println(" => Đã kết nối MQTT!");

      client.subscribe("trashbin/control");

    } else {

      Serial.println(" => Thất bại, thử lại sau 5s.");

      Serial.print("Mã lỗi MQTT: ");

      Serial.println(client.state());

      delay(5000);

    }

  }

}

// ====== Callback khi nhận lệnh từ MQTT ======

void callback(char\* topic, byte\* payload, unsigned int length) {

  String message = "";

  for (unsigned int i = 0; i < length; i++) {

    message += (char)payload[i];

  }

  Serial.print("Nhận lệnh từ MQTT: ");

  Serial.println(message);

  if (String(topic) == "trashbin/control") {

    if (message == "open") {

      myservo.write(GOC\_MO);

      digitalWrite(LED\_PIN, HIGH);

      tone(BUZZER\_PIN, 1500);

      delay(100);

      noTone(BUZZER\_PIN);

      client.publish("trashbin/servo", "open");

      client.publish("trashbin/buzzer", "on");

      client.publish("trahbin/led", "on");

      openCount++;

      Serial.print("Số lần mở nắp (MQTT): ");

      Serial.println(openCount);

      // Gửi lên MQTT

      String countStr = String(openCount);

      client.publish("trashbin/opencount", countStr.c\_str());

    }

    else if (message == "close") {

      myservo.write(GOC\_DONG);

      digitalWrite(LED\_PIN, LOW);

      noTone(BUZZER\_PIN); // tắt buzzer phòng trường hợp đang kêu

      client.publish("trashbin/servo", "close");

      client.publish("trashbin/buzzer", "off");

      client.publish("trashbin/led", "off");

    }

  }

}

// ====== Cảm biến khoảng cách ======

long getDistance() {

  digitalWrite(trigPin, LOW);

  delayMicroseconds(2);

  digitalWrite(trigPin, HIGH);

  delayMicroseconds(10);

  digitalWrite(trigPin, LOW);

  long duration = pulseIn(echoPin, HIGH);

  return duration \* 0.034 / 2; // đổi ra cm

}

// ====== Cài đặt ban đầu ======

void setup() {

  Serial.begin(115200);

  setup\_wifi();

  client.setServer(mqtt\_server, 1883);

  client.setCallback(callback);

  pinMode(trigPin, OUTPUT);

  pinMode(echoPin, INPUT);

  pinMode(BUZZER\_PIN, OUTPUT);

  myservo.attach(servoPin);

  myservo.write(GOC\_DONG);

  pinMode(LED\_PIN, OUTPUT);

  digitalWrite(LED\_PIN, LOW);

  dht.begin();

  String json = "{\"temperature\":25.3}";

  client.publish("trashbin/temperature", json.c\_str());

  pinMode(BLUE\_PIN, OUTPUT);  // Đổi LED\_BLUE\_PIN thành BLUE\_PIN

  pinMode(GREEN\_PIN, OUTPUT); // Đổi LED\_GREEN\_PIN thành GREEN\_PIN

  pinMode(RED\_PIN, OUTPUT);   // Đổi LED\_RED\_PIN thành RED\_PIN

}

// ====== Vòng lặp chính ======

void loop() {

  if (!client.connected()) reconnect();

  client.loop();

  if (WiFi.status() != WL\_CONNECTED) {

    Serial.println("WiFi bị mất kết nối. Đang kết nối lại...");

    setup\_wifi();

  }

  if (millis() - previousMillis >= 200) {

    previousMillis = millis();

    int distance = getDistance();

    Serial.print("Khoảng cách đo được: ");

    Serial.println(distance);

    Serial.println(" cm");

    static int lastDistance = -1;

    if (abs(distance - lastDistance) > 2) {

      lastDistance = distance;

      String msg = String(distance);

      Serial.print("Gửi lên MQTT: ");

      Serial.println(msg);

      if (client.publish("trashbin/distance", msg.c\_str())) {

        Serial.println("Đã gửi lên MQTT!");

      } else {

        Serial.println("Không thể gửi dữ liệu lên MQTT!");

      }

    }

    // Tự động mở nắp nếu có người lại gần

    if (distance < 10) {

      autoTrigger = 1;

      autoMillis = millis();

      myservo.write(GOC\_MO);

      tone(BUZZER\_PIN, 1500);

      delay(100);

      noTone(BUZZER\_PIN);

      digitalWrite(LED\_PIN, HIGH);

      client.publish("trashbin/servo", "open");

      client.publish("trashbin/buzzer", "on");

      client.publish("trashbin/led", "on");

      openCount++;

      Serial.print("Số lần mở nắp (MQTT): ");

      Serial.println(openCount);

      // Gửi lên MQTT

      String countStr = String(openCount);

      client.publish("trashbin/opencount", countStr.c\_str());

    }

  }

  // Đóng nắp sau 2 giây nếu đã mở tự động

  if (millis() - autoMillis >= 2000 && autoTrigger == 1) {

    autoTrigger = 0;

    myservo.write(GOC\_DONG);

    digitalWrite(LED\_PIN, LOW);

    client.publish("trashbin/servo", "close");

    client.publish("trashbin/buzzer", "off");

    client.publish("trashbin/led", "off");

  }

  // Đọc nhiệt độ/độ ẩm

  unsigned long lastTempHumMillis = 0;

  const unsigned long tempHumInterval = 1000;

  float temperature = dht.readTemperature();

  float humidity = dht.readHumidity();

  if (!isnan(temperature) && !isnan(humidity)) {

    String tempStr = String(temperature);

    String humStr = String(humidity);

    Serial.print("Nhiệt độ: ");

    Serial.print(tempStr);

    Serial.print(" °C, Độ ẩm: ");

    Serial.print(humStr);

    Serial.println(" %");

    client.publish("trashbin/temperature", tempStr.c\_str());

    client.publish("trashbin/humidity", humStr.c\_str());

  } else {

    Serial.println("Không đọc được dữ liệu từ DHT11");

  }

  // 🚨 Nếu nhiệt độ ≥ 40 độ C => cảnh báo

  if (temperature >= 40 && !overheatTriggered) {

    overheatTriggered = true;

    Serial.println("🔥 CẢNH BÁO: Nhiệt độ quá cao! MỞ NẮP!");

    myservo.write(GOC\_MO);

    digitalWrite(LED\_PIN, HIGH);

    tone(BUZZER\_PIN, 1000); // kêu liên tục

    client.publish("trashbin/servo", "open");

    client.publish("trashbin/buzzer", "on");

    client.publish("trashbin/led", "on");

    openCount++;

    Serial.print("Số lần mở nắp (MQTT): ");

    Serial.println(openCount);

    // Gửi lên MQTT

    String countStr = String(openCount);

    client.publish("trashbin/opencount", countStr.c\_str());

  }

  // Nếu nhiệt độ giảm < 40 => tắt cảnh báo (tùy bạn muốn giữ mở hay tự đóng lại)

  if (temperature < 40 && overheatTriggered) {

    overheatTriggered = false;

    Serial.println("✅ Nhiệt độ bình thường trở lại");

    noTone(BUZZER\_PIN); // tắt còi

    digitalWrite(LED\_PIN, LOW);

    myservo.write(GOC\_DONG); // Đóng nắp nếu muốn

    client.publish("trashbin/servo", "close");

    client.publish("trashbin/buzzer", "off");

    client.publish("trashbin/led", "off");

  }

  // Thay đổi màu LED sau mỗi 2 giây

  if (millis() - previousMillisLED >= 2000) {

    previousMillisLED = millis(); // Cập nhật thời gian

  // Tắt tất cả các LED

    digitalWrite(BLUE\_PIN, LOW);  // Đổi LED\_BLUE\_PIN thành BLUE\_PIN

    digitalWrite(GREEN\_PIN, LOW); // Đổi LED\_GREEN\_PIN thành GREEN\_PIN

    digitalWrite(RED\_PIN, LOW);   // Đổi LED\_RED\_PIN thành RED\_PIN

  // Đổi trạng thái LED theo giá trị của ledState

    if (ledState == 0) {

      digitalWrite(BLUE\_PIN, HIGH);  // Bật LED xanh dương

      digitalWrite(GREEN\_PIN, LOW);

      digitalWrite(RED\_PIN, LOW);

      client.publish("trashbin/3led", "BLUE");

      ledState = 1;

    } else if (ledState == 1) {

      digitalWrite(GREEN\_PIN, HIGH);  // Bật LED xanh lá

      digitalWrite(BLUE\_PIN, LOW);

      digitalWrite(RED\_PIN, LOW);

      client.publish("trashbin/3led", "GREEN");

      ledState = 2;

    } else if (ledState == 2) {

      digitalWrite(RED\_PIN, HIGH);  // Bật LED đỏ

      digitalWrite(BLUE\_PIN, LOW);

      digitalWrite(GREEN\_PIN, LOW);

      client.publish("trashbin/3led", "RED");

      ledState = 0;

    }

  }

}