Guia 1

ANOTAÇÕES SOBRE OS DOIS CÓDIGOS

**Código 1**

#include <WiFi.h> // Importa a biblioteca para funções Wi-Fi no ESP32

void setup() {

Serial.begin(115200); // Inicia a comunicação serial com o computador na velocidade de 115200 bps

Serial.println("Obtendo endereço MAC..."); // Exibe uma mensagem no Monitor Serial

Serial.print("MAC address: "); // Imprime o texto antes de mostrar o MAC

Serial.println(WiFi.macAddress()); // Obtém e imprime o endereço **MAC** do ESP32

}

**MAC** :

É um **identificador único** atribuído à interface de rede do dispositivo — nesse caso, ao módulo Wi-Fi do seu ESP32.

É como se fosse o "RG" ou "CPF" da sua placa de rede, único para cada dispositivo no mundo.

Esse endereço é usado para comunicação na camada física da rede local (LAN), ajudando os dispositivos a se reconhecerem e se comunicarem dentro da mesma rede.

* A função do código é para saber o endereço MAC do seu ESP32, útil para identificar o dispositivo na rede ou configurar comunicações Wi-Fi.

Guia 2

**Código 2:**

#include <WiFi.h> // Biblioteca para conexão Wi-Fi no ESP32

#include <PubSubClient.h> // Biblioteca para comunicação MQTT

// Credenciais da rede Wi-Fi e endereço do servidor MQTT

const char\* ssid = "REDEWORK"; // Nome da rede Wi-Fi

const char\* password = "Acessonet05"; // Senha da rede Wi-Fi

const char\* mqtt\_server = "192.168.1.4"; // Endereço IP do broker MQTT (no caso, local)

Aqui mostra onde a rede wifi deve se conectar

WiFiClient espClient; // Cria um cliente Wi-Fi

PubSubClient client(espClient); // Cria um cliente MQTT usando o Wi-Fi

unsigned long lastMsg = 0; // Variável para controle de tempo de envio de mensagens

#define MSG\_BUFFER\_SIZE (50) // Define o tamanho máximo do buffer da mensagem

char msg[MSG\_BUFFER\_SIZE]; // Buffer para mensagens MQTT

int value = 0; // Contador simples usado para enviar mensagens numeradas

**WiFiClient espClient:** objeto que cria a conexão de rede básica via Wi-Fi.

**PubSubClient client:** cliente MQTT que usa espClient para enviar e receber dados.

**lastMsg** é usado para controlar quando a próxima mensagem deve ser enviada.

**msg** é um espaço para armazenar mensagens antes de enviá-las.

**value** é um contador que aumenta a cada mensagem enviada para criar mensagens únicas.

// Função que conecta o ESP32 à rede Wi-Fi

void setup\_wifi() {

delay(10); // Pequeno atraso para estabilidade

Serial.println();

Serial.print("Connecting to ");

Serial.println(ssid); // Mostra qual rede está tentando conectar

WiFi.mode(WIFI\_STA); // Define o modo estação (cliente)

WiFi.begin(ssid, password); // Inicia a conexão Wi-Fi

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

delay(500); // Espera meio segundo

Serial.print("."); // Imprime ponto enquanto tenta conectar

}

randomSeed(micros()); // Gera uma semente de números aleatórios baseada no tempo atual

Serial.println("");

Serial.println("WiFi connected"); // Conexão bem-sucedida

Serial.println("IP address: ");

Serial.println(WiFi.localIP()); // Mostra o IP atribuído ao ESP

}

// Função chamada automaticamente quando chega uma mensagem MQTT no tópico inscrito

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

Serial.print("Message arrived [");

Serial.print(topic); // Imprime o tópico da mensagem recebida

Serial.print("] ");

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

Serial.print((char)payload[i]); // Imprime o conteúdo da mensagem caractere por caractere

}

Serial.println();

// Comandos recebidos para controlar LEDs com base no primeiro caractere da mensagem

if ((char)payload[0] == 'd') {

digitalWrite(2, LOW); // Liga o LED no pino 2

}

if ((char)payload[0] == 'l') {

digitalWrite(2, HIGH); // Desliga o LED no pino 2

}

if ((char)payload[0] == 'D') {

digitalWrite(4, LOW); // Liga o LED no pino 4

}

if ((char)payload[0] == 'L') {

digitalWrite(4, HIGH); // Desliga o LED no pino 4

}

}

// Função que tenta reconectar ao servidor MQTT se a conexão cair

void reconnect() {

while (!client.connected()) { // Enquanto não estiver conectado

Serial.print("Attempting MQTT connection...");

String clientId = "ESP8266Client-"; // Cria um ID único para o cliente MQTT

clientId += String(random(0xffff), HEX); // Adiciona valor aleatório ao final

if (client.connect(clientId.c\_str())) { // Tenta conectar com o client ID

Serial.println("connected");

client.publish("outTopic", "hello world"); // Envia mensagem inicial

client.subscribe("topico/timestamp/"); // Se inscreve no tópico para receber comandos

} else {

Serial.print("failed, rc=");

Serial.print(client.state()); // Mostra código de erro

Serial.println(" try again in 5 seconds");

delay(5000); // Espera 5 segundos antes de tentar de novo

}

}

}

// Função de configuração inicial

void setup() {

pinMode(2, OUTPUT); // Define o pino 2 como saída (LED)

pinMode(4, OUTPUT); // Define o pino 4 como saída (LED)

Serial.begin(115200); // Inicia a comunicação serial

setup\_wifi(); // Conecta à rede Wi-Fi

client.setServer(mqtt\_server, 1883); // Define o servidor MQTT e a porta (padrão: 1883)

client.setCallback(callback); // Define a função de callback para mensagens recebidas

}

// Função principal que roda continuamente

void loop() {

if (!client.connected()) {

reconnect(); // Reconecta ao MQTT se a conexão cair

}

client.loop(); // Mantém a conexão MQTT viva

unsigned long now = millis();

if (now - lastMsg > 2000) { // A cada 2 segundos...

lastMsg = now;

int sensorValue = analogRead(A0); // Lê o valor do pino analógico A0

Serial.print("Luminosidade: ");

Serial.print(sensorValue, DEC); // Imprime o valor lido

Serial.print(" \n");

snprintf(msg, MSG\_BUFFER\_SIZE, "%ld", sensorValue); // Converte o valor lido em texto

client.publish("topico/analogico1/", msg); // Publica o valor no tópico MQTT

++value;

snprintf(msg, MSG\_BUFFER\_SIZE, "hello world #%ld", value); // Cria uma mensagem de contagem

Serial.print("Publish message: ");

Serial.println(msg);

client.publish("outTopic", msg); // Publica no tópico "outTopic"

}

}