E-DOMO

Anderley Riaño Polania 20192573010

Harry Giovanny Cadena León 20192573029

Universidad Distrital Francisco José De Caldas

Resumen

[El resumen ha de tener una longitud de un párrafo de entre 150 y 250 palabras, sin sangría. Los títulos de sección, como la palabra *Resumen* anterior, no se consideran títulos, por lo que no se usa formato de título en negrita. En su lugar, use el estilo Título de sección. Este estilo inicia automáticamente la sección en una nueva página, por lo que no es necesario que agregue saltos de página. (Para ver el documento con la paginación, seleccione la pestaña Vista y haga clic en Vista de lectura). Tenga en cuenta que todos los estilos de texto de esta plantilla están disponibles en la pestaña Inicio de la cinta de opciones, en la galería de estilos].

Palabras clave: IOT, Redis, Grafana, MQTT, Node Red, Raspberry pi4, Esp32

Objetivo General

Implementar un sistema de control de electrodomésticos que permita mejorar la comodidad y la eficiencia energética del hogar mediante el uso de dispositivos inteligentes conectados a Internet, utilizando la tecnología IoT (Internet of Things).

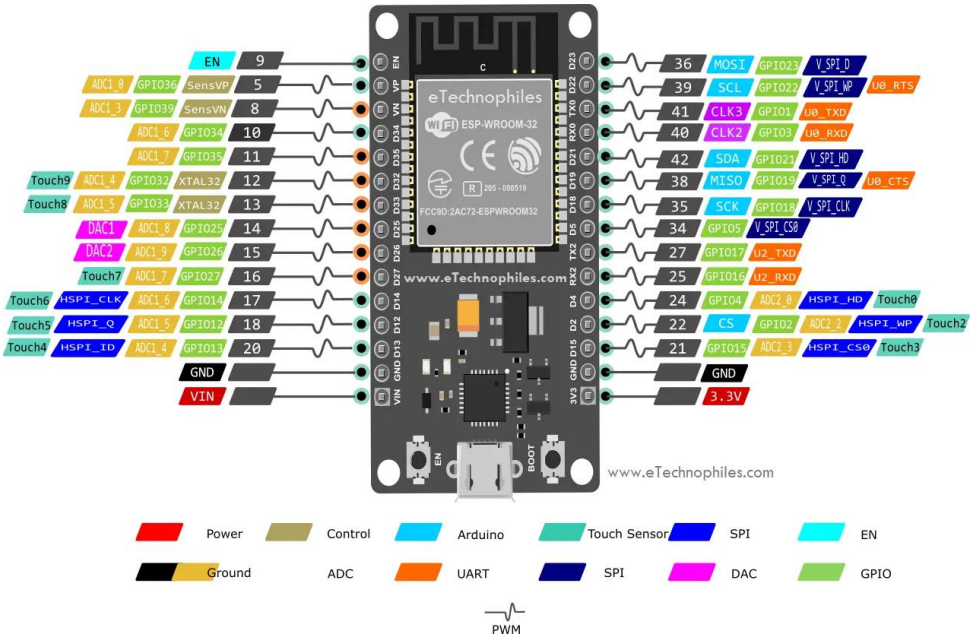
Objetivos Específicos

- Configurar los dispositivos IoT, como sensores, termostatos, medidores de energía y dispositivos de control de electrodomésticos, para que funcionen correctamente y estén conectados a la red Wi-Fi.
- Desarrollar un control inalámbrico que permita controlar los electrodomésticos de manera remota desde cualquier lugar y programar su funcionamiento para optimizar la eficiencia energética. La aplicación también debe contar con medidas de seguridad adecuadas para proteger la privacidad del usuario y evitar el acceso no autorizado a los dispositivos.
- Establecer un sistema de automatización que permita a partir de las preferencias del usuario tomar decisiones en función de las condiciones del hogar, con el objetivo de mejorar la eficiencia energética y la comodidad de este.

Marco Teórico

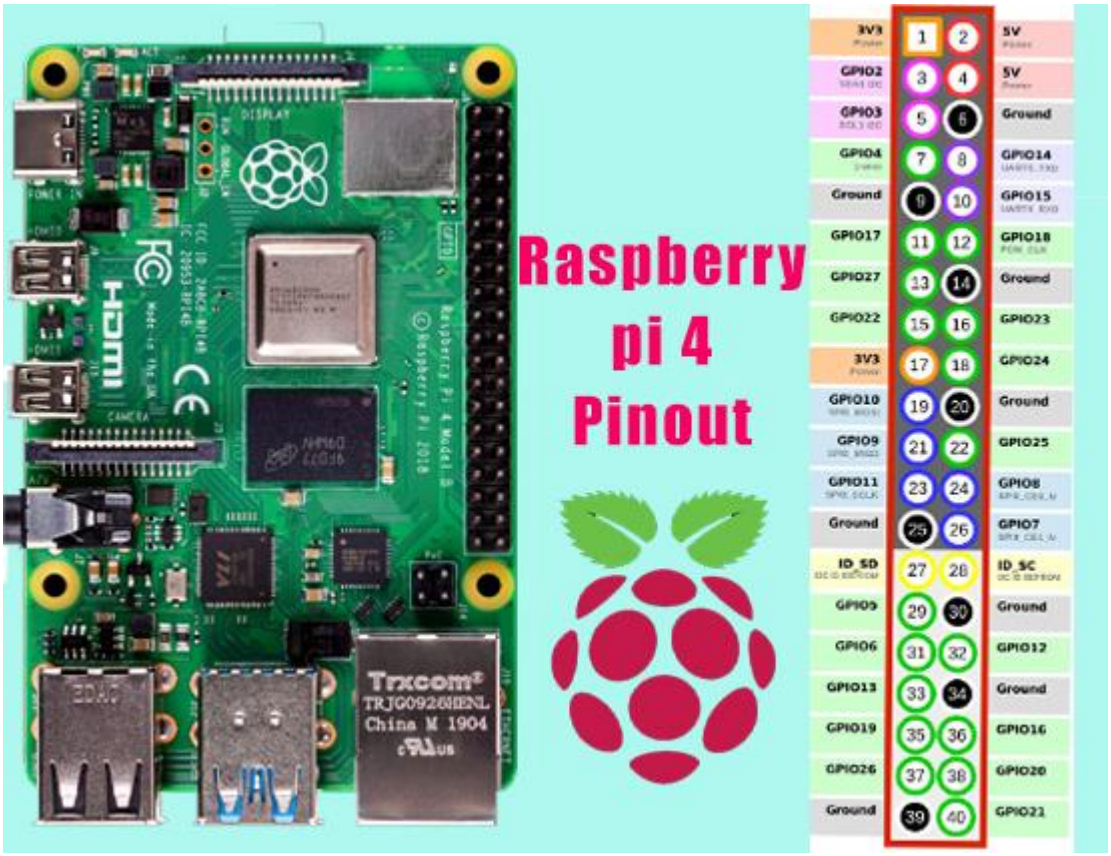
ESP32

El ESP32 es un microcontrolador de bajo costo y alto rendimiento diseñado para aplicaciones de Internet de las cosas (IoT). Cuenta con capacidades de conectividad Wi-Fi y Bluetooth, lo que permite la comunicación inalámbrica con otros dispositivos y servicios en la nube. El ESP32 es ampliamente utilizado en proyectos de IoT debido a su versatilidad, bajo consumo de energía y amplio soporte de desarrollo.



Raspberry Pi 4

La Raspberry Pi 4 es una computadora de placa única (SBC) de bajo costo y alto rendimiento. Posee un procesador de cuatro núcleos, memoria RAM y puertos USB, HDMI, Ethernet, entre otros. La Raspberry Pi 4 es muy popular para proyectos de IoT debido a su capacidad para ejecutar sistemas operativos completos, así como su amplia gama de puertos y conectividad.



NODE-*RED*

Node-RED es un entorno de desarrollo de código abierto basado en Node.js que permite la creación de flujos de trabajo visualmente intuitivos para la automatización y la integración de sistemas. Utiliza una interfaz gráfica basada en nodos y cables para conectar diferentes servicios, sensores y actuadores, lo que facilita el desarrollo de aplicaciones de IoT. Node-RED se ejecuta en la Raspberry Pi 4 y se integra fácilmente con el ESP32 y otros dispositivos.

Redis

Redis es una base de datos en memoria de código abierto y estructura de datos clave-valor. Proporciona un almacenamiento rápido y eficiente para datos temporales y cachés, lo que es especialmente útil en aplicaciones de IoT donde se requiere una alta velocidad de

procesamiento de datos. Redis puede ejecutarse en la Raspberry Pi 4 y permite la comunicación rápida y eficiente entre diferentes componentes del sistema.

Python

Python es un lenguaje de programación de alto nivel ampliamente utilizado en el desarrollo de aplicaciones de IoT y ciencia de datos. Es conocido por su simplicidad y legibilidad, así como por su amplia variedad de bibliotecas y herramientas para el procesamiento y análisis de datos. Tanto el ESP32 como la Raspberry Pi 4 admiten Python, lo que facilita la programación y la integración de los dispositivos.

Ciencia de Datos con Python

Python también es ampliamente utilizado en el campo de la ciencia de datos. Con bibliotecas populares como NumPy, Pandas, Matplotlib y scikit-learn, los desarrolladores pueden realizar análisis de datos, aprendizaje automático y visualización de manera eficiente. La Raspberry Pi 4, en combinación con Node-RED y Redis, puede servir como plataforma de recopilación y procesamiento de datos para aplicaciones de ciencia de datos.

Internet de las Cosas (IoT)

El Internet de las Cosas (IoT) se refiere a la red de dispositivos físicos conectados que pueden intercambiar datos entre sí y con servicios en la nube. Los dispositivos IoT, como el ESP32 y la Raspberry Pi 4, se utilizan para recopilar información del entorno a través de sensores y enviarla a través de conexiones inalámbricas. La integración de IoT en el marco de este proyecto permite controlar y monitorear los electrodomésticos del hogar de forma remota, mejorando la comodidad y la eficiencia energética.

Ciudades Inteligentes

Las ciudades inteligentes son entornos urbanos que utilizan tecnología y datos para mejorar la calidad de vida de los ciudadanos, la eficiencia de los servicios públicos y la sostenibilidad ambiental. En el contexto de este proyecto, la integración de dispositivos IoT, Node-RED, Redis y Python en la Raspberry Pi 4 permite desarrollar soluciones para la creación de una ciudad inteligente. Esto implica la monitorización y gestión de diferentes aspectos, como el consumo de energía, la calidad del aire, la gestión de residuos, el tráfico y la iluminación pública.

Proceso de implementación

1. Diseño del sistema

- Definir los requisitos y objetivos del sistema, incluyendo la funcionalidad deseada y los resultados esperados.
- Realizar un análisis de los componentes necesarios y su integración, teniendo en cuenta el sensor DHT11, el ESP32, Node-RED, Raspberry Pi, Redis, Grafana y Python.
- Diseñar la arquitectura del sistema, identificando las interconexiones y flujos de datos entre los diferentes componentes.

2. Configuración y programación del hardware:

- Realizar las conexiones físicas necesarias entre el sensor DHT11, el ESP32 y la Raspberry Pi, siguiendo las especificaciones del fabricante y los estándares de conexión.
- Programar el ESP32 para leer los datos del sensor DHT11 y enviarlos a través de MQTT (Message Queuing Telemetry Transport) a Node-RED.

3. Configuración de Node-RED:

- Instalar y configurar Node-RED en la Raspberry Pi, asegurándose de que esté conectada a la misma red que el ESP32.
- Crear un flujo en Node-RED que reciba los datos MQTT del ESP32 y los almacene en una base de datos local o en Redis.

4. Configuración de Redis y Grafana:

- Instalar y configurar Redis en la Raspberry Pi, estableciendo los parámetros de conexión y configurando la persistencia de datos.
- Configurar Grafana para conectarse a Redis como fuente de datos y crear paneles personalizados para visualizar los datos de temperatura y humedad en tiempo real.

5. Desarrollo del análisis de datos con Python:

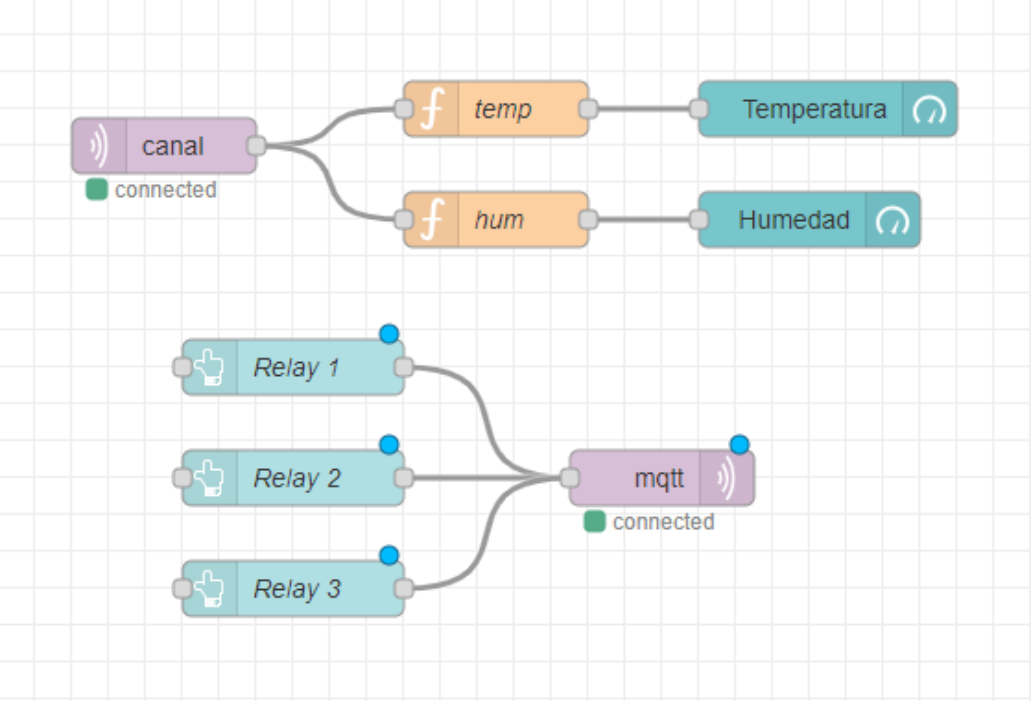
- Desarrollar scripts en Python que se conecten a Redis y recuperen los datos almacenados.
- Utilizar bibliotecas de análisis de datos en Python, como Pandas y NumPy, para realizar el análisis y procesamiento de los datos obtenidos.

6. Pruebas y ajustes del sistema:

- Realizar pruebas exhaustivas del sistema para verificar que los datos se estén capturando correctamente, se almacenen en Redis y se visualicen correctamente en Grafana.
- Ajustar y optimizar la configuración y funcionalidad del sistema según sea necesario.

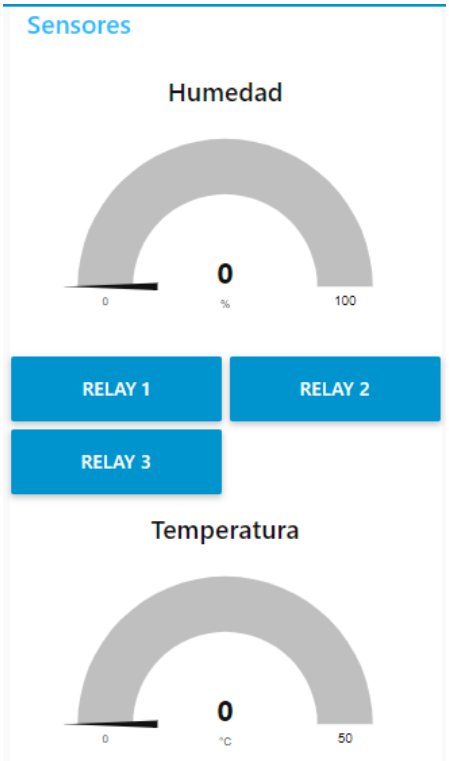
Resultados

En el Node-Red se utilizan los siguientes nodos

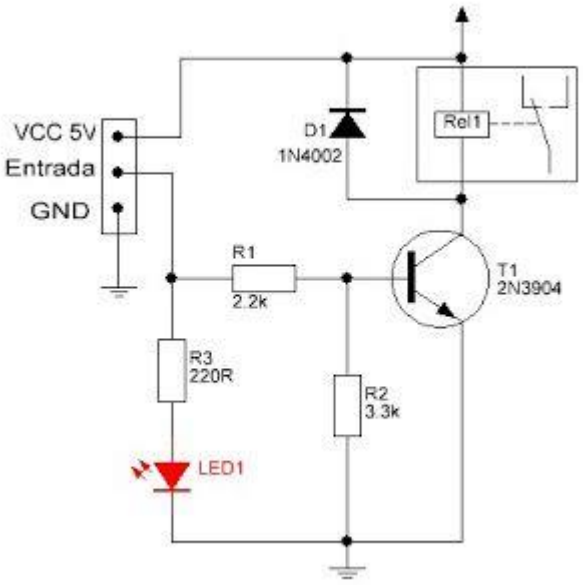


Monitoreo de temperatura y control de relés

Una interfaz que permite visualizar la temperatura y humedad de la habitación y y unos botones para el control de relevadores que funcionan como interruptores para los focos y permiten el paso a aparatos eléctricos.



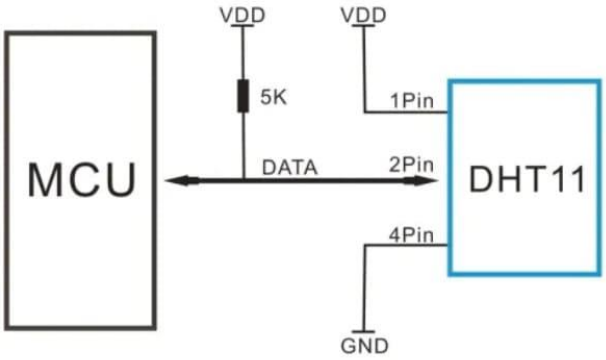
Esquemático Relé

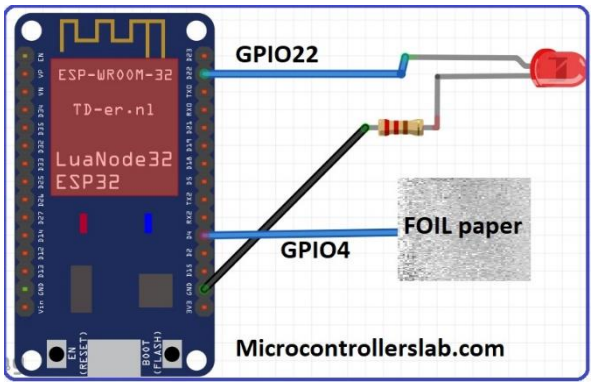


Se utilizan módulos relé donde el modulo trabaja con 5 voltios y en el caso de los relés a trabajar trabajan voltajes de 125V a 250V AC y 28V a 30V en DC a 10A.

El sensor DHT11 es un sensor de temperatura y de Humedad con tres pines uno para vcc uno para GND y uno para los datos que se enviaran al microcontrolador

dicho sensor funciona con voltajes de 3.3V y 5V



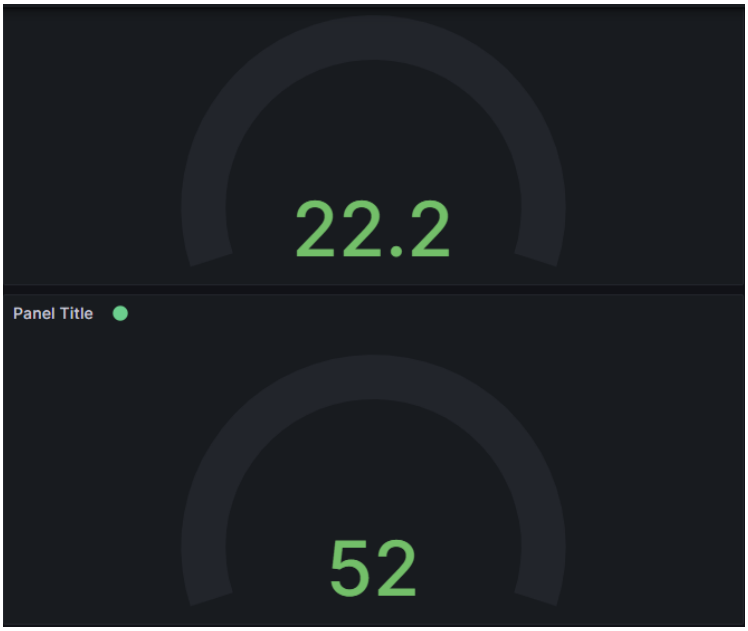


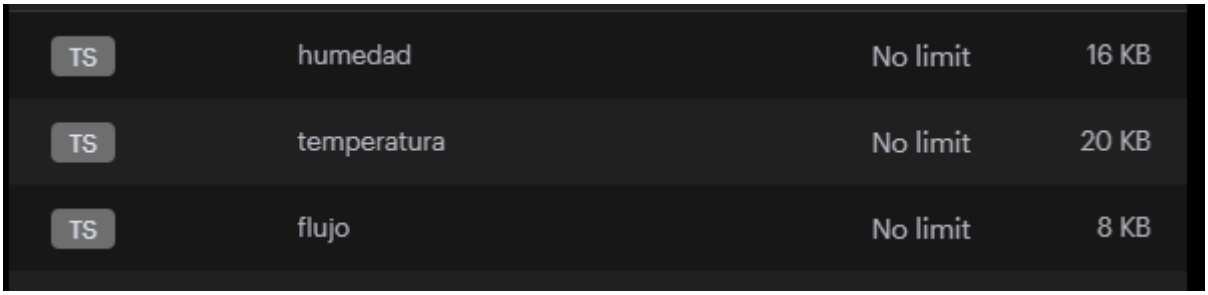
La esp32 cuenta con 9 pines Touch los cuales al sacar un cable y una pequeña placa de metal podemos transformar cualquier superficie en un botón touch los cuales a largo plazo tienen un mejor rendimiento ya que no tendrán desgaste mecánico.

sketch_jun05b §

```
#define TOUCH_PIN_1 27
#define TOUCH_PIN_2 32
#define TOUCH_PIN_3 33
void setup() { Serial.begin(9600);
  touchAttachInterrupt(TOUCH_PIN_1, handleTouch, 40);
  touchAttachInterrupt(TOUCH_PIN_2, handleTouch, 40);
  touchAttachInterrupt(TOUCH_PIN_3, handleTouch, 40);}
void loop() {}
void handleTouch() {
  if (touchRead(TOUCH_PIN_1) < 50) {Serial.println("Se ha detectado un toque en el pin 27.");}
  if (touchRead(TOUCH_PIN_2) < 50) {Serial.println("Se ha detectado un toque en el pin 32.");}
  if (touchRead(TOUCH_PIN_3) < 50) {Serial.println("Se ha detectado un toque en el pin 33.");}
}
```

Dashboard Grafana recibiendo datos de Redis





Código Arduino

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

Bibliotecas necesarias para trabajar con WiFi, MQTT (utilizado para la comunicación entre dispositivos) y el sensor DHT (utilizado para medir la temperatura y la humedad).

```
#define DHT_PIN 23
#define DHT_TIPO DHT11

#define RELAY_PIN_1 2
#define RELAY_PIN_2 15
#define RELAY_PIN_3 4

DHT dht(DHT_PIN, DHT_TIPO);

//const char* ssid = "Omni_lite611D20";
//const char* password = "major8954";
//const char* ssid = "CONEXION-RIANO";
//const char* password = "RP38203290@";
const char* ssid = "ARP";
const char* password = "12345678";
const char* mqtt_server = "broker.mqttdashboard.com";

WiFiClient espClient;
PubSubClient client(espClient);

char msg[16];

bool relayState1 = LOW;
bool relayState2 = LOW;
bool relayState3 = LOW;
```

Aquí se definen las constantes y variables utilizadas en el programa. Se establecen los pines utilizados para el sensor DHT y los relés. También se inicializan los objetos DHT y PubSubClient.

```
void setup() {
  Serial.begin(115200);
  setup_wifi();
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);

  pinMode(RELAY_PIN_1, OUTPUT);
  pinMode(RELAY_PIN_2, OUTPUT);
  pinMode(RELAY_PIN_3, OUTPUT);

  dht.begin();
}
```

En la función setup(), se inicializa la comunicación en serie (Serial.begin()) para la depuración. Luego se llama a la función setup_wifi() para establecer la conexión WiFi. A continuación, se configura el servidor MQTT y se asigna la función de devolución de llamada (callback). Se configuran los pines de los relés como salidas digitales y se inicia el sensor DHT.

```
void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();

  delay(500);
  float temperatura = dht.readTemperature();
  float humedad = dht.readHumidity();

  if (isnan(temperatura) || isnan(humedad)) {
    Serial.println("Error al leer la temperatura y la humedad");
    return;
  }

  Serial.println(temperatura);
  Serial.println(humedad);

  snprintf(msg, 16, "%.2f,%.2f", temperatura, humedad);
  client.publish("canal", msg);
  delay(10);
}
```

En el bucle principal (loop()), se verifica si el cliente MQTT está conectado. Si no lo está, se llama a la función reconnect() para establecer la conexión. A continuación, se procesan los mensajes MQTT pendientes (client.loop()).

Después de eso, se lee la temperatura y la humedad del sensor DHT. Si ocurre algún error en la lectura, se imprime un mensaje de error y se sale del bucle. De lo contrario, se imprimen los valores de temperatura y humedad en el monitor en serie.

Luego, se formatea el mensaje MQTT con la temperatura y la humedad y se publica en el canal llamado "canal". Finalmente, se agrega un pequeño retardo antes de repetir el bucle.

```
void reconnect() {
  while (!client.connected()) {
    Serial.print("Connecting to MQTT...");
    if (client.connect("ESP32")) {
      Serial.println("connected!");
      // Suscribirse a los temas relevantes en Node-RED
      client.subscribe("control/relay1");
      client.subscribe("control/relay2");
      client.subscribe("control/relay3");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" Retrying in 5 seconds...");
      delay(5000);
    }
  }
}
```

La función reconnect() se utiliza para establecer la conexión con el servidor MQTT. Si no se puede establecer la conexión, se muestra un mensaje de error en el monitor en serie y se espera 5 segundos antes de intentar nuevamente.

Cuando se establece la conexión con éxito, se muestra un mensaje de éxito y se suscribe a los temas relevantes en Node-RED para controlar los relés.

```
void setup_wifi() {
  WiFi.begin(ssid, password);

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

  Serial.println(WiFi.localIP());
}
```

La función setup_wifi() se utiliza para establecer la conexión WiFi. Se llama a WiFi.begin() con el nombre de la red y la contraseña proporcionados. Luego, se espera hasta que la conexión esté establecida (WiFi.status() == WL_CONNECTED). Mientras tanto, se muestra un punto en el monitor en serie. Una vez que la conexión está establecida, se muestra la dirección IP local en el monitor en serie.

```
void callback(char* topic, byte* payload, unsigned int length) {
  payload[length] = '\0';
  String message = String((char*)payload);

  // Control de relés
  if (strcmp(topic, "control/relay1") == 0) {
    if (message.equals("ON")) {
      relayState1 = !relayState1;  // Invierte el estado actual
      digitalWrite(RELAY_PIN_1, relayState1);
    }
  } else if (strcmp(topic, "control/relay2") == 0) {
    if (message.equals("ON")) {
      relayState2 = !relayState2;  // Invierte el estado actual
      digitalWrite(RELAY_PIN_2, relayState2);
    }
  } else if (strcmp(topic, "control/relay3") == 0) {
    if (message.equals("ON")) {
      relayState3 = !relayState3;  // Invierte el estado actual
      digitalWrite(RELAY_PIN_3, relayState3);
    }
  }
}
```

La función callback() se llama cuando se recibe un mensaje MQTT. Toma el tema (topic), la carga útil (payload) y la longitud (length) como parámetros.

En esta función, se convierte la carga útil en una cadena (String message) y se verifica el tema para determinar qué relé debe controlarse. Si el mensaje es "ON",

se invierte el estado actual del relé correspondiente (relayState1, relayState2 o relayState3) y se actualiza el estado del relé (digitalWrite()) según el estado invertido.

Esto permite controlar los relés enviando mensajes MQTT con "ON" en los temas "control/relay1", "control/relay2" o "control/relay3".

Referencias

BricoGeek. (s. f.). Introducción a MQTT con Node-RED, Raspberry Pi y Arduino: Ejemplo MQTT con Arduino. Recuperado de https://lab.bricogeek.com/tutorial/introduccion-a-mqtt-con-node-red-raspberry-pi-y-arduino/ejemplo-mqtt-con-arduino

Real Python. (s. f.). Data Science Tutorials. Recuperado de https://realpython.com/tutorials/data-science/

Pimylifeup. (s. f.). Raspberry Pi Redis Tutorial. Recuperado de https://pimylifeup.com/raspberry-pi-redis/

Programar Fácil. (s. f.). Cómo programar el ESP32 con el IDE de Arduino. Recuperado de https://programarfacil.com/esp8266/programar-esp32-ide-arduino/