Universidad Zaragoza
1542

ANEXOS

Diseño de una Estación Meteorológica WiFi basada en un microcontrolador ESP32

Design of a WiFi weather station based on a microcontroller ESP32

Autor/es

JORGE VENTURA GUILLAMÓN

Director/es

JESÚS LÁZARO PLAZA

Grado en Ingeniería Electrónica y Automática

2024

Índice de contenido

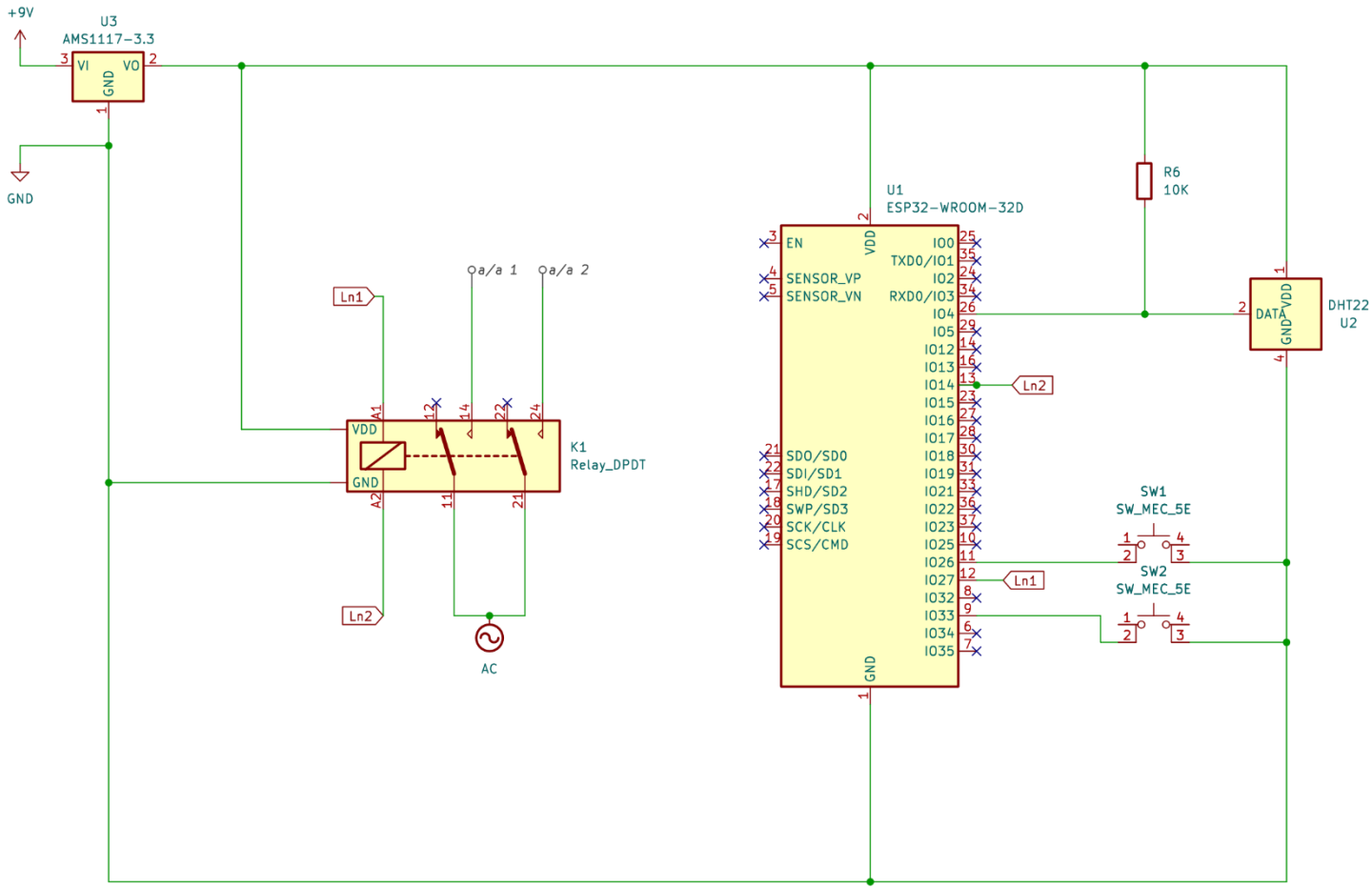
ANEXO I

1. ESQUEMAS ELÉCTRICOS

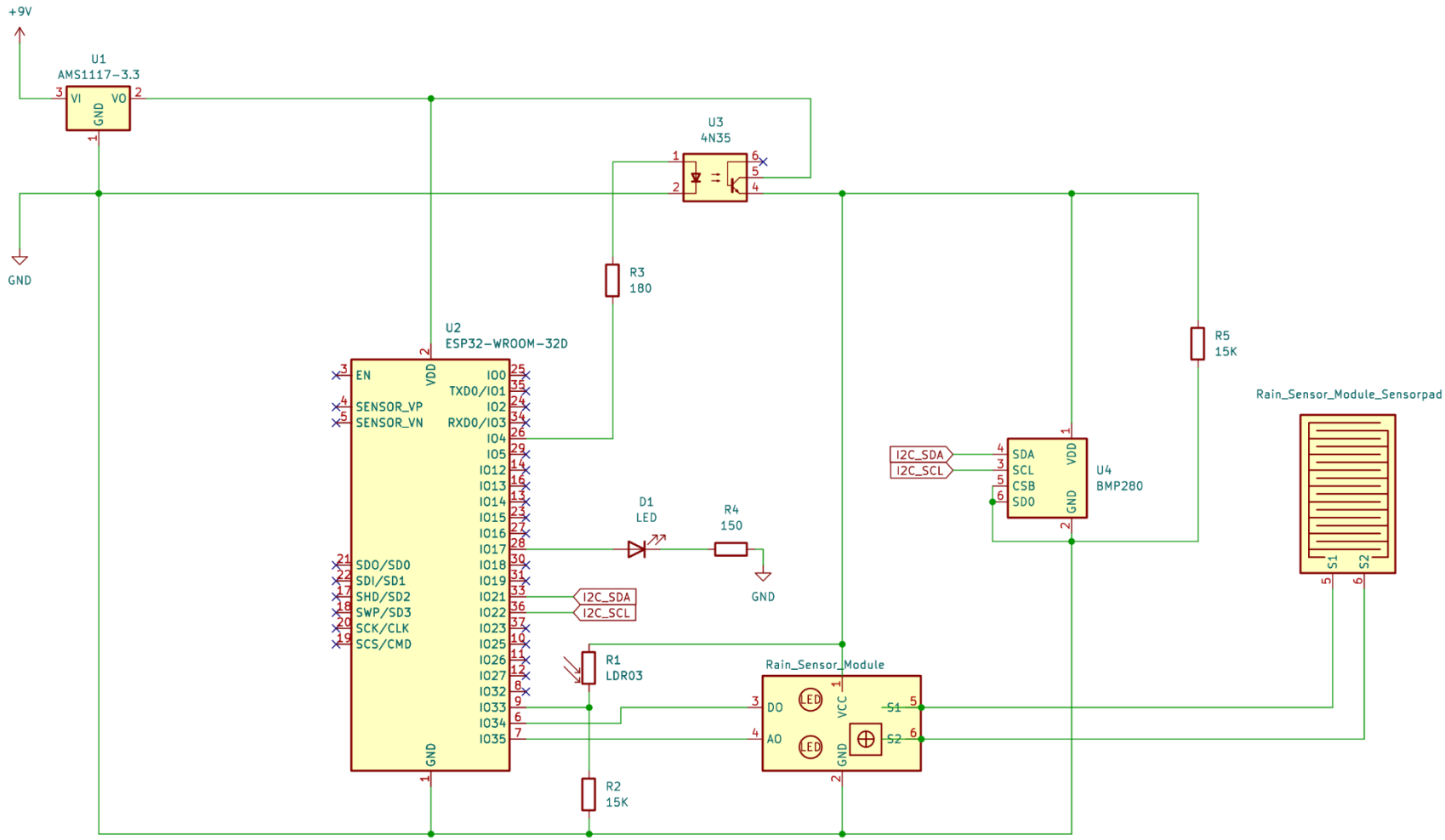
1.1 ESQUEMA ELÉCTRICO ESP32_INTERIOR

1.2 ESQUEMA ELÉCTRICO ESP32_EXTERIOR

1.3 PLANO GENERAL A3

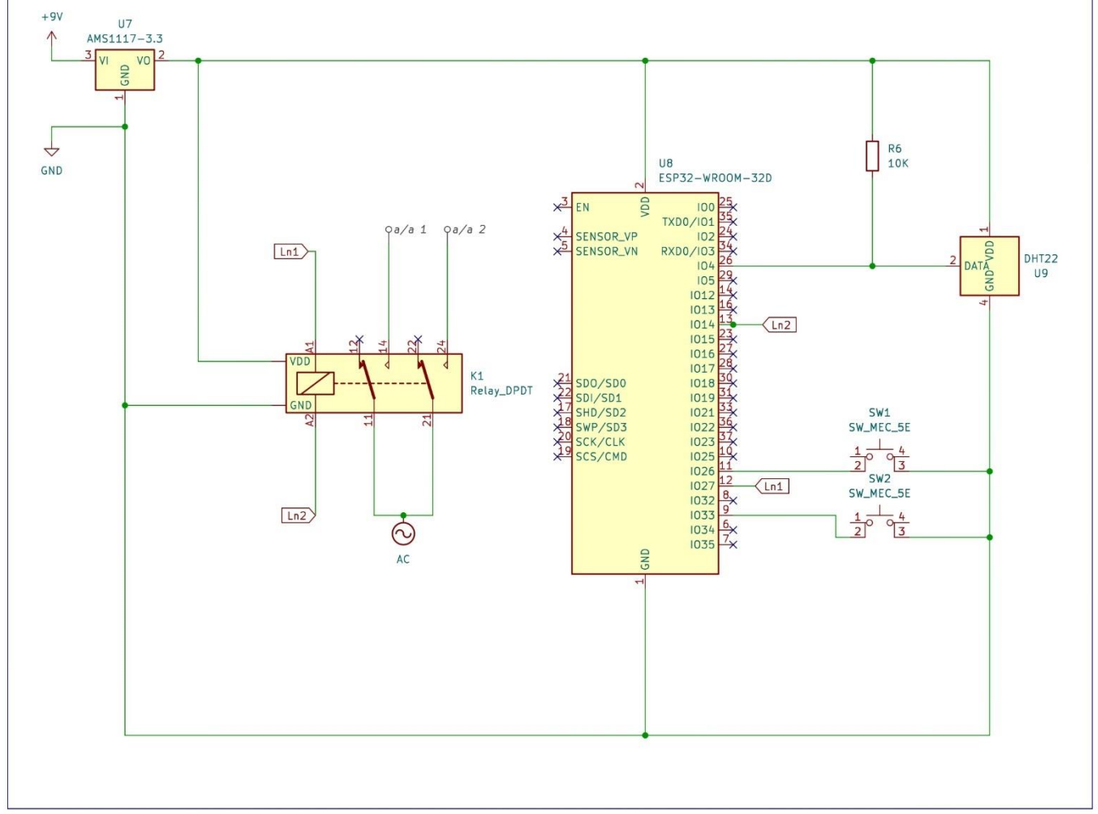


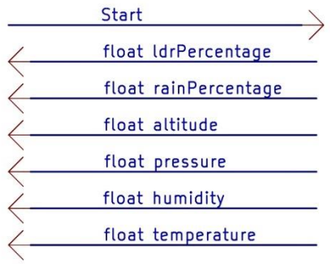
Jorge Ventura Guillamón		
Sheet: /		
File: Esp_Interior.kicad_sch		
Title: Esquema Electrico Esp_Interior		
Size: A4	Date: 2024-11-02	**Rev:**
KiCad E.D.A. 8.0.6		Id: 1/1

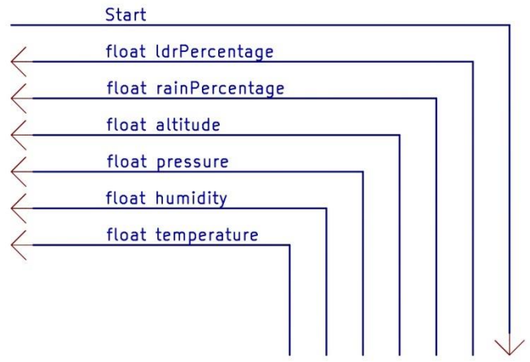


Jorge Ventura Guillamón		
Sheet: /		
File: Esp_Exterior.kicad_sch		
Title: Esquema Electrico Esp_Exterior		
Size: A4	Date: 2024-11-01	**Rev:**
KiCad E.D.A. 8.0.6		Id: 2/1

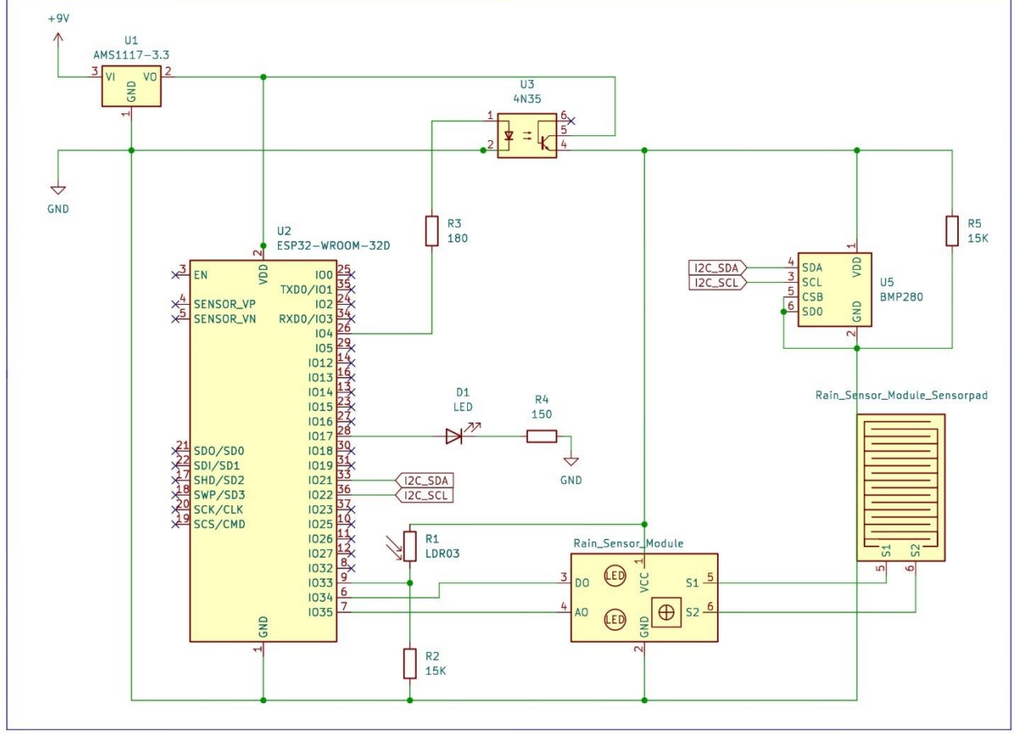
Esp_Interior

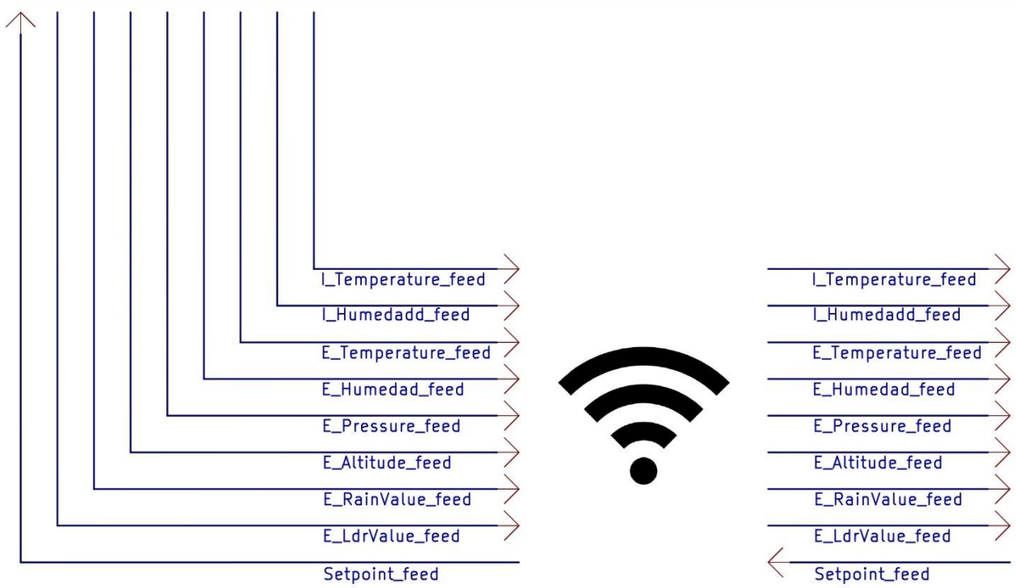






Esp_Exterior





Plataforma de Datos IoT

Jorge Ventura Guillamón
Sheet: /
File: A3.kicad_sch
Title: Plano General
Size: A3 | Date: 2024-11-02 | Rev:
KiCad E.D.A. 8.0.6 | Id: 3/1

ANEXO II

1. CÓDIGOS FUENTE

1.1 ESP32 INTERIOR

1.2 ESP32 EXTERIOR

1.1 ESP32 INTERIOR

```
/*
  ---------------------------------------------------------
  Proyecto:  Diseño de una Estación Meteorológica WiFi
  Descripción:
    El siguiente código controla el microcontrolador ESP32_Interior,
    encargado de recibir datos del ESP32_Exterior,
    recolectar datos ambientales del interior,
    y enviar toda la información al servidor web Adafruit IO.
    También gestiona el bucle de control para activar o desactivar
    dispositivos de climatización según la temperatura interior y
    el valor de consigna fijado por el usuario.

  Autor:     JORGE VENTURA GUILLAMÓN
  Fecha:     11-2024

  Hardware:
    - Sensor DHT22: mide temperatura y humedad.
    - Módulo Relé 5V 2CH: controla dispositivos de climatización.
    - Pulsadores de botón: permiten el control manual de los relés.
    - Fuente de alimentación MB102: proporciona voltaje estable.

  Notas:
    - IMPORTANTE --> Acordarse de activar el Wi-Fi del móvil
  ---------------------------------------------------------
*/



#include <Arduino.h>               // Biblioteca principal de Arduino para
funciones básicas
#include <NimBLEDevice.h>          // Biblioteca para manejar dispositivos
Bluetooth de baja energía
#include <Adafruit_Sensor.h>       // Biblioteca de sensores de Adafruit para
estandarizar la lectura de datos
#include <DHT.h>                   // Biblioteca para sensores de temperatura
y humedad DHT
#include <Wire.h>                  // Biblioteca para comunicación I2C
#include <WiFi.h>                  // Biblioteca para conectar a redes WiFi
#include <WiFiClientSecure.h>      // Biblioteca para manejar conexiones WiFi
seguras
#include <Adafruit_MQTT.h>         // Biblioteca para manejar la comunicación
MQTT
#include <Adafruit_MQTT_Client.h>  // Biblioteca para manejar clientes MQTT
en conjunto con Adafruit IO

#define DHTPIN 4      // Pin digital conectado al DHT22
```

```
#define DHTTYPE DHT22 // Tipo de sensor DHT (DHT22 en este caso)

// Definir UUIDs como constantes para la comunicación Bluetooth
const char* SERVICE_UUID = "12345678-1234-1234-1234-
123456789abc";                // Definir UUIDs como constantes para la
comunicación Bluetooth
const char* CHARACTERISTIC_UUID = "87654321-4321-4321-4321-
cba987654321";         // UUID de la característica principal
const char* CONTROL_CHARACTERISTIC_UUID = "abcdefab-cdef-cdef-cdef-
abcdefabcdef"; // UUID para la característica de control

// Declaraciones de objetos Bluetooth
NimBLEClient* pClient =
nullptr;                                                  // Puntero al
cliente Bluetooth
NimBLERemoteService* pRemoteService =
nullptr;                                    // Puntero al servicio remoto
Bluetooth
NimBLERemoteCharacteristic* pRemoteCharacteristic =
nullptr;                      // Puntero a la característica remota Bluetooth
NimBLERemoteCharacteristic* pControlCharacteristic =
nullptr;                     // Puntero a la característica de control remota
Bluetooth

bool doConnect = false;       // Indica si se debe conectar al dispositivo
Bluetooth
bool connected = false;       // Estado de la conexión Bluetooth
bool doScan = false;          // Indica si se debe escanear para dispositivos
Bluetooth

// Declaración de variables para almacenar datos de sensores exteriores
float E_Temperature = 0.0;
float E_Humedad = 0.0;
float E_Pressure = 0.0;
float E_Altitude = 0.0;
float E_RainPercentage = 0;
float E_LdrPercentage = 0;

// Declaración de variables para almacenar datos de sensores internos
float I_Temperature = 0.0;
float I_Humedad = 0.0;

volatile bool relayHeaterManual = false;// Indica si el relé 1 está en modo
manual
volatile bool relayFanManual = false;   // Indica si el relé 2 está en modo
manual
```

```
float Setpoint = 25.0;                  // Valor de consigna en grados
centigrados
const float EnabledHysteresis = 2.0;    // Histeresis para la activación
const float DisabledHysteresis = 1.0;   // Histeresis para la desactivación

// Declaración de las funciones antes de usarlas
void toggleRelay1();                    // Declaración de la función para
alternar el estado del relé 1
void toggleRelay2();                    // Declaración de la función para
alternar el estado del relé 2

// Inicializa el sensor DHT con el pin y tipo definidos
DHT dht(DHTPIN, DHTTYPE);

// Estructura para almacenar los datos del sensor
struct DataPacket {
  float temperature;
  float humidity;
  float pressure;
  float altitude;
  float rainPercentage;
  float ldrPercentage;
};

static void notifyCallback(NimBLERemoteCharacteristic* pRemoteCharacteristic,
uint8_t* pData, size_t length, bool isNotify) {
  Serial.print("Notificación recibida: ");
  Serial.println("Datos recibidos");

  // Procesar los datos recibidos
  DataPacket* dataPacket = (DataPacket*)pData;

 // Asignar los datos recibidos a las variables correspondientes
  E_Temperature = dataPacket->temperature;
  E_Humedad = dataPacket->humidity;
  E_Pressure = dataPacket->pressure;
  E_Altitude = dataPacket->altitude;
  E_RainPercentage = dataPacket->rainPercentage;
  E_LdrPercentage = dataPacket->ldrPercentage;

  // Imprimir valores por pantalla (Funciona, se podría comentar todo con /*
*/ )
  Serial.print("Temperatura Exterior: ");
  Serial.print(E_Temperature);
  Serial.println(" °C");

  Serial.print("Humedad Exterior: ");
  Serial.print(E_Humedad);
```

```
  Serial.println(" %");

  Serial.print("Presión Exterior: ");
  Serial.print(E_Pressure);
  Serial.println(" hPa");

  Serial.print("Altitud Exterior: ");
  Serial.print(E_Altitude);
  Serial.println(" m");

  Serial.print("Valor del sensor de lluvia Exterior: ");
  Serial.println(E_RainPercentage);

  Serial.print("Valor de la LDR Exterior: ");
  Serial.println(E_LdrPercentage);
  Serial.println();
}

class ClientCallbacks : public NimBLEClientCallbacks {
  void onConnect(NimBLEClient* pClient) {
    Serial.println("Conectado");                // Imprimir que se ha
conectado
    connected = true;                           // Marcar como conectado
  }

  void onDisconnect(NimBLEClient* pClient) {
    Serial.println("Desconectado");             // Imprimir que se ha
conectado
    connected = false;                          // Marcar como desconectado
    doScan = true;                              // Indicar que debe comenzar
un nuevo escaneo
  }
};

class AdvertisedDeviceCallbacks : public NimBLEAdvertisedDeviceCallbacks {
  void onResult(NimBLEAdvertisedDevice* advertisedDevice) {
    Serial.print("Dispositivo encontrado: ");
    Serial.println(advertisedDevice->toString().c_str());                //
Imprimir los detalles del dispositivo encontrado

    if (advertisedDevice->isAdvertisingService(NimBLEUUID(SERVICE_UUID))) {
      Serial.println("Dispositivo con el servicio deseado encontrado");
      NimBLEDevice::getScan()->stop();                                   //
Detener el escaneo
      pClient = NimBLEDevice::createClient();                            //
Crear un cliente Bluetooth
      pClient->setClientCallbacks(new ClientCallbacks(), false);         //
Establecer las devoluciones de llamada del cliente
```

```
      pClient->connect(advertisedDevice);                                    //
Conectar al dispositivo anunciado
      doConnect = true;                                                      //
Marcar que se debe conectar
    }
  }
};


void scanEndedCallback(NimBLEScanResults results) {
  Serial.println("Escaneo finalizado");     // Imprimir que el escaneo ha
finalizado
  doScan = true;                           // Indicar que debe comenzar un
nuevo escaneo
}

// Función para conectar al servidor Bluetooth
bool connectToServer() {
  Serial.print("Conectando a: ");
  Serial.println(pClient->getPeerAddress().toString().c_str()); // Imprimir la
dirección del servidor

  if (!pClient->connect(pClient->getPeerAddress())) {
    Serial.println("Fallo en la conexión");                    // Imprimir
mensaje de error si la conexión falla
    return false;
  }

  Serial.println("Conectado al servidor");
  pRemoteService = pClient->getService(SERVICE_UUID);
  if (pRemoteService == nullptr) {
    Serial.print("Fallo en encontrar el servicio UUID: ");
    Serial.println(SERVICE_UUID);                              // Imprimir
mensaje de error si no se encuentra el servicio
    pClient->disconnect();
    return false;
  }

  pRemoteCharacteristic = pRemoteService-
>getCharacteristic(CHARACTERISTIC_UUID);
  if (pRemoteCharacteristic == nullptr) {
    Serial.print("Fallo en encontrar la característica UUID: ");
    Serial.println(CHARACTERISTIC_UUID);                       // Imprimir
mensaje de error si no se encuentra la característica
    pClient->disconnect();
    return false;
  }
```

```
  if (pRemoteCharacteristic->canNotify()) {
    if (!pRemoteCharacteristic->subscribe(true, notifyCallback)) {
      Serial.println("Fallo en suscribirse a la notificación");   // Imprimir
mensaje de error si no se encuentra la característica de control
      pClient->disconnect();
      return false;
    }
  }

  pControlCharacteristic = pRemoteService-
>getCharacteristic(CONTROL_CHARACTERISTIC_UUID);
  if (pControlCharacteristic == nullptr) {
    Serial.print("Fallo en encontrar la característica de control UUID: ");
    Serial.println(CONTROL_CHARACTERISTIC_UUID);                // Imprimir
mensaje de error si no se encuentra la característica de control
    pClient->disconnect();
    return false;
  }

  // Enviar señal de inicio al Esp_Exterior
  pControlCharacteristic->writeValue("START");                  // Escribir el
valor "START" en la característica de control
  Serial.println("Señal de inicio enviada al ESP32 exterior");

  return true; ;                                                // Devolver
"true" si la conexión y la configuración son exitosas
}


// Definir como constantes los pines de los relés y pulsadores
const int relayHeater = 27;   // Pin del relé 1
const int relayFan = 14;      // Pin del relé 2
const int button1 = 26;       // Pin del pulsador 1
const int button2 = 33;       // Pin del pulsador 2


//Wifi
const char* ssid = "Esp";                         // Nombre de la red WiFi
const char* password = "Aprobadme_por_favor";     // Contraseña de la red WiFi

//Adafruit IO
#define AIO_SERVER      "io.adafruit.com"                     // Servidor de
Adafruit IO
#define AIO_SERVERPORT  1883                                  // Puerto del
servidor (8883 para SSL)
#define AIO_USERNAME    "JrgVG"                               // Nombre de
usuario de Adafruit IO
```

```
#define AIO_KEY         "aio_CSBw35cynvzdQ7VXH66amrAtukiE"      // Contraseña
de Adafruit IO

// Crear una clase WiFiClient para conectar con el servidor MQTT
WiFiClient client;

// Configurar la clase MQTT
Adafruit_MQTT_Client mqtt(&client, AIO_SERVER, AIO_SERVERPORT, AIO_USERNAME,
AIO_KEY);

// Configurar feeds para Adafruit IO
Adafruit_MQTT_Publish E_Temperature_feed = Adafruit_MQTT_Publish(&mqtt,
AIO_USERNAME "/feeds/E_Temperature");
Adafruit_MQTT_Publish E_Humedad_feed = Adafruit_MQTT_Publish(&mqtt,
AIO_USERNAME "/feeds/E_Humedad");
Adafruit_MQTT_Publish E_Pressure_feed = Adafruit_MQTT_Publish(&mqtt,
AIO_USERNAME "/feeds/E_Pressure");
Adafruit_MQTT_Publish E_Altitude_feed = Adafruit_MQTT_Publish(&mqtt,
AIO_USERNAME "/feeds/E_Altitude");
Adafruit_MQTT_Publish E_RainValue_feed = Adafruit_MQTT_Publish(&mqtt,
AIO_USERNAME "/feeds/E_RainValue");
Adafruit_MQTT_Publish E_LdrValue_feed = Adafruit_MQTT_Publish(&mqtt,
AIO_USERNAME "/feeds/E_LdrValue");

Adafruit_MQTT_Publish I_Temperature_feed = Adafruit_MQTT_Publish(&mqtt,
AIO_USERNAME "/feeds/I_Temperature");
Adafruit_MQTT_Publish I_Humedadd_feed = Adafruit_MQTT_Publish(&mqtt,
AIO_USERNAME "/feeds/I_Humedadd");

// Suscribirse al feed para recibir de Adafruit IO el valor de la variable
Setpoint
Adafruit_MQTT_Subscribe Setpoint_feed = Adafruit_MQTT_Subscribe(&mqtt,
AIO_USERNAME "/feeds/Setpoint");


// Función para establecer la conexión MQTT
void MQTT_connect() {
  int8_t ret;                                          // Variable para
almacenar el estado de la conexión

  // Detener si ya está conectado.
  if (mqtt.connected()) {
    return;                                            // Salir de la función
si ya está conectado
  }

  Serial.print("Conectando a MQTT... ");               // Imprimir mensaje de
conexión
```

```
  uint8_t retries = 3;                                  // Número de intentos de
reconexión
  while ((ret = mqtt.connect()) != 0) {                 // Devolverá 0 si está
conectado
        Serial.println(mqtt.connectErrorString(ret));  // Imprimir mensaje de
error de conexión
        Serial.println("Reintento de conexion a MQTT en 5 segundos...");  //
Imprimir mensaje de conexión exitosa
        mqtt.disconnect();                              // Desconectar del
servidor MQTT
        delay(5000);
        retries--;                                      // Decrementar el número
de reintentos en uno
        if (retries == 0) {

          while (1);                                    // Entrar en bucle
infinito
        }
  }
  Serial.println("MQTT Conectado!");                    // Imprimir mensaje de
conexión exitosa
}

void setup() {
  Serial.begin(115200);                 //Velocidad en baudios para
comunicarse con el serial COM
  Serial.println("Iniciando...");       // Mensaje de inicio en el monitor
serial

  NimBLEDevice::init("");               // Inicializar el dispositivo
Bluetooth Low Energy (BLE)

// Configuración del escaneo Bluetooth
  NimBLEScan* pScan = NimBLEDevice::getScan();    // Obtener el objeto de
escaneo BLE
  pScan->setAdvertisedDeviceCallbacks(new AdvertisedDeviceCallbacks());
  pScan->setInterval(45);                         // Establecer el intervalo
de escaneo a 45 ms
  pScan->setWindow(15);                           // Establecer la ventana de
escaneo a 15 ms
  pScan->setActiveScan(true);                     // Habilitar el escaneo
activo
  pScan->start(0, scanEndedCallback);             // Iniciar el escaneo

  // Iniciar el sensor DHT
  dht.begin();
```

```
  // Configuración de pines para los relés y pulsadores
  pinMode(relayHeater, OUTPUT);     // Configurar el pin del relé 1 como
salida
  pinMode(relayFan, OUTPUT);        // Configurar el pin del relé 2 como
salida
  pinMode(button1, INPUT_PULLUP);   // Configurar el pin del pulsador 1 como
entrada con resistencia pull-up
  pinMode(button2, INPUT_PULLUP);   // Configurar el pin del pulsador 2 como
entrada con resistencia pull-up

  // Inicializar los relés en estado apagado (HIGH)
  digitalWrite(relayHeater, HIGH);    // Establecer el relé 1 en estado
apagado
  digitalWrite(relayFan, HIGH);       // Establecer el relé 2 en estado
apagado

  // Suscribirse al feed para establecer la consigna en Adafruit IO
  mqtt.subscribe(&Setpoint_feed);

  // Configurar las interrupciones para los pulsadores
  attachInterrupt(digitalPinToInterrupt(button1), toggleRelay1, FALLING);   //
Configurar interrupción para el pulsador 1
  attachInterrupt(digitalPinToInterrupt(button2), toggleRelay2, FALLING);   //
Configurar interrupción para el pulsador 2


  //Wifi

    WiFi.begin(ssid, password);                  // Iniciar la conexión a la
red WiFi con el SSID y la contraseña

    // Esperar a que la conexión WiFi se establezca
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);                             // Esperar 1 segundo
        Serial.println("Conectando a WiFi...");   // Imprimir mensaje
indicando que está intentando conectar
    }

    Serial.println("Conectado a WiFi");          // Imprimir mensaje
indicando que la conexión se ha establecido

}

void loop() {

    // Intentar conectar al servidor si es necesario
  if (doConnect) {
    if (connectToServer()) {
```

```
      Serial.println("Conectado al servidor");  // Imprimir mensaje de
conexión exitosa
    } else {
      Serial.println("Fallo en conectar al servidor");
    }
    doConnect = false;                          // Resetear la bandera de
conexión
  }

  // Leer valores del DHT22
  I_Temperature = dht.readTemperature();        // Leer la temperatura
interior
  I_Humedad = dht.readHumidity();               // Leer la humedad interior

  // Imprimir valores por pantalla (Funciona, se podría comentar todo con /*
*/ )
  Serial.print("Temperatura Interior: ");
  Serial.print(I_Temperature);
  Serial.println(" °C");

  Serial.print("Humedad Interior: ");
  Serial.print(I_Humedad);
  Serial.println(" %");
  Serial.println();


// Conectar a MQTT
  MQTT_connect();

  // Procesar mensajes MQTT
  Adafruit_MQTT_Subscribe* subscription;                // Variable para
manejar las suscripciones MQTT

  // Leer suscripciones MQTT durante 5 segundos
  while ((subscription = mqtt.readSubscription(5000))) {
    if (subscription == &Setpoint_feed) {               // Verificar si el
mensaje recibido es del feed de Setpoint
      Setpoint = atof((char*)Setpoint_feed.lastread);   // Convertir el
valor recibido a float y asignarlo a Setpoint
      Serial.print("Nuevo valor de consigna: ");
      Serial.println(Setpoint);                         // Imprimir el nuevo
valor de Setpoint en el monitor serial
    }
  }

  // Publicar datos en Adafruit IO
  if (!E_Temperature_feed.publish(E_Temperature)) {
    Serial.println("Error al publicar E_Temperature");
```

```
  }
  if (!E_Humedad_feed.publish(E_Humedad)) {
    Serial.println("Error al publicar E_Humedad");
  }
  if (!E_Pressure_feed.publish(E_Pressure)) {
    Serial.println("Error al publicar E_Pressure");
  }
  if (!E_Altitude_feed.publish(E_Altitude)) {
    Serial.println("Error al publicar E_Altitude");
  }
  if (!E_RainValue_feed.publish(E_RainPercentage)) {
    Serial.println("Error al publicar E_RainValue");
  }
  if (!E_LdrValue_feed.publish(E_LdrPercentage)) {
    Serial.println("Error al publicar E_LdrValue");
  }
  if (!I_Temperature_feed.publish(I_Temperature)) {
    Serial.println("Error al publicar I_Temperature");
  }
  if (!I_Humedadd_feed.publish(I_Humedad)) {
    Serial.println("Error al publicar I_Humedad");
  }


  // Control de relés basado en la temperatura interior y el valor de consigna
Setpoint
  if (!relayHeaterManual) {
    if (I_Temperature < Setpoint - EnabledHysteresis) {             // Si la
temperatura interior baja del valor de consigna menos la histeresis de
activacion
      digitalWrite(relayHeater, LOW);                               // Activar
rele 1 (calefactor)
    } else if (I_Temperature > Setpoint + DisabledHysteresis) {     // Si la
temperatura interior supera el valor de consigna más la histeresis
      digitalWrite(relayHeater, HIGH);                              //
Desactivar rele 1 (calefactor)
    }
  }

  if (!relayFanManual) {
    if (I_Temperature > Setpoint + EnabledHysteresis) {             // Si la
temperatura interior supera el valor de consigna más la histeresis de
activacion
      digitalWrite(relayFan, LOW);                                  // Activar
rele 2 (ventilador)
    } else if (I_Temperature < Setpoint - DisabledHysteresis) {     // Si la
temperatura interior baja del valor de consigna menos la histeresis de
desactivacion
```

```
      digitalWrite(relayFan, HIGH);                                    //
Desactivar rele 2 (ventilador)
    }
  }

  delay(30000); // Esperar 30 segundos antes de intentar leer de nuevo
(Asegurarse de que sea suficiente para no superar el dataRate de Adafruit IO
30 datos por minuto)
}


// Función para alternar el estado del relé 1 Heater
void toggleRelay1() {
  relayHeaterManual = !relayHeaterManual;                  // Cambiar el
estado de relay1
  if (relayHeaterManual) {
    relayFanManual = false;                                // Desactivar
ventilador manualmente
    digitalWrite(relayFan, HIGH);                          // Asegurar que el
ventilador esté apagado
  }
  digitalWrite(relayHeater, relayHeaterManual ? LOW : HIGH);// Activar o
desactivar el calefactor basado en el nuevo estado de relayHeaterManual
}

// Función para alternar el estado del relé 2 Fan
void toggleRelay2() {
  relayFanManual = !relayFanManual;
  if (relayFanManual) {
    relayHeaterManual = false;                             // Desactivar
calefactor manualmente
    digitalWrite(relayHeater, HIGH);                       // Asegurar que el
calefactor esté apagado
  }
  digitalWrite(relayFan, relayFanManual ? LOW : HIGH);     // Activar o
desactivar el ventilador basado en el nuevo estado de relayFanManual
}




    // Imprimir el estado de los pulsadores y las salidas (Funciona, se podría
comentar todo con /* */ )
    /*
  Serial.print("Button 1: ");
  Serial.print(button1State == LOW ? "Pressed" : "Released");
  Serial.print(" | Relay 1: ");
```

```
  Serial.println(digitalRead(relay1) == LOW ? "ON" : "OFF");

  Serial.print("Button 2: ");
  Serial.print(button2State == LOW ? "Pressed" : "Released");
  Serial.print(" | Relay 2: ");
  Serial.println(digitalRead(relay2) == LOW ? "ON" : "OFF");
*/
```

1.2 ESP32 EXTERIOR

```
/*
  ---------------------------------------------------------
  Proyecto:  Diseño de una Estación Meteorológica WiFi
  Descripción:
    El siguiente codigo controla el microcontrolador ESP32_Exterior,
    encargado de recolectar datos ambientales del exterior,
    tales como temperatura, humedad, presión y luminosidad,
    y enviarlos al ESP32_Interior mediante Bluetooth.

  Autor:     JORGE VENTURA GUILLAMÓN
  Fecha:     11-2024

  Hardware:
    - Sensor BME280: mide temperatura, humedad y presión.
    - Sensor FC-37: mide intensidad de lluvia.
    - Sensor LDR: mide la intensidad de la luz.
    - Optoacoplador 4N35: gestiona la alimentación de los sensores.
    - Fuente de alimentación MB102: proporciona voltaje estable.

  ---------------------------------------------------------
*/



#include <Wire.h>                 // Biblioteca para comunicación I2C
#include <Adafruit_Sensor.h>      // Biblioteca para sensores de Adafruit
#include <Adafruit_BME280.h>      // Biblioteca para el sensor BME280
#include <NimBLEDevice.h>         // Biblioteca para comunicación Bluetooth
// #include "esp_sleep.h"         // Biblioteca para modos de sueño (En
prototipo no se usa)

#define LDR_PIN 33                //Pin para LDR
#define RAIN_SENSOR_PIN 35        //Pin para sensor de lluvia
#define LED_PIN 17                //Pin para LED Azul, indica que acaba de
enviar datos por bluetooth
#define OPTO_PIN 4                //Pin para controlar el optoacoplador

// Definir UUIDs como constantes para la comunicación Bluetooth
const char* SERVICE_UUID = "12345678-1234-1234-1234-
123456789abc";                // Definir UUIDs como constantes para la
comunicación Bluetooth
const char* CHARACTERISTIC_UUID = "87654321-4321-4321-4321-
cba987654321";         // UUID de la característica principal
const char* CONTROL_CHARACTERISTIC_UUID = "abcdefab-cdef-cdef-cdef-
abcdefabcdef"; // UUID para la característica de control
```

```
Adafruit_BME280 bme;                                    // Objeto para el
sensor BME280
NimBLEServer* pServer = nullptr;                        // Puntero al
servidor Bluetooth
NimBLECharacteristic* pCharacteristic = nullptr;        // Puntero a la
característica principal
NimBLECharacteristic* pControlCharacteristic = nullptr; // Puntero a la
característica de control
bool startDataCollection = false;                       // Flag para indicar
si se inicia la recopilación de datos

class ControlCallbacks : public NimBLECharacteristicCallbacks {
  void onWrite(NimBLECharacteristic* pCharacteristic) {
    std::string value = pCharacteristic->getValue();                //
Obtener el valor escrito en la característica
    if (value == "START") {                                         // Si el
valor es "START"
      startDataCollection = true;                                   //
Iniciar la recopilación de datos
      Serial.println("Iniciando recopilación de datos");            //
Imprimir mensaje en el monitor serial
    }
  }
};

void setup() {
  //Velocidad en baudios para comunicarse con el serial COM
  Serial.begin(115200);

  // Configurar el pin del optoacoplador como salida
  pinMode(OPTO_PIN, OUTPUT);

  // Encender el optoacoplador para alimentar los sensores
  digitalWrite(OPTO_PIN, HIGH);
  delay(500);

  // Bluetooth
  NimBLEDevice::init("ESP32_Exterior");                             //
Inicializa el dispositivo Bluetooth con nombre "ESP32_Exterior"
  pServer = NimBLEDevice::createServer();                           // Crea
un servidor Bluetooth
  NimBLEService* pService = pServer->createService(SERVICE_UUID);    // Crea
un servicio Bluetooth con el UUID definido

  // Crea una característica para notificaciones
  pCharacteristic = pService->createCharacteristic(
                      CHARACTERISTIC_UUID,
```

```
                    NIMBLE_PROPERTY::NOTIFY
                  );
  // Crea una característica de control para escritura
  pControlCharacteristic = pService->createCharacteristic(
                             CONTROL_CHARACTERISTIC_UUID,
                             NIMBLE_PROPERTY::WRITE
                           );
  // Establece las devoluciones de llamada para la característica de control
  pControlCharacteristic->setCallbacks(new ControlCallbacks());

  pService->start();
  NimBLEAdvertising* pAdvertising = NimBLEDevice::getAdvertising();
  pAdvertising->addServiceUUID(SERVICE_UUID);
  pAdvertising->start();
  Serial.println("El Bluetooth ha sido inicializado");

  // BME280
  while (!Serial);                                    // Espera a que el
puerto serial esté listo

  if (!bme.begin(0x76)) {                             // Direccion
predeterminada si CSB Y SD0 conectados a GND -> 0x76
    Serial.println("No se encuentra sensor BME280");  // Mensaje de error si
no se encuentra el sensor
    while (1);                                        // Entra en un bucle
infinito si no se encuentra el sensor
  }

  pinMode(RAIN_SENSOR_PIN, INPUT);                    // Configura el pin del
sensor de lluvia como entrada
  pinMode(LDR_PIN, INPUT);                            // Configura el pin del
sensor LDR como entrada
  pinMode(LED_PIN, OUTPUT);                           // Configura el pin del
LED como salida
  delay(10000);
}

void loop() {

  // Encender el optoacoplador para alimentar los sensores
  digitalWrite(OPTO_PIN, HIGH);
  delay(500);

 // Reinicializar el bus I2C y el sensor BME280 (Debido a que entre mediciones
se le quita la alimentacion)
    Wire.begin();
    if (!bme.begin(0x76)) {                           // Direccion
predeterminada si CSB Y SD0 conectados a GND -> 0x76
```

```
      Serial.println("No se encuentra sensor BME280");  // Mensaje de error si
no se encuentra el sensor
      while (1);                                        // Entra en un bucle
infinito si no se encuentra el sensor
    }


  // Si la recopilación de datos ha comenzado, leer valores de los sensores
  if (startDataCollection) {
    // Leer valores del BME280
    float temperature = bme.readTemperature();          // Leer la temperatura
del BME280
    float humidity = bme.readHumidity();                // Leer la humedad del
BME280
    float pressure = bme.readPressure() / 100.0F;       // Leer la presión y
convertir a hPa
    float altitude = bme.readAltitude(1013.25);         // Calcular la altitud
asumiendo una presión a nivel del mar de 1013.25 hPa

    // Leer valor del sensor de lluvia
    int rainValue = analogRead(RAIN_SENSOR_PIN);
    // Normalizar el valor del detector de lluvia a un porcentaje
    float rainPercentage = ((4095.0 - rainValue) / 4095.0) * 100.0;

    // Leer valor de la LDR
    int ldrValue = analogRead(LDR_PIN);
    // Normalizar el valor de la LDR a un porcentaje
    float ldrPercentage = (ldrValue / 4035.0) * 100.0;  //Maximo 4035
enfocando con linterna, 3818 luz natural



    // Imprimir valores por pantalla (Funciona, se podría comentar todo con /*
*/ )
    Serial.print("Temperatura: ");
    Serial.print(temperature);
    Serial.println(" °C");

    Serial.print("Humedad: ");
    Serial.print(humidity);
    Serial.println(" %");

    Serial.print("Presión: ");
    Serial.print(pressure);
    Serial.println(" hPa");

    Serial.print("Altitud: ");
    Serial.print(altitude);
```

```
    Serial.println(" m");

    Serial.print("Porcentaje del sensor de lluvia: ");
    Serial.println(rainPercentage);

    Serial.print("Porcentaje de la LDR: ");
    Serial.println(ldrPercentage);

    Serial.print("Valor de la LDR: ");
    Serial.println(ldrValue);
    Serial.println();

    // Estructura para almacenar los datos de los sensores
    struct DataPacket {
      float temperature;
      float humidity;
      float pressure;
      float altitude;
      float rainPercentage;
      float ldrPercentage;
    } dataPacket;

    // Asignar los valores leídos a las respectivas variables en el paquete de
datos
    dataPacket.temperature = temperature;
    dataPacket.humidity = humidity;
    dataPacket.pressure = pressure;
    dataPacket.altitude = altitude;
    dataPacket.rainPercentage = rainPercentage;
    dataPacket.ldrPercentage = ldrPercentage;

    pCharacteristic->setValue((uint8_t*)&dataPacket, sizeof(dataPacket));   //
Establece el valor de la característica con los datos del paquete
    pCharacteristic->notify();                                              //
Envía una notificación a los dispositivos suscritos con el nuevo valor

    // Activar LED durante 1 segundo (Indicando que se ha hecho el envio)
    digitalWrite(LED_PIN, HIGH);
    delay(1000);
    digitalWrite(LED_PIN, LOW);

    // Apagar el optoacoplador para cortar la alimentación de los sensores
    digitalWrite(OPTO_PIN, LOW);

    delay(25000);

  }
}
```

```
// CONFIGURACION PARA PONER EL ESP EN MODO SUSPENSION ENTRE INTERVALOS DE
MEDIDA              // (Necesario borrar los ultimos dos } de arriba mas el
delay)

// No se implementa debido a ser un prototipo en el que para las prubas los
intervalos entre medidas son muy cortos,
// la implementacion entorpecería el correcto funcionamiento

/*

    // Configurar el temporizador para el modo de suspensión ligera
    esp_sleep_enable_timer_wakeup(24 * 1000000); );   // Configuracion para
despertarse después de 24 segundos

    // Entrar en modo de suspensión ligera
    esp_light_sleep_start();                          // Comienzo del modo de
suspensión ligera

  }
}

*/
```