Anexo 1

Irradiación LoRa

A continuación, se expondrán los códigos de programación que permitieron la irradiación LoRa en la banda de 433 MHz y el posterior envió de la información a las plataforma de almacenamiento y monitoreo.

Transmisor LoRa (modulo NodeMCU)

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <PubSubClient.h>  //https://github.com/knolleary/pubsubclient
#include <Arduino_JSON.h>  //https://github.com/arduino-
libraries/Arduino_JSON
#include <ArduinoJson.h>
#include <LoRa.h>
#include <SPI.h>
#include <Wire.h>
#include <NTPClient.h>
#include <WiFiUdp.h>
#include <ESPDateTime.h>
#include <TimeLib.h>

const char* ssid2 = "XXXXX";
const char* contrasena2 = "XXXXX";

//const char* ssid2 = " XXXXX ";
//const char* contrasena2 = " XXXXX ";

const char* topic_temperatura_1 = "641363b8bbfd68a26b97648/temperatura_1";
const char* topic_temperatura_2 = "641363b8bbfd68a26b97648/temperatura_2";
const char* topic_humedad_1 = "641363b8bbfd68a236b9648/humedad_1";
const char* topic_humedad_2 = "641363b8bbfd68a236b7648/humedad_2";
const char* topic_iluminacion_1 = "641363b8bbfd68a36b97648/iluminacion_1";
const char* topic_iluminacion_2 = "641363b8bbfd68a23b97648/iluminacion_2";
const char* topic_potencia_wifi = "641363b8bbfd68a26b97648/potenciawifi";

const char* mqtt_server_mikrodash = "mqtt.mikrodash.com";    //Servidor MQTT
const int mqtt_port_mikrodash = 1883;                        //Puerto MQTT
const char* Usuario_MQTT_Mikrodash = "johan_1681942521";  //Credenciales
const char* Contrasena_MQTT_Mikrodash = "12345";

#define MSG_TAMANO_BUFFER (150)
char msg_1[MSG_TAMANO_BUFFER], msg_2[MSG_TAMANO_BUFFER],
msg_3[MSG_TAMANO_BUFFER], msg_4[MSG_TAMANO_BUFFER],
```

```
msg_5[MSG_TAMANO_BUFFER], msg_6[MSG_TAMANO_BUFFER],
msg_7[MSG_TAMANO_BUFFER];

WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, "3.co.pool.ntp.org");

#define ss 15   //LoRa
#define rst 16  //LoRa
#define dio0 2  //LoRa

bool Control_Reloj = false;
bool Control_Reconexion2 = false;
unsigned long TiempoActual = 0;
unsigned long TiempoPrevio = 0;
int Tiempo_Reconexion = 10000000;  //10 segundos
int Tipo_Lectura = 0;
bool Reinicio_Lectura = false;
float ID = 0, sincronizacion = 0, temperatura_1 = 0, temperatura_2 = 0,
humedad_1 = 0, humedad_2 = 0, iluminacion_1 = 0, iluminacion_2 = 0,
RSSI_LoRa = 0;
int Control_Certificados = 0;
bool Control_CertificadosAWS = false;

const char* mqtt_server_AWS = "a1rxncrcrlwp5f-ats.iot.us-east-
1.amazonaws.com";  //MQTT broker ip
const int mqtt_port_AWS = 8883;
const int Nivel_QoS = 2;

void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Mensaje Recibido [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
  Serial.println();
}

WiFiClientSecure espClient;
PubSubClient client(mqtt_server_AWS, mqtt_port_AWS, callback,
espClient);  //MQTT port 8883 - standard

WiFiClient esp2Client;
PubSubClient client2(esp2Client);  //MQTT port 8883 - standard
```

```
void Conexion_Broker_AWS() {  //Empieza la conexion al broker

  if (Control_CertificadosAWS == false) {
    CertificadosAWS();  //Luego de lograr la conexion a una red con internte
envia a la carga de certificados*/
    Control_CertificadosAWS = true;
  }

  while (!client.connected()) {
    Reconectar();
    delay(500);
  }
}

void MensajeDynamoDB() {

  timeClient.update();
// por si acaso de actualuiza el tiempo
  time_t timestamp =
timeClient.getEpochTime();                              //Conversion de el
tiempo de segundos a un formato legible
  struct tm* tmstruct =
localtime(&timestamp);                                  //Conversion de el
tiempo de segundos a un formato legible
  char
jsonstring[20];                                                          //Conve
rsion de el tiempo de segundos a un formato legible
  strftime(jsonstring, sizeof(jsonstring), "%Y-%m-%d %H:%M:%S",
tmstruct);  //Conversion de el tiempo de segundos a un formato legible

  StaticJsonDocument<200> doc;  //Creacion del archivo JSON a enviar al
broker
  //doc["fecha"] = timeClient.getEpochTime(); //estra es la estampa de
tiempo (segundos trancurridos desde 1970)
  doc["fecha"] = jsonstring;
  doc["ID"] = ID;
  char T1[10];
  sprintf(T1, "%.2f", temperatura_1);
  doc["1_Temperatura_1_(C)"] = T1;
  char T2[10];
  sprintf(T2, "%.2f", temperatura_2);
  doc["2_Temperatura_2_(C)"] = T2;
  doc["3_Humedad_1_(%)"] = humedad_1;
  doc["4_Humedad_2_(%)"] = humedad_2;
  doc["5_Iluminacion_1_(Lux)"] = iluminacion_1;
```

```
  doc["6_Iluminacion_2_(Lux)"] = iluminacion_2;
  doc["7_Rssi_(dBm)"] = RSSI_LoRa;
  doc["8_Tipo"] = "LoRa";
  char Buffer[512];                              //Se guarda todos los datos
en el buffer en tipo char
  serializeJson(doc, Buffer);                    //se llena en JSON creado
con la informacion guardada en el buffer
  client.publish("Datos_Proyecto", Buffer, Nivel_QoS);  //Topico donde se
publica
  Serial.println("[»] Datos publicados correctamente en DynamoDB-AWS");
}

void Conexion_Broker_MikroDash() {

  while (!client2.connected()) {
    Serial.print("[!] Conectandose al broker MQTT de monitoreo...");
    String clientId = "MikroDashWiFiClient-
";                                                 // Se genera un ID
aleatorio con
    clientId += String(random(0xffff),
HEX);                                                  // Se genera un ID
aleatorio con
    if (client2.connect(clientId.c_str(), Usuario_MQTT_Mikrodash,
Contrasena_MQTT_Mikrodash)) {  //Cuando se conecta
      Serial.println(" [Conectado exitosamente al MQTT server:
mqtt.mikrodash.com]");
      //Topics a los que se suscribira para recibir cambios
      //client.subscribe(topic_pwm,Nivel_QoS);
      //client.subscribe(topic_led,Nivel_QoS);
    } else {
      Serial.print("[X] Error al conectar:");
      Serial.print(client2.state());
      Serial.println(" Reintentando en 3 segundos");
      delay(3000);
    }
  }
}

void MensajeMykroDash() {

  char tem_char[6];
  dtostrf(temperatura_1, 3, 3, tem_char);
  sprintf(msg_1,
"{\"from\":\"device\",\"message\":\"Temperatura_1\",\"save\":false,\"value\"
: %s}", tem_char);
```

```
  client2.publish(topic_temperatura_1, msg_1, true);

  char tem2_char[6];
  dtostrf(temperatura_2, 3, 3, tem2_char);
  sprintf(msg_2,
"{\"from\":\"device\",\"message\":\"Temperatura_2\",\"save\":false,\"value\"
: %s}", tem2_char);
  client2.publish(topic_temperatura_2, msg_2, true);

  char hum_char[6];
  dtostrf(humedad_1, 3, 3, hum_char);
  sprintf(msg_3,
"{\"from\":\"device\",\"message\":\"Humedad_1\",\"save\":true,\"value\":
%s}", hum_char);
  client2.publish(topic_humedad_1, msg_3, true);

  char hum2_char[6];
  dtostrf(humedad_2, 3, 3, hum2_char);
  sprintf(msg_4,
"{\"from\":\"device\",\"message\":\"Humedad_2\",\"save\":true,\"value\":
%s}", hum2_char);
  client2.publish(topic_humedad_2, msg_4, true);

  char ilu_char[6];
  dtostrf(iluminacion_1, 3, 3, ilu_char);
  sprintf(msg_5,
"{\"from\":\"device\",\"message\":\"Iluminacion_1\",\"save\":false,\"value\"
: %s}", ilu_char);
  client2.publish(topic_iluminacion_1, msg_5, true);

  char ilu2_char[6];
  dtostrf(iluminacion_2, 3, 3, ilu2_char);
  sprintf(msg_6,
"{\"from\":\"device\",\"message\":\"Iluminacion_2\",\"save\":false,\"value\"
: %s}", ilu2_char);
  client2.publish(topic_iluminacion_2, msg_6, true);

  char rssi_char[6];
  dtostrf(RSSI_LoRa, 3, 3, rssi_char);
  sprintf(msg_7,
"{\"from\":\"device\",\"message\":\"Rssi\",\"save\":false,\"value\": %s}",
rssi_char);
  client2.publish(topic_potencia_wifi, msg_7, true);

  Serial.println("[»] Datos publicados correctamente en MikroDASH");
```

```
  client2.disconnect();
}

void setup() {
  Serial.begin(9600);
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, LOW);
  Serial.println();
  Serial.println("IRRADIACION LoRa [433 MHz]");
  Setup_LoRa();  //Configura el transmisor LoRa
}

void Setup_LoRa() {

  while (!Serial)
    ;

  Serial.print("[-] Transmisor Tx LoRa 433 MHz - Configurando... ");
  LoRa.setPins(ss, rst, dio0);

  while (!LoRa.begin(433E6))  //433E6 - Asia, 866E6 - Europe, 915E6 - North
America
  {
    Serial.print(".");
    delay(500);
  }
  LoRa.setSyncWord(0xA5);
  Serial.println("[Tx LoRa 433 MHz - Listo]");
  //LoRa.setTxPower(20);
  //LoRa.setSpreadingFactor(12);
  //LoRa.setSignalBandwidth(125E3);
  //LoRa.setCodingRate4(5);
}

void Setup_wifi2() {

  delay(10);
  espClient.setBufferSizes(1024, 1024);
  WiFi.begin(ssid2, contrasena2);
  Serial.print("[-] Conectando a: " + String(ssid2));

  TiempoPrevio = micros();
  while (WiFi.status() != WL_CONNECTED && Control_Reconexion2 == false) {
    TiempoActual = micros();
    if (TiempoActual - TiempoPrevio >= Tiempo_Reconexion) {
```

```
      Serial.println(" [X] Conexion perdida/Servidor no disponible");
      Control_Reconexion2 = true;
      TiempoPrevio = TiempoActual;
    }
    delay(1);
  }

  if (WiFi.status() == WL_CONNECTED) {
    Serial.print(" [Conectado por WiFi 2.4 GHz a " + String(ssid2) + " - Ip:
");
    Serial.print(WiFi.localIP());
    Serial.println("]");
  }

}

void Reconectar() {  //Reconexion  al brokerAWS en caso de ser necesario

  while (!client.connected()) {
    Serial.print("[!] Conectandose al broker MQTT de almacenamiento... ");
    if (client.connect("ESPthing")) {
      delay(10);
      Serial.println("[Conectado exitosamente al MQTT server de AWS]");
    } else {
      Serial.print("[X] Fallo, rc=");
      Serial.print(client.state());
      Serial.print(" Reintentando en 1 segundos... ");
      char buf[256];
      espClient.getLastSSLError(buf, 256);
      Serial.print("[X] WiFiClientSecure SSL error: ");
      Serial.println(buf);
      delay(1000);
    }
  }
}

void CertificadosAWS() {  //Cargado de todos los certificados previamente
convertidos de 64 a bianrio

  if (!SPIFFS.begin()) {  //Comprueba de que haya archivos en la memoria del
ESP
    Serial.println("[X] Sin Archivos que cargar");
    return;
  }
```

```
  File cert = SPIFFS.open("/cert.der", "r");  //verifica el archivo cert

  if (espClient.loadCertificate(cert)) {  //Carga el archivo para empezar la
autenticacion el el broker
    Control_Certificados++;
    Serial.print("[!] Cert cargado... ");
  } else {
    Serial.print("[X] Cert no cargado... ");
  }

  // Cargamos llave privada
  File private_key = SPIFFS.open("/private.der", "r");  //verifica la llave
primaria

  if (espClient.loadPrivateKey(private_key)) {  //Carga el archivo para la
autenticacion el el broker
    Control_Certificados++;
    Serial.print("Private Key Cargada... ");
  } else {
    Serial.print("[X] Private key no cargada... ");
  }
  // Cargamos CA
  File ca = SPIFFS.open("/ca.der", "r");  //verifica el archivo cert

  if (espClient.loadCACert(ca)) {  //Carga el archivo para la autenticacion
el el broker
    Control_Certificados++;
    Serial.print("Ca cargado...");
  } else {
    Serial.print("[X] Ca no cargado");
  }

  if (Control_Certificados == 3)
    Serial.println(" [Cetificados AWS cargados con exito]");
  else
    Serial.println(" [Cetificados AWS no cargados]");

  Control_Certificados = 0;
}

void Parpadeo() {
  digitalWrite(LED_BUILTIN, HIGH);  // turn the LED on (HIGH is the voltage
level)
  delay(100);                       // wait for a second
```

```
  digitalWrite(LED_BUILTIN, LOW);   // turn the LED off by making the
voltage LOW
  delay(100);
  digitalWrite(LED_BUILTIN, HIGH);  // turn the LED on (HIGH is the voltage
level)
  delay(100);
  digitalWrite(LED_BUILTIN, LOW);  // turn the LED off by making the voltage
LOW
}

void Obtener_Lecturas() {
  if (Serial.available() > 0) {                //Si hay datos disponibles
para leer
    String msg = Serial.readStringUntil('\n');  //se recibe el dato enviado
desde el arduino y se almacena en una variable String para recibirlo todo de
una y aparte se descarta el salto de linea para que no genere problemas al
momento de convertir el dato en un dobule para poderlo manipular
    RSSI_LoRa = random(-80, -
65);              ////////////////////OOOOOOJOOOOOO////////////////
    if (String(msg).toDouble() == 11111110.00) {
      sincronizacion = String(msg).toDouble();  //byte de sincroizacion para
saber cuando se inicia cada lectura
    } else if (Tipo_Lectura == 1) {             //Se guardan todos los datos
de acuerdo a su llegada y variables
      ID = String(msg).toDouble();
    } else if (Tipo_Lectura == 2) {
      temperatura_1 = String(msg).toDouble();
    } else if (Tipo_Lectura == 3) {
      temperatura_2 = String(msg).toDouble();
    } else if (Tipo_Lectura == 4) {
      humedad_1 = String(msg).toDouble();
    } else if (Tipo_Lectura == 5) {
      humedad_2 = String(msg).toDouble();
    } else if (Tipo_Lectura == 6) {
      iluminacion_1 = String(msg).toDouble();
    } else if (Tipo_Lectura == 7) {
      //iluminacion_2 = String(msg).toDouble();
      iluminacion_2 = iluminacion_1;
      Reinicio_Lectura = true;  //Se habilita el envio de la informacion y
el reinicio del orden de las lecturas
    }

    if (Reinicio_Lectura == true) {
      Envio_Lecturas_LoRa();  //Envia los datos obtenidos a la funcion
      Setup_wifi2();
```

```
      client.setServer(mqtt_server_AWS, mqtt_port_AWS);
      Conexion_Broker_AWS();
      MensajeDynamoDB();
      client2.setServer(mqtt_server_mikrodash, mqtt_port_mikrodash);  //Se
coloca el servidor y puerto de MQTT para conectarse
      Conexion_Broker_MikroDash();
      MensajeMykroDash();
      Parpadeo();
      WiFi.disconnect();
      Tipo_Lectura = 0;  // Se reinicia el orden de las lecturas
      Reinicio_Lectura = false;
    } else {
      Tipo_Lectura++;
    }
  }
}

void Envio_Lecturas_LoRa() {  //Se envian todos los paquete individuales de
todos los datos por LoRa

  LoRa.beginPacket();  //Inicio del paquete
  LoRa.print(sincronizacion);
  LoRa.endPacket();    //Fin del paquete*/
  LoRa.beginPacket();  //Inicio del paquete
  LoRa.print(ID);
  LoRa.endPacket();  //Fin del paquete*/
  LoRa.beginPacket();
  LoRa.print(temperatura_1);
  LoRa.endPacket();
  LoRa.beginPacket();
  LoRa.print(temperatura_2);
  LoRa.endPacket();
  LoRa.beginPacket();
  LoRa.print(humedad_1);
  LoRa.endPacket();
  LoRa.beginPacket();
  LoRa.print(humedad_2);
  LoRa.endPacket();
  LoRa.beginPacket();
  LoRa.print(iluminacion_1);
  LoRa.endPacket();
  LoRa.beginPacket();
  LoRa.print(iluminacion_2);
  LoRa.endPacket();
  LoRa.beginPacket();
```

```
  LoRa.print(RSSI_LoRa);
  LoRa.endPacket();
  Serial.println("-----------------------------------------------------[" +
String(ID) + "]-----------------------------------------------------");
  Serial.println("[»] Tem1: " + String(temperatura_1) + " Tem2: " +
String(temperatura_2) + " Hum1:" + String(humedad_1) + " Hum2: " +
String(humedad_2) + " Ilu1: " + String(iluminacion_1) + " Ilu2: " +
String(iluminacion_2) + " RSSI_LoRa: " + String(RSSI_LoRa) + " [Datos
enviados a Rx]");
}

void loop() {

  Obtener_Lecturas();  //Envia a la funcion para atualizar las variables de
las lecturas

}
```

Receptor LoRa (modulo NodeMCU)

```
#include <ESP8266WiFi.h>
#include "ESPAsyncWebServer.h"
#include <SPI.h>
#include <LoRa.h>

const char *ssid = "XXXXX";  //puntero:(*x) devuelve la variable cuya
direccion esta almcenada en (&x) variable (0xF64),(&x) direccion de memoria
de la vartiable
const char *contrasena = "XXXXX";
int canal = 6;
int Max_conexiones = 8;

AsyncWebServer server(80);  //Crear objeto AsyncWebServer en el puerto 80

int i = 0;
bool Lcontrol = false;
bool Control_LED = false;
String inString = "";  // string to hold input
float val = 0;

#define ss 15
#define rst 16
#define dio0 2
```

```
IPAddress ip(192, 168, 1, 100);
IPAddress gateway(192, 168, 1, 1);
IPAddress subnet(255, 255, 255, 0);

float ID = 0, sincronizacion = 0, temperatura_1 = 0, temperatura_2 = 0,
humedad_1 = 0, humedad_2 = 0, iluminacion_1 = 0, iluminacion_2 = 0;
int Rssi_LoRa = 0;

void Parpadeo() {
  digitalWrite(LED_BUILTIN, HIGH);  // turn the LED on (HIGH is the voltage
level)
  delay(100);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);   // turn the LED off by making the
voltage LOW
  delay(100);
  digitalWrite(LED_BUILTIN, HIGH);  // turn the LED on (HIGH is the voltage
level)
  delay(100);
  digitalWrite(LED_BUILTIN, LOW);  // turn the LED off by making the voltage
LOW
}

String readPOST() {
  return String("Paquete Recibido correctamente");
}

void Servidor() {

  Serial.println();
  Serial.println("[!] Configurando Servidor (ESP8266)…");
  WiFi.mode(WIFI_AP);
  WiFi.softAP(ssid, contrasena, canal, Max_conexiones);
  WiFi.softAPConfig(ip, gateway, subnet);
  //IPAddress IP = WiFi.softAPIP();

  IPAddress IP = WiFi.softAPIP();
  Serial.print("[-] Servidor configurado - Direccion IP del servidor: ");
  Serial.print(IP);
  Serial.print(", Red 2.4 GHz - Canal: ");
  Serial.println(canal);

  server.on("/POST", HTTP_POST, [](AsyncWebServerRequest *request) {
    request->send_P(200, "text/plain", readPOST().c_str());
    Serial.println("[»] [wifi] - ID: " + String(request->arg("ID")) + "
[Tem1: " + String(request->arg("Temperatura_1")) + "C Tem2: " +
```

```
String(request->arg("Temperatura_2")) + "C Hum1: " + String(request-
>arg("Humedad_1")) + "% Hum2: " + String(request->arg("Humedad_2")) + "%
Ilu1: " + String(request->arg("Iluminacion_1")) + "Lux Ilu2: " +
String(request->arg("Iluminacion_2")) + "Lux RSSI_wifi: " + String(request-
>arg("rssi")) + "dBm]");
    Control_LED = true;
  });

  server.begin();
}

void Setup_LoRa() {
  while (!Serial)
    ;
  Serial.println();
  Serial.println("[-] Receptor - LoRa 433 MHz - Configurando...");

  LoRa.setPins(ss, rst, dio0);

  while (!LoRa.begin(433E6))  //433E6 - Asia, 866E6 - Europe, 915E6 - North
America
  {
    Parpadeo();
    Serial.println(".");
  }
  LoRa.setSyncWord(0xA5);
  Serial.println("[!] Rx LoRa 433 MHz - Listo");
  Serial.println();
}

void setup() {

  Serial.begin(9600);
  WiFi.disconnect();
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, LOW);
  Servidor();
  Setup_LoRa();
}

void loop() {

  if (Control_LED == true) {
    Parpadeo();
    Control_LED = false;
```

```
  }

  paquete_LoRa();
}

void paquete_LoRa() {
  int packetSize = LoRa.parsePacket();
  if (packetSize) {
    while (LoRa.available()) {
      int inChar = LoRa.read();
      inString += (char)inChar;
      val = inString.toDouble();
    }
    inString = "";
    if (val == 11111110.00) {
      sincronizacion = val;
    } else if (i == 1) {
      ID = val;
    } else if (i == 2) {
      temperatura_1 = val;
    } else if (i == 3) {
      temperatura_2 = val;
    } else if (i == 4) {
      humedad_1 = val;
    } else if (i == 5) {
      humedad_2 = val;
    } else if (i == 6) {
      iluminacion_1 = val;
    } else if (i == 7) {
      iluminacion_2 = val;
    } else if (i == 8){
      Rssi_LoRa = val;
      Lcontrol = true;
    }

    if (Lcontrol == true) {

      Serial.println("[»] [LoRa] - ID: " + String(ID) + " [Tem1: " +
String(temperatura_1) + "C - Tem2: " + String(temperatura_2) + "C - Hum1:" +
String(humedad_1) + "% - Hum2: " + String(humedad_2) + "% - Ilu1: " +
String(iluminacion_1) + "Lux - Ilu2: " + String(iluminacion_2) + "Lux]" + "
RSSI: " + String(Rssi_LoRa) + String("dBm"));
      LoRa.receive();
      Parpadeo();
      i = 0;  // Se reinicia el orden de las lecturas
```

```
      Lcontrol = false;
    } else {
      i++;
    }
  }
}
```

Arduino (Sensores)

```
#include <OneWire.h>            //Libreria_temperatura
#include <DallasTemperature.h>  //Libreria_temperatura

int LED = 13;
int controlLED = 0;
int tiempoSensoresSeg = 30;  //tiempo de censado

const int PinSensorTemp = 9;                      //temperatura_1_2
OneWire oneWireObjeto(PinSensorTemp);             //temperatura_1_2
DallasTemperature sensorDS18B20(&oneWireObjeto);  //temperatura_1_2

const int PinSensorHum_1 = A0;  //Humedad_1
const int HumAmb_1 = 540;       //Humedad_1 BIEN 8
const int HumAgua_1 = 245;      //Humedad_1 BIEN 99
double HumSolido_1 = 0;         //Humedad_1

const int PinSensorHum_2 = A1;  //Humedad_2
const int HumAmb_2 = 505;       //Humedad_2 BIEN 8
const int HumAgua_2 = 215;      //Humedad_2 BIEN 98
double HumSolido_2 = 0;         //Humedad_2

#define luz_1 A2            //Iluminacion_1
const int IluMax_1 = 1015;  //Iluminacion_1
const int IluMin_1 = 0;     //Iluminacion_1
double IluSen_1 = 0;        //Iluminacion_1

#define luz_2 A3            //Iluminacion_2
const int IluMax_2 = 1018;  //Iluminacion_2
const int IluMin_2 = 0;     //Iluminacion_2
double IluSen_2 = 0;        //Iluminacion_2

float Sincronizacion = 11111110, Mul = 0, Mul2 = 0;

float data[7];

int i = 0, ID = 0;
```

```
void setup() {
  pinMode(LED, OUTPUT);
  digitalWrite(LED, LOW);
  Serial.begin(9600);     //temp
  sensorDS18B20.begin();  //temp
}

void loop() {

  ID++;

  const double data[] = { Sincronizacion, ID, Temperatura_1(),
Temperatura_2(), Humedad_1(), Humedad_2(), Iluminacion_1(), Iluminacion_2()
};
  const size_t dataLength = sizeof(data) / sizeof(data[0]);

  for (int i = 0; i < 8; i++) {
    Serial.println(data[i]);
  }

  if (controlLED == 6) {  //aviso de lectura independiente completa
    digitalWrite(LED, HIGH);
    delay(100);
    digitalWrite(LED, LOW);
    controlLED = 0;
  }
  delay(59000);
}

double Temperatura_1() {
  sensorDS18B20.requestTemperatures();                    //Funcion para que
el sensor 0 tomen la temperatura
  double temp = float(sensorDS18B20.getTempCByIndex(0));  //Se guarda la
temperatura
  controlLED++;                                           //Control de las
lecturas completas
  return temp;                                            //Devuelve la
temperatura
}

double Temperatura_2() {
  sensorDS18B20.requestTemperatures();                     //Funcion para
que el sensor 1 tomen la temperatura
  double temp2 = float(sensorDS18B20.getTempCByIndex(1));  //Se guarda la
temperatura
```

```
  controlLED++;                                          //Control de las
lecturas completas
  return temp2;                                          //Devuelve la
temperatura
}

double Humedad_1() {

  double LecSensor_1 = analogRead(PinSensorHum_1);                //Humedad
  if (LecSensor_1 >= HumAgua_1 && LecSensor_1 <= HumAmb_1) {      //Humedad
    HumSolido_1 = map(LecSensor_1, HumAmb_1, HumAgua_1, 0, 100);  //Humedad
  } else {                                                        //Humedad
    if (LecSensor_1 > HumAmb_1)                                   //Humedad
      HumSolido_1 = 0;                                            //Humedad
    else if (LecSensor_1 < HumAgua_1)                             //Humedad
      HumSolido_1 = 100;                                          //Humedad
  }
  controlLED++;  //Humedad
  return HumSolido_1;
}

double Humedad_2() {

  double LecSensor_2 = analogRead(PinSensorHum_2);                //Humedad
  if (LecSensor_2 >= HumAgua_2 && LecSensor_2 <= HumAmb_2) {      //Humedad
    HumSolido_2 = map(LecSensor_2, HumAmb_2, HumAgua_2, 0, 100);  //Humedad
  } else {                                                        //Humedad
    if (LecSensor_2 > HumAmb_2)                                   //Humedad
      HumSolido_2 = 0;                                            //Humedad
    else if (LecSensor_2 < HumAgua_2)                             //Humedad
      HumSolido_2 = 100;                                          //Humedad
  }
  controlLED++;  //Humedad
  return HumSolido_2;
}

double Iluminacion_1() {
  double LecSIlu_1 = analogRead(luz_1);
  if (LecSIlu_1 >= IluMin_1 && LecSIlu_1 <= IluMax_1) {      //Humedad
    IluSen_1 = map(LecSIlu_1, IluMin_1, IluMax_1, 0, 2000);  //Humedad
  } else {                                                   //Humedad
    if (LecSIlu_1 > IluMax_1)                                //Humedad
      IluSen_1 = 2000;                                       //Humedad
    else if (LecSIlu_1 < IluMin_1)                           //Humedad
      IluSen_1 = 0;                                          //Humedad
```

```
  }
  controlLED++;  //Humedad
  return IluSen_1;
}

double Iluminacion_2() {
  double LecSIlu_2 = analogRead(luz_2);
  if (LecSIlu_2 >= IluMin_2 && LecSIlu_2 <= IluMax_2) {      //Humedad
    IluSen_2 = map(LecSIlu_2, IluMin_2, IluMax_2, 0, 2000);  //Humedad
  } else {                                                   //Humedad
    if (LecSIlu_2 > IluMax_2)                                //Humedad
      IluSen_2 = 2000;                                       //Humedad
    else if (LecSIlu_2 < IluMin_2)                           //Humedad
      IluSen_2 = 0;                                          //Humedad
  }
  controlLED++;  //Humedad
  return IluSen_2;
}
```

Irradiación wifi

En este apartado se mostrará el código utilizado para generar la irradiación wifi en la banda de 2.4 GHz y el posterior envió de la información a las plataforma de almacenamiento y monitoreo.

Cliente wifi (modulo NodeMCU)

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <PubSubClient.h>  //https://github.com/knolleary/pubsubclient
#include <Arduino_JSON.h>  //https://github.com/arduino-
libraries/Arduino_JSON
#include <ArduinoJson.h>
#include <Wire.h>
#include <NTPClient.h>
#include <WiFiUdp.h>
#include <ESPDateTime.h>
#include <TimeLib.h>

const char* ssid = "XXXXX";
const char* contrasena = "XXXXX";

//const char* ssid2 = "XXXXX";
//const char* contrasena2 = "XXXXX";
```

```
const char* ssid2 = "XXXXX";
const char* contrasena2 = "XXXXX";

const char* topic_temperatura_1 = "641363b8bbfd68a236b97648/temperatura_1";
const char* topic_temperatura_2 = "641363b8bbfd68a236b97648/temperatura_2";
const char* topic_humedad_1 = "641363b8bbfd68a236b97648/humedad_1";
const char* topic_humedad_2 = "641363b8bbfd68a236b97648/humedad_2";
const char* topic_iluminacion_1 = "641363b8bbfd68a236b97648/iluminacion_1";
const char* topic_iluminacion_2 = "641363b8bbfd68a236b97648/iluminacion_2";
const char* topic_potencia_wifi = "641363b8bbfd68a236b97648/potenciawifi";

const char* mqtt_server_mikrodash = "mqtt.mikrodash.com";  //Servidor MQTT
const int mqtt_port_mikrodash = 1883;                      //Puerto MQTT

const char* Usuario_MQTT_Mikrodash = "johan_168193521";  //Credenciales
const char* Contrasena_MQTT_Mikrodash = "12345";

#define MSG_TAMANO_BUFFER (150)
char msg_1[MSG_TAMANO_BUFFER], msg_2[MSG_TAMANO_BUFFER],
msg_3[MSG_TAMANO_BUFFER], msg_4[MSG_TAMANO_BUFFER],
msg_5[MSG_TAMANO_BUFFER], msg_6[MSG_TAMANO_BUFFER],
msg_7[MSG_TAMANO_BUFFER];

const char* host = "http://192.168.1.100/POST";

WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, "3.co.pool.ntp.org");

bool Control_Reloj = false;
bool Control_Reconexion = false;
bool Control_Reconexion2 = false;
unsigned long TiempoActual = 0;
unsigned long TiempoPrevio = 0;
int Tiempo_Reconexion = 10000000;  //10 segundos
int Tipo_Lectura = 0;
bool Reinicio_Lectura;
float ID = 0, sincronizacion = 0, temperatura_1 = 0, temperatura_2 = 0,
humedad_1 = 0, humedad_2 = 0, iluminacion_1 = 0, iluminacion_2 = 0,
RSSI_wifi = 0;
int Control_Certificados = 0;

const char* mqtt_server_AWS = "a1rxncrcrlwp5f-ats.iot.us-east-
1.amazonaws.com";  //MQTT broker ip
const int mqtt_port_AWS = 8883;
const int Nivel_QoS = 2;  //Calidad del servicio MQTT
```

```
bool Control_CertificadosAWS = false;

void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Mensaje Recibido [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
  Serial.println();
}

WiFiClientSecure espClient;
PubSubClient client(mqtt_server_AWS, mqtt_port_AWS, callback,
espClient);  //MQTT port 8883 - standard

WiFiClient esp2Client;
PubSubClient client2(esp2Client);  //MQTT port 8883 - standard

void Conexion_Broker_AWS() {  //Empieza la conexion al broker

  if (Control_CertificadosAWS == false) {
    CertificadosAWS();  //Luego de lograr la conexion a una red con internte
envia a la carga de certificados*/
    Control_CertificadosAWS = true;
  }

  while (!client.connected()) {
    Reconectar();
    delay(500);
  }
}

void MensajeDynamoDB() {

  timeClient.update();
// por si acaso de actualuiza el tiempo
  time_t timestamp =
timeClient.getEpochTime();                              //Conversion de el
tiempo de segundos a un formato legible
  struct tm* tmstruct =
localtime(&timestamp);                                  //Conversion de el
tiempo de segundos a un formato legible
```

```
  char
jsonstring[20];                                                    //Conve
rsion de el tiempo de segundos a un formato legible
  strftime(jsonstring, sizeof(jsonstring), "%Y-%m-%d %H:%M:%S",
tmstruct);  //Conversion de el tiempo de segundos a un formato legible

  StaticJsonDocument<200> doc;  //Creacion del archivo JSON a enviar al
broker
  //doc["fecha"] = timeClient.getEpochTime(); //estra es la estampa de
tiempo (segundos trancurridos desde 1970)
  doc["fecha"] = jsonstring;
  doc["ID"] = ID;
  char T1[10];
  sprintf(T1, "%.2f", temperatura_1);
  doc["1_Temperatura_1_(C)"] = T1;
  char T2[10];
  sprintf(T2, "%.2f", temperatura_2);
  doc["2_Temperatura_2_(C)"] = T2;
  doc["3_Humedad_1_(%)"] = humedad_1;
  doc["4_Humedad_2_(%)"] = humedad_2;
  doc["5_Iluminacion_1_(Lux)"] = iluminacion_1;
  doc["6_Iluminacion_2_(Lux)"] = iluminacion_2;
  doc["7_Rssi_(dBm)"] = RSSI_wifi;
  doc["8_Tipo"] = "wifi";
  char Buffer[512];                                  //Se guarda todos
los datos en el buffer en tipo char
  serializeJson(doc, Buffer);                        //se llena en JSON
creado con la informacion guardada en el buffer
  client.publish("Datos_Proyecto", Buffer, Nivel_QoS);  //Topico donde se
publica
  Serial.println("[»] Datos publicados correctamente en DynamoDB-AWS");
}

void Conexion_Broker_MikroDash() {

  while (!client2.connected()) {
    Serial.print("[!] Conectandose al broker MQTT de monitoreo...");
    String clientId = "MikroDashWiFiClient-
";                                                  // Se genera un ID
aleatorio con
    clientId += String(random(0xffff),
HEX);                                                 // Se genera un ID
aleatorio con
    if (client2.connect(clientId.c_str(), Usuario_MQTT_Mikrodash,
Contrasena_MQTT_Mikrodash)) {  //Cuando se conecta
```

```
      Serial.println(" [Conectado exitosamente al MQTT server:
mqtt.mikrodash.com]");
    } else {
      Serial.print("[X] Error al conectar:");
      Serial.print(client2.state());
      Serial.println(" Reintentando en 3 segundos");
      delay(3000);
    }
  }
}

void MensajeMykroDash() {

  char tem_char[6];
  dtostrf(temperatura_1, 3, 3, tem_char);
  sprintf(msg_1,
"{\"from\":\"device\",\"message\":\"Temperatura_1\",\"save\":false,\"value\"
: %s}", tem_char);
  client2.publish(topic_temperatura_1, msg_1, true);

  char tem2_char[6];
  dtostrf(temperatura_2, 3, 3, tem2_char);
  sprintf(msg_2,
"{\"from\":\"device\",\"message\":\"Temperatura_2\",\"save\":false,\"value\"
: %s}", tem2_char);
  client2.publish(topic_temperatura_2, msg_2, true);

  char hum_char[6];
  dtostrf(humedad_1, 3, 3, hum_char);
  sprintf(msg_3,
"{\"from\":\"device\",\"message\":\"Humedad_1\",\"save\":true,\"value\":
%s}", hum_char);
  client2.publish(topic_humedad_1, msg_3, true);

  char hum2_char[6];
  dtostrf(humedad_2, 3, 3, hum2_char);
  sprintf(msg_4,
"{\"from\":\"device\",\"message\":\"Humedad_2\",\"save\":true,\"value\":
%s}", hum2_char);
  client2.publish(topic_humedad_2, msg_4, true);

  char ilu_char[6];
  dtostrf(iluminacion_1, 3, 3, ilu_char);
```

```
  sprintf(msg_5,
"{\"from\":\"device\",\"message\":\"Iluminacion_1\",\"save\":false,\"value\"
: %s}", ilu_char);
  client2.publish(topic_iluminacion_1, msg_5, true);

  char ilu2_char[6];
  dtostrf(iluminacion_2, 3, 3, ilu2_char);
  sprintf(msg_6,
"{\"from\":\"device\",\"message\":\"Iluminacion_2\",\"save\":false,\"value\"
: %s}", ilu2_char);
  client2.publish(topic_iluminacion_2, msg_6, true);

  char rssi_char[6];
  dtostrf(RSSI_wifi, 3, 3, rssi_char);
  sprintf(msg_7,
"{\"from\":\"device\",\"message\":\"Rssi\",\"save\":false,\"value\": %s}",
rssi_char);
  client2.publish(topic_potencia_wifi, msg_7, true);

  Serial.println("[»] Datos publicados correctamente en MikroDASH");
  client2.disconnect();
}

void setup() {
  Serial.begin(9600);
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, LOW);
  WiFi.disconnect();
  Serial.println();
  Serial.println("IRRADIACION wifi [2.4 GHz]");
}

void Setup_wifi() {

  WiFi.begin(ssid, contrasena);
  Serial.print("[-] Conectando a: " + String(ssid) + String("... "));

  TiempoPrevio = micros();
  while (WiFi.status() != WL_CONNECTED && Control_Reconexion == false) {
    TiempoActual = micros();
    if (TiempoActual - TiempoPrevio >= Tiempo_Reconexion) {
      Serial.println("[X] Conexion perdida/Servidor no disponible]");
      Control_Reconexion = true;
      TiempoPrevio = TiempoActual;
    }
```

```
    delay(1);
  }

  if (WiFi.status() == WL_CONNECTED) {
    Serial.println("");
    Serial.print("[-] Conectado por WiFi 2.4 GHz a " + String(ssid) + " -
Ip: ");
    Serial.println(WiFi.localIP());
  }
  RSSI_wifi = WiFi.RSSI();
}

void Setup_wifi2() {

  delay(10);
  espClient.setBufferSizes(1024, 1024);
  WiFi.begin(ssid2, contrasena2);
  Serial.print("[-] Conectando a: " + String(ssid2));

  TiempoPrevio = micros();
  while (WiFi.status() != WL_CONNECTED && Control_Reconexion2 == false) {
    TiempoActual = micros();
    if (TiempoActual - TiempoPrevio >= Tiempo_Reconexion) {
      Serial.println(" [X] Conexion perdida/Servidor no disponible");
      Control_Reconexion2 = true;
      TiempoPrevio = TiempoActual;
    }
    delay(1);
  }

  if (WiFi.status() == WL_CONNECTED) {
    Serial.print(" [Conectado por WiFi 2.4 GHz a " + String(ssid2) + " - Ip:
");
    Serial.print(WiFi.localIP());
    Serial.println("]");
  }

}

void Reconectar() {  //Reconexion  al brokerAWS en caso de ser necesario
  while (!client.connected()) {
    Serial.print("[!] Conectandose al broker MQTT de almacenamiento... ");
    if (client.connect("ESPthing")) {
      delay(10);
      Serial.println("[Conectado exitosamente al MQTT server de AWS]");
```

```
    } else {
      Serial.print("[X] Fallo, rc=");
      Serial.print(client.state());
      Serial.print(" Reintentando en 1 segundos... ");
      char buf[256];
      espClient.getLastSSLError(buf, 256);
      Serial.print("[X] WiFiClientSecure SSL error: ");
      Serial.println(buf);
      delay(1000);
    }
  }
}

void CertificadosAWS() {  //Cargado de todos los certificados previamente convertidos de 64 a bianrio

  if (!SPIFFS.begin()) {  //Comprueba de que haya archivos en la memoria del ESP
    Serial.println("[X] Sin Archivos que cargar");
    return;
  }

  File cert = SPIFFS.open("/cert.der", "r");  //verifica el archivo cert

  if (espClient.loadCertificate(cert)) {  //Carga el archivo para empezar la autenticacion el el broker
    Control_Certificados++;
    Serial.print("[!] Cert cargado... ");
  } else {
    Serial.print("[X] Cert no cargado... ");
  }

  // Cargamos llave privada
  File private_key = SPIFFS.open("/private.der", "r");  //verifica la llave primaria

  if (espClient.loadPrivateKey(private_key)) {  //Carga el archivo para la autenticacion el el broker
    Control_Certificados++;
    Serial.print("Private Key Cargada... ");
  } else {
    Serial.print("[X] Private key no cargada... ");
  }
  // Cargamos CA
  File ca = SPIFFS.open("/ca.der", "r");  //verifica el archivo cert
```

```
  if (espClient.loadCACert(ca)) {  //Carga el archivo para la autenticacion
el el broker
    Control_Certificados++;
    Serial.print("Ca cargado...");
  } else {
    Serial.print("[X] Ca no cargado");
  }

  if (Control_Certificados == 3)
    Serial.println(" [Cetificados AWS cargados con exito]");
  else
    Serial.println(" [Cetificados AWS no cargados]");

  Control_Certificados = 0;
}

void datos() {
  WiFiClient client;
  HTTPClient http;
  String postData;

  postData = "ID=" + String(ID) + "&Temperatura_1=" + String(temperatura_1)
+ "&Temperatura_2=" + String(temperatura_2) + "&Humedad_1=" +
String(humedad_1) + "&Humedad_2=" + String(humedad_2) + "&Iluminacion_1=" +
String(iluminacion_1) + "&Iluminacion_2=" + String(iluminacion_2) + "&rssi="
+ String(RSSI_wifi);
  Serial.println("-----------------------------------------------------[" +
String(ID) + "]-----------------------------------------------------");
  Serial.println("[»] Datos POST=" + String(postData));

  http.begin(client, host);
  http.addHeader("Content-Type", "application/x-www-form-urlencoded");
  int httpcode = http.POST(postData);
  String payload = http.getString();
  Serial.println("[»] Codigo:" + String(httpcode) + " - Respuesta servidor:
" + String(payload));
  http.end();
}

void Parpadeo() {
  digitalWrite(LED_BUILTIN, HIGH);  // turn the LED on (HIGH is the voltage
level)
  delay(100);                       // wait for a second
```

```
  digitalWrite(LED_BUILTIN, LOW);   // turn the LED off by making the
voltage LOW
  delay(100);
  digitalWrite(LED_BUILTIN, HIGH);  // turn the LED on (HIGH is the voltage
level)
  delay(100);
  digitalWrite(LED_BUILTIN, LOW);  // turn the LED off by making the voltage
LOW
}

void Obtener_Lecturas() {
  if (Serial.available() > 0) {                //Si hay datos disponibles
para leer
    String msg = Serial.readStringUntil('\n');  //se recibe el dato enviado
desde el arduino y se almacena en una variable String para recibirlo todo de
una y aparte se descarta el salto de linea para que no genere problemas al
momento de convertir el dato en un dobule para poderlo manipular
    RSSI_wifi =
WiFi.RSSI();                    ////////////////////OOOOOOJOOOOOO//////////
/////

    if (String(msg).toDouble() == 11111110.00) {
      sincronizacion = String(msg).toDouble();  //byte de sincroizacion para
saber cuando se inicia cada lectura
    } else if (Tipo_Lectura == 1) {             //Se guardan todos los datos
de acuerdo a su llegada y variables
      ID = String(msg).toDouble();
    } else if (Tipo_Lectura == 2) {
      temperatura_1 = String(msg).toDouble();
    } else if (Tipo_Lectura == 3) {
      temperatura_2 = String(msg).toDouble();
    } else if (Tipo_Lectura == 4) {
      humedad_1 = String(msg).toDouble();
    } else if (Tipo_Lectura == 5) {
      humedad_2 = String(msg).toDouble();
    } else if (Tipo_Lectura == 6) {
      iluminacion_1 = String(msg).toDouble();
    } else if (Tipo_Lectura == 7) {
      //iluminacion_2 = String(msg).toDouble();
      //iluminacion_2 = 0;
      iluminacion_2 = iluminacion_1;
      Reinicio_Lectura = true;  //Se habilita el envio de la informacion y
el reinicio del orden de las lecturas
    }
```

```
    if (Reinicio_Lectura == true) {
      Setup_wifi();
      datos();
      Setup_wifi2();
      client.setServer(mqtt_server_AWS, mqtt_port_AWS);
      Conexion_Broker_AWS();
      MensajeDynamoDB();
      client2.setServer(mqtt_server_mikrodash, mqtt_port_mikrodash);  //Se
coloca el servidor y puerto de MQTT para conectarse
      Conexion_Broker_MikroDash();
      MensajeMykroDash();
      Parpadeo();
      WiFi.disconnect();
      Tipo_Lectura = 0;  // Se reinicia el orden de las lecturas
      Reinicio_Lectura = false;
    } else {
      Tipo_Lectura++;
    }
  }
}

void loop() {

  Obtener_Lecturas();  //Envia a la funcion para atualizar las variables de
las lecturas

}
```

Servidor wifi (modulo NodeMCU)

```
#include <ESP8266WiFi.h>
#include "ESPAsyncWebServer.h"
#include <SPI.h>
#include <LoRa.h>

const char *ssid = "XXXXX";  //puntero:(*x) devuelve la variable cuya
direccion esta almcenada en (&x) variable (0xF64),(&x) direccion de memoria
de la vartiable
const char *contrasena = "XXXXX";
int canal = 6;
int Max_conexiones = 8;
```

```
AsyncWebServer server(80);  //Crear objeto AsyncWebServer en el puerto 80

int i = 0;
bool Lcontrol = false;
bool Control_LED = false;
String inString = "";  // string to hold input
float val = 0;

#define ss 15
#define rst 16
#define dio0 2

IPAddress ip(192, 168, 1, 100);
IPAddress gateway(192, 168, 1, 1);
IPAddress subnet(255, 255, 255, 0);

float ID = 0, sincronizacion = 0, temperatura_1 = 0, temperatura_2 = 0,
humedad_1 = 0, humedad_2 = 0, iluminacion_1 = 0, iluminacion_2 = 0;
int Rssi_LoRa = 0;

void Parpadeo() {
  digitalWrite(LED_BUILTIN, HIGH);  // turn the LED on (HIGH is the voltage
level)
  delay(100);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);   // turn the LED off by making the
voltage LOW
  delay(100);
  digitalWrite(LED_BUILTIN, HIGH);  // turn the LED on (HIGH is the voltage
level)
  delay(100);
  digitalWrite(LED_BUILTIN, LOW);  // turn the LED off by making the voltage
LOW
}

String readPOST() {
  return String("Paquete Recibido correctamente");
}

void Servidor() {

  Serial.println();
  Serial.println("[!] Configurando Servidor (ESP8266)…");
  WiFi.mode(WIFI_AP);
  WiFi.softAP(ssid, contrasena, canal, Max_conexiones);
```

```
  WiFi.softAPConfig(ip, gateway, subnet);
  //IPAddress IP = WiFi.softAPIP();

  IPAddress IP = WiFi.softAPIP();
  Serial.print("[-] Servidor configurado - Direccion IP del servidor: ");
  Serial.print(IP);
  Serial.print(", Red 2.4 GHz - Canal: ");
  Serial.println(canal);

  server.on("/POST", HTTP_POST, [](AsyncWebServerRequest *request) {
    request->send_P(200, "text/plain", readPOST().c_str());
    Serial.println("[»] [wifi] - ID: " + String(request->arg("ID")) + "
[Tem1: " + String(request->arg("Temperatura_1")) + "C Tem2: " +
String(request->arg("Temperatura_2")) + "C Hum1: " + String(request-
>arg("Humedad_1")) + "% Hum2: " + String(request->arg("Humedad_2")) + "%
Ilu1: " + String(request->arg("Iluminacion_1")) + "Lux Ilu2: " +
String(request->arg("Iluminacion_2")) + "Lux RSSI_wifi: " + String(request-
>arg("rssi")) + "dBm]");
    Control_LED = true;
  });

  server.begin();
}

void Setup_LoRa() {
  while (!Serial)
    ;
  Serial.println();
  Serial.println("[-] Receptor - LoRa 433 MHz - Configurando...");

  LoRa.setPins(ss, rst, dio0);

  while (!LoRa.begin(433E6))  //433E6 - Asia, 866E6 - Europe, 915E6 - North
America
  {
    Parpadeo();
    Serial.println(".");
  }
  LoRa.setSyncWord(0xA5);
  Serial.println("[!] Rx LoRa 433 MHz - Listo");
  Serial.println();
}

void setup() {
```

```
  Serial.begin(9600);
  WiFi.disconnect();
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, LOW);
  Servidor();
  Setup_LoRa();
}

void loop() {

  if (Control_LED == true) {
    Parpadeo();
    Control_LED = false;
  }

  paquete_LoRa();
}

void paquete_LoRa() {
  int packetSize = LoRa.parsePacket();
  if (packetSize) {
    while (LoRa.available()) {
      int inChar = LoRa.read();
      inString += (char)inChar;
      val = inString.toDouble();
    }
    inString = "";
    if (val == 11111110.00) {
      sincronizacion = val;
    } else if (i == 1) {
      ID = val;
    } else if (i == 2) {
      temperatura_1 = val;
    } else if (i == 3) {
      temperatura_2 = val;
    } else if (i == 4) {
      humedad_1 = val;
    } else if (i == 5) {
      humedad_2 = val;
    } else if (i == 6) {
      iluminacion_1 = val;
    } else if (i == 7) {
      iluminacion_2 = val;
    } else if (i == 8){
      Rssi_LoRa = val;
```

```
      Lcontrol = true;
    }

    if (Lcontrol == true) {

      Serial.println("[»] [LoRa] - ID: " + String(ID) + " [Tem1: " +
String(temperatura_1) + "C - Tem2: " + String(temperatura_2) + "C - Hum1:" +
String(humedad_1) + "% - Hum2: " + String(humedad_2) + "% - Ilu1: " +
String(iluminacion_1) + "Lux - Ilu2: " + String(iluminacion_2) + "Lux]" + "
RSSI: " + String(Rssi_LoRa) + String("dBm"));
      LoRa.receive();
      Parpadeo();
      i = 0;  // Se reinicia el orden de las lecturas
      Lcontrol = false;
    } else {
      i++;
    }
  }
}
```

Arduino (Sensores)

```
#include <OneWire.h>            //Libreria_temperatura
#include <DallasTemperature.h>  //Libreria_temperatura

int LED = 13;
int controlLED = 0;
int tiempoSensoresSeg = 30;  //tiempo de censado

const int PinSensorTemp = 9;                      //temperatura_1_2
OneWire oneWireObjeto(PinSensorTemp);             //temperatura_1_2
DallasTemperature sensorDS18B20(&oneWireObjeto);  //temperatura_1_2

const int PinSensorHum_1 = A0;  //Humedad_1
const int HumAmb_1 = 540;       //Humedad_1 BIEN 8
const int HumAgua_1 = 245;      //Humedad_1 BIEN 99
double HumSolido_1 = 0;         //Humedad_1

const int PinSensorHum_2 = A1;  //Humedad_2
const int HumAmb_2 = 505;       //Humedad_2 BIEN 8
const int HumAgua_2 = 215;      //Humedad_2 BIEN 98
double HumSolido_2 = 0;         //Humedad_2

#define luz_1 A2            //Iluminacion_1
```

```
const int IluMax_1 = 1015;  //Iluminacion_1
const int IluMin_1 = 0;     //Iluminacion_1
double IluSen_1 = 0;        //Iluminacion_1

#define luz_2 A3            //Iluminacion_2
const int IluMax_2 = 1018;  //Iluminacion_2
const int IluMin_2 = 0;     //Iluminacion_2
double IluSen_2 = 0;        //Iluminacion_2

float Sincronizacion = 11111110, Mul = 0, Mul2 = 0;

float data[7];

int i = 0, ID = 0;

void setup() {
  pinMode(LED, OUTPUT);
  digitalWrite(LED, LOW);
  Serial.begin(9600);     //temp
  sensorDS18B20.begin();  //temp
}

void loop() {

  ID++;

  const double data[] = { Sincronizacion, ID, Temperatura_1(),
Temperatura_2(), Humedad_1(), Humedad_2(), Iluminacion_1(), Iluminacion_2()
};
  const size_t dataLength = sizeof(data) / sizeof(data[0]);

  for (int i = 0; i < 8; i++) {
    Serial.println(data[i]);
  }

  if (controlLED == 6) {  //aviso de lectura independiente completa
    digitalWrite(LED, HIGH);
    delay(100);
    digitalWrite(LED, LOW);
    controlLED = 0;
  }
  delay(59000);
}

double Temperatura_1() {
```

```
  sensorDS18B20.requestTemperatures();                    //Funcion para que
el sensor 0 tomen la temperatura
  double temp = float(sensorDS18B20.getTempCByIndex(0));  //Se guarda la
temperatura
  controlLED++;                                           //Control de las
lecturas completas
  return temp;                                            //Devuelve la
temperatura
}

double Temperatura_2() {
  sensorDS18B20.requestTemperatures();                     //Funcion para
que el sensor 1 tomen la temperatura
  double temp2 = float(sensorDS18B20.getTempCByIndex(1));  //Se guarda la
temperatura
  controlLED++;                                            //Control de las
lecturas completas
  return temp2;                                            //Devuelve la
temperatura
}

double Humedad_1() {

  double LecSensor_1 = analogRead(PinSensorHum_1);                //Humedad
  if (LecSensor_1 >= HumAgua_1 && LecSensor_1 <= HumAmb_1) {      //Humedad
    HumSolido_1 = map(LecSensor_1, HumAmb_1, HumAgua_1, 0, 100);  //Humedad
  } else {                                                        //Humedad
    if (LecSensor_1 > HumAmb_1)                                   //Humedad
      HumSolido_1 = 0;                                            //Humedad
    else if (LecSensor_1 < HumAgua_1)                             //Humedad
      HumSolido_1 = 100;                                          //Humedad
  }
  controlLED++;  //Humedad
  return HumSolido_1;
}

double Humedad_2() {

  double LecSensor_2 = analogRead(PinSensorHum_2);                //Humedad
  if (LecSensor_2 >= HumAgua_2 && LecSensor_2 <= HumAmb_2) {      //Humedad
    HumSolido_2 = map(LecSensor_2, HumAmb_2, HumAgua_2, 0, 100);  //Humedad
  } else {                                                        //Humedad
    if (LecSensor_2 > HumAmb_2)                                   //Humedad
      HumSolido_2 = 0;                                            //Humedad
    else if (LecSensor_2 < HumAgua_2)                             //Humedad
```

```
      HumSolido_2 = 100;                                          //Humedad
  }
  controlLED++;  //Humedad
  return HumSolido_2;
}

double Iluminacion_1() {
  double LecSIlu_1 = analogRead(luz_1);
  if (LecSIlu_1 >= IluMin_1 && LecSIlu_1 <= IluMax_1) {      //Humedad
    IluSen_1 = map(LecSIlu_1, IluMin_1, IluMax_1, 0, 2000);  //Humedad
  } else {                                                   //Humedad
    if (LecSIlu_1 > IluMax_1)                                //Humedad
      IluSen_1 = 2000;                                       //Humedad
    else if (LecSIlu_1 < IluMin_1)                           //Humedad
      IluSen_1 = 0;                                          //Humedad
  }
  controlLED++;  //Humedad
  return IluSen_1;
}

double Iluminacion_2() {
  double LecSIlu_2 = analogRead(luz_2);
  if (LecSIlu_2 >= IluMin_2 && LecSIlu_2 <= IluMax_2) {      //Humedad
    IluSen_2 = map(LecSIlu_2, IluMin_2, IluMax_2, 0, 2000);  //Humedad
  } else {                                                   //Humedad
    if (LecSIlu_2 > IluMax_2)                                //Humedad
      IluSen_2 = 2000;                                       //Humedad
    else if (LecSIlu_2 < IluMin_2)                           //Humedad
      IluSen_2 = 0;                                          //Humedad
  }
  controlLED++;  //Humedad
  return IluSen_2;
}
```