REINICIO REMOTO POR INTERNET DE UN ROUTER UTILIZANDO EL WIFI QUE SUMINISTRA EL PROPIO ROUTER

Versión 1.03

Por Lino, EB2CTA

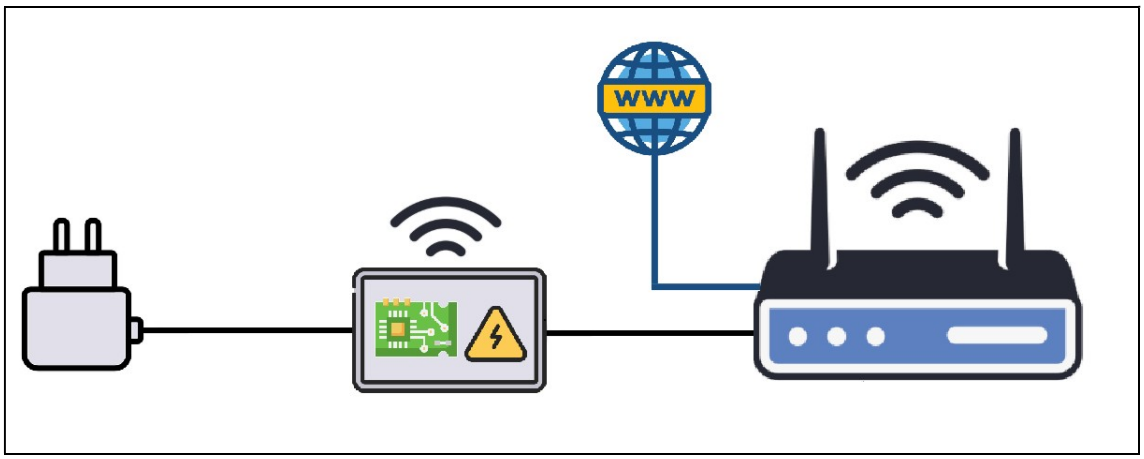
INTRODUCCION Y DESCRIPCION DEL PROYECTO

El proyecto consiste en poder reiniciar un router conectado que gestiona una red y reiniciarlo en remoto a través de Internet desde cualquier dispositivo que permita acceder a un navegador Web.

Un problema habitual es que algunos dispositivos conectados a un Router dejan de funcionar correctamente o incluso desconectarse de la red local y por ende de perder su conectividad a Internet. En la mayoría de los casos es suficiente restablecer el sistema apagando y encendiendo el Router y haciendo que los dispositivos se vuelvan a conectar al Router de nuevo, lo que les hace obtener otra vez una IP correcta.

Cuando el acceso físico al Router no es posible y/o por cualquier razón, no está habilitado la gestión remota del Router (muchas veces por que la operadora no lo permite) podemos instalar un pequeño dispositivo auto construido y basado en un Arduino ESP32 que interrumpa momentáneamente la alimentación al Router y obligue a su reinicio.

La gracia del invento reside en que, el acceso remoto para el reinicio del Router se hace mediante una Web servida por el Arduino ESP32 y que, una vez reiniciado vuelve a estar operativa la red WiFi local y restablecida su acceso completo a Internet.



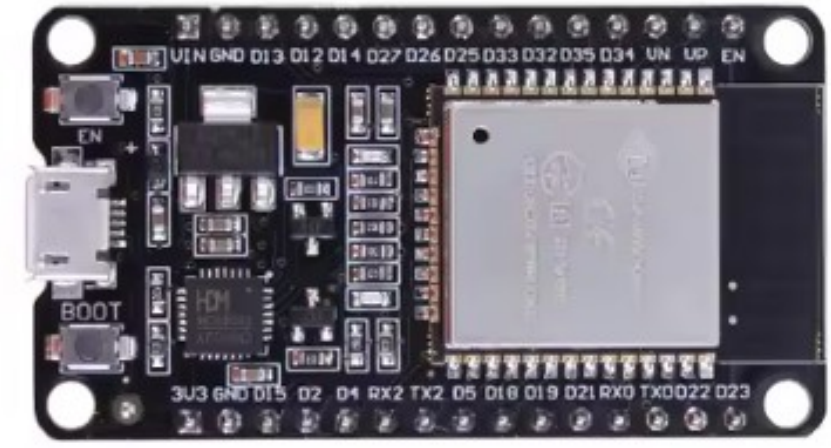
El dispositivo se conecta entre el alimentador y el Router.

HARDWARE

Se describen a continuación los principales materiales necesarios para el montaje:

Materiales:

- **Placa de desarrollo ESP32 CP2102 TYPE-C/MICRO USB WiFi + Bluetooth consumo de energía ultrabajo ESP-32S de doble núcleo para hogar inteligente**

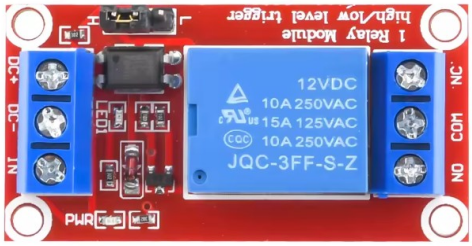


Especificaciones:
Especificaciones: 463B (CH9102X), 463C (CP2102 TYPE-C) (opcional)
SPIFlash: predeterminado 32Mbit
Compatible con Bluetooth: estándares Bluetooth 4.2 BR/EDR y BLE
WiFi: 802.11 b/g/n/
Interfaces compatibles: UART, SPI, SDIO, I2C, PWM, I2S, IR, ADC, DAC
Sensores en chip: sensores Hall, sensores de temperatura, sensores táctiles capacitivos
Velocidad del puerto serie: predeterminado 115200 bps
Rango de espectro: 2412 ~ 2484MHz
Forma de antena: antena PCB integrada con una ganancia de 2dBi
Potencia de transmisión:
802.11b:17±2dBm(@11Mbps)
802.11g: 14+2dBm(@54Mbps)
802.11n:13±2dBm(@MCS7)
Sensibilidad de recepción:
CCK, 1Mbps:-90dBm
CCK, 11Mbps: -85dBm 6Mbps (1/2BPSK): -88dBm 54Mbps(3/464-QAM):-70dBm
MCS7 (65Mbps, 72,2Mbps):-67dBm
Perdida de energía: 300 mA a 3,3 V
Seguridad: WPA/WPA2/WPA2 Enterprise/WPS
Rango de fuente de alimentación: Fuente de alimentación USB: 5 V Fuente de alimentación de pin Vin: 5,0 V ~ 12 V
Temperatura de trabajo: -20~85℃
Entorno de almacenamiento: -40~90,<90% RH
Tamaño del producto: aproximadamente 52 * 29 * 15 mm/2,04 * 1,14 * 0,59 pulgadas

1. Módulo WiFi+BT 802.11b/g/g;
2. CPU de 32 bits de doble núcleo de bajo consumo, que se puede utilizar como procesador de aplicaciones;

3. La frecuencia principal puede alcanzar hasta 240 MHz y la potencia informática puede alcanzar hasta 600 DMIPS;
4. Construido en SRAM de 520 KB;
5. Admite interfaces como UART/SPI/I2C/PWM/DAC/ADC;
6. Admite interfaz de depuración abierta OCD;
7. Admite múltiples modos de suspensión, con una corriente de 6,5 uA durante el sueño profundo;
8. Lwip y FreeRTOS integrados;
9. Admite el modo STA/AP/STA+AP;
10. Admite distribución de red con un clic de Smart Config/Airkiss;
11. Comando AT universal, fácil de usar;
12. Admite actualización del puerto serie local y actualización remota (FOTA).

- **Módulo de relé de 1 canal de 12 V, usando aislamiento de optoacoplador bidireccional, con gran capacidad de conducción y rendimiento estable**



Características del producto:
1. Adoptando aislamiento de optoacoplador bidireccional, con gran capacidad de conducción y rendimiento estable.
2. El módulo se puede activar configurando niveles altos o bajos a través de cables de puente.
3. Hay 4 orificios para tornillos fijos.
4. Equipado con una luz indicadora de encendido (roja).
5. El diseño de la interfaz es fácil de usar y todas las interfaces se pueden conectar directamente a través de bloques de terminales.

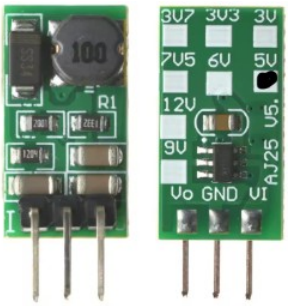
Parámetros del producto:
Corriente de disparo: 5mA
Voltaje de funcionamiento del módulo: DC12V
Corriente de carga: 2A
Tamaño del módulo: 50 * 26 milímetros * 18,5 milímetros (largo * ancho * alto)
Hay cuatro orificios para pernos fijos, con un diámetro de 3,1 milímetros y un espacio de 44,5 milímetros por 20,5 milímetros.

Interfaz del módulo:
DC+: Conecte el polo positivo de la fuente de alimentación (voltaje según los requisitos del relé)
DC-: Conecte el polo negativo de la fuente de alimentación.
IN: Puede controlar el cierre del relé en niveles altos o bajos
Terminal de salida de relé:
NO: Interfaz de relé normalmente abierta, suspendida antes de cerrar, en cortocircuito a COM después del cierre
COM: Interfaz común de relé
NC: Interfaz de relé normalmente cerrada, en cortocircuito a COM antes del cierre del relé, suspendida después del cierre
Terminal de selección de disparador de nivel alto y bajo
Cuando el puente está en cortocircuito a BAJO, se activa un nivel bajo;
Cuando el puente está en cortocircuito a alto, se activa un nivel alto.

- **Módulo de tarjeta Micro SD TF, módulo de tarjeta Mini SD, módulo de memoria para Arduino ARM AVR**

Descripción
Módulo de tarjeta Mini SD TF Módulo de memoria Mini tarjeta SD para Arduino AVR ARM
1. Interfaz de tarjeta MINI SD emergente integrada
2. Los pines relevantes han sido extraídos y marcados
3. Tamaño: 18,5 (mm) x 17,5 (mm)

- **Módulo convertidor reductor DD4012SA: regulador reductor de 5 V ~ 40 V a 5V, salida de 1 A para electrónica de bricolaje**



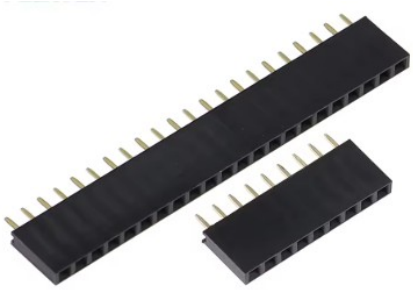
Descripción
Voltaje de entrada 5 ~ 40 V, salida 5 V
Corriente máxima de salida 1A
Frecuencia de trabajo del módulo convertidor reductor DC-DC 550 KHz.La eficiencia es del 76-90%.
Corriente de reposo: alrededor de 1,5 mA
Apagado por sobretemperatura, protección contra subtensión de entrada, protección contra voltaje BS y protección contra cortocircuitos.
Paso de pin de 2,54 mm, placa de desarrollo MCU Arduiuo UNO MEGA2560 compatible con placa de pruebas
Excluyendo el tamaño del pasador 18,3 mm x 10,2 mm x 6,7 mm (pequeño)
Peso: alrededor de 1,3 g (muy ligero)
Atención :
Este es un módulo convertidor de voltaje DC-DC. Se debe tener en cuenta al usar:
1. El voltaje de entrada no puede ser mayor que el rango de entrada máximo
2. La potencia de salida no puede ser mayor que la carga máxima durante mucho tiempo.
3. La potencia de entrada debe ser mayor que la potencia de salida, debido al consumo de energía del propio módulo.

- **Tarjeta de memoria de alta velocidad para teléfono inteligente, MiniSD Clase 10 de 4GB, tarjeta TF**

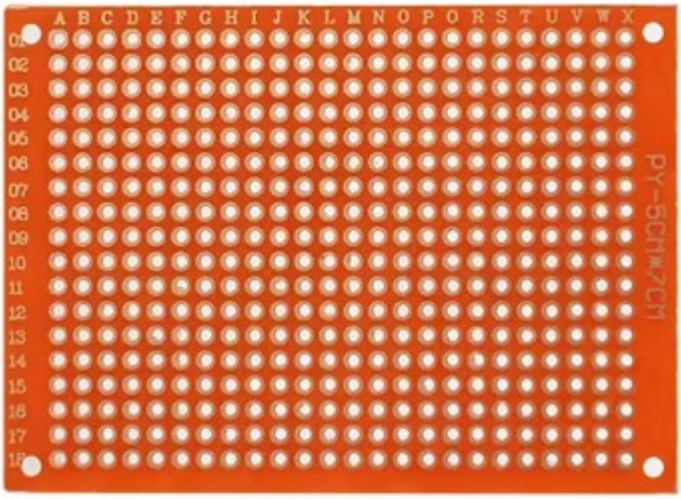
En realidad, cualquier tarjeta MiniSD servirá siempre que esté formateada como FAT (File Allocation Table).

- **Conector DC Macho/Hembra 3A 12V 5.5x2.1/2.5mm DC022B&DC005 con Tuercas para Montaje en Panel Kit DC-022B**

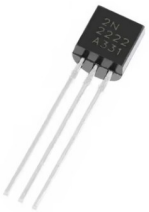
- **Enchufe hembra de una sola fila, 2,54mm, 15p, conector Pin**



- **Placa Universal PCB de un solo lado amarillo 5*7CM,7X9Cm, Kit de tablero de rejilla perforada prototipo de placa PCB, para soldadura DIY**



- **Transistor 2N2222 o 2N2222A encapsulado TO-92**



- **Resistencias varias (4k7, 10k y 100k)**

- **Caja impermeable DIY de 92x58x23mm, caja de proyectos electrónicos, caja de Control de conexiones de almacenamiento de instrumentos blanco**

Diseño:

El Arduino ESP32 comanda un relé activando el pin 12. Al recibir la orden desde la Web alojada en el mismo Arduino ESP32, es decir, se comporta como un servidor Web. Al activar el botón 

se abre el relé y espera 10 segundos antes de reconectar el Router. Es importante que el puente del relé esté en la posición H al objeto de que se active cuando reciba señal HIGH del pin 12.

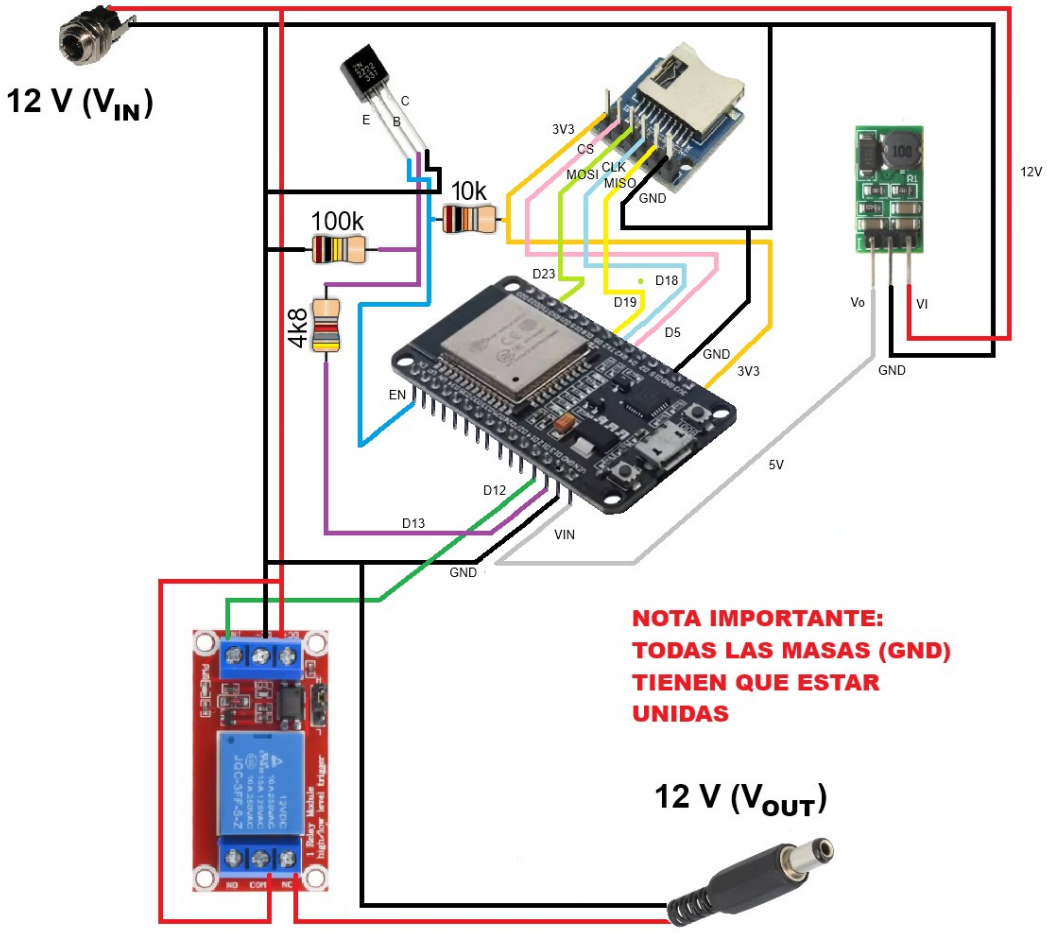
Los datos de la red WiFi se almacenan en la tarjeta MiniSD lo que permite configurar los parámetros de conexión sin cambiar la programación del Arduino.

Así mismo, se alimenta el Arduino ESP32 a 5 voltios por el pin VIN mediante el Módulo convertidor reductor DD4012SA, evitando así el calentamiento excesivo de la placa Arduino ESP32 (que sufriría térmicamente si se alimentara a 12 voltios directamente desde la fuente de alimentación).

El transistor 2N2222 actúa como interruptor, al entrar en saturación, poniendo a tierra el pin EN, lo que hace reiniciar físicamente el Arduino ESP32. para asegurarnos

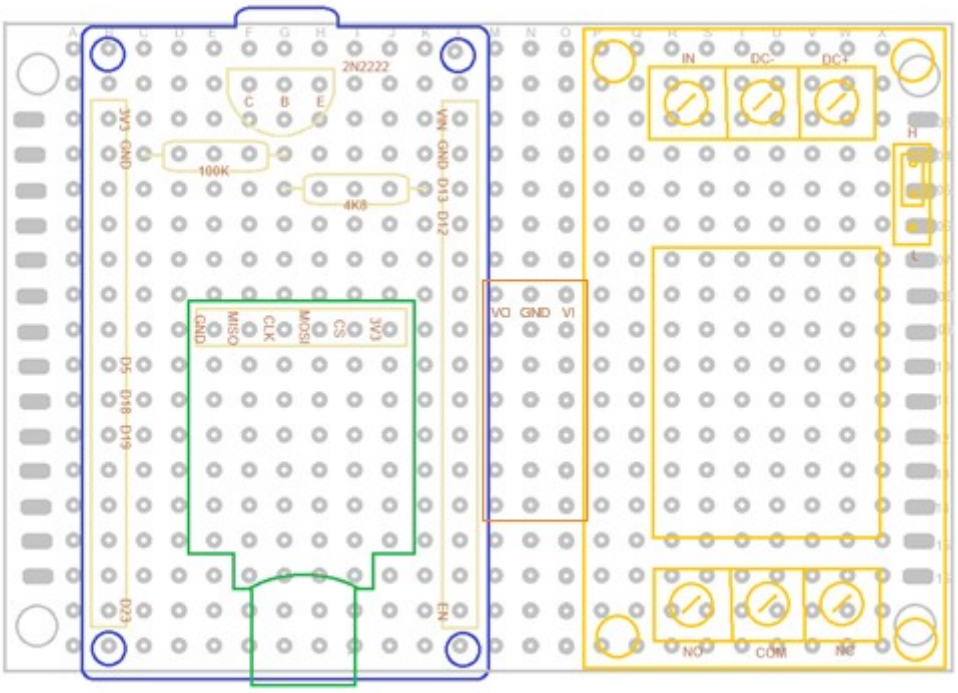
un correcto pulldown[1], la resistencia de 100kΩ nos asegura que la base del transistor permanece a potencial cero (GND) hasta que recibe señal del pin 13 a través de la resistencia 4.7kΩ que se ocupa de gestionar la saturación del transistor. También se conecta el pin EN al pin 3V3 mediante una resistencia de 10 kΩ para estabilizar la tensión del pin EN y hacer el sistema más robusto.

El conexionado es el siguiente:

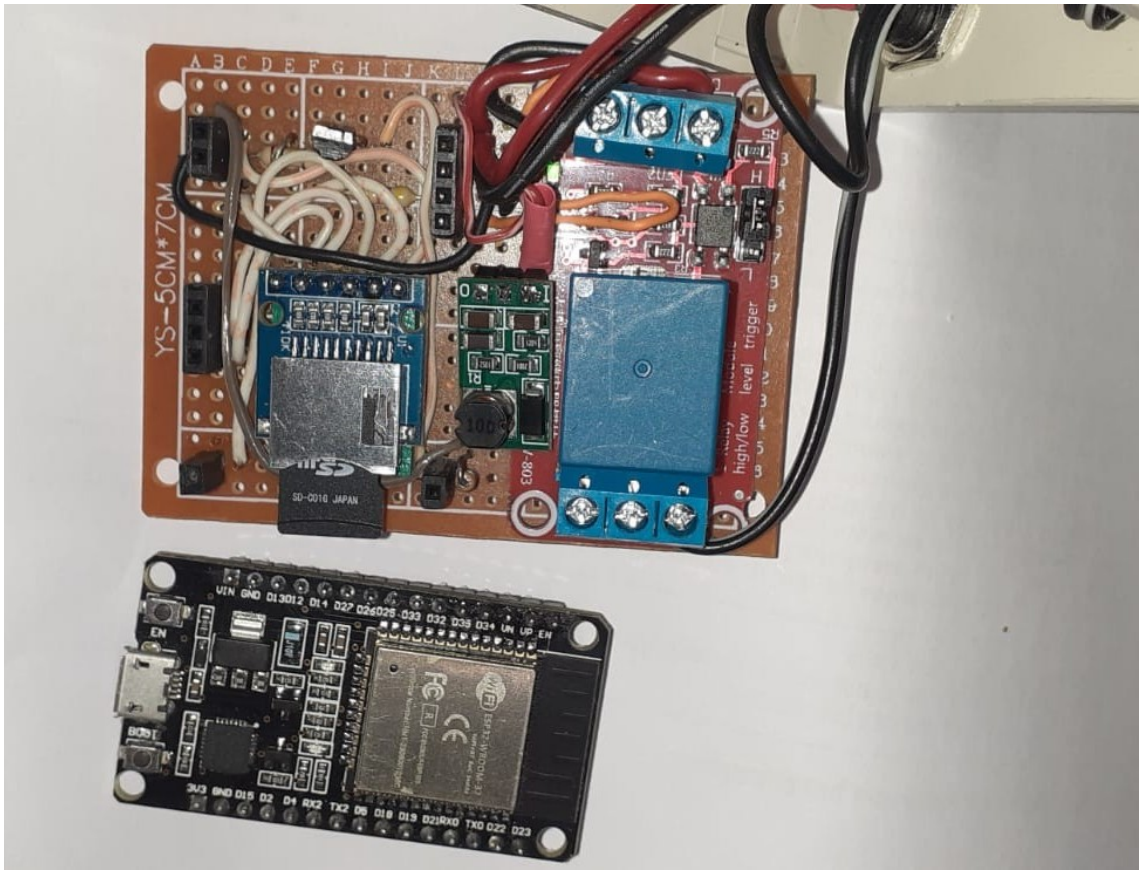


[1] En electrónica, un **pulldown** es una configuración de circuito que usa una resistencia para conectar un pin de entrada digital a tierra (GND), asegurando que su estado predeterminado sea un **cero lógico (LOW)** (0V) cuando no hay una señal activa, evitando que la entrada "flote" y capte ruido. Cuando se presiona un botón o se activa un sensor, la entrada recibe un voltaje (HIGH), cambiando a un **uno lógico (HIGH)** (por ejemplo, 5V), lo que genera un estado lógico estable y predecible en el reposo.

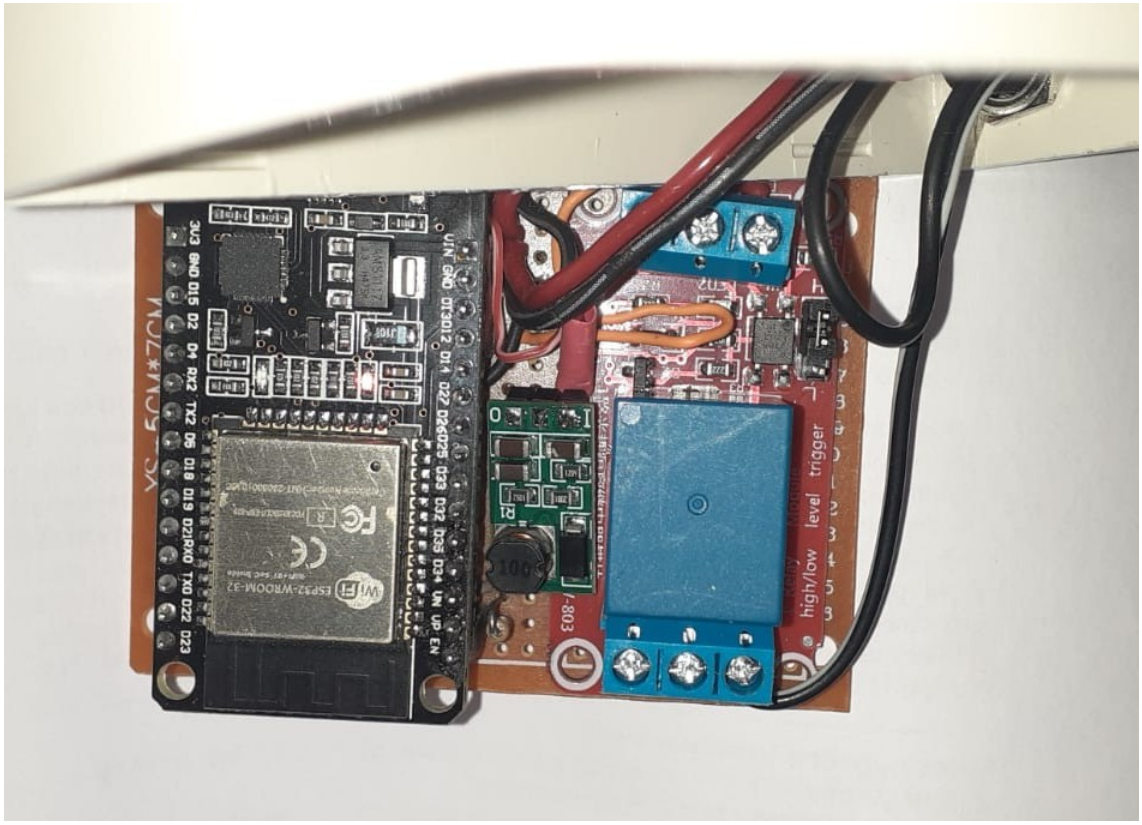
Y la ubicación dentro de la placa es la siguiente:



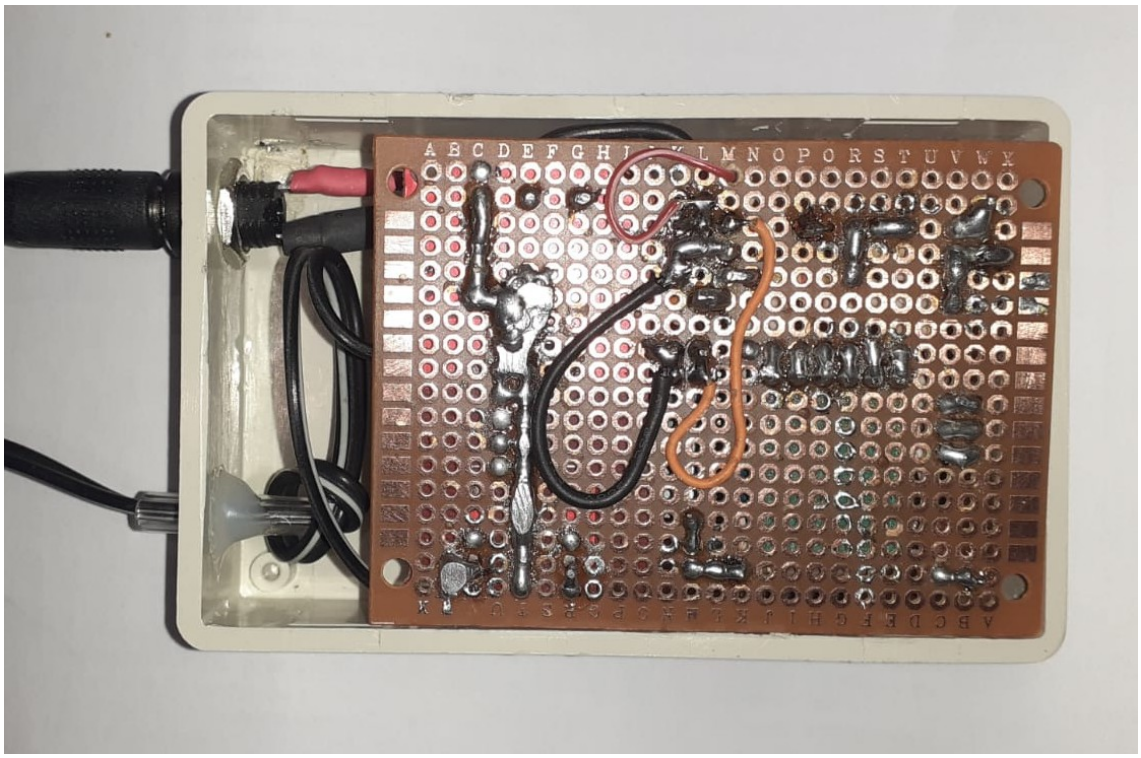
Fotografías del montaje:



En la siguiente fotografía se muestra el montaje del Arduino ESP32 sobre los zócalos correspondientes y la inserción de la tarjeta MiniSD en su lugar.



En la parte trasera de la placa de montaje se hacen las soldaduras.



Todo esto se mete en una caja para proteger el circuito y se conecta entre el alimentador del Router u el propio Router mediante los conectores macho y hembra.

SOFTWARE

Se describe a continuación el código necesario para el funcionamiento del Arduino ESP32:

IMPORTANTE: Recuerda sustituir la cadena de texto por una clave segura que tan sólo tu conozcas en la sección CLAVE SECRETA PARA HMAC del siguiente programa.

SKETCH para ANDUINO ESP32:

```
// ********************************************************
// * REINICIO REMOTO DE ROUTER MEDIANTE APLICACION WEB HTTP *
// * por EB2CTA                                            *
// ********************************************************
//
// CONEXIONADO DEL MODULO PARA TARJETA MICROSD
// ===========================================
      // TARJETA MICROSD     ESP32
      // -------------------------
      //      V3              3V3
      //      CS              D5
      //      MOSI            D23
      //      CLK             D18
      //      MISO            D19
      //      GND             GND
//
// RESET FISICO CON TRANSISTOR 2N2222
// ==================================
      // colocar un puente entre el pin de reinicio y otro pin de E/S y luego puse a nivel bajo
el otro pin de E/S,
      // eso es lo mismo que usar un botón para conectar a tierra el pin de reinicio. Utiliza un
transistor 2N2222 como interruptor
      // Genera pulso: HIGH en GPIO -> transistor conduce -> EN a LOW -> reset
      //
      //                                Pin EN
      //                                   |
      //                                   |
      //                                  /C
      //                      4.7kΩ      /
      //      Pin 13 ---------OOO------B<  2N2222 (NPN)  Cálculo resistencia Base = (3.3V -
0.7V)/Ib; Ib = 0.5 mA para saturación
      //                              |    \
      //                              O     >E
      //                     100kΩ    O     |              100 kΩ (pulldown base, opcional)
      //                              O     |
      //                              |_____|
      //                                 |
      //                              Pin GND
      //
```

```
//
// CONEXIONADO DEL RELE
// ====================
      // TARJETA RELE         ESP32
      // -------------------------------
      //      DC+             12V in
      //      DC-             GND
      //      IN              D12
      //      NO              ---
      //      COM             12V
      //      NC              a 12 V out
      // (puente del relé en H: Señal IN en HIGH activa el relé)
//
// ARCHIVO DE CONFIGURACION A GRABAR EN LA TARJETA MICROSD FORMATEADA COMO FAT
// ==========================================================================
  // config.txt
  //
    // # CONFIGURACIÓN WIFI
    // ssid=MiRedWiFi
    // password=MiClaveWiFi
    //
    // # IP ESTÁTICA
    // local_IP=192.168.1.50
    // gateway=192.168.1.1
    // subnet=255.255.255.0
    // primaryDNS=8.8.8.8
    // secondaryDNS=8.8.4.4
    //
    // # SERVIDOR WEB (80, 81, 8080, 8888, etc)
    // webPort=81
    //
    // # AUTENTICACIÓN WEB
    // authUser=admin
    // authPass=1234

// ------------------------------
// LIBRERIAS
// ------------------------------

#include <WiFi.h>           // Librería que gestiona el WiFi
#include <WebServer.h>      // Librería que gestiona el servidor Web
#include <SPI.h>            // Librería que gestiona el protocolo de comunicación serial
síncrono SPI (Serial Peripheral Interface)
#include <SD.h>             // Librería que gestiona la lectura y escritura de una tarjeta SD
#include "mbedtls/md.h"     // Gestiona los códigos de autenticación de mensajes (HMAC)
#include <esp_task_wdt.h>   // Librería de watchdog

// ------------------------------
// DEFINICIONES WATCHDOG
// ------------------------------
#define WDT_TIMEOUT 420 // Tiempo de espera en segundos (7 minutos)
esp_err_t ESP32_ERROR;  // Almacena si ha habido un error en la variable ESP32_ERROR. Si no hay
error almacena ESP_OK

// ------------------------------
// PINES
// ------------------------------
#define RELE_PIN 12         // Se define el pin que activará el relé
#define RESET_PIN 13        // Se define el pin que activará la base del transistor 2N2222 para
el reinicio físico del ESP32
#define SD_CS 5             // Se define el pin 5 que conecta el pin CS del lector de la tarjeta
MicroDS

#define RESET_PULSE_MS 150  // Milisegundos que dura el pulso de reinicio sobre el pin 13 que
pone a GND el pin EN
#define WAIT_MS 1000        // Milisegundos de espera para el reinicio

// ================ REINICIO CICLICO =================
unsigned long RESTART_INTERVAL_MS = 5UL * 60UL * 1000UL; // Cada 5 minutos se reinicia
fisicamente la placa ESP32
unsigned long lastRestartTime = 0;

// ------------------------------
// VARIABLES CONFIGURABLES (DESDE SD)
// ------------------------------
String ssid;
String password;

IPAddress local_IP;
IPAddress gateway;
IPAddress subnet;
IPAddress primaryDNS;
IPAddress secondaryDNS;
```

```
String authUser;
String authPass;

uint16_t webPort = 8080;   // Puerto configurable desde SD

// ----------------------------
// CLAVE SECRETA PARA HMAC
// ----------------------------
const char* SECRET_KEY =
"EscribeAquiTuClaveSuperSecreta";

// ----------------------------
// SERVIDOR WEB (DINÁMICO)
// ----------------------------
WebServer* server = nullptr;
bool releActive = false;

// ----------------------------
// COOLDOWN (TIEMPO DE ESPERA PARA DAR TIEMPO AL REINICIO DEL ROUTER)
// ----------------------------
unsigned long cooldownDuration = 90000;   // Milisegundos de espera de la barra cuenta atrás
para dar tiempo al reinicio completo del Router
bool cooldownActive = false;
unsigned long cooldownStart = 0;

// =======================================================
// FUNCIÓN: CARGAR CONFIGURACIÓN DESDE microSD
// =======================================================
bool loadConfigFromSD() {                // Comprueba si se puede iniciar la microSD
  if (!SD.begin(SD_CS)) {
    Serial.println("❌Error inicializando microSD");
    return false;
  }

  File file = SD.open("/config.txt");     // Intenta abrir el archivo config.txt
  if (!file) {                            // Comprueba si puede abrir el archivo
    Serial.println("❌No se pudo abrir config.txt");
    return false;
  }

  while (file.available()) {             // Salta las lineas que empiecen por # o estén vacias
    String line = file.readStringUntil('\n');
    line.trim();
    if (line.startsWith("#") || line.length() == 0) continue;

    int sep = line.indexOf('=');
    if (sep < 0) continue;

    String key = line.substring(0, sep);
    String value = line.substring(sep + 1);

    if (key == "ssid") ssid = value;      // Va leyendo líneas del archivo config.txt y va
asignado ,os valores a sus respectivas variables
    else if (key == "password") password = value;
    else if (key == "authUser") authUser = value;
    else if (key == "authPass") authPass = value;
    else if (key == "local_IP") local_IP.fromString(value);
    else if (key == "gateway") gateway.fromString(value);
    else if (key == "subnet") subnet.fromString(value);
    else if (key == "primaryDNS") primaryDNS.fromString(value);
    else if (key == "secondaryDNS") secondaryDNS.fromString(value);
    else if (key == "webPort") webPort = value.toInt();
  }

  file.close();                           // Cierra el archivo
  Serial.println("☑Configuración cargada desde microSD");
  return true;
}

// =======================================================
// HMAC SHA256
// =======================================================
String hmacSHA256(const String& message) {  // Define una función llamada hmacSHA256, recibe un
mensaje (message) como String y devuelve un String (el HMAC en hexadecimal)
  unsigned char output[32];           // Crea un arreglo de 32 bytes, SHA-256 siempre produce
256 bits = 32 bytes y aquí se guardará el resultado binario del HMAC
  const mbedtls_md_info_t* md_info = mbedtls_md_info_from_type(MBEDTLS_MD_SHA256);  // Obtiene
la información del algoritmo SHA-256, mbedtls es la librería criptográfica (muy usada en
ESP32/ESP8266) y md_info describe qué algoritmo se va a usar
  mbedtls_md_hmac(md_info,            // Inicia el cálculo del HMAC
    (const unsigned char*)SECRET_KEY, strlen(SECRET_KEY),     // Convierte SECRET_KEY a bytes y
strlen(SECRET_KEY) indica la longitud de la clave
```

```
    (const unsigned char*)message.c_str(), message.length(),  // Convierte el String message a
un arreglo de bytes, c_str() obtiene el puntero al texto y strlen(SECRET_KEY) indica la longitud
de la clave
    output);                          // Guarda el resultado final del HMAC en el arreglo output

  String hex = "";                    // Vacia la variable string hex
  for (int i = 0; i < 32; i++) {      // Recorre los 32 bytes del HMAC
    if (output[i] < 16) hex += "0";   // Si el byte es menor que 0x10, añade un 0 para mantener
siempre 2 caracteres hexadecimales
    hex += String(output[i], HEX);    // Convierte el byte a texto hexadecimal y lo agrega a la
cadena hex
  }
  Serial.print("🔑 HMAC: ");
  Serial.println(hex);                // Muestra en el monitor Serie la HMAC en hexadecimal
  return hex;                         // Devuelve el HMAC completo como texto hexadecimal
}

// =======================================================
// TOKENS
// =======================================================
String generarToken() {                         // Define una función llamada generarToken que
devuelve un String
  unsigned long expiry = millis() + 60000;      // Calcula el milisegundo que expirará el token
(60 segundos) y lo llamo expiry
  String payload = String(expiry);              // Se convierte a texto (no es preciso quesea
secreto)
  Serial.print("🔑 Token: ");
  Serial.print(payload);                        // Muestra en el Monitor Serie el valor que
caduca el Token
  Serial.println(":HMAC");                      // Muestra en el Monitor Serie : seguid de la
HMAC
  return payload + ":" + hmacSHA256(payload);   // Devuelve un mensaje con el valor de cuando
caduca el token y su firma separado por :
}

bool validarToken() {                           // Define una función para validar el Token
  String token = server->arg("token");          // Extrae el parámetro token de la URL o del
cuerpo HTTP
  int sep = token.indexOf(':');                 // El token tiene este formato:
<expiración>:<firma>
  if (sep < 0) { server->send(401,"text/plain","Token inválido"); return false; } // Si no hay
:, el token no es válido, se envia error 401 y la función termina

  String expiryStr = token.substring(0, sep);   // Tiempo de expiración (en milisegundos)
  String signature = token.substring(sep+1);    // Firma HMAC del tiempo
  unsigned long expiry = expiryStr.toInt();     // Convierte el texto a número para poder
compararlo con millis()

  if (millis() > expiry) { server->send(401,"text/plain","Token expirado"); return false; } //
Si el tiempo actual es mayor que expiry entonces el token caducó
  if (hmacSHA256(expiryStr) != signature) {     // Genera una firma usando una clave secreta y
se compara con la firma recibida; Si no coinciden, el token fue manipulado
    server->send(401,"text/plain","Firma inválida");  // el token no es válido, se envia error
401
    Serial.println("🔑 Token no válido");
    return false;                                // y la función termina
  }
  Serial.println("🔑 Token válido");
  return true;                                   // Si pasó todas las validaciones, el token es
válido y la petición puede continuar
}

// =======================================================
// AUTENTICACION BASICA (BASIC AUTH) MEDIANTE USUARIO Y CONTRASEÑA
// =======================================================
bool isAuthenticated() {                // Crea la funcion isAuthenticated que devolverá false o
true
  if (!server->authenticate(authUser.c_str(), authPass.c_str())) {  // Comprueba si el Usuario y
la Contraseña son correctas y coinckiden con authUser y con authPass
    server->requestAuthentication();    // El navegador muestra el clásico popup: USUARIO y
CONTRASEÑA
    Serial.println("🔑 Usuario no autenticado");                      // Muestra en el Monitor
Serie el valor que caduca el Token
    return false;                       // Devuelve false si Usuario no autenticado
  }
  Serial.println("🔑 Usuario autenticado");
  return true;                          // Devuelve true si Usuario autenticado
}

// =======================================================
// WEB PRINCIPAL
// =======================================================
void handleRoot() {                     // Función que muestra la página Web
```

```
  if (!isAuthenticated()) return;       // Si no se ha autenticado el Usuario con su Contraseña
se sale de la función y no muestra la Web

  String html = R"====(
<html>
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>Reinicio del Router</title>
<style>
body { font-family: Arial, sans-serif; background:#f0f0f0; margin:0; padding:0; text-
align:center; padding-bottom:50px; }
.container { width:90%; max-width:600px; margin:30px auto; padding:25px; background:white; box-
shadow:0 0 12px rgba(0,0,0,0.2); border-radius:12px; }
button { font-size:20px; padding:12px 20px; width:80%; max-width:300px; border-radius:10px;
border:none; background:#2196F3; color:white; cursor:pointer; margin-bottom:25px; }
button:disabled { background:#888; }
#countdownBar { width:100%; max-width:350px; height:14px; border:1px solid #333; border-
radius:7px; margin:0 auto 25px auto; overflow:hidden; background:#ddd; }
#countdownFill { height:100%; width:0%; background:#2196F3; transition: width 0.5s linear; }
h2 { margin-top:25px; margin-bottom:5px; }
.status-circle { width:18px; height:18px; border-radius:50%; margin:8px auto 0 auto; border:2px
solid #555; }
.footer { position: fixed; bottom: 0; left: 0; width: 100%; background: #ffffff; color: #555;
font-size:14px; padding:8px 0; border-top:1px solid #ccc; }
#alertMsg { position: fixed; top:10px; left:50%; transform:translateX(-50%); background:#ff9800;
color:white; padding:10px 20px; border-radius:8px; display:none; box-shadow:0 0 10px
rgba(0,0,0,0.3); z-index:1000; }
#alertMsg.online { background:#4CAF50; }
#alertMsg.offline { background:#f44336; }
</style>
<script>
let ledCountdown = 0;
let ledCounter = 0;
let cooldownCountdown = 0;
let cooldownCounter = 0;
const ledDuration = 10;
const cooldownDuration = 90;
let reinicioActivo = false;

async function actualizarInternet() {
  try {
    const res = await fetch("/status");
    const status = await res.text();
    const statusEl = document.getElementById("wifiStatus");
    const ledWifi = document.getElementById("ledWifi");

    if(!reinicioActivo){
      if(status.trim() === "online") {
        statusEl.innerHTML = "Conectado a Internet";
        ledWifi.style.background = "green";
      } else {
        statusEl.innerHTML = "Desconectado a Internet";
        ledWifi.style.background = "red";
      }
    }
  } catch(e) {
    if(!reinicioActivo){
      const statusEl = document.getElementById("wifiStatus");
      const ledWifi = document.getElementById("ledWifi");
      statusEl.innerHTML = "Desconectado a Internet";
      ledWifi.style.background = "red";
    }
  }
}

function mostrarAlerta(mensaje, tipo){
  const el = document.getElementById("alertMsg");
  el.innerHTML = mensaje;
  el.className = tipo;
  el.style.display = "block";
  setTimeout(()=>{ el.style.display="none"; }, 3000);
}

async function reiniciar() {
  if(cooldownCountdown > 0) return;

  reinicioActivo = true;

  ledCountdown = ledDuration;
  ledCounter = 0;
  document.getElementById("estado").innerHTML = "ROUTER desconectado";
  document.getElementById("led").style.background = "red";
```

```
  const statusEl = document.getElementById("wifiStatus");
  const ledWifi = document.getElementById("ledWifi");
  statusEl.innerHTML = "Desconectado a Internet";
  ledWifi.style.background = "red";

  document.getElementById("btnReiniciar").disabled = true;

  try {
    const tokenRes = await fetch("/token");
    const token = await tokenRes.text();
    await fetch("/encender?token=" + token, { method:"POST" });
  } catch(e){
    mostrarAlerta("ESP32 no disponible, simulando localmente", "offline");
  }

  setTimeout(() => {
    reinicioActivo = false;
  }, ledDuration*1000);

  cooldownCountdown = cooldownDuration;
  cooldownCounter = 0;
}

function actualizarSimulacion() {
  if(ledCountdown > 0){
    ledCounter++;
    if(ledCounter >= ledDuration){
      ledCountdown = 0;
      ledCounter = 0;
      document.getElementById("estado").innerHTML = "ROUTER conectado";
      document.getElementById("led").style.background = "green";
    }
  }

  if(cooldownCountdown > 0){
    cooldownCounter++;
    document.getElementById("countdownFill").style.width =
((cooldownCounter/cooldownDuration)*100)+"%";
    if(cooldownCounter >= cooldownDuration){
      cooldownCountdown = 0;
      cooldownCounter = 0;
      document.getElementById("countdownFill").style.width = "0%";
      document.getElementById("btnReiniciar").disabled = false;
      setTimeout(() => {
        location.reload();
      }, 500);
    }
  }
}

setInterval(actualizarSimulacion, 1000);
setInterval(actualizarInternet, 1000);
</script>
</head>
<body>
<div id="alertMsg"></div>
<div class="container">
<h1>Reinicio del Router</h1>
<button id="btnReiniciar" onclick="reiniciar()">Reiniciar</button>
<div id="countdownBar"><div id="countdownFill"></div></div>

<h2>Alimentación ROUTER:</h2>
<div id="estado">ROUTER conectado</div>
<div id="led" class="status-circle" style="background:green;"></div>

<h2>Estado Internet:</h2>
<div id="wifiStatus">Conectado a Internet</div>
<div id="ledWifi" class="status-circle" style="background:green;"></div>
</div>
<div class="footer">Powered by Lino, EB2CTA</div>
</body>
</html>
)====";

  server->send(200,"text/html",html);
}

// ======================================================
// ENDPOINTS (INTERACCION DE DATOS CON LA WEB)
// ======================================================
void handleToken() {
  if (!isAuthenticated()) {                          // Comprueba que el Usuario esté
autenticado
```

```
    server->send(401, "text/plain", "No autorizado"); // Si no está autenticado envia un mensaje
"No autorizado"
    return;                                          // Sale y no continua
  }
  server->send(200,"text/plain",generarToken());      // Genera un Token y lo envia como
respuesta HTTP
}

void handleEncender() {
  if (!isAuthenticated()) return; // Verifica que el Usuario esté autenticado; si no, hace
return para salir
  if (!validarToken()) return;    // Verifica que el Token de seguridad sea válido; si no, hace
return para salir
  if (releActive) return;         // Indica si el relé ya está activado; si está activado hace
return para salir y evita que se active el relé dos veces al mismo tiempo
  server->send(200,"text/plain","ROUTER desconectado"); // Envía código 200 OK, tipo texto plano
y con el mensaje "ROUTER desconectado"

  xTaskCreate([](void*) {  // Activación del relé que desconecta la alimentación del Router,
crea una tarea independiente usando una función lambda lo que hace que no se bloquee el loop() y
ejecutar el temporizador sin usar delay()
    Serial.println("🔌 Alimentación desconectada");
    releActive = true;
    digitalWrite(RELE_PIN, HIGH);             // Se activa el relé
    vTaskDelay(10000 / portTICK_PERIOD_MS);   // Se pausa la tarea 10 segundos (Tiempo que el
relé esta activado) sin bloquear el microprocesador y siguen funcionando el resto de tareas
    digitalWrite(RELE_PIN, LOW);              // Se desactiva el relé
    releActive = false;
    Serial.println("🔌 Alimentación conectada");
    vTaskDelete(NULL);          // Destruye la tarea y libera memoria
  }, "Relay", 4096, NULL, 1, NULL);   // Nombre e la tarea: "Relay"; Tamaño de la pila (stack):
4096; Parámetros de entrada: NULL; Prioridad: 1; No guarda el handle: NULL

  cooldownActive = true;        // Inicia un tiempo de espera
  cooldownStart = millis();     // Guarda en cooldownStart el momento de inicio
}

void handleStatus() {           // Envía una respuesta HTTP al cliente, indicando si el
dispositivo está conectado a la red WiFi
  server->send(200,"text/plain",
    (WiFi.status() == WL_CONNECTED) ? "online" : "offline");
}

// =======================================================
// SETUP
// =======================================================
void setup() {
  Serial.begin(115200);           // Inicia la comunicación serie del Monitor Serie del IDE

  // Configuración Watchdog
    delay(100); // Pequeña pausa para asegurarse de que el sistema ya arrancó correctamente
antes de tocar el WDT.
    Serial.println("⚙️ Configurando Watchdog");
    Serial.print("🔌 Watchdog establecido en: ");
    Serial.print(WDT_TIMEOUT);
    Serial.println(" en segundos");
    esp_task_wdt_deinit();  // Desactiva cualquier configuración previa de tareas watchdog
    // Configuración del vigilante de tareas
      esp_task_wdt_config_t wdt_config = {                // Se crea una estructura con los
parámetros del WDT
        .timeout_ms = WDT_TIMEOUT * 1000,                 // Convierte a milisegundos
        .idle_core_mask = (1 << portNUM_PROCESSORS) - 1,  // Vigila todos los núcleos del
procesador (generalmente tiene 2)
        .trigger_panic = true                             // Habilita el reinicio del ESP32
      };
    // Inicializa el WDT
      ESP32_ERROR = esp_task_wdt_init(&wdt_config);       // Inicializa el WDT con esa
configuración y guards resultado en ESP32_ERROR (OK o ERROR)
      Serial.println("🔌 Último reinicio del Watchdog: " + String(esp_err_to_name(ESP32_ERROR)));
// Muestra el estado: OK, ESP_ERR_INVALID_STATE, etc.
      esp_task_wdt_add(NULL);  // Agregar el hilo actual al WDT, es decir, le dices que vigile
esta tarea
    // Fin Configuración Watchdog


  pinMode(RELE_PIN, OUTPUT);      // Define ese pin como pin de salida
  pinMode(RESET_PIN, OUTPUT);     // Define ese pin como pin de salida
  digitalWrite(RELE_PIN, LOW);    // Envia a ese pin 0V
  digitalWrite(RESET_PIN, LOW);   // Envia a ese pin 0V

  if (!loadConfigFromSD()) {
    Serial.println("⚠️ ERROR SD - Reinicio bloqueado");
    while (true);
  }
```

```
  server = new WebServer(webPort);

  delay(WAIT_MS);                                  // Espera forzada

  WiFi.config(local_IP, gateway, subnet, primaryDNS, secondaryDNS);  // Configura una IP
estática segun los parametros leidos de la MicroSD
  delay(WAIT_MS);                                  // Espera forzada
  WiFi.begin(ssid.c_str(), password.c_str());      // Se conecta a la red Wifi SSID con su
correpondiente contraseña
  delay(WAIT_MS);                                  // Espera forzada
  Serial.print("🔄 Conectando WiFi");
  while (WiFi.status() != WL_CONNECTED) {          // Mientras no haya conexión WiFi va mostrando
puntos en el Monitor Serie
    delay(500);                                    // Cada 500 miisegundos
    Serial.print(".");
  }

  Serial.println("\n☑WiFi conectado");           // Muestra en el monitor Serie que la placa
ESP32 e haa conectado al WiFi
  Serial.print("📡 Dirección MAC: ");
  Serial.println(WiFi.macAddress());               // Muestra en el monitor Serie la MAC de la
placa ESP32
  Serial.print("🌐 Dirección IP: ");
  Serial.println(WiFi.localIP());                  // Muestra en el monitor Serie la IP de la
placa ESP32

  server->on("/", handleRoot);                     // Cuando se solicita la pagina principal en
la IP:port ejecuta esa función (que es la WEB)
  server->on("/token", handleToken);               // Se gestiona la función del Token
  server->on("/encender", handleEncender);         // Se gestiona la función para activar el relé
verificando el Token
  server->on("/status", handleStatus);             // Se gestiona la función para ver si hay ya
internet despues del reinicio

  server->begin();                                 // Inicia el servidor Web y espera conexones
de clientes
  Serial.print("🚀 Puerto del Servidor Web: ");
  Serial.println(webPort);                         // Muestra en el Monitor Serie el puerto
seleccionado

  Serial.println("☑Servidor Web iniciado");     // Muestra aviso en Monitor Serie de que el
servidor Web está levandado
  Serial.print("🧠 Memoria RAM disponible: ");
  Serial.print(ESP.getFreeHeap());                 // Muestran la memoria disponible en el ESP32
  Serial.println(" bytes");
}

// =======================================================
// LOOP
// =======================================================
void loop() {

    // Resetea el contador Watchdog
        esp_task_wdt_reset(); // Resetea el WDT
        delay(1); // Importante para que el reset se aplique
    // Fin Resetea el contador Whatchdog

  if (server) server->handleClient();              // Comprueba y maneja peticiones HTTP

  if (WiFi.status() != WL_CONNECTED) {             // Si el ESP32 no está conectado a WiFi
    WiFi.disconnect();                             // Termina cualquier conexion existente a WiFi
    delay(WAIT_MS * 2UL);                          // Espera forzada
    WiFi.begin(ssid.c_str(), password.c_str());    // Se conecta a la red Wifi SSID con su
correpondiente contraseña
  }

  if (cooldownActive && millis() - cooldownStart >= cooldownDuration) {
    cooldownActive = false;

    if (server) {       // Cerrar los sockets TCP y libera recursos asociados antes de reiniciar
el ESP32
      server->stop();
      delete server;
     server = nullptr;  // Desapunta el puntero (hace que el puntro no apunte a ningun sitio)
    }

    Serial.println("🔄 Reinicio de la placa ESP32");                      // Llama a la función
de reseteo por software
    delay(WAIT_MS * 2UL);             // Espera WAIT_MS milisegundos para reiniciar
    digitalWrite(RESET_PIN, HIGH);    // Excita el transistor 2N2222
    delay(RESET_PULSE_MS);            // Tiempo que mantiene EN a GND
    digitalWrite(RESET_PIN, LOW);     // Libera EN
  }
```

```
  // ================ REINICIO PERIODICO ================
  if (millis() - lastRestartTime >= RESTART_INTERVAL_MS) {    //Comprueba si han pasado
RESTART_INTERVAL_MS para reiniciar periodicamente el ESP32

    lastRestartTime = millis();                            // Pone el contador otra vez a cero
para esperar otro reinicio

    if (server) {       // Cerrar los sockets TCP y libera recursos asociados antes de reiniciar
el ESP32
      server->stop();
      delete server;
     server = nullptr;  // Desapunta el puntero (hace que el puntro no apunte a ningun sitio)
    }
    server = new WebServer(webPort);
    Serial.println("🔄 Reinicio de la placa ESP32");
    delay(WAIT_MS);             // Espera forzada
    digitalWrite(RESET_PIN, HIGH);    // Excita el transistor 2N2222
    delay(RESET_PULSE_MS);            // Tiempo que mantiene EN a GND
    digitalWrite(RESET_PIN, LOW);     // Libera EN
  }
}
```

ARCHIVO DE CONFIGURACION para la tarjeta MiniSD:

En la tarjeta MiniSD debe escribirse en el directorio raíz del archivo `config.txt` que contiene la configuración relativa a la red en la que va a funcionar el dispositivo:

IMPORTANTE: La tarjeta MiniSD debe estar formateada como FAT (File Allocation Table) y recuerda sustituir las cadenas de texto de los parámetros siguientes por los que se adecuen a los datos de tu red.

```
# CONFIGURACIÓN WIFI
ssid=MiRedWiFi
password=MiClaveWiFi

# IP ESTÁTICA
local_IP=192.168.1.50
gateway=192.168.1.1
subnet=255.255.255.0
primaryDNS=8.8.8.8
secondaryDNS=8.8.4.4

# SERVIDOR WEB (80, 81, 8080, 8888, etc)
webPort=81

# AUTENTICACIÓN WEB
authUser=admin
authPass=1234
```

EXPLICACIÓN DEL CÓDIGO

Se cargan inicialmente las librerías que se van a utilizar:

```
// ------------------------------
// LIBRERIAS
// ------------------------------
#include <WiFi.h>           // Librería que gestiona el WiFi
#include <WebServer.h>      // Librería que gestiona el servidor Web
#include <SPI.h>            // Librería que gestiona el protocolo de comunicación serial
síncrono SPI (Serial Peripheral Interface)
#include <SD.h>             // Librería que gestiona la lectura y escritura de una tarjeta SD
#include "mbedtls/md.h"     // Gestiona los códigos de autenticación de mensajes (HMAC)
#include <esp_task_wdt.h>   // Librería de watchdog
```

Definimos el perro guardián (Watchdog) cuya función es reiniciar la placa ESP32 si se queda colgada y no responde. Se ha decidido que si la placa no responde en

7 minutos, es decir, que no se ejecuta la instrucción `esp_task_wdt_reset()` en ese tiempo prefijado, la placa se reinicie automáticamente.

```
// ----------------------------
// DEFINICIONES WATCHDOG
// ----------------------------
#define WDT_TIMEOUT 420 // Tiempo de espera en segundos (7 minutos)
esp_err_t ESP32_ERROR;  // Almacena si ha habido un error en la variable ESP32_ERROR. Si no hay
error almacena ESP_OK
```

Se definen los pines del ESP32 definiendo que el pin 12 servirá para activar el relé que desconecta la alimentación y el pin 13 que sirve para energizar la base del transistor 2N2222 para que entre en saturación y conduzca entre el colector y emisor del transistor y ponga el pin EN a GND para reiniciar físicamente la placa ESP32. Así mismo, se define que el pin CS del lector de la tarjeta MicroSD será comandado por el pin 5. También se define que la duración de tiempo del pulso que energiza la base del transistor será de 159 milisegundos y se define una variable `WAIT_MS` en 2000 milisegundos (2 segundos) como tiempo de espera estándar:

```
// ----------------------------
// PINES
// ----------------------------
#define RELE_PIN 12         // Se define el pin que activará el relé
#define RESET_PIN 13        // Se define el pin que activará la base del transistor 2N2222 para
el reinicio físico del ESP32
#define SD_CS 5             // Se define el pin 5 que conecta el pin CS del lector de la tarjeta
MicroSD

#define RESET_PULSE_MS 150  // Milisegundos que dura el pulso de reinicio sobre el pin 13 que
pone a GND el pin EN
#define WAIT_MS 1000        // Milisegundos de espera para el reinicio
```

En las líneas siguientes se define la variable `RESTART_INTERVAL_MS` que indica que se reinicializará la placa ESP32 repetidamente cada 5 minutos e inicia el contador `lastRestartTime` a 0:

```
// ================ REINICIO CICLICO =================
unsigned long RESTART_INTERVAL_MS = 5UL * 60UL * 1000UL; // Cada 5 minutos se reinicia
fisicamente la placa ESP32
unsigned long lastRestartTime = 0;
```

A continuación, se definen las variables que se leerán desde la tarjeta MicroSD, como son la SSID y contraseña de la Red local Wifi; la IP que asignamos a la placa ESP32, la puerta de enlace al Router, la subred, las DNS primaria y secundaria y por último el puerto que se usará para redirigir el tráfico de Internet hasta la placa ESP32. La idea de usar este sistema de almacenamiento en la tarjeta MicroSD es para que sea sencillo redefinir estas variables introduciendo la tarjeta MicroSD en un ordenador y no tener que modificar el código grabado en la placa ESP32 y cambiamos de acceso a Internet o de red. La estructura del archivo `config.txt` que contiene estos datos se explica más adelante:

```
// ----------------------------
// VARIABLES CONFIGURABLES (DESDE SD)
// ----------------------------
String ssid;
String password;

IPAddress local_IP;
IPAddress gateway;
IPAddress subnet;
```

```
IPAddress primaryDNS;
IPAddress secondaryDNS;

String authUser;
String authPass;

uint16_t webPort = 8080;   // Puerto configurable desde SD
```

Seguidamente se define la clave HMAC (Hash-based Message Authentication Code) es una clave secreta compartida que se usa junto con una función hash criptográfica (como SHA-256) para crear un código único (firma) para un mensaje, verificando así su integridad (no ha sido alterado) y autenticidad (proviene de una fuente confiable). El remitente calcula el HMAC y lo adjunta al mensaje; el receptor lo recalcula usando la misma clave y, si coinciden, se confirma que el mensaje es legítimo y no ha sido modificado en tránsito. Por seguridad se ha decidido que este clave no se guarde en la tarjeta MicroSD sino que se modifique en el código y que quede segura tras la compilación. Recuerda modificarla y utilizar una clave segura:

```
// ----------------------------
// CLAVE SECRETA PARA HMAC
// ----------------------------
const char* SECRET_KEY =
"EscribeAquiTuClaveSuperSecreta";
```

Ahora se define un puntero llamado `server`, del tipo `WebServer`, y lo inicializa como nulo y se crea una variable booleana `releActive` y le asigna el valor de falso, es decir, que el relé esta desactivado:

```
// ----------------------------
// SERVIDOR WEB (DINÁMICO)
// ----------------------------
WebServer* server = nullptr;
bool releActive = false;
```

Aquí se define el tiempo de espera que tiene que esperar el programa para dar tiempo al router que se conecte ras la pérdida de la alimentación. Se ha estimado en 90 segundos para que el Router se vuelva operativo después de que el relé haya cortado y repuesto la alimentación al mismo. Si el Router es más klento en volver a estar operativo se debe aumentar el valor de la variable `cooldownDuration`. También se inician las variables `cooldownActive` y `cooldownStart`:

```
// ----------------------------
// COOLDOWN (TIEMPO DE ESPERA PARA DAR TIEMPO AL REINICIO DEL ROUTER)
// ----------------------------
unsigned long cooldownDuration = 90000;   // Milisegundos de espera de la barra cuenta atrás
para dar tiempo al reinicio completo del Router
bool cooldownActive = false;
unsigned long cooldownStart = 0;
```

En el siguiente grupo de líneas de código se accede a la tarjeta MicroSD y se leen los datos ahí almacenados sobre la red local:

```
// ======================================================
// FUNCIÓN: CARGAR CONFIGURACIÓN DESDE microSD
// ======================================================
bool loadConfigFromSD() {                 // Comprueba si se puede iniciar la microSD
  if (!SD.begin(SD_CS)) {
    Serial.println("❌Error inicializando microSD");
    return false;
  }

  File file = SD.open("/config.txt");     // Intenta abrir el archivo config.txt
```

```
  if (!file) {                          // Comprueba si puede abrir el archivo
    Serial.println("❌No se pudo abrir config.txt");
    return false;
  }

  while (file.available()) {            // Salta las lineas que empiecen por # o estén vacias
    String line = file.readStringUntil('\n');
    line.trim();
    if (line.startsWith("#") || line.length() == 0) continue;

    int sep = line.indexOf('=');
    if (sep < 0) continue;

    String key = line.substring(0, sep);
    String value = line.substring(sep + 1);

    if (key == "ssid") ssid = value;      // Va leyendo líneas del archivo config.txt y va
asignado ,os valores a sus respectivas variables
    else if (key == "password") password = value;
    else if (key == "authUser") authUser = value;
    else if (key == "authPass") authPass = value;
    else if (key == "local_IP") local_IP.fromString(value);
    else if (key == "gateway") gateway.fromString(value);
    else if (key == "subnet") subnet.fromString(value);
    else if (key == "primaryDNS") primaryDNS.fromString(value);
    else if (key == "secondaryDNS") secondaryDNS.fromString(value);
    else if (key == "webPort") webPort = value.toInt();
  }

  file.close();                         // Cierra el archivo
  Serial.println("☑Configuración cargada desde microSD");
  return true;
}
```

Se gestiona la parte de autenticación mediante HMAC SHA256 utiliando la clave secreta guardada en la variable `SECRET_KEY` y crea la variable `hex` que devuelve el HMAC completo como texto hexadecimal:

```
// ======================================================
// HMAC SHA256
// ======================================================
String hmacSHA256(const String& message) {  // Define una función llamada hmacSHA256, recibe un
mensaje (message) como String y devuelve un String (el HMAC en hexadecimal)
  unsigned char output[32];           // Crea un arreglo de 32 bytes, SHA-256 siempre produce
256 bits = 32 bytes y aquí se guardará el resultado binario del HMAC
  const mbedtls_md_info_t* md_info = mbedtls_md_info_from_type(MBEDTLS_MD_SHA256);  // Obtiene
la información del algoritmo SHA-256, mbedtls es la librería criptográfica (muy usada en
ESP32/ESP8266) y md_info describe qué algoritmo se va a usar
  mbedtls_md_hmac(md_info,            // Inicia el cálculo del HMAC
    (const unsigned char*)SECRET_KEY, strlen(SECRET_KEY),     // Convierte SECRET_KEY a bytes y
strlen(SECRET_KEY) indica la longitud de la clave
    (const unsigned char*)message.c_str(), message.length(),  // Convierte el String message a
un arreglo de bytes, c_str() obtiene el puntero al texto y strlen(SECRET_KEY) indica la longitud
de la clave
    output);                          // Guarda el resultado final del HMAC en el arreglo output

  String hex = "";                    // Vacia la variable string hex
  for (int i = 0; i < 32; i++) {      // Recorre los 32 bytes del HMAC
    if (output[i] < 16) hex += "0";   // Si el byte es menor que 0x10, añade un 0 para mantener
siempre 2 caracteres hexadecimales
    hex += String(output[i], HEX);    // Convierte el byte a texto hexadecimal y lo agrega a la
cadena hex
  }
  Serial.print("🔑 HMAC: ");
  Serial.println(hex);                // Muestra en el monitor Serie la HMAC en hexadecimal
  return hex;                         // Devuelve el HMAC completo como texto hexadecimal
}
```

De la misma manera, se gestiona a continuación el token (que actúa como una “llave”) para la autenticación, y que caduca a los 60 segundos. Devuelve `FALSE` si no se autentica y devuelve `TRUE` y sigue el proceso si se ha autenticado correctamente.

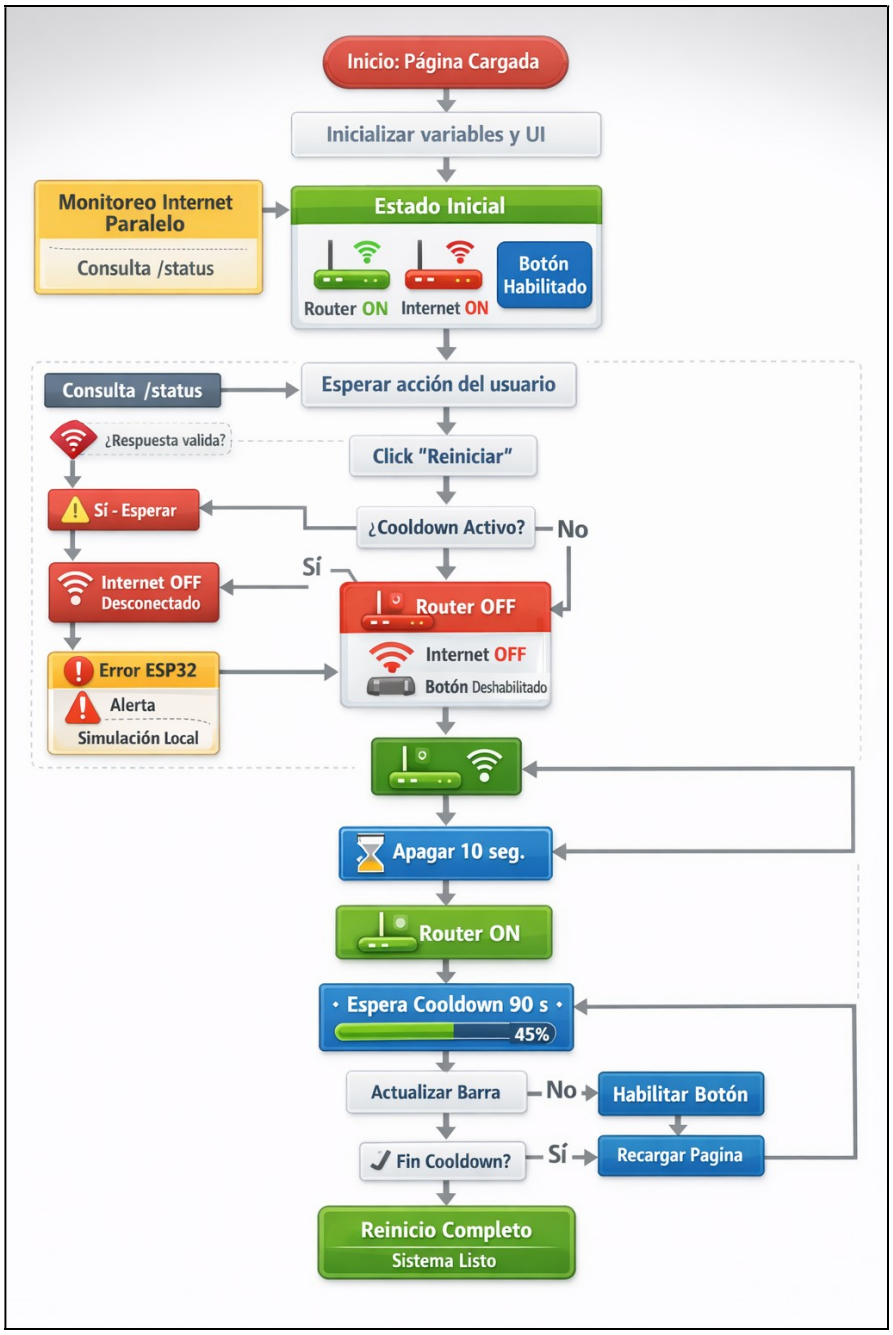
```
// ======================================================
// TOKENS
// ======================================================
String generarToken() {                         // Define una función llamada generarToken que
devuelve un String
```

```
  unsigned long expiry = millis() + 60000;      // Calcula el milisegundo que expirará el token
(60 segundos) y lo llamo expiry
  String payload = String(expiry);              // Se convierte a texto (no es preciso quesea
secreto)
  Serial.print("🔑 Token: ");
  Serial.print(payload);                        // Muestra en el Monitor Serie el valor que
caduca el Token
  Serial.println(":HMAC");                      // Muestra en el Monitor Serie : seguid de la
HMAC
  return payload + ":" + hmacSHA256(payload);   // Devuelve un mensaje con el valor de cuando
caduca el token y su firma separado por :
}

bool validarToken() {                           // Define una función para validar el Token
  String token = server->arg("token");          // Extrae el parámetro token de la URL o del
cuerpo HTTP
  int sep = token.indexOf(':');                 // El token tiene este formato:
<expiración>:<firma>
  if (sep < 0) { server->send(401,"text/plain","Token inválido"); return false; } // Si no hay
:, el token no es válido, se envia error 401 y la función termina

  String expiryStr = token.substring(0, sep);   // Tiempo de expiración (en milisegundos)
  String signature = token.substring(sep+1);    // Firma HMAC del tiempo
  unsigned long expiry = expiryStr.toInt();     // Convierte el texto a número para poder
compararlo con millis()

  if (millis() > expiry) { server->send(401,"text/plain","Token expirado"); return false; } //
Si el tiempo actual es mayor que expiry entonces el token caducó
  if (hmacSHA256(expiryStr) != signature) {     // Genera una firma usando una clave secreta y
se compara con la firma recibida; Si no coinciden, el token fue manipulado
    server->send(401,"text/plain","Firma inválida");  // el token no es válido, se envia error
401
    Serial.println("❌ Token no válido");
    return false;                                // y la función termina
  }
  Serial.println("✅ Token válido");
  return true;                                  // Si pasó todas las validaciones, el token es
válido y la petición puede continuar
}
```

Ahora se gestiona la validación de Usuario y Contraseña generándose una función `isAuthenticated` que devuelve `FALSE` si no es correcto el Usuario y la Contraseña y devuelve `TRUE` y es correcto:

```
// ======================================================
// AUTENTICACION BASICA (BASIC AUTH) MEDIANTE USUARIO Y CONTRASEÑA
// ======================================================
bool isAuthenticated() {                // Crea la funcion isAuthenticated que devolverá false o
true
  if (!server->authenticate(authUser.c_str(), authPass.c_str())) {  // Comprueba si el Usuario y
la Contraseña son correctas y coinckiden con authUser y con authPass
    server->requestAuthentication();    // El navegador muestra el clásico popup: USUARIO y
CONTRASEÑA
    Serial.println("❌ Usuario no autenticado");                      // Muestra en el Monitor
Serie el valor que caduca el Token
    return false;                       // Devuelve false si Usuario no autenticado
  }
  Serial.println("✅ Usuario autenticado");
  return true;                          // Devuelve true si Usuario autenticado
}
```

El siguiente grupo de líneas contiene el código HTML + CSS + JavaScript de la página Web que sirve la placa ESP32 recogida en la función `handleRoot`. Se ha marcado en texto azul el código de la página Web con características “responsive” (diseño Web adaptable). El diagrama de flujo simplificado es el siguiente:



Y el código HTML + CSS + JavaScript embebido está remarcado en color azul:

```
// ======================================================
// WEB PRINCIPAL
// ======================================================
void handleRoot() {                    // Función que muestra la página Web
  if (!isAuthenticated()) return;      // Si no se ha autenticado el Usuario con su Contraseña
se sale de la función y no muestra la Web

  String html = R"====(
<html>
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>Reinicio del Router</title>
<style>
body { font-family: Arial, sans-serif; background:#f0f0f0; margin:0; padding:0; text-
align:center; padding-bottom:50px; }
.container { width:90%; max-width:600px; margin:30px auto; padding:25px; background:white; box-
shadow:0 0 12px rgba(0,0,0,0.2); border-radius:12px; }
button { font-size:20px; padding:12px 20px; width:80%; max-width:300px; border-radius:10px;
border:none; background:#2196F3; color:white; cursor:pointer; margin-bottom:25px; }
button:disabled { background:#888; }
#countdownBar { width:100%; max-width:350px; height:14px; border:1px solid #333; border-
radius:7px; margin:0 auto 25px auto; overflow:hidden; background:#ddd; }
#countdownFill { height:100%; width:0%; background:#2196F3; transition: width 0.5s linear; }
h2 { margin-top:25px; margin-bottom:5px; }
```

```
.status-circle { width:18px; height:18px; border-radius:50%; margin:8px auto 0 auto; border:2px
solid #555; }
.footer { position: fixed; bottom: 0; left: 0; width: 100%; background: #ffffff; color: #555;
font-size:14px; padding:8px 0; border-top:1px solid #ccc; }
#alertMsg { position: fixed; top:10px; left:50%; transform:translateX(-50%); background:#ff9800;
color:white; padding:10px 20px; border-radius:8px; display:none; box-shadow:0 0 10px
rgba(0,0,0,0.3); z-index:1000; }
#alertMsg.online { background:#4CAF50; }
#alertMsg.offline { background:#f44336; }
</style>
<script>
let ledCountdown = 0;
let ledCounter = 0;
let cooldownCountdown = 0;
let cooldownCounter = 0;
const ledDuration = 10;
const cooldownDuration = 90;
let reinicioActivo = false;

async function actualizarInternet() {
  try {
    const res = await fetch("/status");
    const status = await res.text();
    const statusEl = document.getElementById("wifiStatus");
    const ledWifi = document.getElementById("ledWifi");

    if(!reinicioActivo){
      if(status.trim() === "online") {
        statusEl.innerHTML = "Conectado a Internet";
        ledWifi.style.background = "green";
      } else {
        statusEl.innerHTML = "Desconectado a Internet";
        ledWifi.style.background = "red";
      }
    }
  } catch(e) {
    if(!reinicioActivo){
      const statusEl = document.getElementById("wifiStatus");
      const ledWifi = document.getElementById("ledWifi");
      statusEl.innerHTML = "Desconectado a Internet";
      ledWifi.style.background = "red";
    }
  }
}

function mostrarAlerta(mensaje, tipo){
  const el = document.getElementById("alertMsg");
  el.innerHTML = mensaje;
  el.className = tipo;
  el.style.display = "block";
  setTimeout(()=>{ el.style.display="none"; }, 3000);
}

async function reiniciar() {
  if(cooldownCountdown > 0) return;

  reinicioActivo = true;

  ledCountdown = ledDuration;
  ledCounter = 0;
  document.getElementById("estado").innerHTML = "ROUTER desconectado";
  document.getElementById("led").style.background = "red";

  const statusEl = document.getElementById("wifiStatus");
  const ledWifi = document.getElementById("ledWifi");
  statusEl.innerHTML = "Desconectado a Internet";
  ledWifi.style.background = "red";

  document.getElementById("btnReiniciar").disabled = true;

  try {
    const tokenRes = await fetch("/token");
    const token = await tokenRes.text();
    await fetch("/encender?token=" + token, { method:"POST" });
  } catch(e){
    mostrarAlerta("ESP32 no disponible, simulando localmente", "offline");
  }

  setTimeout(() => {
    reinicioActivo = false;
  }, ledDuration*1000);

  cooldownCountdown = cooldownDuration;
  cooldownCounter = 0;
```

```
}

function actualizarSimulacion() {
  if(ledCountdown > 0){
    ledCounter++;
    if(ledCounter >= ledDuration){
      ledCountdown = 0;
      ledCounter = 0;
      document.getElementById("estado").innerHTML = "ROUTER conectado";
      document.getElementById("led").style.background = "green";
    }
  }

  if(cooldownCountdown > 0){
    cooldownCounter++;
    document.getElementById("countdownFill").style.width =
((cooldownCounter/cooldownDuration)*100)+"%";
    if(cooldownCounter >= cooldownDuration){
      cooldownCountdown = 0;
      cooldownCounter = 0;
      document.getElementById("countdownFill").style.width = "0%";
      document.getElementById("btnReiniciar").disabled = false;
      setTimeout(() => {
        location.reload();
      }, 500);
    }
  }
}

setInterval(actualizarSimulacion, 1000);
setInterval(actualizarInternet, 1000);
</script>
</head>
<body>
<div id="alertMsg"></div>
<div class="container">
<h1>Reinicio del Router</h1>
<button id="btnReiniciar" onclick="reiniciar()">Reiniciar</button>
<div id="countdownBar"><div id="countdownFill"></div></div>

<h2>Alimentación ROUTER:</h2>
<div id="estado">ROUTER conectado</div>
<div id="led" class="status-circle" style="background:green;"></div>

<h2>Estado Internet:</h2>
<div id="wifiStatus">Conectado a Internet</div>
<div id="ledWifi" class="status-circle" style="background:green;"></div>
</div>
<div class="footer">Powered by Lino, EB2CTA</div>
</body>
</html>
)====";

  server->send(200,"text/html",html);
}
```

A continuación se realizan las interacciones entre la placa ESP32 y la página Web intercambiando datos entre ellos, es decir, el usuario interacciona con la página Web y comanda el reinicio mediante el corte de la alimentación del Router (10 segundos) por la activación del relé. Así mismo, se informa al usuario de la situación en tiempo real de la alimentación y conexión a Internet del Router:

```
// ======================================================
// ENDPOINTS (INTERACCION DE DATOS CON LA WEB)
// ======================================================
void handleToken() {
  if (!isAuthenticated()) {                          // Comprueba que el Usuario esté
autenticado
    server->send(401, "text/plain", "No autorizado"); // Si no está autenticado envia un mensaje
"No autorizado"
    return;                                          // Sale y no continua
  }
  server->send(200,"text/plain",generarToken());     // Genera un Token y lo envia como
respuesta HTTP
}

void handleEncender() {
```

```
  if (!isAuthenticated()) return; // Verifica que el Usuario esté autenticado; si no, hace
return para salir
  if (!validarToken()) return;    // Verifica que el Token de seguridad sea válido; si no, hace
return para salir
  if (releActive) return;         // Indica si el relé ya está activado; si está activado hace
return para salir y evita que se active el relé dos veces al mismo tiempo
  server->send(200,"text/plain","ROUTER desconectado"); // Envía código 200 OK, tipo texto plano
y con el mensaje "ROUTER desconectado"

  xTaskCreate([](void*) {  // Activación del relé que desconecta la alimentación del Router,
crea una tarea independiente usando una función lambda lo que hace que no se bloquee el loop() y
ejecutar el temporizador sin usar delay()
    Serial.println("🔌 Alimentación desconectada");
    releActive = true;
    digitalWrite(RELE_PIN, HIGH);             // Se activa el relé
    vTaskDelay(10000 / portTICK_PERIOD_MS);   // Se pausa la tarea 10 segundos (Tiempo que el
relé esta activado) sin bloquear el microprocesador y siguen funcionando el resto de tareas
    digitalWrite(RELE_PIN, LOW);              // Se desactiva el relé
    releActive = false;
    Serial.println("🔌 Alimentación conectada");
    vTaskDelete(NULL);          // Destruye la tarea y libera memoria
  }, "Relay", 4096, NULL, 1, NULL);   // Nombre e la tarea: "Relay"; Tamaño de la pila (stack):
4096; Parámetros de entrada: NULL; Prioridad: 1; No guarda el handle: NULL

  cooldownActive = true;        // Inicia un tiempo de espera
  cooldownStart = millis();     // Guarda en cooldownStart el momento de inicio
}

void handleStatus() {           // Envía una respuesta HTTP al cliente, indicando si el
dispositivo está conectado a la red WiFi
  delay(WAIT_MS);               // Espera forzada
  server->send(200,"text/plain",
    (WiFi.status() == WL_CONNECTED) ? "online" : "offline");
}
```

En el bloque siguiente de código se define el preceptivo `setup()` de Arduino, done se abre el canal de comunicación con el Monitor Serie del IDE para ayudar a la depuración y monitorización del correcto funcionamiento del programa. Se definen los pines que gestionan el relé y el reset de la placa como pines de salida y se les asigna el valor `LOW` (0 V) y se conecta la plaza ESP32 a la red local y se levanta el servidor Web. También se configura el Watchdog, se mata cualquier tarea watchdog previa, se le define que vigile todos los núcleos del procesador y la tarea en curso a vigilar:

```
// ======================================================
// SETUP
// ======================================================
void setup() {
  Serial.begin(115200);           // Inicia la comunicación serie del Monitor Serie del IDE

  // Configuración Watchdog
    delay(100); // Pequeña pausa para asegurarse de que el sistema ya arrancó correctamente
antes de tocar el WDT.
    Serial.println("⚙️ Configurando Watchdog");
    Serial.print("🔌 Watchdog establecido en: ");
    Serial.print(WDT_TIMEOUT);
    Serial.println(" en segundos");
    esp_task_wdt_deinit();  // Desactiva cualquier configuración previa de tareas watchdog
    // Configuración del vigilante de tareas
      esp_task_wdt_config_t wdt_config = {                // Se crea una estructura con los
parámetros del WDT
        .timeout_ms = WDT_TIMEOUT * 1000,                 // Convierte a milisegundos
        .idle_core_mask = (1 << portNUM_PROCESSORS) - 1,  // Vigila todos los núcleos del
procesador (generalmente tiene 2)
        .trigger_panic = true                             // Habilita el reinicio del ESP32
      };
    // Inicializa el WDT
      ESP32_ERROR = esp_task_wdt_init(&wdt_config);       // Inicializa el WDT con esa
configuración y guards resultado en ESP32_ERROR (OK o ERROR)
      Serial.println("🔌 Último reinicio del Watchdog: " + String(esp_err_to_name(ESP32_ERROR)));
// Muestra el estado: OK, ESP_ERR_INVALID_STATE, etc.
      esp_task_wdt_add(NULL);  // Agregar el hilo actual al WDT, es decir, le dices que vigile
esta tarea
    // Fin Configuración Watchdog

  pinMode(RELE_PIN, OUTPUT);      // Define ese pin como pin de salida
  pinMode(RESET_PIN, OUTPUT);     // Define ese pin como pin de salida
```

```
  digitalWrite(RELE_PIN, LOW);    // Envia a ese pin 0V
  digitalWrite(RESET_PIN, LOW);   // Envia a ese pin 0V

  if (!loadConfigFromSD()) {
    Serial.println("⚠ ERROR SD - Reinicio bloqueado");
    while (true);
  }

  server = new WebServer(webPort);

  delay(WAIT_MS);                                  // Espera forzada

  WiFi.config(local_IP, gateway, subnet, primaryDNS, secondaryDNS);  // Configura una IP
estática segun los parametros leidos de la MicroSD
  delay(WAIT_MS);                                  // Espera forzada
  WiFi.begin(ssid.c_str(), password.c_str());      // Se conecta a la red Wifi SSID con su
correpondiente contraseña
  delay(WAIT_MS);    // Espera forzada
  Serial.print("🔄 Conectando WiFi");
  while (WiFi.status() != WL_CONNECTED) {          // Mientras no haya conexión WiFi va mostrando
puntos en el Monitor Serie
    delay(500);                                    // Cada 500 milisegundos
    Serial.print(".");
  }

  Serial.println("\n☑WiFi conectado");             // Muestra en el monitor Serie que la placa
ESP32 ee ha conectado al WiFi
  Serial.print("📡 Dirección MAC: ");
  Serial.println(WiFi.macAddress());               // Muestra en el monitor Serie la MAC de la
placa ESP32
  Serial.print("🌐 Dirección IP: ");
  Serial.println(WiFi.localIP());                  // Muestra en el monitor Serie la IP de la
placa ESP32

  server->on("/", handleRoot);                     // Cuando se solicita la pagina principal en
la IP:port ejecuta esa función (que es la WEB)
  server->on("/token", handleToken);               // Se gestiona la función del Token
  server->on("/encender", handleEncender);         // Se gestiona la función para activar el relé
verificando el Token
  server->on("/status", handleStatus);             // Se gestiona la función para ver si hay ya
internet despues del reinicio

  server->begin();                                 // Inicia el servidor Web y espera conexones
de clientes
  Serial.print("🔌 Puerto del Servidor Web: ");
  Serial.println(webPort);                         // Muestra en el Monitor Serie el puerto
seleccionado

  Serial.println("☑Servidor Web iniciado");        // Muestra aviso en Monitor Serie de que el
servidor Web está levandado
  Serial.print("💾 Memoria RAM disponible: ");
  Serial.print(ESP.getFreeHeap());                 // Muestran la memoria disponible en el ESP32
  Serial.println(" bytes");
}
```

El último bloque corresponde al código `loop()` de Arduino que se reiterará una y otra vez mientras la placa esté energizada. En primer lugar se resetea el contador Watchdog para que no se reinicie la placa ESP32. Al pasar por estas instrucciones entiende en menos de `WDT_TIMEOUT` minutos, entiende que no está bloqueada la placa y vuelve a contar de nuevo la cuenta atrás del watchdog.

Desde aquí se hacen las llamadas a las funciones de conexión a WiFi, cierra sockets TCP y libera recursos asociados, detiene el servidor Web, y reinicia la placa mediante un reinicio físico al energizar la base del transistor 2N2222 activando el pin 13 haciendo funcionar dicho transistor como un interruptor poniendo el pin EN a nivel `LOW`, es decir conectándolo a 0V (GND) lo que fuerza el reinicio de la placa ESP32. Como refuerzo de la estabilidad del sistema, se realiza un reinicio periódico cada `RESTART_INTERVAL_MS` minutos (en este caso cada 5 min; no obstante dicho valor puede ser cambiado en esa variable definida en el código. Cabe reseñar que se han incluido

algunas líneas de espera forzada que aporta estabilidad ya que permite los rearmes adecuados.

```
// ======================================================
// LOOP
// ======================================================
void loop() {

    // Resetea el contador Watchdog
        esp_task_wdt_reset(); // Resetea el WDT
        delay(1); // Importante para que el reset se aplique
    // Fin Resetea el contador Whatchdog

  if (server) server->handleClient();             // Comprueba y maneja peticiones HTTP

  if (WiFi.status() != WL_CONNECTED) {            // Si el ESP32 no está conectado a WiFi
    WiFi.disconnect();                            // Termina cualquier conexion existente a WiFi
    delay(WAIT_MS * 2UL);                         // Espera forzada
    WiFi.begin(ssid.c_str(), password.c_str());   // Se conecta a la red Wifi SSID con su
correpondiente contraseña
  }

  if (cooldownActive && millis() - cooldownStart >= cooldownDuration) {
    cooldownActive = false;

    if (server) {       // Cerrar los sockets TCP y libera recursos asociados antes de reiniciar
el ESP32
      server->stop();
      delete server;
     server = nullptr;  // Desapunta el puntero (hace que el puntro no apunte a ningun sitio)
    }

    Serial.println("🔄 Reinicio de la placa ESP32");   // Llama a la función de reseteo por
software
    delay(WAIT_MS * 2UL);             // Espera WAIT_MS milisegundos para reiniciar
    digitalWrite(RESET_PIN, HIGH);    // Excita el transistor 2N2222
    delay(RESET_PULSE_MS);            // Tiempo que mantiene EN a GND
    digitalWrite(RESET_PIN, LOW);     // Libera EN
  }

  // ================ REINICIO PERIODICO ================
  if (millis() - lastRestartTime >= RESTART_INTERVAL_MS) {    //Comprueba si han pasado
RESTART_INTERVAL_MS para reiniciar periodicamente el ESP32

    lastRestartTime = millis();                          // Pone el contador otra vez a cero
para esperar otro reinicio

    if (server) {       // Cerrar los sockets TCP y libera recursos asociados antes de reiniciar
el ESP32
      server->stop();
      delete server;
     server = nullptr;  // Desapunta el puntero (hace que el puntro no apunte a ningun sitio)
    }
    server = new WebServer(webPort);
    Serial.println("🔄 Reinicio de la placa ESP32");
    delay(WAIT_MS);                   // Espera forzada
    digitalWrite(RESET_PIN, HIGH);    // Excita el transistor 2N2222
    delay(RESET_PULSE_MS);            // Tiempo que mantiene EN a GND
    digitalWrite(RESET_PIN, LOW);     // Libera EN
  }
}
```

CONFIGURACION DEL ROUTER Y DEL REINICIO

ACCESO desde la RED LOCAL:

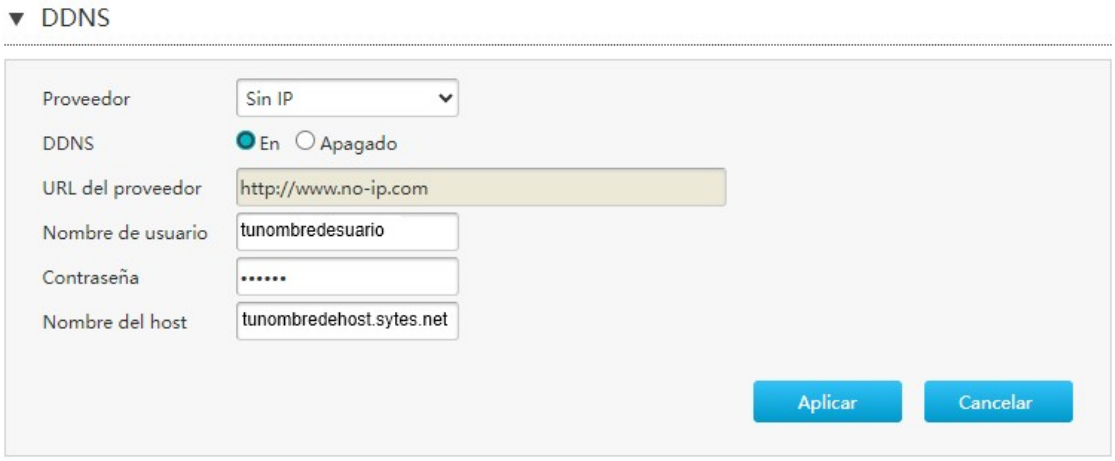
Para acceder a la Web que sirve el Arduino ESP32 y comandar el reinicio del Router se puede hacer desde la red local, y para ello, solo seria necesario escribir desde cualquier navegador la dirección IP y el puerto que hemos indicado en nuestro archivo de configuración. En nuestro caso: `192.168.1.50:81` y autenticarnos con el usuario `admin` y contraseña `1234` previamente elegidos. Previamente, debes seleccionar la red WiFi a la

que te vas a conectar, por ejemplo a le red `MiRedWiFi` cuya contraseña es `MiClaveWiFi`. Recuerda sustituir estos datos por los específicos de tu red local.

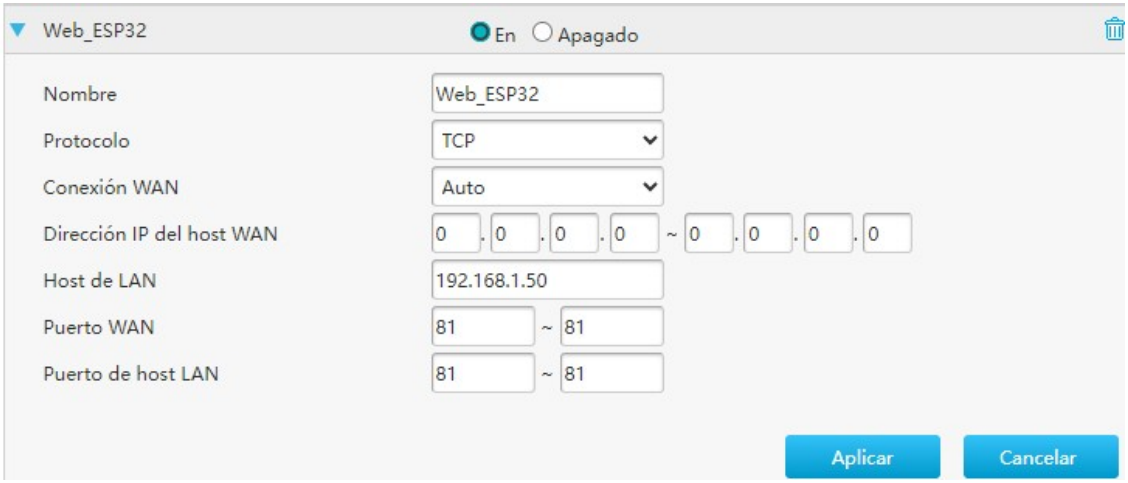
ACCESO desde INTERNET:

Para acceder a la Web que sirve el Arduino ESP32 y comandar el reinicio del Router se puede hacer desde Internet, y para ello, debemos de contar de un servicio que nos proporcione una IP pública fija. Existen muchos servicios gratuitos y de pago que nos permita tener un acceso a nuestra IP pública mediante una DDNS. Puedes utilizar https://www.noip.com/

Algunos Routers permiten automatizar este proceso una vez que te hayas dado de alta en el servicio:



La operativa es similar al caso anterior, pero tendrás que tener la precaución de redirigir el puerto (en nuestro caso de ejemplo el puerto `81`) a la dirección local `192.168.1.50` y tendrás que abrir el puerto correspondiente en tu Router:



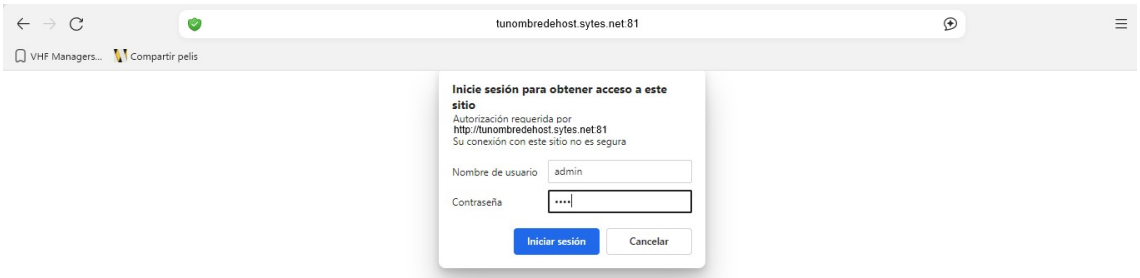
Es interesante también, fijar la asignación de la IP local en el Router mediante la MAC del dispositivo, para asegurar que ésta IP queda reservada y es solo utilizada por el dispositivo reiniciador del Router que hemos montado.

Tendrás que buscar en tu Router una pantalla parecida a esta y asignar la MAC a la IP:

La forma específica de definir estos parámetros en cada Router difiere del modelo y de la marca del mismo, pero en líneas generales todos se comportan de una manera muy similar. Consulta el manual de tu Router para hacerlo correctamente.

OPERATIVA DESDE LA WEB

Nada más acceder a la página Web nos solicitara el usuario y la contraseña para permitir el acceso:



Al iniciar sesión, entramos en la aplicación que permite reiniciar el Router:

Al pinchar sobre el botón **Reiniciar**, la alimentación del Router se interrumpe durante 10 segundos y vuelve a conectarse. La Web muestra gráficamente el proceso de reinicio y de la conexión efectiva del Router a Internet.

CONCLUSION

Este pequeño dispositivo intercalado entre el alimentador del Router y el propio Router lo reinicia para reestablecerlo cuando sea necesario por un mal funcionamiento del mismo o conflictos entre las IP utilizadas o asignadas de manera automática. Como consejo, es importante que la IP asignada al Arduino ESP32 sea fija y reservada en el Router para este propósito.

73 y buenos DX de Lino, EB2CTA.