PROTOTIPO

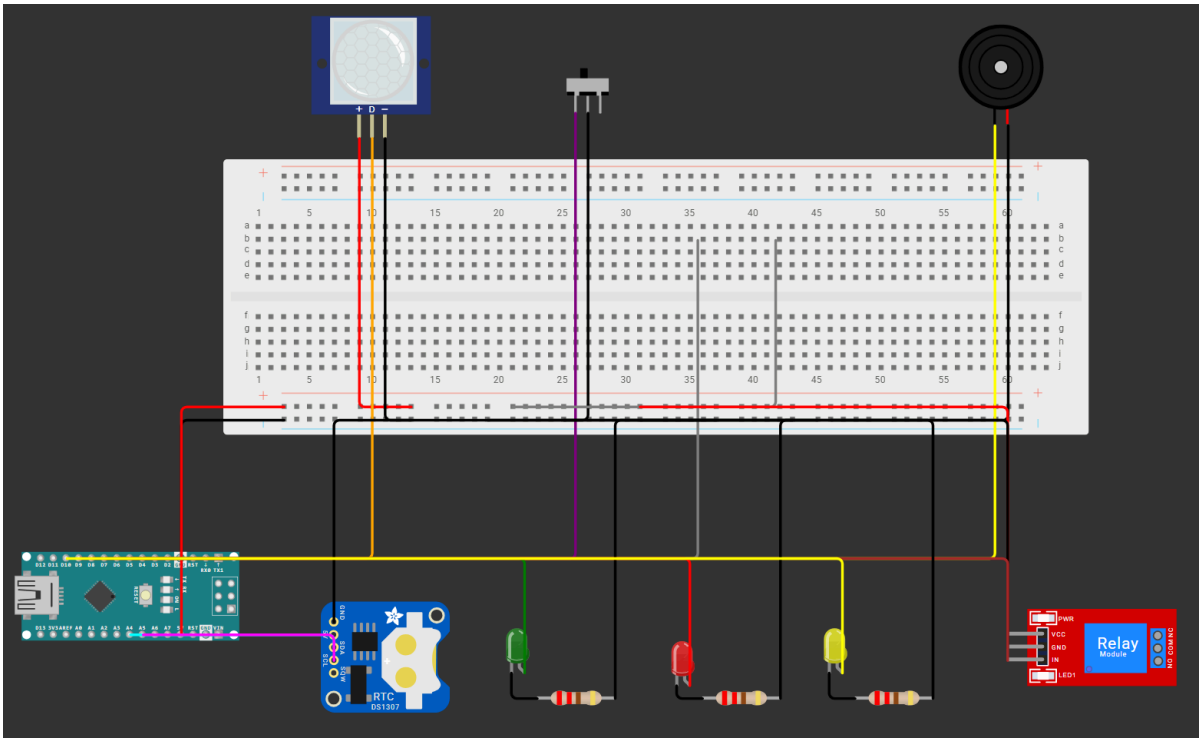

DIAGRAMA:

JSON

```
{
  "version": 1,
  "author": "Sistema de Geolocalización - Alzheimer (CORREGIDO v2)",
  "editor": "wokwi",
  "parts": [
    { "type": "wokwi-arduino-nano", "id": "nano", "top": 340.8, "left":
-365.3, "attrs": {} },
    { "type": "wokwi-esp8266-01", "id": "esp8266", "top": -100, "left": 200,
"attrs": {} },
    { "type": "wokwi-pir-motion-sensor", "id": "pir", "top": -63.2, "left":
-141.78, "attrs": {} },
    { "type": "wokwi-slide-switch", "id": "reed", "top": -24.4, "left":
51.1, "attrs": {} },
    { "type": "wokwi-buzzer", "id": "buzzer", "top": -64.8, "left": 347.4,
"attrs": {} },
    { "type": "wokwi-relay-module", "id": "relay", "top": 384.2, "left":
384, "attrs": {} },
    { "type": "wokwi-ds1307", "id": "rtc", "top": 378.6, "left": -134.3,
"attrs": {} },
```

```
    { "type": "wokwi-led", "id": "led_green", "top": 390, "left": -5.8,
"attrs": { "color": "green" } },
    { "type": "wokwi-led", "id": "led_red", "top": 399.6, "left": 119,
"attrs": { "color": "red" } },
    { "type": "wokwi-led", "id": "led_yellow", "top": 390, "left": 234.2,
"attrs": { "color": "yellow" } },
    { "type": "wokwi-resistor", "id": "r1", "top": 445.55, "left": 28.8,
"attrs": { "value": "220" } },
    { "type": "wokwi-resistor", "id": "r2", "top": 445.55, "left": 153.6,
"attrs": { "value": "220" } },
    { "type": "wokwi-resistor", "id": "r3", "top": 445.55, "left": 268.8,
"attrs": { "value": "220" } },
    { "type": "wokwi-resistor", "id": "r4", "top": 100, "left": 150,
"attrs": { "value": "10000" } },
    { "type": "wokwi-breadboard", "id": "bb", "top": 45, "left": -208.4,
"attrs": {} }
  ],
  "connections": [
    [ "nano:GND.2", "bb:bn.1", "black", [ "v0" ] ],
    [ "nano:5V", "bb:bp.1", "red", [ "v0" ] ],

    [ "esp8266:VCC", "bb:bp.5", "red", [ "v0" ] ],
    [ "esp8266:GND", "bb:bn.5", "black", [ "v0" ] ],
    [ "esp8266:TX", "nano:2", "blue", [ "v0" ] ],
    [ "esp8266:RX", "nano:3", "green", [ "v0" ] ],

    [ "pir:VCC", "bb:bp.10", "red", [ "v0" ] ],
    [ "pir:GND", "bb:bn.10", "black", [ "v0" ] ],
    [ "pir:OUT", "nano:4", "orange", [ "v0" ] ],

    [ "reed:1", "nano:5", "purple", [ "v0" ] ],
    [ "reed:2", "bb:bn.15", "black", [ "v0" ] ],
    [ "r4:1", "nano:5", "gray", [ "v0" ] ],
    [ "r4:2", "bb:bp.16", "gray", [ "v0" ] ],

    [ "buzzer:1", "nano:6", "yellow", [ "v0" ] ],
    [ "buzzer:2", "bb:bn.20", "black", [ "v0" ] ],

    [ "relay:VCC", "bb:bp.25", "red", [ "v0" ] ],
    [ "relay:GND", "bb:bn.25", "black", [ "v0" ] ],
    [ "relay:IN", "nano:7", "brown", [ "v0" ] ],

    [ "rtc:VCC", "bb:bp.30", "red", [ "v0" ] ],
    [ "rtc:GND", "bb:bn.30", "black", [ "v0" ] ],
    [ "rtc:SDA", "nano:A4", "cyan", [ "v0" ] ],
    [ "rtc:SCL", "nano:A5", "magenta", [ "v0" ] ],

    [ "led_green:A", "nano:8", "green", [ "v0" ] ],
```

```
    [ "led_green:C", "r1:1", "black", [ "v0" ] ],
    [ "r1:2", "bb:bn.35", "black", [ "v0" ] ],

    [ "led_red:A", "nano:9", "red", [ "v0" ] ],
    [ "led_red:C", "r2:1", "black", [ "v0" ] ],
    [ "r2:2", "bb:bn.36", "black", [ "v0" ] ],

    [ "led_yellow:A", "nano:10", "yellow", [ "v0" ] ],
    [ "led_yellow:C", "r3:1", "black", [ "v0" ] ],
    [ "r3:2", "bb:bn.37", "black", [ "v0" ] ]
  ],
  "dependencies": {}
}
```

CÓDIGO

C/C++

```
/*
Sistema de Geolocalización para Personas con Alzheimer - VERSIÓN CORREGIDA
Detección de abandono de perímetro por WiFi y sensores físicos

CORRECCIONES APLICADAS:
- Lógica Reed Switch corregida (HIGH = abierto)
- Baudrate ESP8266 optimizado para Wokwi (115200)
- Eliminación de comando RSSI duplicado
- Patrón de buzzer corregido
- Strings optimizados con buffers estáticos
- Mejor manejo de memoria RAM

Componentes:
- Arduino Nano
- ESP8266-01 (WiFi) - En Wokwi: alimentar a 3.3V o 5V
- Sensor PIR (movimiento)
- Reed Switch (apertura) con resistencia pull-up 10kΩ
- Buzzer (alarma sonora)
- Módulo Relé (alarma externa)
- DS1307 (RTC)
- LEDs indicadores (Verde, Rojo, Amarillo)
- Resistencias limitadoras 220Ω para LEDs
*/

#include <Wire.h>
#include <SoftwareSerial.h>
#include <avr/wdt.h>
```

```
// ==================== DEFINICIÓN DE PINES ====================
#define ESP_RX 2
#define ESP_TX 3
#define PIR_PIN 4
#define REED_PIN 5
#define BUZZER_PIN 6
#define RELAY_PIN 7
#define LED_GREEN 8
#define LED_RED 9
#define LED_YELLOW 10

// ==================== CONFIGURACIÓN WIFI ====================
// Para Wokwi usar: "Wokwi-GUEST" sin contraseña
const char* WIFI_SSID = "Wokwi-GUEST";
const char* WIFI_PASS = "";

// ==================== CONFIGURACIONES DEL SISTEMA ====================
const int SIGNAL_THRESHOLD = -70;      // dBm - señal mínima aceptable
const int SIGNAL_WARNING = -65;        // dBm - señal débil, advertencia
const unsigned long CHECK_INTERVAL = 5000;           // Verificar WiFi cada
5 segundos
const unsigned long ALARM_DURATION = 30000;          // Duración de alarma:
30 segundos
const unsigned long PIR_DELAY = 2000;                // Delay después de
detección PIR
const unsigned long DEBOUNCE_DELAY = 50;             // Delay para
debouncing (ms)
const unsigned long WIFI_RECONNECT_INTERVAL = 30000; // Intentar reconexión
cada 30s
const int MAX_RECONNECT_ATTEMPTS = 3;                // Intentos de
reconexión antes de alarma

// ==================== MÓDULO WIFI ====================
SoftwareSerial espSerial(ESP_RX, ESP_TX);

// ==================== VARIABLES DE ESTADO ====================
bool wifiConnected = false;
bool alarmActive = false;
bool perimeterBreach = false;
int wifiSignalStrength = 0;
unsigned long lastCheckTime = 0;
unsigned long alarmStartTime = 0;
unsigned long lastReconnectAttempt = 0;
int reconnectAttempts = 0;

// Variables de sensores
bool pirState = false;
```

```
bool reedState = false;
bool lastReedState = false;
unsigned long lastDebounceTime = 0;

// Variables de control
bool systemReady = false;

// Dirección del DS1307 en I2C
#define DS1307_ADDRESS 0x68

// ==================== ENUMERACIÓN DE ESTADOS ====================
enum SystemState {
  STATE_INIT,
  STATE_NORMAL,
  STATE_WARNING,
  STATE_ALARM,
  STATE_ERROR
};
SystemState currentState = STATE_INIT;

// ==================== SETUP ====================
void setup() {
  // Desactivar watchdog inicialmente
  wdt_disable();

  // Inicializar comunicación serial
  Serial.begin(9600);
  espSerial.begin(115200); // 115200 para mejor compatibilidad con Wokwi

  // Configurar pines
  pinMode(PIR_PIN, INPUT);
  pinMode(REED_PIN, INPUT_PULLUP); // PULL-UP INTERNO ACTIVADO
  pinMode(BUZZER_PIN, OUTPUT);
  pinMode(RELAY_PIN, OUTPUT);
  pinMode(LED_GREEN, OUTPUT);
  pinMode(LED_RED, OUTPUT);
  pinMode(LED_YELLOW, OUTPUT);

  // Estado inicial - todo apagado
  digitalWrite(BUZZER_PIN, LOW);
  digitalWrite(RELAY_PIN, LOW);
  digitalWrite(LED_RED, LOW);
  digitalWrite(LED_YELLOW, LOW);
  digitalWrite(LED_GREEN, LOW);

  Serial.println(F("================================"));
  Serial.println(F("Sistema de Geolocalización"));
  Serial.println(F("================================"));
```

```
  // Animación de LEDs en inicio
  startupAnimation();

  // Inicializar RTC (DS1307)
  Serial.println(F("Inicializando RTC..."));
  Wire.begin();
  initRTC();

  // Inicializar WiFi
  Serial.println(F("Inicializando WiFi..."));
  delay(1000);
  initWiFi();

  // Sistema listo
  systemReady = true;
  currentState = STATE_NORMAL;

  // Activar watchdog timer (8 segundos)
  wdt_enable(WDTO_8S);

  Serial.println(F("Sistema listo!"));
  Serial.println(F("Monitoreando perímetro..."));
  Serial.println(F("Comandos disponibles: STATUS, RESET, TEST, WIFI, TIME,
HELP"));
  Serial.println();
}

// ==================== LOOP PRINCIPAL ====================
void loop() {
  // Resetear watchdog timer
  wdt_reset();

  unsigned long currentTime = millis();

  // Procesar comandos serial si hay
  processSerialCommands();

  // Verificar conexión WiFi periódicamente
  if (currentTime - lastCheckTime >= CHECK_INTERVAL) {
    lastCheckTime = currentTime;
    checkWiFiConnection();
  }

  // Intentar reconexión si está desconectado
  if (!wifiConnected && (currentTime - lastReconnectAttempt >=
WIFI_RECONNECT_INTERVAL)) {
    lastReconnectAttempt = currentTime;
```

```
    attemptReconnection();
  }

  // Leer sensores físicos
  checkPhysicalSensors();

  // Evaluar condiciones de perímetro
  evaluatePerimeter();

  // Actualizar estado del sistema
  updateSystemState();

  // Actualizar LEDs de estado
  updateStatusLEDs();

  // Gestionar alarma
  manageAlarm();

  delay(100); // Pequeño delay para estabilidad
}

// ==================== INICIALIZACIÓN WIFI ====================
void initWiFi() {
  Serial.println(F("Reseteando módulo ESP8266..."));

  // Resetear módulo ESP8266
  sendATCommand("AT+RST", 2000);

  // Configurar modo estación
  sendATCommand("AT+CWMODE=1", 1000);

  // Construir comando de conexión con buffer estático
  char connectCmd[100];
  snprintf(connectCmd, sizeof(connectCmd), "AT+CWJAP=\"%s\",\"%s\"",
WIFI_SSID, WIFI_PASS);

  Serial.print(F("Conectando a: "));
  Serial.println(WIFI_SSID);

  String response = sendATCommand(connectCmd, 8000);

  if (response.indexOf("OK") != -1 || response.indexOf("CONNECTED") != -1) {
    wifiConnected = true;
    reconnectAttempts = 0;
    Serial.println(F("WiFi conectado exitosamente"));

    digitalWrite(LED_GREEN, HIGH);
    delay(500);
```

```
    digitalWrite(LED_GREEN, LOW);
  } else {
    wifiConnected = false;
    Serial.println(F("Error al conectar WiFi"));
    Serial.println(F("Verifique SSID y contraseña"));
    digitalWrite(LED_YELLOW, HIGH);
  }
}

// ==================== VERIFICACIÓN WIFI ====================
void checkWiFiConnection() {
  String response = sendATCommand("AT+CWJAP?", 1000);

  // Verificar si hay respuesta válida
  if (response.indexOf("No AP") != -1 || response.indexOf("ERROR") != -1) {
    if (wifiConnected) {
      wifiConnected = false;
      Serial.println(F("ALERTA: Conexión WiFi perdida"));
      logEvent("WiFi perdido");
    }
  } else if (response.indexOf("+CWJAP:") != -1) {
    wifiConnected = true;
    reconnectAttempts = 0;

    // Obtener RSSI de la misma respuesta (sin duplicar comando)
    int rssiIndex = response.indexOf(",-");
    if (rssiIndex != -1) {
      String rssiStr = response.substring(rssiIndex + 1, rssiIndex + 4);
      wifiSignalStrength = rssiStr.toInt();

      if (wifiSignalStrength < SIGNAL_THRESHOLD) {
        Serial.print(F("Señal débil: "));
        Serial.print(wifiSignalStrength);
        Serial.println(F(" dBm"));
      }
    }
  }
}

// ==================== RECONEXIÓN AUTOMÁTICA ====================
void attemptReconnection() {
  if (reconnectAttempts < MAX_RECONNECT_ATTEMPTS) {
    reconnectAttempts++;
    Serial.print(F("Intento de reconexión WiFi #"));
    Serial.println(reconnectAttempts);

    initWiFi();
```

```
    if (!wifiConnected && reconnectAttempts >= MAX_RECONNECT_ATTEMPTS) {
      Serial.println(F("FALLO: No se pudo reconectar después de múltiples
intentos"));
      logEvent("Fallo reconexión WiFi");
    }
  }
}

// ==================== LECTURA DE SENSORES ====================
void checkPhysicalSensors() {
  // Leer sensor PIR
  pirState = digitalRead(PIR_PIN);
  if (pirState == HIGH) {
    Serial.println(F("Movimiento detectado por PIR"));
    logEvent("Movimiento PIR");
    delay(PIR_DELAY);
  }

  // Leer reed switch con debouncing
  int reading = digitalRead(REED_PIN);

  if (reading != lastReedState) {
    lastDebounceTime = millis();
  }

  if ((millis() - lastDebounceTime) > DEBOUNCE_DELAY) {
    if (reading != reedState) {
      reedState = reading;

      // Reed switch con PULL-UP: HIGH = abierto (alarma), LOW = cerrado
(normal)
      if (reedState == HIGH) {
        Serial.println(F("ALERTA: Apertura física detectada"));
        logEvent("Apertura física");
        perimeterBreach = true;
      } else {
        Serial.println(F("Cierre físico detectado"));
      }
    }
  }

  lastReedState = reading;
}

// ==================== EVALUACIÓN DE PERÍMETRO ====================
void evaluatePerimeter() {
  bool breachDetected = false;
  String breachCause = "";
```

```
  // Verificar WiFi
  if (!wifiConnected) {
    breachDetected = true;
    breachCause = "Pérdida de conexión WiFi";
  } else if (wifiSignalStrength < SIGNAL_THRESHOLD && wifiSignalStrength !=
0) {
    breachDetected = true;
    breachCause = "Señal WiFi muy débil";
  }

  // Verificar Reed Switch (HIGH = abierto con pull-up)
  if (reedState == HIGH) {
    breachDetected = true;
    if (breachCause.length() > 0) breachCause += " + ";
    breachCause += "Apertura física";
  }

  // Actualizar estado de violación
  if (breachDetected && !perimeterBreach) {
    perimeterBreach = true;
    Serial.println();
    Serial.println(F("*** VIOLACIÓN DE PERÍMETRO DETECTADA ***"));
    Serial.print(F("Causa: "));
    Serial.println(breachCause);
    Serial.println(F("Activando alarmas..."));
    logEvent("PERÍMETRO VIOLADO: " + breachCause);
  } else if (!breachDetected && perimeterBreach && !alarmActive) {
    perimeterBreach = false;
    Serial.println(F("Perímetro restaurado"));
    logEvent("Perímetro restaurado");
  }
}

// ==================== ACTUALIZACIÓN DE ESTADO ====================
void updateSystemState() {
  if (alarmActive) {
    currentState = STATE_ALARM;
  } else if (perimeterBreach) {
    currentState = STATE_WARNING;
  } else if (!wifiConnected) {
    currentState = STATE_ERROR;
  } else if (wifiSignalStrength < SIGNAL_WARNING && wifiSignalStrength != 0)
{
    currentState = STATE_WARNING;
  } else {
    currentState = STATE_NORMAL;
  }
```

```
}

// ==================== ACTUALIZACIÓN DE LEDS ====================
void updateStatusLEDs() {
  unsigned long currentTime = millis();

  switch (currentState) {
    case STATE_ALARM:
      // LED rojo parpadeando rápido (250ms)
      digitalWrite(LED_RED, (currentTime / 250) % 2);
      digitalWrite(LED_GREEN, LOW);
      digitalWrite(LED_YELLOW, LOW);
      break;

    case STATE_WARNING:
      // LED amarillo parpadeando lento (1000ms)
      digitalWrite(LED_YELLOW, (currentTime / 1000) % 2);
      digitalWrite(LED_GREEN, LOW);
      digitalWrite(LED_RED, LOW);
      break;

    case STATE_ERROR:
      // LED amarillo y rojo alternando
      bool state = (currentTime / 500) % 2;
      digitalWrite(LED_YELLOW, state);
      digitalWrite(LED_RED, !state);
      digitalWrite(LED_GREEN, LOW);
      break;

    case STATE_NORMAL:
      // LED verde fijo
      digitalWrite(LED_GREEN, HIGH);
      digitalWrite(LED_RED, LOW);
      digitalWrite(LED_YELLOW, LOW);
      break;

    case STATE_INIT:
      // Todos parpadeando
      digitalWrite(LED_GREEN, (currentTime / 300) % 2);
      digitalWrite(LED_YELLOW, (currentTime / 300) % 2);
      digitalWrite(LED_RED, (currentTime / 300) % 2);
      break;
  }
}

// ==================== GESTIÓN DE ALARMA ====================
void manageAlarm() {
  unsigned long currentTime = millis();
```

```
  // Activar alarma si hay violación
  if (perimeterBreach && !alarmActive) {
    alarmActive = true;
    alarmStartTime = currentTime;
    digitalWrite(RELAY_PIN, HIGH);
    Serial.println(F(">>> ALARMA ACTIVADA <<<"));
  }

  // Desactivar alarma después del timeout
  if (alarmActive && (currentTime - alarmStartTime >= ALARM_DURATION)) {
    alarmActive = false;
    digitalWrite(BUZZER_PIN, LOW);
    digitalWrite(RELAY_PIN, LOW);
    Serial.println(F("Alarma desactivada (timeout)"));

    if (perimeterBreach) {
      Serial.println(F("ATENCIÓN: Perímetro aún comprometido"));
    }
  }

  // Patrón de beep intermitente durante alarma (200ms ON, 100ms OFF)
  if (alarmActive) {
    unsigned long alarmTime = currentTime - alarmStartTime;
    bool beepState = (alarmTime % 300) < 200;
    digitalWrite(BUZZER_PIN, beepState ? HIGH : LOW);
  } else {
    digitalWrite(BUZZER_PIN, LOW);
  }
}

// ==================== INICIALIZACIÓN RTC ====================
void initRTC() {
  Wire.beginTransmission(DS1307_ADDRESS);
  Wire.write(0x00);
  Wire.endTransmission();

  // Verificar si el RTC está funcionando
  Wire.requestFrom(DS1307_ADDRESS, 1);

  if (Wire.available()) {
    byte segundos = Wire.read();

    if (segundos & 0x80) {
      Serial.println(F("RTC detenido, iniciando..."));
      // Iniciar el oscilador
      Wire.beginTransmission(DS1307_ADDRESS);
      Wire.write(0x00);
```

```
      Wire.write(0x00);
      Wire.endTransmission();
    }

    Serial.println(F("RTC inicializado correctamente"));
  } else {
    Serial.println(F("ADVERTENCIA: RTC no detectado"));
  }
}

// ==================== REGISTRO DE EVENTOS ====================
void logEvent(String event) {
  String timestamp = getTimestamp();
  Serial.print(F("["));
  Serial.print(timestamp);
  Serial.print(F("] "));
  Serial.println(event);
}

// ==================== OBTENER TIMESTAMP ====================
String getTimestamp() {
  Wire.beginTransmission(DS1307_ADDRESS);
  Wire.write(0x00);
  Wire.endTransmission();

  Wire.requestFrom(DS1307_ADDRESS, 7);

  if (Wire.available() >= 7) {
    int seconds = bcdToDec(Wire.read() & 0x7F);
    int minutes = bcdToDec(Wire.read());
    int hours = bcdToDec(Wire.read() & 0x3F);
    Wire.read(); // día de la semana
    int day = bcdToDec(Wire.read());
    int month = bcdToDec(Wire.read());
    int year = bcdToDec(Wire.read()) + 2000;

    char buffer[20];
    sprintf(buffer, "%02d/%02d/%04d %02d:%02d:%02d",
            day, month, year, hours, minutes, seconds);
    return String(buffer);
  }

  return F("00/00/0000 00:00:00");
}

// ==================== CONVERSIONES BCD ====================
byte bcdToDec(byte val) {
  return ((val / 16 * 10) + (val % 16));
```

```
}

byte decToBcd(byte val) {
  return ((val / 10 * 16) + (val % 10));
}

// ==================== COMANDOS AT OPTIMIZADOS ====================
String sendATCommand(const char* cmd, unsigned long timeout) {
  String response = "";
  espSerial.println(cmd);

  unsigned long startTime = millis();

  while (millis() - startTime < timeout) {
    while (espSerial.available()) {
      char c = espSerial.read();
      response += c;
    }

    if (response.indexOf("OK") != -1 || response.indexOf("ERROR") != -1) {
      break;
    }
  }

  return response;
}

// ==================== COMANDOS SERIAL ====================
void processSerialCommands() {
  if (Serial.available()) {
    String command = Serial.readStringUntil('\n');
    command.trim();
    command.toUpperCase();

    if (command == "STATUS") {
      printStatus();
    } else if (command == "RESET") {
      Serial.println(F("Reiniciando sistema..."));
      delay(100);
      wdt_enable(WDTO_15MS);
      while(1) {}
    } else if (command == "TEST") {
      testAlarm();
    } else if (command == "WIFI") {
      Serial.println(F("Reintentando conexión WiFi..."));
      initWiFi();
    } else if (command == "TIME") {
      Serial.print(F("Hora actual: "));
```

```
      Serial.println(getTimestamp());
    } else if (command == "HELP") {
      printHelp();
    } else {
      Serial.println(F("Comando no reconocido. Escriba HELP"));
    }
  }
}

// ==================== IMPRIMIR ESTADO ====================
void printStatus() {
  Serial.println(F("\n========== ESTADO DEL SISTEMA =========="));
  Serial.print(F("Estado: "));

  switch(currentState) {
    case STATE_INIT: Serial.println(F("INICIALIZANDO")); break;
    case STATE_NORMAL: Serial.println(F("NORMAL")); break;
    case STATE_WARNING: Serial.println(F("ADVERTENCIA")); break;
    case STATE_ALARM: Serial.println(F("ALARMA ACTIVA")); break;
    case STATE_ERROR: Serial.println(F("ERROR")); break;
  }

  Serial.print(F("WiFi: "));
  Serial.println(wifiConnected ? F("CONECTADO") : F("DESCONECTADO"));

  Serial.print(F("Señal WiFi: "));
  Serial.print(wifiSignalStrength);
  Serial.println(F(" dBm"));

  Serial.print(F("Perímetro: "));
  Serial.println(perimeterBreach ? F("VIOLADO") : F("SEGURO"));

  Serial.print(F("Alarma: "));
  Serial.println(alarmActive ? F("ACTIVA") : F("INACTIVA"));

  Serial.print(F("Sensor PIR: "));
  Serial.println(pirState ? F("MOVIMIENTO") : F("SIN MOVIMIENTO"));

  Serial.print(F("Reed Switch: "));
  Serial.println(reedState == HIGH ? F("ABIERTO") : F("CERRADO"));

  Serial.print(F("Hora actual: "));
  Serial.println(getTimestamp());

  Serial.print(F("Intentos reconexión: "));
  Serial.println(reconnectAttempts);

  Serial.println(F("========================================\n"));
```

```
}

// ==================== PRUEBA DE ALARMA ====================
void testAlarm() {
  Serial.println(F("\n=== PRUEBA DE ALARMA ==="));

  Serial.println(F("Activando buzzer..."));
  digitalWrite(BUZZER_PIN, HIGH);
  delay(1000);
  digitalWrite(BUZZER_PIN, LOW);

  Serial.println(F("Activando relé..."));
  digitalWrite(RELAY_PIN, HIGH);
  delay(1000);
  digitalWrite(RELAY_PIN, LOW);

  Serial.println(F("Probando LEDs..."));
  digitalWrite(LED_GREEN, HIGH);
  delay(500);
  digitalWrite(LED_GREEN, LOW);

  digitalWrite(LED_YELLOW, HIGH);
  delay(500);
  digitalWrite(LED_YELLOW, LOW);

  digitalWrite(LED_RED, HIGH);
  delay(500);
  digitalWrite(LED_RED, LOW);

  Serial.println(F("Prueba completada\n"));
}

// ==================== AYUDA ====================
void printHelp() {
  Serial.println(F("\n========== COMANDOS DISPONIBLES =========="));
  Serial.println(F("STATUS - Mostrar estado del sistema"));
  Serial.println(F("RESET  - Reiniciar el Arduino"));
  Serial.println(F("TEST   - Probar alarmas y LEDs"));
  Serial.println(F("WIFI   - Reintentar conexión WiFi"));
  Serial.println(F("TIME   - Mostrar hora actual del RTC"));
  Serial.println(F("HELP   - Mostrar esta ayuda"));
  Serial.println(F("==========================================\n"));
}

// ==================== ANIMACIÓN DE INICIO ====================
void startupAnimation() {
  // Secuencia de LEDs en inicio
  for (int i = 0; i < 3; i++) {
```

```
    digitalWrite(LED_GREEN, HIGH);
    delay(150);
    digitalWrite(LED_GREEN, LOW);

    digitalWrite(LED_YELLOW, HIGH);
    delay(150);
    digitalWrite(LED_YELLOW, LOW);

    digitalWrite(LED_RED, HIGH);
    delay(150);
    digitalWrite(LED_RED, LOW);
  }

  // Todos encendidos brevemente
  digitalWrite(LED_GREEN, HIGH);
  digitalWrite(LED_YELLOW, HIGH);
  digitalWrite(LED_RED, HIGH);
  delay(500);

  // Apagar todos
  digitalWrite(LED_GREEN, LOW);
  digitalWrite(LED_YELLOW, LOW);
  digitalWrite(LED_RED, LOW);
}
```