

**Universidad Nacional
Autónoma de México
Facultad de Estudios
Superiores Cuautitlán**



**Ingeniería en Telecomunicaciones, Sistemas y
Electrónica**

Microcontroladores

**Profesor: Ing. Jorge Ricardo Gersenowies Rosas
Grupo 2809A
Semestre 2025 – II**

Reporte de proyecto 2
“Alerta sísmica con ESP8266”

Alumnos colaboradores:

- ❖ Reyes Hernández Miguel Ángel
- ❖ González Morales Luis Alfonso
- ❖ San Pedro Rivera Roberto Yael

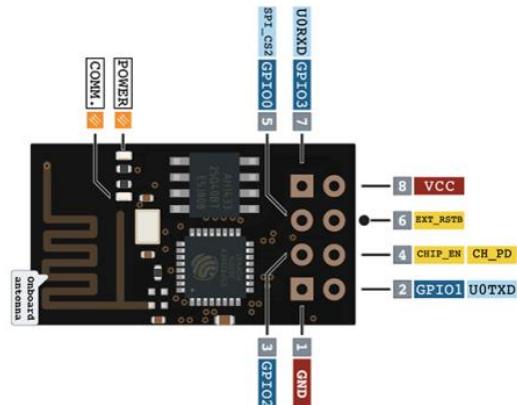
Objetivos

La prevención ante desastres naturales es fundamental para proteger vidas humanas. En este proyecto se implementa un sistema de alerta sísmica automatizada utilizando un microcontrolador ESP8266, que consulta constantemente datos en tiempo real sobre actividad sísmica en México, proporcionados por el servicio geológico de Estados Unidos (USGS). Si se detecta un evento sísmico con magnitud mayor a 5 grados y cuya ubicación esté relacionada con México, el sistema envía una alerta inmediata por Telegram y activa una alarma sonora de advertencia. Los objetivos que se desean y deben cumplir son los siguientes:

- ❖ Desarrollar un sistema autónomo de alerta sísmica en tiempo real, enfocado en eventos que afecten el territorio mexicano.
- ❖ Conectar el ESP8266 a internet mediante Wi-Fi para consumir información desde la API pública del USGS.
- ❖ Analizar los datos sísmicos y determinar si cumplen con los criterios de alerta: magnitud mayor a 5 y ubicación en México.
- ❖ Enviar notificaciones a través de Telegram a usuarios registrados, utilizando un bot programado para ese fin.
- ❖ Activar una señal de alarma sonora local durante un tiempo determinado para advertir presencialmente del sismo.
- ❖ Incluir un modo de prueba manual mediante un botón físico que simula el envío de una alerta.
- ❖ Optimizar el consumo de memoria y recursos del ESP8266 para mantener estabilidad y confiabilidad durante el monitoreo constante.

Introducción

El ESP8266 M-01 es un módulo compacto y versátil de conectividad Wi-Fi basado en el chip ESP8266 de Espressif Systems. Este módulo destaca en los sistemas embebidos e IoT por su bajo costo, facilidad de programación y capacidad para conectarse a redes inalámbricas, lo que le permite interactuar con servicios en la nube, APIs, bases de datos remotas o aplicaciones móviles. El M-01, en particular, es una variante minimalista que expone las funcionalidades básicas del ESP8266, siendo ideal para integrarse en dispositivos que requieren conectividad Wi-Fi sin sacrificar demasiado espacio físico.



¿Cómo funciona cuando hacemos una solicitud GET a través de una API?

Cuando un dispositivo como el ESP8266 M-01 realiza una solicitud HTTP GET a través de una API (Application Programming Interface), sigue una serie de pasos:

1. Conexión a la red Wi-Fi: Primero, el ESP8266 se conecta a una red Wi-Fi mediante sus librerías de software, como `ESP8266WiFi.h`, donde se especifican el SSID (nombre de la red) y la contraseña.
2. Resolución de la URL y conexión al servidor: Luego, el microcontrolador interpreta el nombre de dominio (como `earthquake.usgs.gov`) mediante DNS y establece una conexión TCP/IP con el servidor remoto.
3. Envío de la solicitud GET: Usando librerías como `ESP8266HTTPClient.h`, el módulo crea una solicitud en formato HTTP estándar, por ejemplo:

```
GET /fdsnws/event/1/query?format=geojson&minmagnitude=5&region=Mexico  
HTTP/1.1
```

Host: earthquake.usgs.gov

Esta solicitud pide datos al servidor sobre, por ejemplo, sismos recientes de magnitud mayor a 5 en México.

4. Recepción y procesamiento de la respuesta: El servidor responde con un archivo JSON que contiene la información solicitada. El ESP8266 puede parsear (analizar y descomponer una estructura de datos o texto para entender su contenido y extraer información útil) este JSON usando librerías como `ArduinoJson`, para extraer datos como la magnitud, ubicación, hora y profundidad de un sismo.
5. Acción local o remota: Con la información obtenida, el ESP8266 puede tomar decisiones como encender un LED, activar una sirena, enviar una alerta por Telegram o almacenar la información en una base de datos en la nube.

¿Cómo funciona la memoria del ESP8266?

El ESP8266 cuenta con tres tipos principales de memoria:

1. Memoria Flash: Esta es la memoria no volátil (externa al chip) donde se almacena el firmware (el código que cargamos desde el entorno de desarrollo). Su tamaño varía según el modelo; en el caso del M-01, típicamente es de 1 MB, aunque puede llegar hasta 4 MB.
2. SRAM (RAM de uso general): El ESP8266 tiene unos 80 kB de SRAM utilizables por el usuario, pero en la práctica una parte está reservada para el sistema. Esta memoria se usa para variables temporales, buffers, estructuras de datos, etc. Debido a su tamaño limitado, es esencial usarla con cuidado, especialmente al manipular cadenas o archivos JSON grandes.
3. RTC Memory: Una pequeña porción de RAM que se mantiene activa durante el deep sleep, útil para guardar estados temporales o contadores entre ciclos de sueño profundo.

Dado que la RAM es escasa, es recomendable trabajar con estructuras optimizadas, usar `StaticJsonDocument` en vez de `DynamicJsonDocument`, y liberar buffers después de su uso.

¿Qué es el USGS?

El USGS (United States Geological Survey) es una agencia científica del gobierno de Estados Unidos que estudia el paisaje, los recursos y los riesgos naturales del país, incluyendo terremotos, volcanes, inundaciones y otros fenómenos geológicos. A través de su sistema de datos en tiempo real, el USGS proporciona una API pública (<https://earthquake.usgs.gov/fdsnws/event/1/>) que permite acceder a información actualizada sobre eventos sísmicos a nivel mundial.

Por medio de esta API, cualquier dispositivo, incluido el ESP8266, puede consultar datos específicos sobre sismos usando filtros como magnitud mínima, zona geográfica, fecha, etc. Esto convierte al USGS en una fuente confiable y valiosa para proyectos de monitoreo sísmico o sistemas de alerta temprana implementados en plataformas como el ESP8266.

Funcionamiento del ESP8266 en la práctica con una API como la del USG

El ESP8266 no solo se limita a enviar solicitudes HTTP; también maneja protocolos seguros como HTTPS, lo cual es importante cuando se comunica con servicios públicos como el del USGS, que requieren una conexión segura. Para ello, el ESP8266 utiliza `WiFiClientSecure`, que permite establecer conexiones TLS/SSL. Sin embargo, esto implica un mayor uso de RAM y procesamiento, por lo que es fundamental optimizar cada byte disponible.

Cuando el ESP8266 se conecta a la API del USGS para obtener información sísmica, como, por ejemplo:

```
https://earthquake.usgs.gov/fdsnws/event/1/query?format=geojson&minmagnitude=5&region=Mexico
```

lo que está haciendo es solicitar un documento GeoJSON. Este archivo contiene una estructura de datos que incluye una lista de eventos sísmicos recientes, cada uno descrito mediante un objeto con propiedades como magnitud (`mag`), lugar (`place`), hora (`time`), profundidad (`depth`), coordenadas (`geometry.coordinates`), entre otras. Para procesar este contenido en un microcontrolador con recursos limitados, se utiliza un objeto JSON estático cuidadosamente dimensionado, como:

```
StaticJsonDocument<1024> doc;
```

Aquí, el número 1024 representa los bytes asignados para decodificar el JSON. Si el documento excede esta cantidad, se descartan datos o la lectura falla, lo cual puede generar errores silenciosos si no se maneja con cuidado. Por eso se recomienda hacer pruebas con datos reales y ajustar el tamaño según la respuesta esperada.

Las nociones anteriores ayudarán a entender parte fundamental del código trabajado en el microcontrolador, el cual es la mayor parte del proyecto.

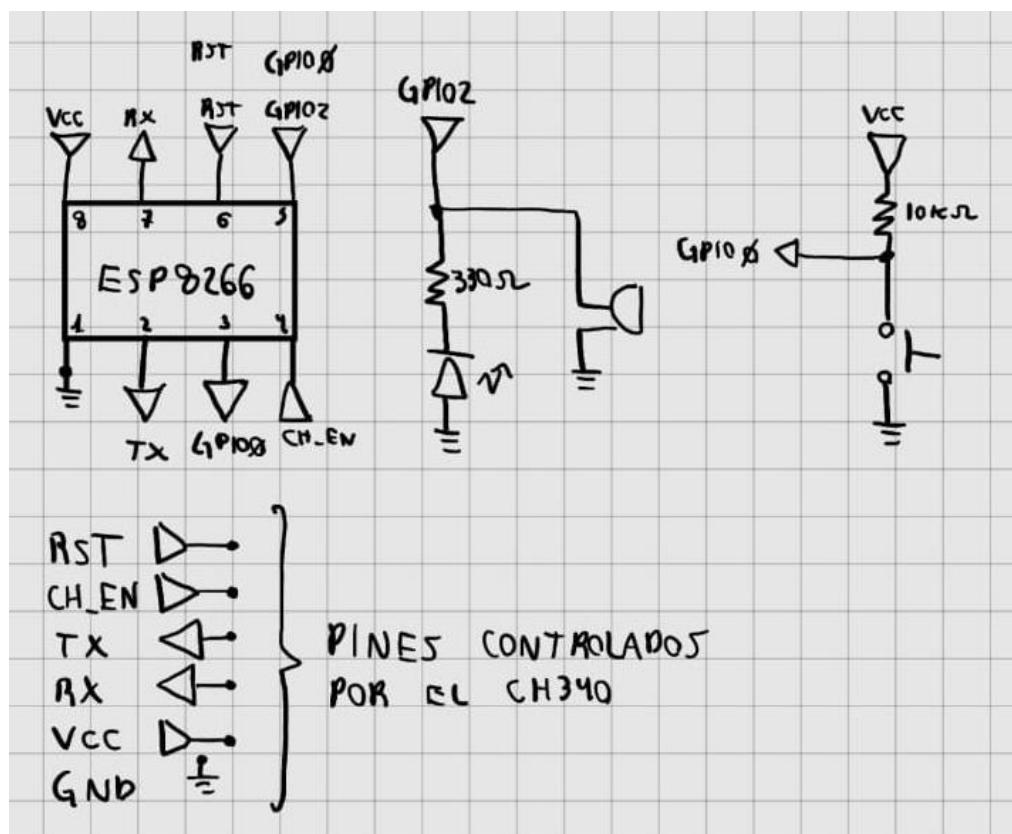
Equipo y material utilizado

- ❖ 1 programador de ESP8266 CH340 (Se puede utilizar otro programador o incluso un Arduino)
- ❖ 1 ESP8266 M01
- ❖ 1 Protoboard
- ❖ 1 Buzzer de 5V
- ❖ 1 Led Rojo
- ❖ 1 Push botón
- ❖ 1 Resistencia de 330 Ohm
- ❖ 1 Resistencia de 1 K Ohm
- ❖ Jumpers y alambre para conexiones
- ❖ Software: Arduino IDE

Procedimiento

Conexiones para la alarma física y programar el microcontrolador

El diagrama que utilizamos para una alarma modesta y la comunicación serial con una laptop por medio del IDE de Arduino, y que también sirvió para programar, es el siguiente:



Importante destacar la resistencia pullup que va de GPIO2 a los componentes del LED y el buzzer, ya que esta configuración se especifica en el código fuente.

Elaboración y configuración del Bot de telegram

A continuación, se muestra cómo crear un bot de Telegram, obtener su token, conocer el ID de usuario que usaremos posteriormente en el código fuente.

Para que el ESP8266 pueda enviar mensajes de alerta a Telegram, primero se necesita tener un bot que actúe como "mensajero":

1. Abrimos una conversación con BotFather `@BotFather (el bot oficial de Telegram para crear y administrar otros bots)

Escribimos: */start*

/newbot

2. Configuramos el nombre del bot `Alarma Sísmica ITSE` y el nombre de usuario del bot, que debe terminar en `bot`, como, por ejemplo: `AlarmaSismicaITSEbot`

3. Si el nombre de usuario está disponible, generará un token de autenticación, copiamos el token del bot y lo guardamos.

El token luce como: 123456789:AAFakeTokenForDemoPurposesOnly999xyz. Este token es único y sirve como contraseña del bot.

El ESP8266 necesita saber a qué usuario enviar los mensajes. Para eso debemos conocer el ID numérico de Telegram.

1. En Telegram, buscamos este bot: `@userinfobot`

Iniciamos el chat y escribimos */start*

Responderá con la información de usuario, incluyendo una línea como esta:

Your Telegram ID: XXXXXXXX

Ese lo copiamos y pagamos en el código así:

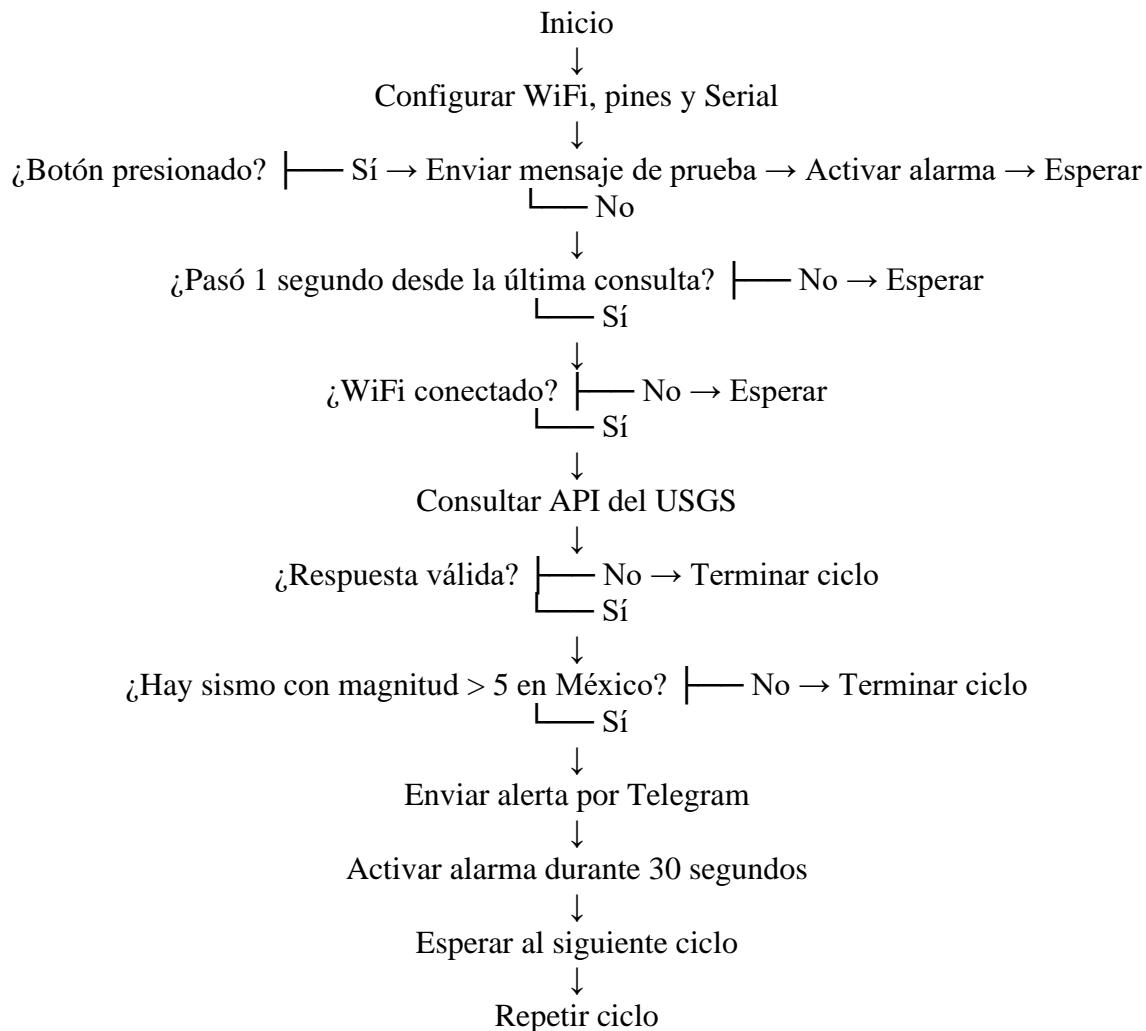
```
const String chatIds[] = {  
    "XXXXXX"  
};
```

Esto permite que el bot envíe mensajes de alerta directamente a tu cuenta.

También podemos agregar más IDs si queremos que otras personas reciban la alerta, simplemente separamos los IDs con comas dentro del arreglo.

Descripción del código fuente

El programa desarrollado para este sistema de alerta sísmica fue escrito en C++ utilizando el entorno de Arduino, siguiendo la lógica del siguiente diagrama de flujo:



El ciclo fuente completo es el siguiente:

```

//Alerta sísmica para eventos en México con magnitud mayor a 5
//La alerta manda un mensaje a telegram y enciende una alarma

#include <ESP8266WiFi.h>
#include <WiFiClientSecure.h>
#include <ESP8266HTTPClient.h>
#include <ArduinoJson.h>

//WiFi
const char* ssid = "Migus";
const char* password = "ITSE123456789";
    
```

Aleta sísmica

```
//Token del bot de Telegram (https://t.me/AlarmaSismicaITSEbot)
const String botToken = "123456789:AAFakeTokenForDemoPurposesOnly999xyz
";

//IDs de los chats (https://t.me/userinfobot)
const String chatIds[] = {
    "XXXXXXX" //Agregamos otro y ponemos ","
};
const int numChats = sizeof(chatIds) / sizeof(chatIds[0]);

//Pines
const int botonPin = 2;    //GPIO2 (PIN5)
const int alarmaPin = 0;   //GPIO0 (PIN3)

//Variables
unsigned long ultimoCheck = 0;
unsigned long intervalo = 1000; //1 segundo (Checa el USGS)

void setup() {
    Serial.begin(9600);

    pinMode(botonPin, INPUT_PULLUP); //El botón de prueba está en pull-up en el
    circuito
    pinMode(alarmaPin, OUTPUT);
    digitalWrite(alarmaPin, LOW);

    //Mensaje serial
    Serial.println("Conectando a WiFi...");
    WiFi.begin(ssid, password);

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

    Serial.println("\nWiFi conectado!");
    Serial.print("Dirección IP: ");
    Serial.println(WiFi.localIP());
}

void loop() {
    if(digitalRead(botonPin) == LOW) {
        Serial.println("Botón presionado, enviando alerta de prueba");
        enviarAlerta("⚠ Mensaje de prueba activada desde ITSE");
        activarAlarma();
        delay(500);
    }
}
```

Aleta sísmica

```
}

if (millis() - ultimoCheck > intervalo && WiFi.status() == WL_CONNECTED) {
    ultimoCheck = millis();
    verificarSismo();
}
}

//Consultar la API de sismos del USGS
void verificarSismo() {
    Serial.println("Consultando sismos... ");

    WiFiClientSecure wifiClient;
    wifiClient.setInsecure();
    HTTPClient http;

    String url = "https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_hour.geojson";
    http.begin(wifiClient, url);

    int httpCode = http.GET();

    if (httpCode == HTTP_CODE_OK) {
        String payload = http.getString();
        StaticJsonDocument<4096> doc; // Usamos StaticJsonDocument para evitar el
        uso de memoria dinámica
        DeserializationError error = deserializeJson(doc, payload);

        if (!error) {
            JSONArray features = doc["features"];

            for (JsonObject sismo : features) {
                float magnitud = sismo["properties"]["mag"];
                const char* lugar = sismo["properties"]["place"];

                if (magnitud > 5.0 && strstr(lugar, "Mexico") != NULL) {
                    String mensaje = "⚠ Sismo en México\nMagnitud: " + String(magnitud) +
                    "\nLugar: " + String(lugar);
                    Serial.println("Sismo detectado, enviando alerta...");
                    enviarAlerta(mensaje);
                    activarAlarma();
                    break;
                }
            }
        } else {
            Serial.print("Error al leer JSON: ");
        }
    }
}
```

```
        Serial.println(error.c_str());
    }
} else {
    Serial.print("Error HTTP: ");
    Serial.println(httpCode);
}

http.end();
limpiarMemoria(); // Limpia todo después de cada consulta
}

//Activar la alarma
void activarAlarma() {
    Serial.println("ALERTA ACTIVADA");

    unsigned long inicio = millis();
    while (millis() - inicio < 30000) { //30 segundos de alarma
        digitalWrite(alarmaPin, HIGH);
        delay(100); //ON 100 ms
        digitalWrite(alarmaPin, LOW);
        delay(100); //OFF APAGADO 100 ms
    }

    Serial.println("ALERTA FINALIZADA");
}

//Enviar una alerta a Telegram usando POST
void enviarAlerta(String mensaje) {
    for (int i = 0; i < numChats; i++) {
        WiFiClientSecure client;
        client.setInsecure();
        HTTPClient http;

        String url = "https://api.telegram.org/bot" + botToken + "/sendMessage";

        http.begin(client, url);
        http.addHeader("Content-Type", "application/x-www-form-urlencoded");

        String postData = "chat_id=" + chatIds[i] + "&text=" + urlencode(mensaje);

        int httpCode = http.POST(postData);

        if (httpCode == HTTP_CODE_OK) {
            Serial.println("Mensaje enviado a Telegram.");
        } else {
            Serial.print("Error al enviar mensaje: ");
            Serial.println(httpCode);
        }
    }
}
```

```

        Serial.println(http.getString());
    }

    http.end();
    delay(100);
}
limpiarMemoria(); //Limpia después de mandar cada alerta
}

//Codificar URL (emojis)
String urlencode(String str) {
    String encoded = "";
    char c;
    char code0, code1;

    for (int i = 0; i < str.length(); i++) {
        c = str.charAt(i);
        if (isalnum(c)) {
            encoded += c;
        } else if (c == ' ') {
            encoded += '+';
        } else {
            code1 = (c & 0xf) + '0';
            if ((c & 0xf) > 9) code1 = (c & 0xf) - 10 + 'A';
            code0 = ((c >> 4) & 0xf) + '0';
            if (((c >> 4) & 0xf) > 9) code0 = ((c >> 4) & 0xf) - 10 + 'A';
            encoded += '%';
            encoded += code0;
            encoded += code1;
        }
    }
    return encoded;
}

```

Examinando parte por parte el código, recalcamos que en el fragmento:

```

const char* ssid = "Migus";
const char* password = "ITSE123456789";
const String botToken = "8056164262:AAGOr9i8X6waPttWvFR7HWmXiyFoHSnwffw";
const String chatIds[] = { "1757783621" };
const int numChats = sizeof(chatIds) / sizeof(chatIds[0]);

```

Se especifican el nombre de la red Wi-Fi (SSID) y su contraseña para conectar el ESP8266 a internet, el token del bot de Telegram, necesario para autenticar las solicitudes hacia la API

de Telegram. Un arreglo de IDs de chat, es decir, los usuarios que recibirán la alerta, y se calcula automáticamente el número de usuarios (`numChats`) con `sizeof()`.

Las variables de control para establecer un intervalo entre consultas a la API del USGS está dado en:

```
unsigned long ultimoCheck = 0;  
unsigned long intervalo = 1000; //1 segundo
```

Donde el microcontrolador verifica si ha pasado más de un segundo desde la última verificación antes de hacer una nueva solicitud.

La función `setup()` se ejecuta una sola vez al iniciar el ESP8266. Aquí se configura la comunicación serial, define los pines como entrada o salida, conecta a la red Wi-Fi e imprime en el monitor serie información útil para depuración o corrección de errores.

La función `loop()` se ejecuta de forma continua. Se compone de dos partes:

- ❖ Si se presiona el botón (`LOW` en configuración `INPUT_PULLUP`), se envía un mensaje de prueba y se activa la alarma.
- ❖ Si ha pasado más de un segundo (`intervalo`) desde la última revisión, se consulta la API de sismos mediante `verificarSismo()`.

La función `verificarSismo()` es la más importante. Su propósito es consultar la API del USGS (https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_hour.geojson). En donde se crea una conexión HTTPS y se obtiene una respuesta en formato GeoJSON, que contiene una lista de sismos recientes. Se deserializa (es decir, se convierte en estructura interna) usando `ArduinoJson` para buscar en ese arreglo (`features`) si hay algún sismo con magnitud mayor a 5 en México. Si se encuentra un sismo con esas características, se construye un mensaje y se llama a `enviarAlerta()` y `activarAlarma()`, donde:

Función `activarAlarma()` activa la alarma durante 30 segundos, alternando el encendido y apagado cada 100 milisegundos para producir un parpadeo o sonido intermitente. También se realiza una solicitud POST a la API de Telegram, usando la URL <https://api.telegram.org/bot<TOKEN>/sendMessage>, el mensaje se codifica (usando `urlencode()`) para caracteres especiales, como emojis, y se envía al chat correspondiente. Telegram responde con un código HTTP que el ESP8266 imprime en la consola para confirmar si fue exitoso.

`urlencode(String str)` convierte los caracteres especiales previamente mencionados en un formato válido para URL, esto evita errores al enviar texto complejo en la URL.

`limpiarMemoria()` fue parte fundamental del código ya que soluciona una limitación del hardware, y es que limpia procesos internos del sistema y le da tiempo al microcontrolador para liberar recursos. Es por ello que también usamos la clase *StaticJsonDocument<4096>* que aparece reserva exactamente 4 KB de memoria de forma estática, es decir, desde antes de ejecutar el programa para los JSON de la API del USGS.

Conclusión

El ESP8266 M-01, pese a su tamaño y precio reducido, es una herramienta poderosa para el desarrollo de aplicaciones conectadas, especialmente en entornos de monitoreo y alerta como los sistemas sísmicos. Su capacidad de realizar solicitudes HTTP/HTTPS, procesar información en tiempo real desde APIs como la del USGS y tomar decisiones locales lo convierte en un aliado ideal en proyectos de bajo costo, pero alto impacto. Sin embargo, su eficiencia depende en gran medida de una programación cuidadosa y una gestión meticulosa de su limitada memoria. Usado con criterio, el ESP8266 puede ser el núcleo de soluciones IoT que salvan vidas, automatizan procesos y acercan la ciencia a la comunidad.

El código fuente realiza de forma automática un proceso de monitoreo sísmico que típicamente requeriría múltiples dispositivos o software; además de presentar la ventaja de ser modular y clara que permite una fácil expansión (por ejemplo, añadir más chats, cambiar la magnitud mínima, o agregar sensores físicos).

Sin duda alguna, este es un ejemplo completo y funcional de un sistema embebido conectado a la nube, aplicable en contextos reales de protección civil.

Referencias

- ❖ Telegram. (n.d.). *Telegram Bot API*. Recuperado de <https://core.telegram.org/bots/api>
- ❖ Arduino. (n.d.). *ESP8266WiFi Library*. Recuperado de <https://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/readme.html>
- ❖ U.S. Geological Survey. (n.d.). *USGS Earthquake Hazards Program*. Recuperado de <https://earthquake.usgs.gov/>
- ❖ Arduino. (n.d.). *ArduinoJson Library*. Recuperado de <https://arduinojson.org/>
- ❖ Espressif Systems. (n.d.). *ESP8266EX Datasheet*. Recuperado de https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf
- ❖ Telegram. (n.d.). *BotFather*. Recuperado de <https://t.me/botfather>
- ❖ Telegram. (n.d.). *User Info Bot*. Recuperado de <https://t.me/userinfobot>