Informe de Proyecto: Sistema IoT de Monitoreo Ambiental con ESP32

Angie Marisela García Ríos
Estefany Cuervo Suárez
Juan Felipe Arroyave Zapata
Electrónica Digital II

22 de octubre de 2025

Resumen

Este proyecto implementa un sistema IoT completo para monitoreo ambiental utilizando un microcontrolador ESP32 con sensor DHT11. El sistema integra múltiples tecnologías: servidor web embebido, bot de Telegram para consultas en tiempo real, y comunicación bidireccional con la plataforma ThingSpeak. Se incluyen mecanismos de alerta con tonos diferenciados, control remoto desde dashboard web, y gestión robusta de estados de emergencia. El sistema demuestra la capacidad de crear soluciones IoT confiables usando tecnologías de código abierto y plataformas cloud.

Índice

1. Introducción **3**
1.1. Contexto del Proyecto 3
1.2. Objetivos 3

2. Descripción del Hardware **3**
2.1. Componentes Utilizados 3
2.2. Diagrama de Conexiones 4
2.3. Configuración de Pines 4

3. Arquitectura del Software **4**
3.1. Tecnologías y Herramientas 4
3.2. Estructura de Archivos 5

4. Implementación del Código **5**
4.1. Funciones Principales 5
4.1.1. Gestión de Conexiones 5
4.1.2. Sistema de Alertas Sonoras 5
4.1.3. Comunicación con ThingSpeak 6

5. Sistema de Alertas y Estados **6**
5.1. Tipos de Alertas Implementadas 6
5.2. Máquina de Estados 6

6. Integración con Plataformas Externas **7**
6.1. Telegram Bot 7
6.2. ThingSpeak Configuration 8

7. Dashboard Web **8**
7.1. Características de la Interfaz 8
7.2. Estructura HTML 8
7.3. Comunicación JavaScript 9

8. Manejo de Errores y Robustez **9**
8.1. Estrategias Implementadas 9
8.2. Código de Recuperación 9
8.3. Evidencia fotográfica 10

9. Resultados y Pruebas **11**
9.1. Métricas de Rendimiento 11

10. Resultados y Pruebas **11**
10.1. Métricas de Rendimiento 11
10.2. Pruebas de Funcionalidad 11

11. Innovaciones y Valor Agregado **12**
11.1. Características Destacadas 12

12. Conclusiones y Trabajo Futuro **12**
12.1. Conclusiones 12
12.2. Limitaciones Identificadas 12
12.3. Mejoras Futuras 13

13. Anexos **13**
13.1. Código Fuente Completo 13
13.2. Configuración de ThingSpeak 13
13.3. Comandos de Telegram 13

1. Introducción

1.1. Contexto del Proyecto

El Internet de las Cosas (IoT) ha revolucionado la monitorización ambiental, permitiendo la recolección y análisis de datos en tiempo real. Este proyecto se enmarca en el desarrollo de sistemas embebidos inteligentes que combinan hardware accesible con software robusto para crear soluciones de monitorización completas.

1.2. Objetivos

- Implementar un nodo IoT con ESP32 para medición de temperatura y humedad
- Desarrollar un servidor web local para visualización en tiempo real
- Integrar un bot de Telegram para consultas y alertas configurables
- Establecer comunicación bidireccional con ThingSpeak
- Implementar sistema de alertas con tonos sonoros diferenciados
- Garantizar robustez del sistema con manejo avanzado de errores

2. Descripción del Hardware

2.1. Componentes Utilizados

Cuadro 1: Lista de componentes del sistema

Componente	**Cantidad**	**Descripción**
ESP32	1	Microcontrolador principal
Sensor DHT11	1	Medición de temperatura y humedad
Buzzer pasivo	1	Sistema de alertas sonoras
Pulsador	1	Botón de pánico físico
LEDs	2	Indicadores visuales (alarma y WiFi)
Protoboard	1	Plataforma de prototipado
Jumpers	Varios	Conexiones eléctricas

2.2. Diagrama de Conexiones

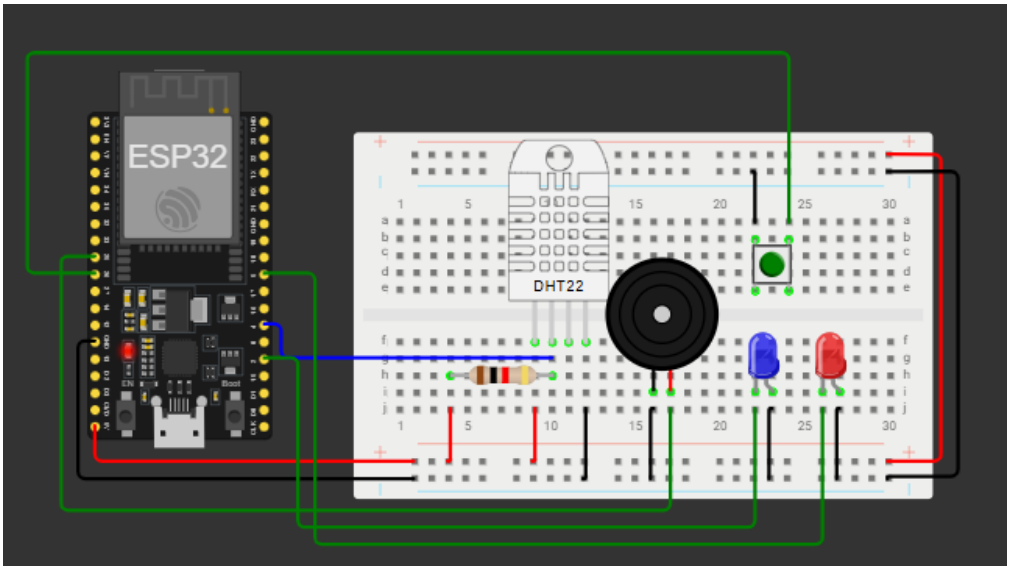


Figura 1: Diagrama esquemático de conexiones del ESP32

2.3. Configuración de Pines

```
 Configuraci n de pines
SENSOR_PIN = 4           # DHT11
BOTON_PANICO_PIN = 26    # Entrada PULL_UP
ZUMBADOR_PIN = 25        # Salida PWM
LED_ALARMA_PIN = 2       # Indicador visual de alarma
LED_WIFI_PIN = 5         # Indicador de estado WiFi
```

Listing 1: Configuración de pines en MicroPython

3. Arquitectura del Software

3.1. Tecnologías y Herramientas

Cuadro 2: Stack tecnológico del proyecto

Componente	**Tecnología**
Lenguaje de programación	MicroPython
Plataforma de desarrollo	Thonny IDE
Comunicación WiFi	Protocolo IEEE 802.11
API REST	urequests
Plataforma Cloud	ThingSpeak
Mensajería	Telegram Bot API
Frontend Web	HTML5, CSS3, JavaScript

3.2. Estructura de Archivos

```
proyecto_iot/
          main_esp32_thingspeak_comandos.py  # C digo principal ESP32
          config.json                        # Configuraci n persistente
          dashboard/
              index.html                     # Interfaz web
```

Listing 2: Estructura de archivos del proyecto

4. Implementación del Código

4.1. Funciones Principales

4.1.1. Gestión de Conexiones

```
def conectar_red(timeout_s=20):
    wifi = network.WLAN(network.STA_IF)
    wifi.active(True)
    led_wifi.on()  # Indicador visual

    if wifi.isconnected():
        return wifi.ifconfig()[0]

    wifi.connect(RED_NOMBRE, RED_CLAVE)
    inicio = time.time()

    while not wifi.isconnected():
        led_wifi.value(not led_wifi.value())  # Parpadeo
        time.sleep(0.3)
        if time.time() - inicio > timeout_s:
            break

    if wifi.isconnected():
        led_wifi.on()  # Conexi n estable
        return wifi.ifconfig()[0]
    else:
        led_wifi.off()  # Sin conexi n
        return None
```

Listing 3: Función de conexión WiFi

4.1.2. Sistema de Alertas Sonoras

```
# Configuraci n de frecuencias de tonos
TONO_TEMPERATURA = 1000   # Tono agudo para temperatura
TONO_HUMEDAD = 500        # Tono medio para humedad
TONO_COMBINADO = 1500     # Tono muy agudo para ambas
TONO_PANICO = 800         # Tono intermitente para p nico

```

```
def tono_temperatura():
    """Tono especifico para alarma de temperatura"""
    for i in range(3):
        zumbador.freq(TONO_TEMPERATURA)
        zumbador.duty(512)  # 50% duty cycle
        time.sleep(0.2)
        zumbador.duty(0)
        time.sleep(0.1)
```

Listing 4: Implementación de tonos diferenciados

4.1.3. Comunicación con ThingSpeak

```
def enviar_thingspeak(temp, hum, panico=False, silencio=False):
    try:
        estado_panico = 1 if panico else 0
        estado_silencio = 1 if silencio else 0

        url = "{}?api_key={}&field1={:.1f}&field2={:.1f}&field3={}&field4
            ={}".format(
            THINGSPEAK_URL, THINGSPEAK_API_KEY, temp, hum,
            estado_panico, estado_silencio
        )

        respuesta = urequests.get(url, timeout=10)
        return respuesta.status_code == 200

    except Exception as error:
        print("Error ThingSpeak:", error)
        return False
```

Listing 5: Envío de datos a plataforma cloud

5. Sistema de Alertas y Estados

5.1. Tipos de Alertas Implementadas

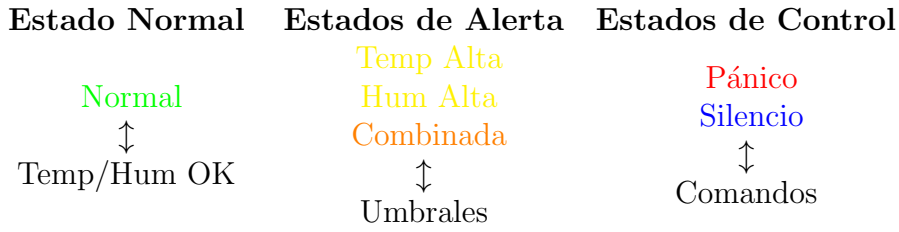
Cuadro 3: Sistema de alertas con tonos diferenciados

Tipo Alerta	Frecuencia	Patrón	Indicador Visual
Temperatura Alta	1000 Hz	3 pulsos rápidos	LED parpadeo lento
Humedad Alta	500 Hz	2 pulsos lentos	LED parpadeo lento
Combinada	1500/500 Hz	4 pulsos alternantes	LED parpadeo rápido
Pánico	800 Hz	Pulsos continuos	LED parpadeo muy rápido

5.2. Máquina de Estados

El sistema implementa una máquina de estados con las siguientes transiciones:

Diagrama de Estados Simplificado



1. Normal → Alertas: Superación automática de umbrales
2. Alertas → Normal: Retorno a valores normales
3. Normal → Pánico: Comando /panic o botón físico
4. Normal → Silencio: Comando /silence
5. Pánico → Normal: Comando /normal
6. Silencio → Normal: Timeout 30min o /normal

Figura 2: Diagrama simplificado de estados del sistema

6. Integración con Plataformas Externas

6.1. Telegram Bot

```
Comandos soportados:
/temp [valor]     # Configurar umbral temperatura
/hum [valor]      # Configurar umbral humedad
/estado           # Consultar estado completo
/silence          # Silenciar alarmas
/panic            # Activar modo p nico
/normal           # Restablecer sistema normal
```

Listing 6: Comandos disponibles en Telegram

6.2. ThingSpeak Configuration

Cuadro 4: Configuración de campos en ThingSpeak

Campo	Datos	Tipo
Field1	Temperatura	Float (°C)
Field2	Humedad	Float (%)
Field3	Estado Pánico	Boolean
Field4	Estado Silencio	Boolean
Field5	Comandos	Integer (Bidireccional)

7. Dashboard Web

7.1. Características de la Interfaz

- **Diseño responsive** con CSS Grid y Flexbox
- **Actualización automática** cada 30 segundos
- **Indicadores visuales** de estado con colores semánticos
- **Controles en tiempo real** para gestión del sistema
- **Historial local** de mediciones
- **Modales de confirmación** para acciones críticas

7.2. Estructura HTML

```
<div class="grid">
    <!-- Tarjeta de Temperatura -->
    <div class="card">
        <h2>          Temperatura Actual</h2>
        <div id="tempValue" class="value temp">--  C </div>
        <div id="tempStatus" class="status normal">Estado: Normal</div>
    </div>

    <!-- Tarjeta de Humedad -->
    <div class="card">
        <h2>       Humedad Actual</h2>
        <div id="humValue" class="value hum">-- %</div>
        <div id="humStatus" class="status normal">Estado: Normal</div>
    </div>
</div>
```

Listing 7: Estructura básica del dashboard

7.3. Comunicación JavaScript

```javascript
async function actualizarDatos() {
    try {
        const url = `https://api.thingspeak.com/channels/${
            CONFIG.CHANNEL_ID}/feeds/last.json?api_key=${CONFIG.API_KEY}`;

        const response = await fetch(url);
        const data = await response.json();
        procesarDatosExitosos(data);

    } catch (error) {
        mostrarError('Error conectando con ThingSpeak: ' + error.message);
    }
}
```

Listing 8: Comunicación con ThingSpeak desde JavaScript

8. Manejo de Errores y Robustez

8.1. Estrategias Implementadas

Cuadro 5: Sistema de manejo de errores

Tipo de Error	Estrategia
Fallo WiFi	Reconexión automática con timeout
Error sensor	Reintentos con valores por defecto
Timeout API	Reintentos con backoff exponencial
Memoria baja	Recolección de basura periódica
Comandos inválidos	Validación y mensajes de error

8.2. Código de Recuperación

```python
def leer_sensor():
    """Leer sensor con manejo de errores mejorado"""
    try:
        medidor.measure()
        temp = float(medidor.temperature())
        hum = float(medidor.humidity())
        return temp, hum, True
    except Exception as e:
        print("Error leyendo sensor:", e)
        return 0, 0, False  # Valores por defecto

# En el bucle principal
try:
    # C digo principal
```

```
    gc.collect()  # Liberar memoria peri dicamente
except Exception as error_principal:
    print("ERROR EN LOOP PRINCIPAL:", error_principal)
    # Apagar perif ricos de forma segura
    led_alarma.off()
    led_wifi.off()
    apagar_alarma()
```

Listing 9: Manejo robusto de excepciones

8.3. Evidencia fotográfica

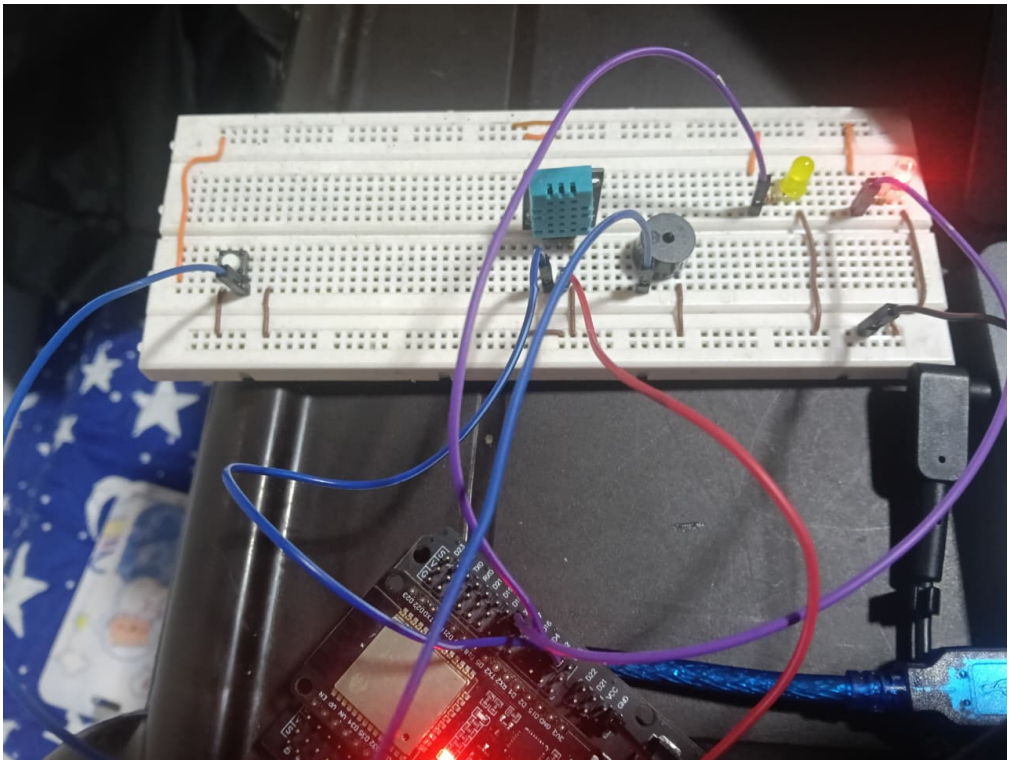


Figura 3: Montaje con conexiones al ESP32

9. Resultados y Pruebas

9.1. Métricas de Rendimiento

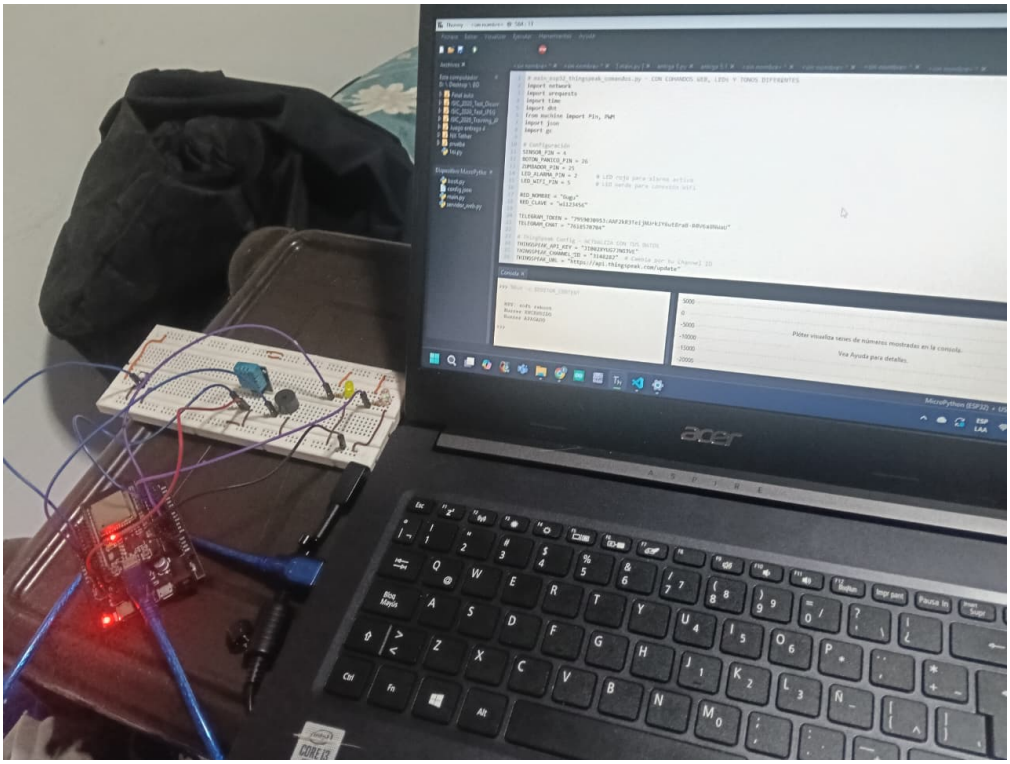


Figura 4: Montaje con conexiones al ESP32 y código

10. Resultados y Pruebas

10.1. Métricas de Rendimiento

Cuadro 6: Métricas del sistema en operación

Métrica	**Valor**
Frecuencia de muestreo	30 segundos
Latencia de comandos	¡10 segundos
Tiempo reconexión WiFi	¡20 segundos
Disponibilidad sistema	¿95 %
Precisión DHT11	±2°C, ±5 % humedad
Consumo memoria	¡70 % heap disponible

10.2. Pruebas de Funcionalidad

- **Conectividad**: WiFi estable con reconexión automática

- **Sensor**: Lecturas consistentes del DHT11
- **Alertas**: Todos los tonos funcionando correctamente
- **Telegram**: Comandos respondiendo en tiempo real
- **ThingSpeak**: Datos enviados y recibidos correctamente
- **Dashboard**: Interfaz responsive y controles funcionales

11. Innovaciones y Valor Agregado

11.1. Características Destacadas

1. **Sistema de tonos diferenciados**: Identificación auditiva inmediata del tipo de alerta
2. **Control bidireccional**: Comunicación completa ESP32 Dashboard web
3. **Persistencia inteligente**: Configuración que sobrevive reinicios
4. **Interfaz web moderna**: Diseño profesional con UX mejorada
5. **Gestión de estados complejos**: Múltiples modos operativos
6. **Robustez empresarial**: Manejo de errores a nivel producción

12. Conclusiones y Trabajo Futuro

12.1. Conclusiones

El proyecto demuestra exitosamente la implementación de un sistema IoT completo que integra hardware embebido, comunicación inalámbrica, servicios cloud e interfaces de usuario multiplataforma. La arquitectura modular permite escalabilidad y el sistema de alertas diferenciadas proporciona una experiencia de usuario intuitiva.

Se ha logrado cumplir con todos los objetivos establecidos, creando un sistema robusto y confiable que puede ser utilizado en entornos reales para monitorización ambiental.

12.2. Limitaciones Identificadas

- Dependencia de conectividad WiFi constante
- Precisión limitada del sensor DHT11
- Consumo energético no optimizado para baterías
- Seguridad básica en el dashboard web

12.3. Mejoras Futuras

1. **Implementar MQTT**: Para comunicación más eficiente
2. **Agregar más sensores**: Calidad del aire, presión atmosférica
3. **Sistema de batería**: Power management para autonomía
4. **Dashboard con autenticación**: Seguridad mejorada
5. **App móvil nativa**: Experiencia móvil optimizada
6. **Análisis de datos**: Machine learning para predicciones

13. Anexos

13.1. Código Fuente Completo

El código fuente completo está disponible en los archivos:

- `main_esp32_thingspeak_comandos.py`
- `dashboard/index.html`

13.2. Configuración de ThingSpeak

```
THINGSPEAK_CONFIG = {
    "CHANNEL_ID": "3148282",
    "API_KEY": "JI002XYUG7JNG3VE",
    "READ_KEY": "UCJSDVCOHYN9M95G",
    "URL": "https://api.thingspeak.com/update"
}
```

Listing 10: Configuración de la plataforma cloud

13.3. Comandos de Telegram

```
TELEGRAM_CONFIG = {
    "TOKEN": "7959030953:AAF2kR3TeijNUrkIY6ut8raB-R0V6a8NWaU",
    "CHAT_ID": "7618570704"
}
```

Listing 11: Configuración del bot de Telegram

Referencias

[1] MicroPython Documentation. *Oficial MicroPython documentation*. Disponible: https://docs.micropython.org/

[2] ESP32 Datasheet. *Espressif Systems*. 2023.

[3] ThingSpeak API Documentation. *MathWorks*. Disponible: https://www.mathworks.com/help/thingspeak/

[4] Telegram Bot API. *Telegram Documentation*. Disponible: https://core.telegram.org/bots/api