



UNIVERSIDAD
CATÓLICA
BOLIVIANA
LA PAZ



“Sistema IoT de Prevención y Alerta Temprana de Incendios”

Nombre de Equipo: OnFire

Carrera: Ing. De Sistemas

Semestre: 6to Semestre

Integrantes:

Guzmán Daniela

Imaña Jesús

Carvajal Franz

Leonardini Damaris

Fecha de entrega: 03/11/2024

1.	Introducción.....	1
1.1.	Objetivos del proyecto	1
1.1.1.	Objetivo General	1
1.1.2.	Objetivos Específicos	1
1.2.	Importancia	1
2.	Diseño de la Base de Datos	2
2.1.	Descripción de la estructura:	2
2.2.	Campos y tipos de datos:	2
3.	Desarrollo del Script en PHP.....	4
3.1.	Inserción de datos:.....	4
3.2.	Lectura de datos:	13
3.3.	Validaciones y seguridad:	13
4.	Desarrollo del Script en MicroPython	14
4.1.	Conexión y configuración:	14
4.2.	Inserción de datos:.....	14
4.3.	Manejo de excepciones y reconexión:	17
5.	Desarrollo del Script en Python para ABM.....	17
5.1.	Descripción general de las operaciones ABM:.....	17
5.1.1.	ABM para datos y sensores	17
5.1.2.	ABM de Usuarios	18
5.2.	Código fuente del script en Python:	18
5.2.1.	ABM para datos y sensores	18
5.2.2.	ABM de Usuarios	20
5.3.	Interfaz de usuario:	21
5.3.1.	ABM para datos	21
5.3.2.	ABM de Usuarios	22

5.4.	Manejo de errores y validaciones:.....	25
5.4.1.	ABM para datos	25
5.4.2.	ABM de Usuarios	25
6.	Desarrollo del Dashboard en Python para Visualización en Tiempo Real	26
6.1.	Diseño del Dashboard:	26
6.2.	Código fuente del dashboard:	26
6.3.	Interactividad y actualización:.....	33
7.	Conclusión y Lecciones Aprendidas.....	34
8.	Anexos	35



1. Introducción

En respuesta a una creciente crisis ambiental en Bolivia, este proyecto busca desarrollar un sistema automatizado para el monitoreo y detección temprana de condiciones de riesgo de incendios en áreas susceptibles. En Bolivia, los chaqueos (una práctica agrícola que implica la quema de vegetación para preparar la tierra) se han extendido en gran medida, causando incendios descontrolados que afectan amplias áreas de bosques y ecosistemas naturales. Esta situación no solo ha generado daños irreparables en la biodiversidad, sino que también ha puesto en grave peligro la vida de las comunidades locales, las cuales a menudo enfrentan los devastadores efectos de estos incendios.

Actualmente, la falta de un sistema de monitoreo continuo y de alertas tempranas ha sido un factor crítico en la rápida propagación de estos incendios. La detección tardía limita la capacidad de respuesta oportuna, lo cual agrava el impacto en el medio ambiente y eleva el riesgo de pérdidas humanas y materiales. Este proyecto surge como una solución integral que utiliza sensores para recopilar datos de variables ambientales clave, como el dióxido de carbono (CO₂), la humedad, el humo y la temperatura. Estos datos son analizados para identificar condiciones de riesgo que permitan alertar rápidamente a las autoridades y a las comunidades afectadas, promoviendo una intervención rápida y efectiva.

1.1. Objetivos del proyecto

1.1.1. Objetivo General

Desarrollar un sistema automatizado de monitoreo y alerta temprana que detecte condiciones de riesgo de incendio en áreas afectadas por chaqueos en Bolivia, facilitando una respuesta rápida y efectiva que contribuya a la preservación del medio ambiente y a la seguridad de las comunidades.

1.1.2. Objetivos Específicos

- Implementar una red de sensores para la captura de datos en tiempo real de variables críticas como CO₂, temperatura, humedad y niveles de humo, que son indicadores clave para la identificación de incendios potenciales.
- Desarrollar una base de datos que almacene y organice los datos ambientales, permitiendo su análisis y consulta histórica para la detección de patrones que preceden a la propagación de incendios.
- Crear una interfaz de usuario amigable y funcional para que los operadores puedan visualizar en tiempo real los datos y recibir alertas automáticas ante condiciones críticas que puedan derivar en un incendio.

1.2. Importancia

Este proyecto tiene una importancia crítica para la protección ambiental y la seguridad pública en Bolivia. Los incendios provocados por los chaqueos representan una amenaza devastadora para la biodiversidad, afectando tanto a los ecosistemas como a las especies endémicas, y ponen en riesgo directo a las comunidades locales. La implementación de un sistema automatizado de monitoreo y detección temprana ofrece una solución innovadora y eficaz para abordar esta problemática.



Al permitir una respuesta temprana y eficaz ante condiciones de riesgo, el sistema ayudará a minimizar los impactos negativos de los incendios, reduciendo así la pérdida de recursos naturales y la afectación a la salud y la vida de los habitantes. Además, este proyecto se alinea con los esfuerzos globales por mitigar el cambio climático y preservar los recursos naturales, lo que posiciona a Bolivia como un país proactivo en la protección de su medio ambiente y en la gestión sostenible de sus prácticas agrícolas.

El sistema no solo beneficia a las comunidades locales, sino que también optimiza los recursos de respuesta de las autoridades, al dirigir los esfuerzos hacia áreas específicas y priorizar la seguridad y protección de la biodiversidad. Este proyecto, por lo tanto, representa una herramienta estratégica para Bolivia en su lucha contra el cambio climático y la protección de su entorno natural, contribuyendo de manera directa al bienestar de las generaciones presentes y futuras.

2. Diseño de la Base de Datos

2.1. Descripción de la estructura:

La base de datos onfireDB almacena información de los sensores que tiene nuestro proyecto tales como CO₂, humedad, humo y temperatura en un entorno monitorizado y también gestiona datos de usuarios y ubicaciones de microcontroladores. Su diseño facilita la administración y análisis de esta toma de datos mediante tablas dedicadas para cada sensor, una tabla para los usuarios, otra para los microcontroladores, y una para sus ubicaciones. Esta estructura organizada permite el acceso seguro y eficiente a la información, siendo el núcleo de un sistema de monitoreo en tiempo real.

Tablas de la base de datos onfireDB:

- **co2:** Almacena las mediciones de dióxido de carbono, con campos de fecha, hora y valor.
- **humedad:** Registra los niveles de humedad, incluyendo fecha, hora y valor.
- **humo:** Contiene datos sobre la concentración de humo en el ambiente.
- **temperatura:** Guarda las mediciones de temperatura, también con fecha, hora y valor.
- **usuarios:** Gestiona la información de los usuarios que tienen acceso al sistema, con nombre de usuario y contraseña.
- **microcontroladores:** Almacena información sobre los microcontroladores que toman las mediciones ambientales.
- **ubicaciones:** Describe las ubicaciones de cada microcontrolador.

Cada tabla de cada sensor incluye un registro único de medición junto con campos de fecha y hora, permitiendo un seguimiento cronológico detallado de cada sensor.

2.2. Campos y tipos de datos:

Cada tabla tiene los siguientes campos, configurados con tipos de datos específicos y restricciones para asegurar la integridad y precisión de los datos:



Tabla: co2

id: INT – Identificador único, PRIMARY KEY, AUTO_INCREMENT.

fecha: DATE – Fecha de la medición, NOT NULL.

hora: TIME – Hora de la medición, NOT NULL.

valor: DOUBLE(30,3) – Concentración de CO2 con tres decimales de precisión, NOT NULL.

microcontroladores_id: INT – Identificador de microcontrolador, FOREIGN KEY, NOT NULL.

Tabla: humedad

id: INT – Identificador único, PRIMARY KEY, AUTO_INCREMENT.

fecha: DATE – Fecha de la medición, NOT NULL.

hora: TIME – Hora de la medición, NOT NULL.

valor: DOUBLE(30,3) – Porcentaje de humedad, con tres decimales de precisión, NOT NULL.

microcontroladores_id: INT – Identificador de microcontrolador, FOREIGN KEY, NOT NULL.

Tabla: humo

id: INT – Identificador único, PRIMARY KEY, AUTO_INCREMENT.

fecha: DATE – Fecha de la medición, NOT NULL.

hora: TIME – Hora de la medición, NOT NULL.

valor: DOUBLE(30,3) – Concentración de humo en el ambiente, con tres decimales de precisión, NOT NULL.

microcontroladores_id: INT – Identificador de microcontrolador, FOREIGN KEY, NOT NULL.

Tabla: temperatura

id: INT – Identificador único, PRIMARY KEY, AUTO_INCREMENT.

fecha: DATE – Fecha de la medición, NOT NULL.

hora: TIME – Hora de la medición, NOT NULL.

valor: DOUBLE(30,3) – Valor de temperatura en grados, con tres decimales de precisión, NOT NULL.

microcontroladores_id: INT – Identificador de microcontrolador, FOREIGN KEY, NOT NULL.

Tabla: microcontroladores

id: INT – Identificador único, PRIMARY KEY, AUTO_INCREMENT.

nombre: VARCHAR(50) – Nombre del microcontrolador, NOT NULL.



ubicaciones_id: INT – Identificador de ubicación, FOREIGN KEY, NOT NULL.

Tabla: ubicaciones

id: INT – Identificador único, PRIMARY KEY, AUTO_INCREMENT.

descripcion: VARCHAR(200) – Descripción de la ubicación del microcontrolador, NOT NULL.

Tabla: usuarios

id: INT – Identificador único, PRIMARY KEY, AUTO_INCREMENT.

usuario: VARCHAR(50) – Nombre de usuario, NOT NULL.

password: VARCHAR(30) – Contraseña del usuario, NOT NULL.

Descripción de los campos clave

Campos id (INT, AUTO_INCREMENT): El campo id en cada tabla sirve como clave primaria y está configurado como AUTO_INCREMENT, lo cual asegura un identificador único para cada registro.

Campos fecha (DATE) y hora (TIME): En las tablas de mediciones ambientales, estos campos registran la fecha y hora de cada lectura, facilitando un análisis temporal preciso de cada variable.

Campos valor (DOUBLE(30,3)): Los valores de CO2, humedad, humo y temperatura se almacenan como DOUBLE, con tres decimales de precisión, lo que permite manejar una amplia gama de valores numéricos con la exactitud necesaria para lecturas ambientales.

Campos usuario y password (VARCHAR): En la tabla usuarios, el campo usuario está limitado a 50 caracteres y password a 30 caracteres. Esto asegura que la información del usuario sea gestionada de forma eficiente, manteniendo flexibilidad sin desperdicio de espacio.

3. Desarrollo del Script en PHP

3.1. Inserción de datos:

El código PHP permite insertar datos de diferentes sensores (temperatura, humo, CO2, humedad) en la base de datos onfireBD, en tablas específicas para cada tipo de sensor. A continuación, se presenta el código y una explicación detallada de cada parte:

Código fuente del script en PHP para la inserción de datos:

PHPTemp (inserción de temperatura):

```
<?php  
  
// Configuración de la conexión a la base de datos  
  
$servername = "localhost";  
  
$username = "root";
```



```
$password = "";  
  
$dbname = "onfireBD";  
  
try {  
  
    // Crear conexión a la base de datos  
  
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,  
    $password);  
  
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);  
  
    // Leer el contenido de la solicitud POST  
  
    $json = file_get_contents('php://input');  
  
    $data = json_decode($json, true);  
  
    // Verificar que los datos JSON contengan las claves necesarias  
  
    //proyeccion a futuro  
  
    if (isset($data['Cos_Taylor'])) { // Cambia 'valor' por 'Cos_Taylor'  
  
        // Definir la fecha y la hora actuales  
  
        $fecha = date('Y-m-d');  
  
        $hora = date('H:i:s');  
  
        // Preparar la inserción en la tabla temperatura  
  
        $stmt = $conn->prepare("INSERT INTO temperatura (fecha, hora, valor) VALUES  
        (:fecha, :hora, :valor)");  
  
        $stmt->bindParam(':fecha', $fecha);  
  
        $stmt->bindParam(':hora', $hora);  
  
        $stmt->bindParam(':valor', $data['Cos_Taylor']); // Cambia 'valor' por 'Cos_Taylor'  
  
        // Ejecutar la consulta  
  
        if ($stmt->execute()) {  
  
            echo json_encode(["message" => "Dato de temperatura insertado exitosamente"]);
```



```
 } else {  
  
    echo json_encode(["message" => "Error al insertar el dato de temperatura"]);  
  
}  
  
} else {  
  
    echo json_encode(["message" => "Datos incompletos en el JSON recibido"]);  
  
}  
  
} catch (PDOException $e) {  
  
    echo json_encode(["error" => "Error en la conexión: " . $e->getMessage()]);  
  
}  
  
// Cerrar la conexión  
  
$conn = null;  
  
?>
```

PHPCO2 (inserción de CO2):

```
<?php  
  
// Configuración de la conexión a la base de datos  
  
$servername = "localhost";  
  
$username = "root";  
  
$password = "";  
  
$dbname = "onfireBD";  
  
try {  
  
    // Crear conexión a la base de datos  
  
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,  
    $password);  
  
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);  
  
    <?php
```



```
// Configuración de la conexión a la base de datos

$servername = "localhost";

$username = "root";

$password = "";

$dbname = "onfireBD";

try {

    // Crear conexión a la base de datos

    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
    $password);

    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    // Leer el contenido de la solicitud POST

    $json = file_get_contents('php://input');

    $data = json_decode($json, true);

    // Verificar que los datos JSON contengan las claves necesarias

    if (isset($data['Exp_Taylor'])) {

        $fecha = date('Y-m-d');

        $hora = date('H:i:s');

        // Preparar la inserción en la tabla

        $stmt = $conn->prepare("INSERT INTO co2 (fecha, hora, valor) VALUES (:fecha,
        :hora, :valor)");

        $stmt->bindParam(':fecha', $fecha);

        $stmt->bindParam(':hora', $hora);

        $stmt->bindParam(':valor', $data['Exp_Taylor']); // Cambia 'valor' por 'Exp_Taylor'

        // Ejecutar la consulta

        if ($stmt->execute()) {

            echo json_encode(["message" => "Dato de CO2 insertado exitosamente"]);

        }

    }

}
```



```
    } else {

        echo json_encode(["message" => "Error al insertar el dato de temperatura"]);

    }

} else {

    echo json_encode(["message" => "Datos incompletos en el JSON recibido"]);

}

} catch (PDOException $e) {

    echo json_encode(["error" => "Error en la conexión: " . $e->getMessage()]);

}

// Cerrar la conexión

$conn = null;

?>

// Leer el contenido de la solicitud POST

$json = file_get_contents('php://input');

$data = json_decode($json, true);

// Verificar que los datos JSON contengan las claves necesarias

if (isset($data['Cos_Taylor'])) {

    $fecha = date('Y-m-d');

    $hora = date('H:i:s');

    // Preparar la inserción en la tabla

    $stmt = $conn->prepare("INSERT INTO co2 (fecha, hora, valor) VALUES (:fecha,
:hora, :valor)");

    $stmt->bindParam(':fecha', $fecha);

    $stmt->bindParam(':hora', $hora);

    $stmt->bindParam(':valor', $data['Cos_Taylor']); // Cambia 'valor' por 'Cos_Taylor'

    // Ejecutar la consulta
```



```
if ($stmt->execute()) {  
  
    echo json_encode(["message" => "Dato de temperatura insertado exitosamente"]);  
  
} else {  
  
    echo json_encode(["message" => "Error al insertar el dato de temperatura"]);  
  
}  
  
}  
  
} catch (PDOException $e) {  
  
    echo json_encode(["error" => "Error en la conexión: " . $e->getMessage()]);  
  
}  
  
// Cerrar la conexión  
  
$conn = null;  
  
?>
```

PHPHumedad (inserción de humedad):

```
<?php  
  
// Configuración de la conexión a la base de datos  
  
$servername = "localhost";  
  
$username = "root";  
  
$password = "";  
  
$dbname = "onfireBD";  
  
try {  
  
    // Crear conexión a la base de datos  
  
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,  
    $password);  
  
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```



```
// Leer el contenido de la solicitud POST

$json = file_get_contents('php://input');

$data = json_decode($json, true);

// Verificar que los datos JSON contengan las claves necesarias

if (isset($data['Tan_Taylor'])) { // Cambia 'valor' por 'Tan_Taylor'

    // Definir la fecha y la hora actuales

    $fecha = date('Y-m-d');

    $hora = date('H:i:s');

    // Preparar la inserción en la tabla

    $stmt = $conn->prepare("INSERT INTO humedad (fecha, hora, valor) VALUES

(:fecha, :hora, :valor)");

    $stmt->bindParam(':fecha', $fecha);

    $stmt->bindParam(':hora', $hora);

    $stmt->bindParam(':valor', $data['Tan_Taylor']); // Cambia 'valor' por 'Tan_Taylor'

    // Ejecutar la consulta

    if ($stmt->execute()) {

        echo json_encode(["message" => "Dato de humedad insertado exitosamente"]);

    } else {

        echo json_encode(["message" => "Error al insertar el dato de temperatura"]);

    }

} else {

    echo json_encode(["message" => "Datos incompletos en el JSON recibido"]);

}

} catch (PDOException $e) {

    echo json_encode(["error" => "Error en la conexión: " . $e->getMessage()]);

}
```



// Cerrar la conexión

\$conn = null;

?>

PHPHumo (inserción de Humo):

```
<?php
```

// Configuración de la conexión a la base de datos

\$servername = "localhost";

\$username = "root";

\$password = "";

\$dbname = "onfireBD";

try {

// Crear conexión a la base de datos

```
$conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,  
$password);
```

```
$conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

// Leer el contenido de la solicitud POST

```
$json = file_get_contents('php://input');
```

```
$data = json_decode($json, true);
```

// Verificar que los datos JSON contengan las claves necesarias

```
if (isset($data['Sin_Taylor'])) { // Cambia 'valor' por 'Cos_Taylor'
```

// Definir la fecha y la hora actuales

```
$fecha = date('Y-m-d');
```

```
$hora = date('H:i:s');
```

// Preparar la inserción en la tabla



```
$stmt = $conn->prepare("INSERT INTO humo (fecha, hora, valor) VALUES (:fecha,  
:hora, :valor);  
  
$stmt->bindParam(':fecha', $fecha);  
  
$stmt->bindParam(':hora', $hora);  
  
$stmt->bindParam(':valor', $data['Sin_Taylor']); // Cambia 'valor' por 'Cos_Taylor'  
  
// Ejecutar la consulta  
  
if ($stmt->execute()) {  
  
    echo json_encode(["message" => "Dato de humo insertado exitosamente"]);  
  
} else {  
  
    echo json_encode(["message" => "Error al insertar el dato de temperatura"]);  
  
}  
  
} else {  
  
    echo json_encode(["message" => "Datos incompletos en el JSON recibido"]);  
  
}  
  
} catch (PDOException $e) {  
  
    echo json_encode(["error" => "Error en la conexión: " . $e->getMessage()]);  
  
}  
  
// Cerrar la conexión  
  
$conn = null;  
  
?>
```

Explicación del código:

- **Conexión a la base de datos:** En cada script, se utiliza PDO para establecer la conexión a onfireBD. El manejo de errores se configura con PDO::ERRMODE_EXCEPTION para capturar cualquier problema en la conexión.
- **Lectura de solicitud POST:** El contenido de la solicitud POST se lee usando:



```
$json = file_get_contents('php://input');
$data = json_decode($json, true);
```

- Esto convierte los datos JSON enviados en un array asociativo que facilita su procesamiento.
- **Verificación de datos JSON:** El script verifica que la clave correspondiente esté presente en los datos JSON (como Cos_Taylor para temperatura o Sin_Taylor para humo) para asegurar que se inserten datos válidos.
- **Inserción en la base de datos:** Los valores de fecha, hora y valor se preparan y se insertan en la tabla adecuada mediante una consulta preparada para evitar ataques SQL y mejorar la seguridad.

3.2. Lectura de datos:

Los scripts PHP también pueden leer datos de las tablas y mostrarlos en consola a través de la ejecución de código en Python y el sistema ABM de datos. La lectura de datos se realiza de forma instantánea para monitorear los valores ingresados.

Código fuente del script en PHP para lectura de datos posterior al POST:

```
$json = file_get_contents('php://input');
```

Esta línea lee los datos en formato JSON que fueron enviados al script PHP a través de una solicitud POST. En este contexto, php://input permite acceder directamente al cuerpo crudo de la solicitud, lo cual es útil cuando se están recibiendo datos en JSON (o en cualquier formato que no sea application/x-www-form-urlencoded).

```
$data = json_decode($json, true);
```

Aquí, json_decode() convierte el texto JSON en un array asociativo de PHP. Esto significa que los datos recibidos ahora están disponibles en una estructura que PHP puede manejar fácilmente, y pueden ser accedidos mediante sus claves.

3.3. Validaciones y seguridad:

Se aplican validaciones y medidas de seguridad para garantizar que los datos sean seguros y consistentes al insertarlos o leerlos.

Validaciones:

- Verificación de la existencia de claves necesarias en el JSON antes de la inserción, evitando errores por datos incompletos.
- Validación de que cada valor cumpla con los requisitos de formato esperado.

Medidas de seguridad básicas:

Sanitización de datos: Los datos JSON se manejan de manera cuidadosa para evitar inyecciones SQL y asegurar que solo datos válidos ingresen a la base de datos.

Manejo de errores de conexión: PDO::ERRMODE_EXCEPTION permite capturar errores sin exponer detalles de la base de datos, aumentando la seguridad del sistema.



4. Desarrollo del Script en MicroPython

4.1. Conexión y configuración:

Este script utiliza un ESP32 para enviar datos calculados al servidor. La conexión Wi-Fi es fundamental para permitir que el microcontrolador se comunique con la base de datos o el servidor donde se almacenan los datos. Para mantener las credenciales seguras, se almacenan en un archivo separado secrets.py, que incluye el nombre de la red (SSID) y la contraseña. Este archivo es importado al inicio del código.

Configuración de la conexión Wi-Fi:

```
from secrets import secrets # Archivo con credenciales Wi-Fi

from Wifi_lib import wifi_init # Inicialización de conexión Wi-Fi

wifi_init()
```

La función wifi_init() configura y conecta el ESP32 a la red especificada en secrets.py.

El microcontrolador determina su tipo de tarjeta para configurar los pines y el LED correspondiente. En el caso del ESP32, se utilizan varios pines para manejar los LEDs RGB y los botones. La función detect_board_type() clasifica la tarjeta en función del sistema y nombre de la máquina, lo que permite configurar los pines de manera precisa.

```
class Board:
    # Configuración del tipo de tarjeta
    def detect_board_type(self):
        # Código de detección y asignación del tipo de tarjeta
```

4.2. Inserción de datos:

El objetivo de esta sección es enviar datos generados (cálculos matemáticos) al servidor, utilizando la función send_data(). El script calcula el coseno usando la serie de Taylor, asimismo, de forma respectiva para cada tabla de datos y envía el resultado junto con el valor trigonométrico exacto y un valor de error al servidor.

Cálculo del coseno :

```
def calculate_cos_taylor(x, n):
    result = 1
    sign = -1
    factorial = 1
```



```
power = x * x

for i in range(1, n):

    factorial *= (2 * i) * (2 * i - 1)

    result += sign * power / factorial

    power *= x * x

    sign *= -1

return result
```

Cálculo del exponencial:

```
def calculate_exp_taylor(x, n):

    result = 1 # e^0 = 1

    factorial = 1

    power = 1 # x^0

    for i in range(1, n):

        factorial *= i # Calcula i! para el siguiente término

        power *= x # Aumenta el exponente

        result += power / factorial

    return result
```

Cálculo del seno:

```
def calculate_sin_taylor(x, n):

    result = 0

    sign = 1

    factorial = 1

    power = x # Inicia con x^1

    for i in range(1, n + 1):
```



```
result += sign * power / factorial  
  
power *= x # Aumenta el exponente  
  
factorial *= (2 * i) * (2 * i + 1)  
  
sign *= -1 # Alterna el signo  
  
return result
```

Cálculo de la tangente:

```
def calculate_tan_taylor(x, n):  
  
    sin_taylor = 0  
  
    cos_taylor = 0  
  
    for i in range(n):  
  
        sin_taylor += ((-1) ** i) * (x ** (2 * i + 1)) / math.factorial(2 * i + 1)  
  
        cos_taylor += ((-1) ** i) * (x ** (2 * i)) / math.factorial(2 * i)  
  
    if cos_taylor != 0:  
  
        return sin_taylor / cos_taylor  
  
    else:  
  
        return float('inf') # Si el coseno es 0, la tangente es indefinida
```

De esta forma todas las respectivas funciones calculadas de `calculate_sin_taylor()`, `calculate_cos_taylor()`, `calculate_exp_taylor()`, `calculate_tan_taylor()` estiman el valor utilizando una serie de Taylor, cuyo número de términos es controlado por `NTaylor`.

Envío de datos al servidor:

```
def send_data():  
  
    data = {  
  
        "Cos_Taylor": cos_taylor_value,  
  
        "Cos_Trig": cos_trig_value,
```



```
"Error": error_value,  
"RGB": calculate_rgb(NTaylor),  
"NTaylor": NTaylor  
}  
  
headers = {'Content-Type': 'application/json'}  
  
response = requests.post(url, data=ujson.dumps(data), headers=headers)
```

En este caso, el ESP32 envía un JSON al servidor. La función `requests.post()` realiza la solicitud HTTP POST y transmite los datos en formato JSON. De la misma manera con las funciones correspondientes en cada inserción.

4.3. Manejo de excepciones y reconexión:

- **Captura de Errores de Red:** Uso de `try-except` en el envío de datos para evitar que el programa se detenga por errores de conexión.
- **Reconexión Automática a Wi-Fi:** Si hay una desconexión, `wifi_init()` intenta restablecer la conexión automáticamente al punto de acceso configurado.
- **Reintento de Envío de Datos:** El ESP32 reintenta el envío hasta que se complete exitosamente, controlado por la bandera `data_sent`.

Detalles de la estrategia implementada para gestionar desconexiones o errores en la red, incluyendo el procedimiento de reconexión.

5. Desarrollo del Script en Python para ABM

5.1. Descripción general de las operaciones ABM:

5.1.1. ABM para datos y sensores

Las operaciones de Alta, Baja y Modificación (ABM) se implementan tanto para los **datos de sensores** como para los propios **sensores** en la base de datos **onfireBD**.

- **ABM de Datos:** Estas operaciones permiten gestionar registros individuales capturados por cada sensor (por ejemplo, temperatura, CO2, humedad), almacenando datos como la fecha, hora y valor del sensor.
- **ABM de Sensores:** Este módulo gestiona las tablas que representan cada tipo de sensor. Permite agregar nuevos sensores, modificar la estructura de las tablas de sensores añadiendo columnas, y eliminar sensores junto con todos sus datos.



5.1.2. ABM de Usuarios

Las operaciones ABM (Alta, Baja y Modificación) se aplican a la tabla usuarios de la base de datos MySQL llamada onfireBD.

Alta (creación): Permite al usuario agregar un nuevo registro a la tabla usuarios, capturando el nombre de usuario y la contraseña.

Baja (eliminación): Permite eliminar un usuario existente, identificando cada registro por un ID único.

Modificación: Permite modificar la información de un usuario específico, proporcionando la posibilidad de actualizar tanto el nombre de usuario como la contraseña.

Estas operaciones se manejan mediante consultas SQL (INSERT, DELETE y UPDATE) en la base de datos, conectada a través de mysql.connector.

5.2. Código fuente del script en Python:

5.2.1. ABM para datos y sensores

ABM de Datos:

- Alta (creación) de datos:

```
def guardar_dato():
    valor = entry_valor.get()
    if not valor:
        messagebox.showwarning("Campos incompletos", "Por favor complete todos los campos.")
        return
    query = f"INSERT INTO {sensor} (fecha, hora, valor) VALUES (CURDATE(), CURTIME(), %s)"
    cursor.execute(query, (valor,))
    conn.commit()
    messagebox.showinfo("Éxito", "Dato agregado exitosamente.")
    agregar_win.destroy()
    mostrar_datos_sensor(sensor)
```

- Baja (eliminación) de datos:

```
def eliminar_dato(dato_id):
    if messagebox.askyesno("Confirmar", f"¿Estás seguro de que deseas eliminar este dato? (Id:{dato_id})"):
        cursor.execute(f"DELETE FROM {sensor} WHERE id = %s", (dato_id,))
        conn.commit()
        mostrar_datos_sensor(sensor)
        messagebox.showinfo("Éxito", "Dato eliminado exitosamente.")
```

- Modificación de datos:

```
def guardar_modificacion():
    nuevo_valor = entry_valor.get()
```



```
if not nuevo_valor:  
    messagebox.showwarning("Campos incompletos", "Por favor complete  
    todos los campos.")  
    return  
    query = f"UPDATE {sensor} SET valor=%s WHERE id=%s"  
    cursor.execute(query, (nuevo_valor, dato_id))  
    conn.commit()  
    messagebox.showinfo("Éxito", "Dato modificado exitosamente.")  
    modificar_win.destroy()  
    mostrar_datos_sensor(sensor)
```

ABM de Sensores:

- Alta (creación) de sensores:

```
def crear_sensor():  
    sensor_nombre = simpledialog.askstring("Agregar Sensor", "Ingrese el  
    nombre del nuevo sensor:")  
    if sensor_nombre:  
        cursor.execute(f"""\n            CREATE TABLE {sensor_nombre} (\n                id INT NOT NULL AUTO_INCREMENT,  
                fecha DATE NOT NULL,  
                hora TIME NOT NULL,  
                valor DOUBLE(30,3) NOT NULL,  
                PRIMARY KEY (id)\n            )\n        """)  
        conn.commit()  
        agregar_boton_sensor(sensor_nombre)  
        messagebox.showinfo("Éxito", f"Sensor '{sensor_nombre}' creado  
        exitosamente.")
```

- Baja (eliminación) de sensores:

```
def eliminar_sensor():  
    sensor_nombre = simpledialog.askstring("Eliminar Sensor", "Ingrese el  
    nombre del sensor a eliminar:")  
    if sensor_nombre:  
        cursor.execute(f"DROP TABLE {sensor_nombre}")  
        conn.commit()  
        if sensor_nombre in botones_sensores:  
            botones_sensores[sensor_nombre].destroy()  
            del botones_sensores[sensor_nombre]  
            messagebox.showinfo("Éxito", f"Sensor '{sensor_nombre}' eliminado  
            exitosamente.")
```

- Modificación de sensores:

```
def modificar_sensor():
```



```
sensor_nombre = simpledialog.askstring("Modificar Sensor", "Ingrese el  
nombre del sensor a modificar:")  
nueva_columna = simpledialog.askstring("Modificar Sensor", "Ingrese el  
nombre de la nueva columna (opcional):")  
if nueva_columna:  
    cursor.execute(f"ALTER TABLE {sensor_nombre} ADD COLUMN  
{nueva_columna} DOUBLE(30,3)")  
    conn.commit()  
    messagebox.showinfo("Éxito", f"Columna '{nueva_columna}' añadida al  
sensor '{sensor_nombre}'.")
```

5.2.2. ABM de Usuarios

Alta (creación):

```
def guardar_usuario():  
  
    usuario = entry_usuario.get()  
  
    password = entry_password.get()
```

```
if not usuario or not password:
```

```
    messagebox.showwarning("Campos incompletos", "Por favor complete todos los  
campos.")  
  
    return
```

```
query = "INSERT INTO usuarios (usuario, password) VALUES (%s, %s)"  
  
cursor.execute(query, (usuario, password))  
  
conn.commit()
```

```
messagebox.showinfo("Éxito", "Usuario agregado exitosamente.")  
  
agregar_win.destroy()  
  
mostrar_usuarios()
```

Baja (eliminación):

```
def eliminar_usuario(usuario_id):  
  
    if messagebox.askyesno("Confirmar", f"¿Estás seguro de que deseas eliminar este  
usuario? (Id:{usuario_id})?"):  
  
        cursor.execute("DELETE FROM usuarios WHERE id = %s", (usuario_id,))  
  
        conn.commit()
```



```
mostrar_usuarios()
```

```
messagebox.showinfo("Éxito", "Usuario eliminado exitosamente.")
```

Modificación:

```
def guardar_modificacion():
```

```
    nuevo_usuario = entry_usuario.get()
```

```
    nueva_password = entry_password.get()
```

```
if not nuevo_usuario or not nueva_password:
```

```
    messagebox.showwarning("Campos incompletos", "Por favor complete todos los campos.")
```

```
    return
```

```
query = "UPDATE usuarios SET usuario=%s, password=%s WHERE id=%s"
```

```
cursor.execute(query, (nuevo_usuario, nueva_password, usuario_id))
```

```
conn.commit()
```

```
messagebox.showinfo("Éxito", "Usuario modificado exitosamente.")
```

```
modificar_win.destroy()
```

```
mostrar_usuarios()
```

5.3. Interfaz de usuario:

5.3.1. ABM para datos

Ventana Principal:

La ventana principal posee todas las opciones que tienen ambos ABM.



Las tablas que se manejan para los ABM de datos son iguales a las tablas de ABM de usuarios.

ID	Fecha	Hora	Valor

Agregar Dato

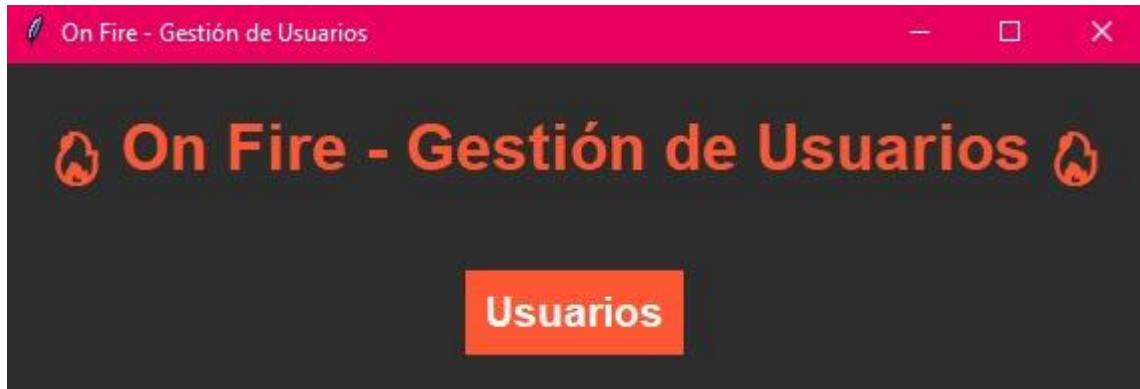
Exportar CSV

5.3.2. ABM de Usuarios

Ventana Principal:

La ventana principal de la aplicación es el punto de entrada donde el usuario puede ver el título y un botón "Usuarios". Al hacer clic en este botón, se llama a la función

mostrar_usuarios, que abre una ventana con la lista de usuarios registrados y las opciones de gestión.



Mostrar Usuarios (mostrar_usuarios):

Cuando se llama a mostrar_usuarios, se crea una nueva ventana llamada usuarios_win que muestra una tabla con la lista de usuarios actuales en la base de datos. Esta ventana tiene una estructura que facilita el acceso a las operaciones de Alta, Baja y Modificación de los usuarios. La tabla (Treeview de ttk) muestra las columnas de ID, Usuario, y Contraseña. Cada fila representa un usuario y contiene dos botones: Modificar y Eliminar.

Los datos en la tabla se obtienen de la base de datos ejecutando la consulta SELECT * FROM usuarios. Cada registro recuperado se agrega como una fila en la tabla.



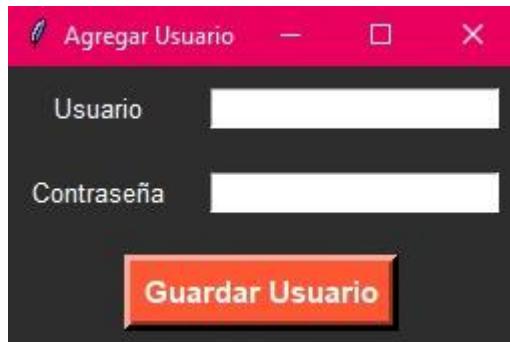
Agregar Usuario (agregar_usuario)

La función agregar_usuario abre una nueva ventana (agregar_win) para permitir al usuario registrar un nuevo usuario en la base de datos.

La ventana contiene dos campos (Entry) donde se ingresa el nombre de usuario y la contraseña.

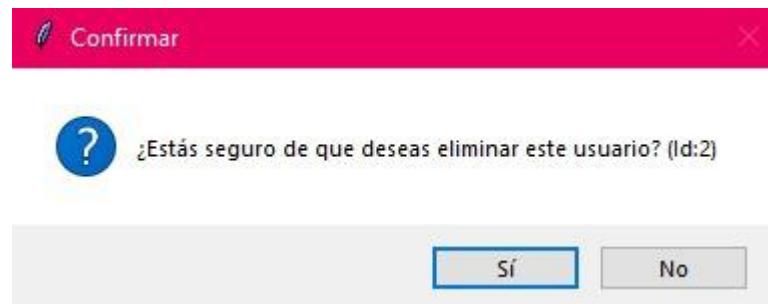


Al hacer clic en el botón "Guardar Usuario", se llama a la función guardar_usuario, que realiza varias acciones: Primero, verifica que los campos de usuario y contraseña no estén vacíos. Si alguno de los dos campos está vacío, se muestra un mensaje de advertencia mediante messagebox.showwarning. Si ambos campos están llenos, se ejecuta la consulta INSERT INTO usuarios (usuario, password) VALUES (%s, %s) para añadir el nuevo usuario en la tabla usuarios. Luego de insertar correctamente el nuevo registro, muestra un mensaje de éxito y cierra la ventana, refrescando la lista de usuarios con mostrar_usuarios.



Eliminar Usuario (eliminar_usuario):

En cada fila de la tabla de usuarios, se muestra un botón "Eliminar" que permite al usuario eliminar el registro seleccionado. Al presionar el botón "Eliminar", se llama a la función eliminar_usuario, que solicita una confirmación al usuario con un cuadro de diálogo (messagebox.askyesno). Esto asegura que la eliminación de un usuario sea intencional. Si el usuario confirma, se ejecuta una consulta DELETE FROM usuarios WHERE id = %s que elimina el registro de la base de datos. Después de eliminar el registro, se recarga la lista de usuarios llamando a mostrar_usuarios para reflejar el cambio. Finalmente, se muestra un mensaje de éxito.



Modificar Usuario (modificar_usuario):

Al presionar el botón "Modificar" en la tabla de usuarios, se abre una ventana (modificar_win) para actualizar el nombre y la contraseña del usuario seleccionado.

Al abrir la ventana de modificación, la función modificar_usuario ejecuta una consulta SQL (SELECT * FROM usuarios WHERE id = %s) para obtener los datos actuales del usuario que se desea modificar. Estos datos se insertan en los campos de entrada para permitir su edición.

La ventana muestra un formulario similar al de agregar usuario, con campos para el nombre de usuario y la contraseña. Estos campos están prellenados con los datos actuales del usuario. Al hacer clic en el botón "Guardar cambios", se llama a la función guardar_modificacion, que sigue varios pasos: Se verifica que el nombre de usuario y la contraseña no estén vacíos. En caso contrario, muestra un mensaje de advertencia para evitar guardar registros incompletos. Si ambos campos están llenos, se ejecuta la consulta UPDATE usuarios SET usuario=%s, password=%s WHERE id=%s para actualizar el registro en la tabla usuarios. Después de realizar la actualización, muestra un mensaje de éxito, cierra la ventana y recarga la lista de usuarios llamando a mostrar_usuarios para reflejar el cambio.



5.4. Manejo de errores y validaciones:

5.4.1. ABM para datos

Incluye validaciones de campos vacíos, confirmaciones antes de eliminación y manejo de errores de conexión a la base de datos, aplicables a ambos módulos de ABM, muchas de estas validaciones funcional igual que para ABM de usuarios.

5.4.2. ABM de Usuarios

Validación de campos vacíos en el formulario de Agregar y Modificar Usuario:

Antes de agregar un nuevo usuario o modificar uno existente, se verifica que tanto el campo de nombre de usuario como el de contraseña no estén vacíos. Si alguno de los campos está vacío, la aplicación muestra un mensaje de advertencia (messagebox.showwarning("Campos incompletos", "Por favor complete todos los campos.")) y cancela la operación.

Por ejemplo, si un usuario intenta guardar un nuevo registro sin proporcionar un nombre de usuario o contraseña, la operación se detiene, evitando la inserción de registros incompletos en la base de datos.

Confirmación antes de eliminar un usuario:

Antes de proceder con la eliminación de un usuario, la aplicación solicita confirmación al usuario mediante un cuadro de diálogo (messagebox.askyesno). Esto permite asegurarse de que la acción es intencional y evita eliminaciones accidentales.



Por ejemplo, si un usuario selecciona "No" en el cuadro de confirmación, la operación de eliminación se cancela, y el registro permanece intacto.

Validación de la existencia del registro antes de eliminar o modificar:

Al intentar eliminar o modificar un usuario, el sistema verifica que el registro aún existe en la base de datos. Esto es importante en aplicaciones donde múltiples usuarios podrían estar accediendo a los mismos datos o cuando un registro pudo haber sido eliminado en una sesión previa.

Por ejemplo, si un usuario intenta eliminar un registro que ya no existe, la aplicación podría ejecutar una consulta de verificación previa (SELECT * FROM usuarios WHERE id = %s). Si la consulta no devuelve resultados, la aplicación muestra un mensaje de error como messagebox.showerror("Error", "El usuario que intenta eliminar ya no existe.") y cancela la operación.

Similarmente, al abrir la ventana de modificación, la aplicación ejecuta una consulta para cargar los datos del usuario (SELECT * FROM usuarios WHERE id = %s). Si la consulta no devuelve datos (indicando que el usuario no existe), la ventana de modificación no se abrirá y se mostrará un mensaje de error.

Manejo de errores de conexión a la base de datos:

Durante la conexión a MySQL, es posible que ocurran errores de conexión (por ejemplo, si la base de datos no está activa o las credenciales son incorrectas). El sistema podría incluir un try-except alrededor de la conexión para capturar estos errores, mostrando un mensaje claro al usuario.

Por ejemplo, si la conexión a la base de datos falla, se podría capturar el error específico (mysql.connector.Error) y mostrar messagebox.showerror("Error de conexión", "No se pudo conectar a la base de datos. Verifique la conexión e intente nuevamente.").

6. Desarrollo del Dashboard en Python para Visualización en Tiempo Real

6.1. Diseño del Dashboard:

Este dashboard está diseñado para visualizar datos de temperatura, humedad, humo y CEO2 en tiempo real, específicamente enfocado en mostrar tendencias y variaciones horarias. Su estructura incluye gráficos que representan la variación de temperatura en distintos momentos del día, con componentes interactivos que permiten a los usuarios analizar datos específicos o seleccionar rangos de tiempo en la cual se usan librerías como pandas, mysql y matplotlib para graficar

6.2. Código fuente del dashboard:

Temperatura



```
import pandas as pd
import mysql.connector
import matplotlib.pyplot as plt

# Configuración de la base de datos
db_config = {
    'host': 'localhost',
    'user': 'root',
    'password': '',
    'database': 'onfirebd'
}

# Conectar a la base de datos MySQL
conn = mysql.connector.connect(**db_config)

# Consulta SQL para leer todos los datos de la tabla temperatura
query = "SELECT fecha, hora, valor FROM temperatura"
data = pd.read_sql(query, conn)

# Cerrar la conexión
conn.close()

# Convertir fecha y hora en un solo campo de tipo datetime especificando el formato
data['fecha'] = pd.to_datetime(data['fecha'], format='%Y-%m-%d', errors='coerce')
data['hora'] = pd.to_timedelta(data['hora'])
data['fecha_hora'] = data['fecha'] + data['hora']

# Eliminar filas con fechas que no pudieron ser convertidas
data = data.dropna(subset=['fecha_hora'])
```



```
# Ordenar los datos por fecha y hora en caso de que estén desordenados
```

```
data = data.sort_values(by='fecha_hora')
```

```
# Graficar los datos de temperatura
```

```
plt.figure(figsize=(12, 8))
```

```
plt.plot(data['fecha_hora'], data['valor'], label='Temperatura', color='blue')
```

```
# Configuración del gráfico
```

```
plt.title('Evolución de la Temperatura a lo largo del Tiempo')
```

```
plt.xlabel('Fecha y Hora')
```

```
plt.ylabel('Valor de Temperatura')
```

```
plt.legend()
```

```
plt.grid(True)
```

```
# Mostrar el gráfico
```

```
plt.show()
```

Humedad

```
import pandas as pd
```

```
import mysql.connector
```

```
import matplotlib.pyplot as plt
```

```
# Configuración de la base de datos
```

```
db_config = {
```

```
    'host': 'localhost',
```

```
    'user': 'root',
```

```
    'password': '',
```

```
    'database': 'onfirebd'
```



}

```
# Conectar a la base de datos MySQL
conn = mysql.connector.connect(**db_config)

# Consulta SQL para leer todos los datos de la tabla humedad
query = "SELECT fecha, hora, valor FROM humedad"
data = pd.read_sql(query, conn)

# Cerrar la conexión
conn.close()

# Convertir fecha y hora en un solo campo de tipo datetime especificando el formato
data['fecha'] = pd.to_datetime(data['fecha'], format='%Y-%m-%d', errors='coerce')
data['hora'] = pd.to_timedelta(data['hora'])
data['fecha_hora'] = data['fecha'] + data['hora']

# Eliminar filas con fechas que no pudieron ser convertidas
data = data.dropna(subset=['fecha_hora'])

# Ordenar los datos por fecha y hora en caso de que estén desordenados
data = data.sort_values(by='fecha_hora')

# Graficar los datos de humedad
plt.figure(figsize=(12, 8))
plt.plot(data['fecha_hora'], data['valor'], label='Humedad', color='green')

# Configuración del gráfico
plt.title('Evolución de la Humedad a lo largo del Tiempo')
```



```
plt.xlabel('Fecha y Hora')
plt.ylabel('Valor de Humedad (%)')
plt.legend()
plt.grid(True)
```

```
# Mostrar el gráfico
plt.show()
```

Humo

```
import pandas as pd
import mysql.connector
import matplotlib.pyplot as plt

# Configuración de la base de datos
```

```
db_config = {
    'host': 'localhost',
    'user': 'root',
    'password': '',
    'database': 'onfirebd'
}
```

```
# Conectar a la base de datos MySQL
conn = mysql.connector.connect(**db_config)
```

```
# Consulta SQL para leer todos los datos de la tabla humo
query = "SELECT fecha, hora, valor FROM humo"
data = pd.read_sql(query, conn)
```



```
# Cerrar la conexión
```

```
conn.close()
```

```
# Convertir fecha y hora en un solo campo de tipo datetime especificando el formato
```

```
data['fecha'] = pd.to_datetime(data['fecha'], format='%Y-%m-%d', errors='coerce')
```

```
data['hora'] = pd.to_timedelta(data['hora'])
```

```
data['fecha_hora'] = data['fecha'] + data['hora']
```

```
# Eliminar filas con fechas que no pudieron ser convertidas
```

```
data = data.dropna(subset=['fecha_hora'])
```

```
# Ordenar los datos por fecha y hora en caso de que estén desordenados
```

```
data = data.sort_values(by='fecha_hora')
```

```
# Graficar los datos de humo
```

```
plt.figure(figsize=(12, 8))
```

```
plt.plot(data['fecha_hora'], data['valor'], label='Concentración de Humo', color='purple')
```

```
# Configuración del gráfico
```

```
plt.title('Evolución de la Concentración de Humo a lo largo del Tiempo')
```

```
plt.xlabel('Fecha y Hora')
```

```
plt.ylabel('Valor de Concentración de Humo')
```

```
plt.legend()
```

```
plt.grid(True)
```

```
# Mostrar el gráfico
```

```
plt.show()
```



CEO2

```
import pandas as pd
import mysql.connector
import matplotlib.pyplot as plt

# Configuración de la base de datos
db_config = {
    'host': 'localhost',
    'user': 'root',
    'password': '',
    'database': 'onfirebd'
}

# Conectar a la base de datos MySQL
conn = mysql.connector.connect(**db_config)

# Consulta SQL para leer todos los datos de la tabla co2
query = "SELECT fecha, hora, valor FROM co2"
data = pd.read_sql(query, conn)

# Cerrar la conexión
conn.close()

# Convertir fecha y hora en un solo campo de tipo datetime especificando el formato
data['fecha'] = pd.to_datetime(data['fecha'], format='%Y-%m-%d', errors='coerce')
data['hora'] = pd.to_timedelta(data['hora'])
data['fecha_hora'] = data['fecha'] + data['hora']
```



```
# Eliminar filas con fechas que no pudieron ser convertidas
data = data.dropna(subset=['fecha_hora'])

# Ordenar los datos por fecha y hora en caso de que estén desordenados
data = data.sort_values(by='fecha_hora')

# Graficar los datos de CO2
plt.figure(figsize=(12, 8))
plt.plot(data['fecha_hora'], data['valor'], label='Concentración de CO2', color='brown')

# Configuración del gráfico
plt.title('Evolución de la Concentración de CO2 a lo largo del Tiempo')
plt.xlabel('Fecha y Hora')
plt.ylabel('Valor de CO2 (ppm)')
plt.legend()
plt.grid(True)

# Mostrar el gráfico
plt.show()
```

6.3. Interactividad y actualización:

El dashboard incluye interactividad y actualización en tiempo real, permitiendo a los usuarios seleccionar rangos de fechas y aplicar filtros de valores para enfocar la visualización en datos específicos, como niveles elevados de CO2 o períodos determinados. La actualización automática asegura que los datos reflejen el estado más reciente, sin necesidad de recargar la página, mientras que el servidor gestiona las solicitudes de manera eficiente para mantener el rendimiento, incluso con varios usuarios simultáneos. Este enfoque brinda una experiencia dinámica y práctica para el monitoreo continuo de las condiciones registradas por los sensores.



7. Conclusión y Lecciones Aprendidas

En este proyecto de Sistema IoT de Prevención y Alerta Temprana de Incendios, se lograron varios avances importantes, incluyendo el desarrollo para la captura de datos a través de sensores para monitoreo en tiempo real de variables críticas como CO₂, temperatura, humedad y niveles de humo, que momentáneamente fueron adaptados a distintas funciones de Taylor.

Con estos logros se permiten a futuro del proyecto, la detección temprana de riesgos de incendio, promoviendo una respuesta más rápida y efectiva por parte de las autoridades.

Se cumplieron los objetivos de diseño e implementación de una base de datos estructurada que organiza y almacena la información de los sensores de manera eficaz, logrando manipular el microcontrolador. De igual forma el desarrollo de un dashboard para la visualización de datos en tiempo real, facilitando la toma de decisiones.

Asimismo, la implementación de operaciones de Alta, Baja y Modificación (ABM) tanto para la gestión de datos de sensores como para los usuarios del sistema, permitiendo:

- Gestión de datos de sensores: Se diseñaron y configuraron módulos para agregar, modificar y eliminar registros individuales capturados por los sensores (como CO₂, temperatura, humedad y humo). Esto facilitó la actualización y el mantenimiento de la base de datos de manera ágil y organizada.
- Administración de usuarios: Las operaciones ABM también se aplicaron a la gestión de usuarios del sistema, permitiendo registrar nuevos usuarios, modificar sus credenciales y eliminar registros obsoletos. Esto contribuyó a la seguridad y control de acceso en el sistema, garantizando que solo usuarios autorizados puedan monitorear y administrar los datos.

La implementación de estos módulos de ABM demostró ser fundamental para la adaptabilidad y eficiencia del sistema, permitiendo una administración eficaz tanto de la información de sensores como de los usuarios.

Resolución de Desafíos y Errores

Durante el desarrollo, se enfrentaron desafíos técnicos en la configuración de los scripts, particularmente en la conexión del microcontrolador ESP32 a la red y en la transmisión de datos al servidor. También se resolvieron errores en el chip y en el código, aplicando técnicas de manejo de excepciones y reconexión automática para garantizar una comunicación estable y continua.

Reflexión sobre el Aprendizaje

La implementación de este sistema permitió al equipo profundizar en conocimientos sobre bases de datos, programación en MicroPython y PHP, y manejo de datos en tiempo real. A través de



este proceso, el equipo adquirió habilidades para resolver problemas de integración de hardware con software, lo cual será fundamental en futuros proyectos de IoT y monitoreo ambiental.

8. Anexos

login.py

```
import tkinter as tk

from tkinter import messagebox
from tkinter import font
import mysql.connector

# Conexión a la base de datos MySQL
conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="",
    database="onfireBD"
)
cursor = conn.cursor()

# Función para verificar el inicio de sesión
def login():
    username = entry_username.get()
    password = entry_password.get()

    if not username or not password:
        messagebox.showwarning("Campos incompletos", "Por favor complete todos los campos.")
    return
```



```
query = "SELECT * FROM usuarios WHERE usuario = %s AND password = %s"
cursor.execute(query, (username, password))
result = cursor.fetchone()

if result:
    messagebox.showinfo("Inicio de sesión exitoso", f"Bienvenido, {username}!")
else:
    messagebox.showerror("Error de inicio de sesión", "Usuario o contraseña incorrectos.")

# Configuración de la ventana de inicio de sesión
root = tk.Tk()
root.title("On Fire - Inicio de Sesión")
root.geometry("400x400")
root.configure(bg="#2d2d2d") # Color de fondo oscuro para un estilo moderno

# Título estilizado
title_font = font.Font(family="Helvetica", size=24, weight="bold")
title_label = tk.Label(root, text="🔥 On Fire 🔥", font=title_font, fg="#FF5733", bg="#2d2d2d")
title_label.pack(pady=20)

# Marco contenedor de los campos
frame = tk.Frame(root, bg="#404040", padx=20, pady=20, relief="ridge", bd=2)
frame.pack(pady=20)

# Estilo de etiquetas y campos de entrada
lbl_font = font.Font(family="Helvetica", size=12)
entry_font = font.Font(family="Helvetica", size=10)

lbl_username = tk.Label(frame, text="Usuario:", font=lbl_font, fg="white", bg="#404040")
```



```
lbl_username.grid(row=0, column=0, padx=10, pady=10)
entry_username = tk.Entry(frame, font=entry_font, width=20)
entry_username.grid(row=0, column=1, padx=10, pady=10)

lbl_password = tk.Label(frame, text="Contraseña:", font=lbl_font, fg="white", bg="#404040")
lbl_password.grid(row=1, column=0, padx=10, pady=10)
entry_password = tk.Entry(frame, show="*", font=entry_font, width=20)
entry_password.grid(row=1, column=1, padx=10, pady=10)

# Botón para iniciar sesión estilizado
btn_login = tk.Button(
    frame, text="Iniciar sesión", font=lbl_font, fg="white", bg="#FF5733",
    width=15, height=1, command=login, activebackground="#C70039", relief="flat"
)
btn_login.grid(row=2, column=0, columnspan=2, pady=20)

root.mainloop()
```



usersABM.py

```
import tkinter as tk

from tkinter import ttk, messagebox, font
import mysql.connector

# Conectar a la base de datos MySQL
conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="",
    database="onfireBD"
)
cursor = conn.cursor()
```



```
# Variables globales para ventanas abiertas
usuarios_win = None
modificar_win = None
agregar_win = None

# Función para abrir la ventana y agregar un nuevo usuario
def agregar_usuario():
    global agregar_win
    if agregar_win is not None:
        agregar_win.destroy()

    agregar_win = tk.Toplevel(root)
    agregar_win.title("Agregar Usuario")
    agregar_win.configure(bg="#2d2d2d")

# Crear campos del formulario
lbl_usuario = tk.Label(agregar_win, text="Usuario", font=('Helvetica', 10), fg="white",
bg="#2d2d2d")
lbl_usuario.grid(row=0, column=0, padx=10, pady=10)
entry_usuario = tk.Entry(agregar_win, font=('Helvetica', 10), width=20)
entry_usuario.grid(row=0, column=1, padx=10, pady=10)

lbl_password = tk.Label(agregar_win, text="Contraseña", font=('Helvetica', 10), fg="white",
bg="#2d2d2d")
lbl_password.grid(row=1, column=0, padx=10, pady=10)
entry_password = tk.Entry(agregar_win, show="*", font=('Helvetica', 10), width=20)
entry_password.grid(row=1, column=1, padx=10, pady=10)

def guardar_usuario():
```



```
usuario = entry_usuario.get()
password = entry_password.get()

if not usuario or not password:
    messagebox.showwarning("Campos incompletos", "Por favor complete todos los
campos.")

    return

query = "INSERT INTO usuarios (usuario, password) VALUES (%s, %s)"
cursor.execute(query, (usuario, password))
conn.commit()

messagebox.showinfo("Éxito", "Usuario agregado exitosamente.")
agregar_win.destroy()
mostrar_usuarios()

btn_guardar = tk.Button(agregar_win, text="Guardar Usuario", command=guardar_usuario,
bg="#FF5733", fg="white", font=('Arial', 11, 'bold'), relief="raised", bd=5)
btn_guardar.grid(row=2, column=0, columnspan=2, pady=10)

# Función para mostrar la tabla de usuarios

def mostrar_usuarios():
    global usuarios_win
    if usuarios_win is not None:
        usuarios_win.destroy()

    usuarios_win = tk.Toplevel(root)
    usuarios_win.title("Usuarios")
    usuarios_win.configure(bg="#2d2d2d")
```



```
frame = tk.Frame(usuarios_win, bg="#404040", padx=10, pady=10)
frame.pack(expand=True, fill='both')

btn_agregar = tk.Button(usuarios_win, text="Agregar Usuario", command=agregar_usuario,
bg="#FF5733", fg="white", font=('Arial', 11, 'bold'), relief="raised", bd=5)
btn_agregar.pack(pady=10)

cols = ("ID", "Usuario", "Contraseña")
tree = ttk.Treeview(frame, columns=cols, show='headings')

for col in cols:
    tree.heading(col, text=col)

tree.pack(side=tk.LEFT, expand=True, fill='both')

scrollbar = ttk.Scrollbar(frame, orient="vertical", command=tree.yview)
scrollbar.pack(side=tk.RIGHT, fill='y')
tree.configure(yscroll=scrollbar.set)

cursor.execute("SELECT * FROM usuarios")
usuarios = cursor.fetchall()

def eliminar_usuario(usuario_id):
    if messagebox.askyesno("Confirmar", f"¿Estás seguro de que deseas eliminar este
usuario? (Id:{usuario_id})"):
        cursor.execute("DELETE FROM usuarios WHERE id = %s", (usuario_id,))
        conn.commit()
        mostrar_usuarios()
```



```
messagebox.showinfo("Éxito", "Usuario eliminado exitosamente.")
```

```
def modificar_usuario(usuario_id):  
    global modificar_win  
  
    if modificar_win is not None:  
        modificar_win.destroy()  
  
    modificar_win = tk.Toplevel(root)  
    modificar_win.title("Modificar Usuario")  
    modificar_win.configure(bg="#2d2d2d")  
  
    cursor.execute("SELECT * FROM usuarios WHERE id = %s", (usuario_id,))  
    usuario = cursor.fetchone()  
  
    lbl_usuario = tk.Label(modificar_win, text="Usuario", font=('Helvetica', 10), fg="white",  
    bg="#2d2d2d")  
    lbl_usuario.grid(row=0, column=0, padx=10, pady=10)  
    entry_usuario = tk.Entry(modificar_win, font=('Helvetica', 10), width=20)  
    entry_usuario.insert(0, usuario[1])  
    entry_usuario.grid(row=0, column=1, padx=10, pady=10)  
  
    lbl_password = tk.Label(modificar_win, text="Contraseña", font=('Helvetica', 10),  
    fg="white", bg="#2d2d2d")  
    lbl_password.grid(row=1, column=0, padx=10, pady=10)  
    entry_password = tk.Entry(modificar_win, show="*", font=('Helvetica', 10), width=20)  
    entry_password.insert(0, usuario[2])  
    entry_password.grid(row=1, column=1, padx=10, pady=10)  
  
    def guardar_modificacion():
```



```
nuevo_usuario = entry_usuario.get()
nueva_password = entry_password.get()

if not nuevo_usuario or not nueva_password:
    messagebox.showwarning("Campos incompletos", "Por favor complete todos los
campos.")

return

query = "UPDATE usuarios SET usuario=%s, password=%s WHERE id=%s"
cursor.execute(query, (nuevo_usuario, nueva_password, usuario_id))
conn.commit()

messagebox.showinfo("Éxito", "Usuario modificado exitosamente.")
modificar_win.destroy()
mostrar_usuarios()

btn_guardar = tk.Button(modificar_win, text="Guardar cambios",
command=guardar_modificacion, bg="#FF5733", fg="white", font=('Arial', 11, 'bold'),
relief="raised", bd=5)
btn_guardar.grid(row=2, column=0, columnspan=2, pady=10)

for usuario in usuarios:
    tree.insert("", "end", values=(usuario[0], usuario[1], usuario[2]))

botones_frame = tk.Frame(frame, bg="#404040")
botones_frame.pack(fill='x')

btn_modificar = tk.Button(botones_frame, text="Modificar", command=lambda
id=usuario[0]: modificar_usuario(id), bg="#2082AA", fg="white", font=('Arial', 6, 'bold'))
btn_modificar.pack(side=tk.LEFT, padx=5, pady=5)
```



```
btn_eliminar = tk.Button(botones_frame, text="Eliminar", command=lambda id=usuario[0]:  
eliminar_usuario(id), bg="#FF5733", fg="white", font=('Arial', 6, 'bold'))  
btn_eliminar.pack(side=tk.LEFT, padx=5, pady=5)  
  
# Ventana principal  
root = tk.Tk()  
root.title("On Fire - Gestión de Usuarios")  
root.configure(bg="#2d2d2d")  
  
# Título estilizado  
title_font = font.Font(family="Helvetica", size=24, weight="bold")  
title_label = tk.Label(root, text="🔥 On Fire - Gestión de Usuarios 🔥", font=title_font,  
fg="#FF5733", bg="#2d2d2d")  
title_label.pack(padx=20, pady=20)  
  
btn_usuarios = tk.Button(  
    root, text="Usuarios", command=mostrar_usuarios, bg="#FF5733", fg="white",  
    font=('Helvetica', 16, 'bold'), relief="flat",  
    activebackground="#C70039"  
)  
btn_usuarios.pack(pady=20)  
  
root.mainloop()
```

datosABM.py

```
import tkinter as tk  
from tkinter import font  
from tkinter import ttk, messagebox, filedialog, simpledialog  
import mysql.connector
```



```
import csv

# Conectar a la base de datos MySQL
conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="",
    database="onfireBD"
)
cursor = conn.cursor()

# Variables globales para ventanas abiertas
sensor_win = None
modificar_win = None
agregar_win = None
botones_sensores = {}

# Función para mostrar datos del sensor
def mostrar_datos_sensor(sensor):
    global sensor_win
    if sensor_win is not None:
        sensor_win.destroy()

    sensor_win = tk.Toplevel(root)
    sensor_win.title(f"Datos de {sensor.capitalize()}")
    sensor_win.configure(bg="#2d2d2d")

    frame = tk.Frame(sensor_win, bg="#404040", padx=10, pady=10)
    frame.pack(expand=True, fill='both')
```



```
btn_agregar = tk.Button(sensor_win, text="Agregar Dato", command=lambda:  
agregar_dato(sensor), bg="#FF5733", fg="white", font=('Arial', 11, 'bold'), relief="raised", bd=5)  
btn_agregar.pack(pady=10)  
  
# Crear tabla para mostrar datos  
  
cols = ("ID", "Fecha", "Hora", "Valor")  
tree = ttk.Treeview(frame, columns=cols, show='headings')  
  
for col in cols:  
    tree.heading(col, text=col)  
  
tree.pack(side=tk.LEFT, expand=True, fill='both')  
  
scrollbar = ttk.Scrollbar(frame, orient="vertical", command=tree.yview)  
scrollbar.pack(side=tk.RIGHT, fill='y')  
tree.configure(yscroll=scrollbar.set)  
  
# Mostrar datos del sensor  
  
cursor.execute(f"SELECT * FROM {sensor}")  
datos = cursor.fetchall()  
  
for dato in datos:  
    tree.insert("", "end", values=dato)  
  
# Exportar datos a CSV  
  
def exportar_csv():  
    file_path = filedialog.asksaveasfilename(defaultextension=".csv", filetypes=[("CSV files",  
"*.csv")])
```



```
if file_path:  
    with open(file_path, mode='w', newline="") as file:  
        writer = csv.writer(file)  
        writer.writerow(cols)  
        writer.writerows(datos)  
  
    messagebox.showinfo("Exportación exitosa", "Datos exportados a CSV exitosamente.")
```

Botón de exportar

```
btn_exportar = tk.Button(sensor_win, text="Exportar CSV", command=exportar_csv,  
bg="#2082AA", fg="white", font=('Arial', 11, 'bold'))  
  
btn_exportar.pack(pady=10)
```

Función para eliminar dato

```
def eliminar_dato(dato_id):  
  
    if messagebox.askyesno("Confirmar", f"¿Estás seguro de que deseas eliminar este dato?  
(Id:{dato_id})?"):  
  
        cursor.execute(f"DELETE FROM {sensor} WHERE id = %s", (dato_id,))  
        conn.commit()  
  
        mostrar_datos_sensor(sensor)  
  
        messagebox.showinfo("Éxito", "Dato eliminado exitosamente.")
```

Función para modificar dato

```
def modificar_dato(dato_id):  
  
    global modificar_win  
  
    if modificar_win is not None:  
        modificar_win.destroy()  
  
    modificar_win = tk.Toplevel(sensor_win)  
    modificar_win.title("Modificar Dato")
```



```
modificar_win.configure(bg="#2d2d2d")
```

```
cursor.execute(f"SELECT * FROM {sensor} WHERE id = %s", (dato_id,))  
dato = cursor.fetchone()
```

```
lbl_valor = tk.Label(modificar_win, text="Valor", font=('Helvetica', 10), fg="white",  
bg="#2d2d2d")
```

```
lbl_valor.grid(row=0, column=0, padx=10, pady=10)  
entry_valor = tk.Entry(modificar_win, font=('Helvetica', 10), width=20)  
entry_valor.insert(0, dato[3])  
entry_valor.grid(row=0, column=1, padx=10, pady=10)
```

```
def guardar_modificacion():
```

```
    nuevo_valor = entry_valor.get()
```

```
    if not nuevo_valor:
```

```
        messagebox.showwarning("Campos incompletos", "Por favor complete todos los  
campos.")
```

```
    return
```

```
query = f"UPDATE {sensor} SET valor=%s WHERE id=%s"
```

```
cursor.execute(query, (nuevo_valor, dato_id))
```

```
conn.commit()
```

```
messagebox.showinfo("Éxito", "Dato modificado exitosamente.")
```

```
modificar_win.destroy()
```

```
mostrar_datos_sensor(sensor)
```

```
btn_guardar = tk.Button(modificar_win, text="Guardar cambios",  
command=guardar_modificacion, bg="#FF5733", fg="white", font=('Arial', 11, 'bold'),  
relief="raised", bd=5)
```

```
btn_guardar.grid(row=1, column=0, columnspan=2, pady=10)
```



```
# Crear menú contextual para la modificación y eliminación de datos

menu_contextual = tk.Menu(sensor_win, tearoff=0)

menu_contextual.add_command(label="Modificar", command=lambda:
modificar_dato(selected_id))

menu_contextual.add_command(label="Eliminar", command=lambda:
eliminar_dato(selected_id))

def mostrar_menu(event):

    item = tree.identify_row(event.y)

    if item:

        global selected_id

        selected_id = tree.item(item, "values")[0]

        menu_contextual.post(event.x_root, event.y_root)

tree.bind("<Button-3>", mostrar_menu)

# Función para agregar un nuevo dato

def agregar_dato(sensor):

    global agregar_win

    if agregar_win is not None:

        agregar_win.destroy()

    agregar_win = tk.Toplevel(root)

    agregar_win.title("Agregar Dato")

    agregar_win.configure(bg="#2d2d2d")

    lbl_valor = tk.Label(agregar_win, text="Valor", font=('Helvetica', 10), fg="white",
bg="#2d2d2d")

    lbl_valor.grid(row=0, column=0, padx=10, pady=10)
```



```
entry_valor = tk.Entry(agregar_win, font=('Helvetica', 10), width=20)
entry_valor.grid(row=0, column=1, padx=10, pady=10)

def guardar_dato():
    valor = entry_valor.get()
    if not valor:
        messagebox.showwarning("Campos incompletos", "Por favor complete todos los campos.")
    return

    query = f"INSERT INTO {sensor} (fecha, hora, valor) VALUES (CURDATE(), CURTIME(), %s)"
    cursor.execute(query, (valor,))
    conn.commit()
    messagebox.showinfo("Éxito", "Dato agregado exitosamente.")
    agregar_win.destroy()
    mostrar_datos_sensor(sensor)

btn_guardar = tk.Button(agregar_win, text="Guardar Dato", command=guardar_dato,
bg="#FF5733", fg="white", font=('Arial', 11, 'bold'), relief="raised", bd=5)
btn_guardar.grid(row=1, column=0, columnspan=2, pady=10)

# Funciones ABM de sensores (agregar, modificar, eliminar)
def crear_sensor():
    sensor_nombre = simpledialog.askstring("Agregar Sensor", "Ingrese el nombre del nuevo sensor:")
    if sensor_nombre:
        try:
            cursor.execute(f"""
CREATE TABLE {sensor_nombre} (

```



```
        id INT NOT NULL AUTO_INCREMENT,  
        fecha DATE NOT NULL,  
        hora TIME NOT NULL,  
        valor DOUBLE(30,3) NOT NULL,  
        PRIMARY KEY (id)  
    )  
    """")  
    conn.commit()  
  
    agregar_boton_sensor(sensor_nombre)  
  
    messagebox.showinfo("Éxito", f"Sensor '{sensor_nombre}' creado exitosamente.")  
  
except mysql.connector.Error as e:  
    messagebox.showerror("Error", f"No se pudo crear el sensor: {e}")  
  
  
def modificar_sensor():  
    sensor_nombre = simpledialog.askstring("Modificar Sensor", "Ingrese el nombre del sensor a  
modificar:")  
  
    if sensor_nombre:  
  
        nueva_columna = simpledialog.askstring("Modificar Sensor", "Ingrese el nombre de la  
nueva columna (opcional):")  
  
        if nueva_columna:  
  
            cursor.execute(f"ALTER TABLE {sensor_nombre} ADD COLUMN {nueva_columna}  
DOUBLE(30,3)")  
  
            conn.commit()  
  
            messagebox.showinfo("Éxito", f"Columna '{nueva_columna}' añadida al sensor  
'{sensor_nombre}'.")  
  
  
def eliminar_sensor():  
    sensor_nombre = simpledialog.askstring("Eliminar Sensor", "Ingrese el nombre del sensor a  
eliminar:")  
  
    if sensor_nombre:
```



```
if messagebox.askyesno("Confirmar", f"¿Estás seguro de que deseas eliminar el sensor '{sensor_nombre}' y todos sus datos?"):
```

```
    cursor.execute(f"DROP TABLE {sensor_nombre}")
```

```
    conn.commit()
```

```
    if sensor_nombre in botones_sensores:
```

```
        botones_sensores[sensor_nombre].destroy()
```

```
        del botones_sensores[sensor_nombre]
```

```
    messagebox.showinfo("Éxito", f"Sensor '{sensor_nombre}' eliminado exitosamente.")
```

Agregar botones de sensores a la ventana principal

```
def agregar_boton_sensor(sensor):
```

```
    btn_sensor = tk.Button(frame_sensores, text=sensor.capitalize(), command=lambda:  
mostrar_datos_sensor(sensor), bg="#5e17eb", fg="white", font=('Arial', 12, 'bold'))
```

```
    btn_sensor.pack(pady=5)
```

```
    botones_sensores[sensor] = btn_sensor
```

Cargar todos los sensores y agregar sus botones, excluyendo la tabla 'usuarios'

```
def cargar_sensores():
```

```
    cursor.execute("SHOW TABLES")
```

```
    sensores = [row[0] for row in cursor.fetchall() if row[0] != "usuarios"]
```

```
    for sensor in sensores:
```

```
        agregar_boton_sensor(sensor)
```

Crear interfaz principal

```
root = tk.Tk()
```

```
root.title("Gestión de Sensores y Datos")
```

```
root.geometry("600x600")
```

```
root.configure(bg="#2d2d2d")
```



Título estilizado

```
title_font = font.Font(family="Helvetica", size=18, weight="bold")  
  
title_label = tk.Label(root, text="🔥 On Fire - Gestión de Datos y Sensores 🔥", font=title_font,  
fg="#FF5733", bg="#2d2d2d")  
  
title_label.pack(padx=20, pady=20)
```

```
frame_principal = tk.Frame(root, bg="#2d2d2d")  
frame_principal.pack(pady=10)
```

```
btn_crear_sensor = tk.Button(frame_principal, text="Crear Sensor", command=crear_sensor,  
bg="#FF5733", fg="white", font=('Arial', 11, 'bold'))  
  
btn_crear_sensor.pack(side=tk.LEFT, padx=5, pady=5)
```

```
btn_modificar_sensor = tk.Button(frame_principal, text="Modificar Sensor",  
command=modificar_sensor, bg="#FF5733", fg="white", font=('Arial', 11, 'bold'))  
  
btn_modificar_sensor.pack(side=tk.LEFT, padx=5, pady=5)
```

```
btn_eliminar_sensor = tk.Button(frame_principal, text="Eliminar Sensor",  
command=eliminar_sensor, bg="#FF5733", fg="white", font=('Arial', 11, 'bold'))  
  
btn_eliminar_sensor.pack(side=tk.LEFT, padx=5, pady=5)
```

```
frame_sensores = tk.Frame(root, bg="#2d2d2d", padx=10, pady=10)  
frame_sensores.pack(expand=True, fill='both')
```

```
cargar_sensores()
```

```
root.mainloop()
```