Tecnicatura Superior en Telecomunicaciones

Espacio Curricular: Programación

DISPOSITIVO IoT PARA CONTROL DE CONCENTRACION DE CO2 EN AMBIENTES CERRADOS

Entrega Nº 2

Alumnos:
Ares, Diego Ezequiel
Gimenez Coria, Fernando
Roldan, Patricio Leandro

Profesor: Lanfranco Lisandro

13 de septiembre de 2024

Resumen

El presente informe detalla el desarrollo de un dispositivo IoT diseñado para medir la concentración de CO2 en el aire, junto con la temperatura y humedad de un ambiente.

El sistema está construido en torno a un microcontrolador ESP32-S, que utiliza un sensor MQ135 para la medición de gases y un sensor AHT25 para obtener lecturas precisas de temperatura y humedad.

Con base en las lecturas de CO2, el dispositivo controla un extractor mediante un relé para mantener la calidad del aire dentro de niveles aceptables.

Además, el sistema monitorea el nivel de batería del dispositivo.

En esta entrega se presenta la creación de una base de datos y el programa en Python para almacenar la información que genere nuestro sistema de control ambiental.

Contenido

Introducción

En la primera entrega de este proyecto se desarrolló un controlador que tomaría datos del medioambiente en un entorno cerrado, como temperatura, humedad, y concentración de CO2, actuando sobre un extractor en caso de excederse un umbral indicado de CO2 y además chequeando el nivel de la batería que alimenta al controlador.

Toda esta información generada periódicamente dispara la necesidad de crear una base de datos para almacenar todos esos datos de forma ordenada y con sentido para su posterior análisis en busca de mejoras o atención a casos imprevistos mejorando en nuestro caso la prevención de daños a la salud. Con estas necesidades en mente este proyecto propone la creación de una base de datos adecuada para almacenar la información obtenida por el controlador y a su vez el código necesario para integrar al controlador con la base de datos.

Aprovechando los recursos del controlador ESP32, se propone hacer el envío de datos a través de la red WiFi donde se encuentre el dispositivo, para lo cual crearemos una API con el micro framework Flask donde recibiremos las solicitudes HTTP enviadas por el controlador e implementaremos las rutas para acceder a la base de datos.

Este enfoque nos proporciona acceso al análisis de datos implementando nuevas rutas para acceder desde una computadora por ejemplo.

Objetivos

Crear la base de datos, interpretando la forma en que los datos deben ser cargados evitando redundancia y garantizando la unicidad de los mismos (3FN).

Crear API con FLASK, para gestionar las peticiones HTTP provenientes del controlador y poder almacenar correctamente la información en la BD.

Generar código modular, encapsulando funcionalidades de tal forma que sea práctico entender, mantener y mejorar el desarrollo.

Documentar el manual de la API, para facilitar su uso.

Desarrollo del objetivo en fases

Fase 1: Investigación y planificación: Aquí proponemos la forma en cómo van a almacenarse los datos en la BD de forma relevante para el proyecto, asi como un repaso del microframework Flask que trabajamos en otra materia.

Fase 2: Implementación técnica En esta etapa, desarrollamos el diagrama relacional y construimos la base de datos con sus respectivas tablas. A su vez desarrollamos el código Python para crear la API.

Fase 3: Validación y ajustes Con la ayuda de un programa para simular la carga de datos desde el controlador (todavía no construimos el dispositivo Edge), probamos que la API funciona y carga efectivamente la información a la base de datos.

Fase 4: Extensión del proyecto con el enfoque de código encapsulado y las funcionalidades de la API es posible extender el proyecto, montar la base de datos y la API en un contenedor, servirlos desde alguna nube donde se pueda guardar datos de muchos clientes en cualquier parte del mundo, que puedan analizar la información generada por sus dispositivos y actuar en consecuencia.

Metodología

Diseño de la base de datos

El sistema propuesto puede manejar varios dispositivos dentro de una red WiFi, a su vez cada dispositivo generara sus propias lecturas. En base a las lecturas de cada controlador, la API calculara alarmas que serán almacenadas en la BD, asi como el historial de carga de batería del dispositivo y eventos del extractor asociados a la información de los sensores al momento de ocurrir dicho evento.

A continuación se muestra el modelo relacional de nuestra base de datos **(fig. 1)**.

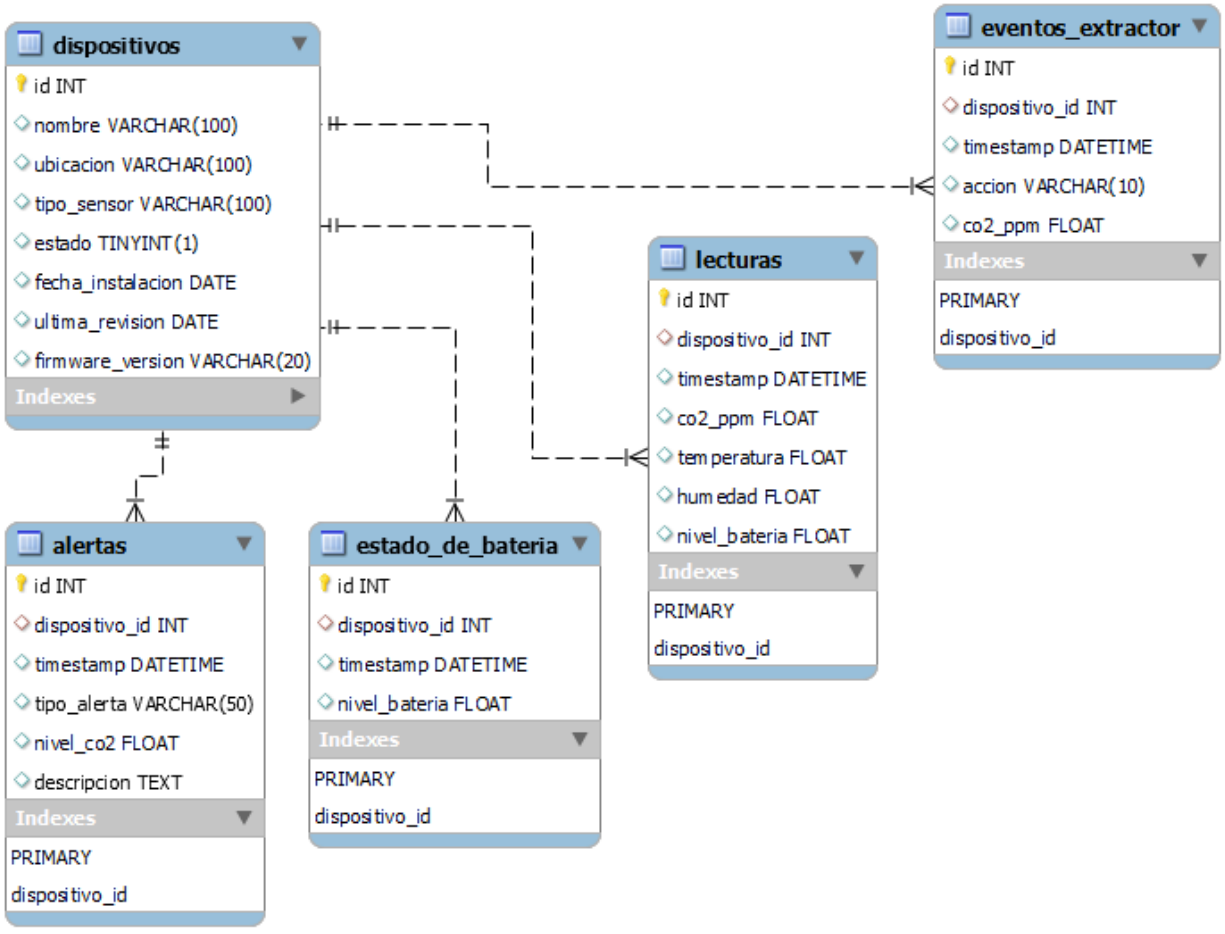

Fig. 1 Diagrama relacional de la base de datos propuesta

Script de creación de la base de datos:

A continuación se presenta el script SQL para la creación de la base de datos, donde se puede apreciar la definición de sus tablas y la relación entre ellas.

```
-- -----------------------------------------------------
-- Schema control_ambiental
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `control_ambiental` DEFAULT CHARACTER SET utf8mb4
COLLATE utf8mb4_0900_ai_ci ;
USE `control_ambiental` ;

-- -----------------------------------------------------
-- Table `control_ambiental`.`dispositivos`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `control_ambiental`.`dispositivos` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `nombre` VARCHAR(100) NULL DEFAULT NULL,
  `ubicacion` VARCHAR(100) NULL DEFAULT NULL,
  `tipo_sensor` VARCHAR(100) NULL DEFAULT NULL,
  `estado` TINYINT(1) NULL DEFAULT '1',
  `fecha_instalacion` DATE NULL DEFAULT NULL,
  `ultima_revision` DATE NULL DEFAULT NULL,
  `firmware_version` VARCHAR(20) NULL DEFAULT NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB
AUTO_INCREMENT = 3
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

-- -----------------------------------------------------
-- Table `control_ambiental`.`alertas`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `control_ambiental`.`alertas` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `dispositivo_id` INT NULL DEFAULT NULL,
  `timestamp` DATETIME NULL DEFAULT CURRENT_TIMESTAMP,
  `tipo_alerta` VARCHAR(50) NULL DEFAULT NULL,
  `nivel_co2` FLOAT NULL DEFAULT NULL,
  `descripcion` TEXT NULL DEFAULT NULL,
  PRIMARY KEY (`id`),
  INDEX `dispositivo_id` (`dispositivo_id` ASC) VISIBLE,
  CONSTRAINT `alertas_ibfk_1`
    FOREIGN KEY (`dispositivo_id`)
    REFERENCES `control_ambiental`.`dispositivos` (`id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
AUTO_INCREMENT = 8
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

-- -----------------------------------------------------
-- Table `control_ambiental`.`estado_de_bateria`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `control_ambiental`.`estado_de_bateria` (
```

```
  `id` INT NOT NULL AUTO_INCREMENT,
  `dispositivo_id` INT NULL DEFAULT NULL,
  `timestamp` DATETIME NULL DEFAULT CURRENT_TIMESTAMP,
  `nivel_bateria` FLOAT NULL DEFAULT NULL,
  PRIMARY KEY (`id`),
  INDEX `dispositivo_id` (`dispositivo_id` ASC) VISIBLE,
  CONSTRAINT `estado_de_bateria_ibfk_1`
    FOREIGN KEY (`dispositivo_id`)
    REFERENCES `control_ambiental`.`dispositivos` (`id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
AUTO_INCREMENT = 6
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

-- -----------------------------------------------------
-- Table `control_ambiental`.`eventos_extractor`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `control_ambiental`.`eventos_extractor` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `dispositivo_id` INT NULL DEFAULT NULL,
  `timestamp` DATETIME NULL DEFAULT CURRENT_TIMESTAMP,
  `accion` VARCHAR(10) NULL DEFAULT NULL,
  `co2_ppm` FLOAT NULL DEFAULT NULL,
  PRIMARY KEY (`id`),
  INDEX `dispositivo_id` (`dispositivo_id` ASC) VISIBLE,
  CONSTRAINT `eventos_extractor_ibfk_1`
    FOREIGN KEY (`dispositivo_id`)
    REFERENCES `control_ambiental`.`dispositivos` (`id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

-- -----------------------------------------------------
-- Table `control_ambiental`.`lecturas`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `control_ambiental`.`lecturas` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `dispositivo_id` INT NULL DEFAULT NULL,
  `timestamp` DATETIME NULL DEFAULT CURRENT_TIMESTAMP,
  `co2_ppm` FLOAT NULL DEFAULT NULL,
  `temperatura` FLOAT NULL DEFAULT NULL,
  `humedad` FLOAT NULL DEFAULT NULL,
  `nivel_bateria` FLOAT NULL DEFAULT NULL,
  PRIMARY KEY (`id`),
  INDEX `dispositivo_id` (`dispositivo_id` ASC) VISIBLE,
  CONSTRAINT `lecturas_ibfk_1`
    FOREIGN KEY (`dispositivo_id`)
    REFERENCES `control_ambiental`.`dispositivos` (`id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
AUTO_INCREMENT = 6
DEFAULT CHARACTER SET = utf8mb4
```

```
COLLATE = utf8mb4_0900_ai_ci;

SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;


-- -----------------------------------------------------
-- Schema control_ambiental
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `control_ambiental` DEFAULT CHARACTER SET utf8mb4
COLLATE utf8mb4_0900_ai_ci ;
USE `control_ambiental` ;

-- -----------------------------------------------------
-- Table `control_ambiental`.`dispositivos`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `control_ambiental`.`dispositivos` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `nombre` VARCHAR(100) NULL DEFAULT NULL,
  `ubicacion` VARCHAR(100) NULL DEFAULT NULL,
  `tipo_sensor` VARCHAR(100) NULL DEFAULT NULL,
  `estado` TINYINT(1) NULL DEFAULT '1',
  `fecha_instalacion` DATE NULL DEFAULT NULL,
  `ultima_revision` DATE NULL DEFAULT NULL,
  `firmware_version` VARCHAR(20) NULL DEFAULT NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB
AUTO_INCREMENT = 3
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

-- -----------------------------------------------------
-- Table `control_ambiental`.`alertas`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `control_ambiental`.`alertas` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `dispositivo_id` INT NULL DEFAULT NULL,
  `timestamp` DATETIME NULL DEFAULT CURRENT_TIMESTAMP,
  `tipo_alerta` VARCHAR(50) NULL DEFAULT NULL,
  `nivel_co2` FLOAT NULL DEFAULT NULL,
  `descripcion` TEXT NULL DEFAULT NULL,
  PRIMARY KEY (`id`),
  INDEX `dispositivo_id` (`dispositivo_id` ASC) VISIBLE,
  CONSTRAINT `alertas_ibfk_1`
    FOREIGN KEY (`dispositivo_id`)
    REFERENCES `control_ambiental`.`dispositivos` (`id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
AUTO_INCREMENT = 8
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

-- -----------------------------------------------------
-- Table `control_ambiental`.`estado_de_bateria`
-- -----------------------------------------------------
```

```
CREATE TABLE IF NOT EXISTS `control_ambiental`.`estado_de_bateria` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `dispositivo_id` INT NULL DEFAULT NULL,
  `timestamp` DATETIME NULL DEFAULT CURRENT_TIMESTAMP,
  `nivel_bateria` FLOAT NULL DEFAULT NULL,
  PRIMARY KEY (`id`),
  INDEX `dispositivo_id` (`dispositivo_id` ASC) VISIBLE,
  CONSTRAINT `estado_de_bateria_ibfk_1`
    FOREIGN KEY (`dispositivo_id`)
    REFERENCES `control_ambiental`.`dispositivos` (`id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
AUTO_INCREMENT = 6
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

-- -----------------------------------------------------
-- Table `control_ambiental`.`eventos_extractor`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `control_ambiental`.`eventos_extractor` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `dispositivo_id` INT NULL DEFAULT NULL,
  `timestamp` DATETIME NULL DEFAULT CURRENT_TIMESTAMP,
  `accion` VARCHAR(10) NULL DEFAULT NULL,
  `co2_ppm` FLOAT NULL DEFAULT NULL,
  PRIMARY KEY (`id`),
  INDEX `dispositivo_id` (`dispositivo_id` ASC) VISIBLE,
  CONSTRAINT `eventos_extractor_ibfk_1`
    FOREIGN KEY (`dispositivo_id`)
    REFERENCES `control_ambiental`.`dispositivos` (`id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

-- -----------------------------------------------------
-- Table `control_ambiental`.`lecturas`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `control_ambiental`.`lecturas` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `dispositivo_id` INT NULL DEFAULT NULL,
  `timestamp` DATETIME NULL DEFAULT CURRENT_TIMESTAMP,
  `co2_ppm` FLOAT NULL DEFAULT NULL,
  `temperatura` FLOAT NULL DEFAULT NULL,
  `humedad` FLOAT NULL DEFAULT NULL,
  `nivel_bateria` FLOAT NULL DEFAULT NULL,
  PRIMARY KEY (`id`),
  INDEX `dispositivo_id` (`dispositivo_id` ASC) VISIBLE,
  CONSTRAINT `lecturas_ibfk_1`
    FOREIGN KEY (`dispositivo_id`)
    REFERENCES `control_ambiental`.`dispositivos` (`id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
AUTO_INCREMENT = 6
```

```
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

Creación del código en Python

A continuación se presentan tres archivos que conforman el código Python encargado de recibir los datos enviados por el controlador y almacenarlos en la base de datos.

main.py

```
from server import app  # Importamos el objeto `app` del servidor Flask

def iniciar_servidor():
    """Función para inicializar el servidor Flask"""
    app.run(debug=True, host='0.0.0.0', port=5000)

if __name__ == "__main__":
    iniciar_servidor()
```

Este archivo es el encargado de inicializar el servidor Flask donde se ejecuta la API, simplemente llamando al método iniciar_servidor().

Server.py

```
from flask import Flask, request, jsonify
from metodos_db import ConexionBD

app = Flask(__name__)

# Inicializar la conexión a la base de datos
bd = ConexionBD('localhost', 'root', 'root', 'control_ambiental')
bd.conectar()
```

```
@app.route('/api/lecturas', methods=['POST'])
def recibir_lecturas():
    """Endpoint para recibir las lecturas de los sensores vía POST"""
    # Obtener la clave API desde el encabezado HTTP
    clave_api = request.headers.get('X-API-KEY')

    # Validar clave API
    if clave_api not in ['silver', 'gold', 'adamantium']:
        return jsonify({'error': 'Clave API no válida'}), 401

    # Extraer los datos del cuerpo de la solicitud
    data = request.json
    dispositivo_id = data.get('dispositivo_id')
    co2_ppm = data.get('co2_ppm')
    temperatura = data.get('temperatura')
    humedad = data.get('humedad')
    nivel_bateria = data.get('nivel_bateria')

    # Verificar si todas las claves existen en el JSON
    if dispositivo_id is None or co2_ppm is None or temperatura is None or humedad
is None or nivel_bateria is None:
        return jsonify({'error': 'Datos incompletos'}), 400

    # Insertar datos en la base de datos
    try:
        bd.insertar_lectura(dispositivo_id, co2_ppm, temperatura, humedad,
nivel_bateria)

        # Verificar si hay que generar alertas
        if co2_ppm > 1000:
            bd.insertar_alerta(dispositivo_id, "Encendido extractor", co2_ppm,
"CO2 excede los 1000 ppm")
        else:
            bd.insertar_alerta(dispositivo_id, "Apagado extractor", co2_ppm, "CO2
en nivel aceptable")

        if nivel_bateria < 3.5:
            bd.insertar_alerta(dispositivo_id, "Batería baja", nivel_bateria,
"Nivel de batería bajo")

        # Insertar estado de la batería
        bd.insertar_estado_bateria(dispositivo_id, nivel_bateria)

        return jsonify({'status': 'Datos recibidos e insertados correctamente'}),
200
    except Exception as e:
        return jsonify({'error': str(e)}), 500
```

En el archivo ***server.py*** encontramos la definición del servidor Flask, encargado de tomar los datos provenientes del controlador y enviar informacion a la BD.
El servidor posee las siguientes características:

1. La ruta **/api/lecturas** es el endpoint al que se enviarán los datos. El ESP32, hará solicitudes POST a esta ruta, enviando los datos en formato JSON.
2. **Autenticación con X-API-KEY**: los datos se autenticarán, con una clave X-API-KEY integrada en el encabezado de las peticiones, antes de ser insertados en la base de datos.
3. **Validación de datos**: Se verifica que los datos sean válidos y completos antes de ser insertados.
4. **Inserciones en la base de datos**: Los datos se insertan en las tablas correspondientes, y se generan alertas si los niveles de CO2 son altos, si la batería es baja y si surge un evento del extractor.

metodos_db.py

```
import pymysql

class ConexionBD:
    def __init__(self, host, usuario, contraseña, base_datos):
        self.host = host
        self.usuario = usuario
        self.contraseña = contraseña
        self.base_datos = base_datos
        self.conexion = None
        self.cursor = None

    def conectar(self):
        """Conectar a la base de datos."""
        try:
            self.conexion = pymysql.connect(
                host=self.host,
                user=self.usuario,
                password=self.contraseña,
                database=self.base_datos
            )
            self.cursor = self.conexion.cursor()
            print("Conexión a la base de datos exitosa")
        except Exception as e:
            print(f"Error al conectar a la base de datos: {e}")

    def cerrar(self):
        """Cerrar la conexión a la base de datos."""
        if self.conexion:
```

```
            self.conexion.close()
            print("Conexión cerrada")

    def insertar_lectura(self, dispositivo_id, co2_ppm, temperatura, humedad,
nivel_bateria):
        """Insertar una nueva lectura de sensores."""
        query = f"""
        INSERT INTO lecturas (dispositivo_id, co2_ppm, temperatura, humedad,
nivel_bateria)
        VALUES ({dispositivo_id}, {co2_ppm}, {temperatura}, {humedad},
{nivel_bateria});
        """
        try:
            self.cursor.execute(query)
            self.conexion.commit()
            print("Lectura insertada correctamente")
        except Exception as e:
            print(f"Error al insertar la lectura: {e}")

    def insertar_alerta(self, dispositivo_id, tipo_alerta, nivel_co2,
descripcion):
        """Insertar una nueva alerta."""
        query = f"""
        INSERT INTO alertas (dispositivo_id, tipo_alerta, nivel_co2, descripcion)
        VALUES ({dispositivo_id}, '{tipo_alerta}', {nivel_co2}, '{descripcion}');
        """
        try:
            self.cursor.execute(query)
            self.conexion.commit()
            print("Alerta insertada correctamente")
        except Exception as e:
            print(f"Error al insertar la alerta: {e}")

    def insertar_estado_bateria(self, dispositivo_id, nivel_bateria):
        """Insertar un nuevo estado de batería."""
        query = f"""
        INSERT INTO estado_de_bateria (dispositivo_id, nivel_bateria)
        VALUES ({dispositivo_id}, {nivel_bateria});
        """
        try:
            self.cursor.execute(query)
            self.conexion.commit()
            print("Estado de batería insertado correctamente")
        except Exception as e:
            print(f"Error al insertar el estado de batería: {e}")
```

El archivo ***metodos_db.py*** contiene la clase que maneja la conexión a la base de datos y las funciones para insertar datos en las tablas correspondientes.

Manual de uso de la API

- Descripción General de la API

Esta API permite a los dispositivos IoT enviar lecturas periódicas de sensores de CO2, temperatura, humedad y estado de batería a un servidor. Los datos se almacenan en una base de datos MySQL.

La API está protegida con autenticación mediante clave API, la cual debe ser incluida en cada solicitud. La clave API se envía a través del encabezado X-API-KEY.

- **Endpoints Disponibles**
    - Enviar Lecturas de Sensores

Este endpoint permite a los dispositivos enviar lecturas de CO2, temperatura, humedad y nivel de batería.

Método HTTP:

POST

URL:

```
/api/lecturas
```

Encabezados HTTP:

- **X-API-KEY**: Clave API para la autenticación.
    - Ejemplo: "adamantium"

Cuerpo de la Solicitud (JSON):

```
{
    "dispositivo_id": 1,
    "co2_ppm": 850.5,
    "temperatura": 25.3,
    "humedad": 45.8,
    "nivel_bateria": 3.9
}
```

Descripción de los Parámetros:

- **dispositivo_id**: ID del dispositivo IoT (debe coincidir con un dispositivo en la base de datos).
- **co2_ppm**: Nivel de CO2 en partes por millón (ppm).
- **temperatura**: Temperatura en grados Celsius.

- **humedad**: Porcentaje de humedad relativa.
- **nivel_bateria**: Nivel de batería del dispositivo en voltios.

*Ejemplo de Solicitud con **curl**:*

```
curl -X POST http://localhost:5000/api/lecturas \
-H "Content-Type: application/json" \
-H "X-API-KEY: adamantium" \
-d '{
    "dispositivo_id": 1,
    "co2_ppm": 850.5,
    "temperatura": 25.3,
    "humedad": 45.8,
    "nivel_bateria": 3.9
}'
```

Códigos de Respuesta:

- **200**: Datos recibidos e insertados correctamente.
- **400**: Datos incompletos o mal formateados.
- **401**: Clave API no válida.
- **500**: Error en el servidor.

Errores Comunes:

- **"Datos incompletos"**: Si alguno de los campos *dispositivo_id, co2_ppm, temperatura, humedad* o *nivel_bateria* no está presente o es *null*.
- **"Clave API no válida"**: Si no se proporciona una clave API válida en el encabezado `X-API-KEY`.

- **Formato de los Datos para el Envío**

Para que los datos sean cargados correctamente en la base de datos, deben enviarse en formato JSON como se describe anteriormente. Aquí tienes una guía de los valores esperados:

Campo	Tipo de Dato	Descripción	Restricciones
dispositivo_id	int	Identificador único del dispositivo en la base de datos	Debe existir en la tabla dispositivos.
co2_ppm	float	Nivel de CO2 medido en partes por millón (ppm)	Valor positivo.
temperatura	float	Temperatura en grados Celsius	Valor numérico.
humedad	float	Humedad relativa en porcentaje	Valor entre 0 y 100.
nivel_bateria	float	Nivel de batería en voltios	Valor entre 3.0 y 4.2.

Validaciones Automáticas del Servidor:

- Si el ***dispositivo_id*** no existe en la base de datos, la inserción fallará.
- Todos los campos son obligatorios. Si falta algún campo, la solicitud será rechazada.

Validacion y ajustes

Para validar el correcto funcionamiento del programa propuesto en conjunto con la base de datos se usara el siguiente script de Python para simular datos como si fueran emitidos por el controlador, ralizar las solicitudes HTTP correspondientes para luego comprobar la salida por consola de los mensajes de respuesta. Tambien podemos comprobar mediante mySQL Workbench la carga de la informacion dentro de las tablas de la base de datos.

simulador_http.py

```
import requests
import random

# URL del servidor Flask
url = 'http://localhost:5000/api/lecturas'

def enviar_datos():
    """Simula el envío de datos del ESP32 a la API vía HTTP"""
    headers = {
        'X-API-KEY': 'adamantium'  # Clave API en el encabezado
    }

    data = {
        'dispositivo_id': 1,
        'co2_ppm': round(random.uniform(400, 2000), 2),
```

```
        'temperatura': round(random.uniform(18, 30), 2),
        'humedad': round(random.uniform(40, 60), 2),
        'nivel_bateria': round(random.uniform(3.0, 4.2), 2)  # Aseguramos un valor
válido
    }

    # Enviar los datos al servidor
    response = requests.post(url, json=data, headers=headers)
    print(response.json())

if __name__ == "__main__":
    for _ in range(5):  # Simular 5 lecturas
        enviar_datos()
```

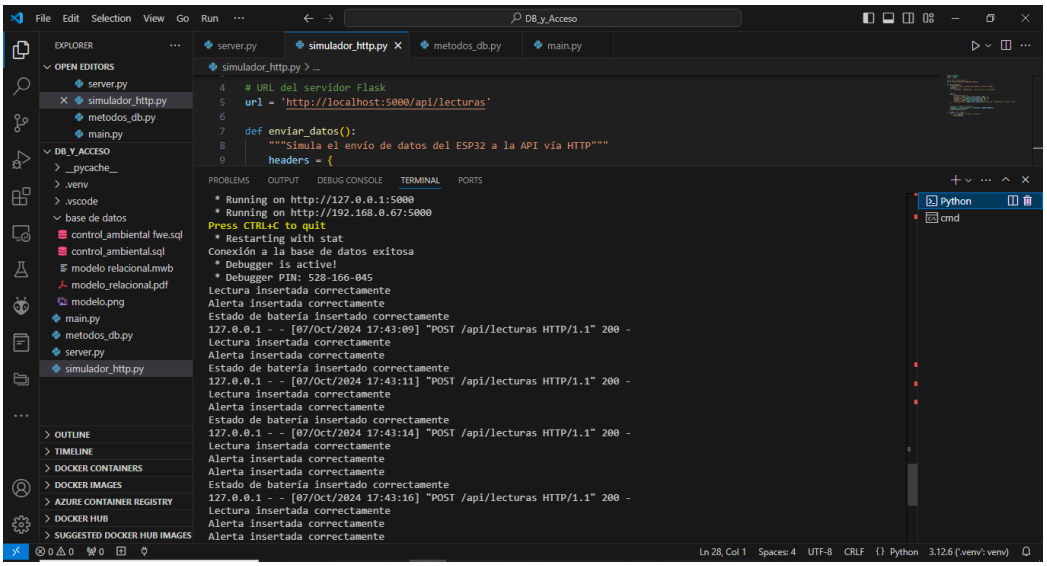

Fig.2 terminal mostrando la actividad del servidor al recibir solicitudes

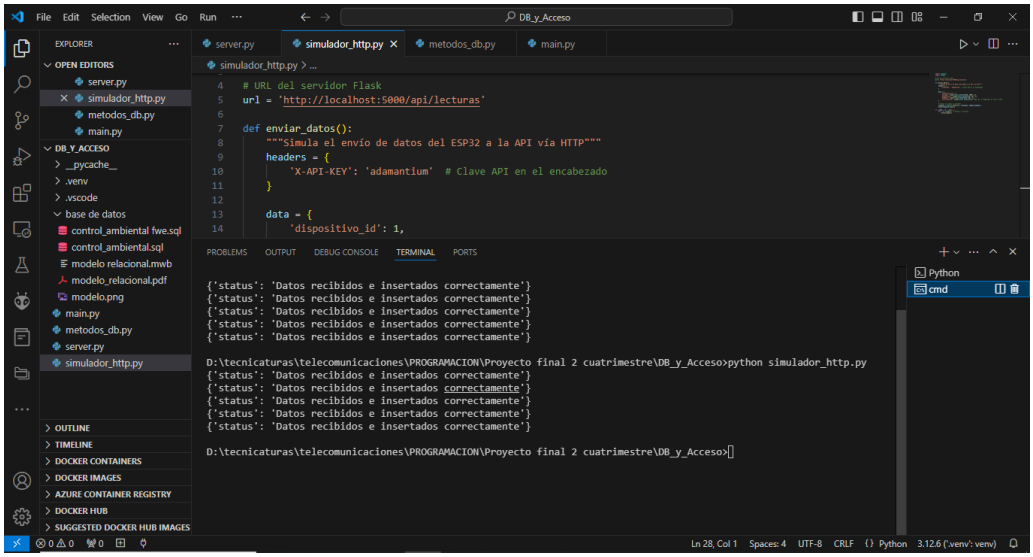

Fig. 3 terminal mostrando las respuestas del servidor

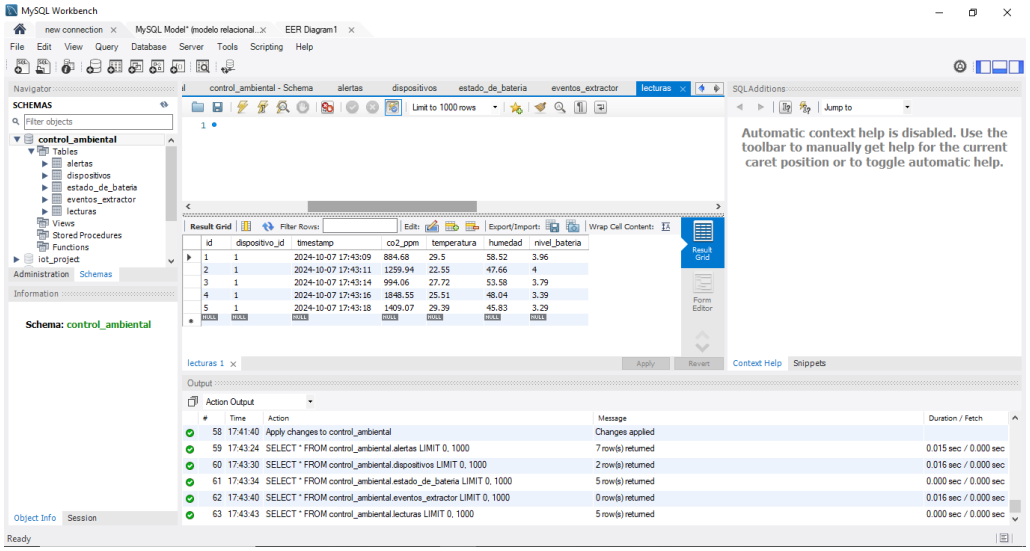

Fig. 4 Comprobamos la carga de la informacion en la base de datos en Workbench

Extensión del proyecto para el Futuro

Se puede implementar una ruta para el acceso a los datos en el servidor Flask con objeto de crear una aplicación que permita a un cliente acceder a la información generada por sus controladores y así poder realizar el análisis de esos datos. Por ejemplo agregando el siguiente código a las rutas existentes:

Obtener Lecturas de Sensores

Descripción:

Este endpoint permite obtener las lecturas de los sensores almacenadas en la base de datos para un análisis posterior.

Método HTTP:

`GET`

URL:
`/api/lecturas`
Parámetros de Consulta (`Query Parameters`):

- **`dispositivo_id`** (opcional): Si se especifica, devolverá solo las lecturas de ese dispositivo.
- **`fecha_inicio`** (opcional): Fecha de inicio para filtrar las lecturas (formato `YYYY-MM-DD`).
- **`fecha_fin`** (opcional): Fecha de fin para filtrar las lecturas (formato `YYYY-MM-DD`).

Ejemplo de Solicitud con `curl`:

```
curl -X GET http://localhost:5000/api/lecturas?dispositivo_id=1&fecha_inicio=2024-10-01&fecha_fin=2024-10-07
```

Respuesta (`JSON`):

```
json
Copiar código
[
    {
        "id": 1,
        "dispositivo_id": 1,
        "co2_ppm": 850.5,
        "temperatura": 25.3,
        "humedad": 45.8,
        "nivel_bateria": 3.9,
        "timestamp": "2024-10-07 12:00:00"
    },
    {
        "id": 2,
        "dispositivo_id": 1,
        "co2_ppm": 950.1,
        "temperatura": 26.7,
        "humedad": 50.2,
        "nivel_bateria": 3.8,
        "timestamp": "2024-10-07 12:05:00"
    }
]
```

Códigos de Respuesta:

- **200**: Lecturas obtenidas correctamente.
- **400**: Solicitud incorrecta.
- **404**: No se encontraron lecturas para los criterios de búsqueda.

Endpoint para Obtener Lecturas (`GET /api/lecturas`)

Código en Flask:

```
@app.route('/api/lecturas', methods=['GET'])
def obtener_lecturas():
    """Obtener lecturas de sensores."""
    dispositivo_id = request.args.get('dispositivo_id')
    fecha_inicio = request.args.get('fecha_inicio')
    fecha_fin = request.args.get('fecha_fin')

    query = "SELECT * FROM lecturas WHERE 1=1"

    if dispositivo_id:
        query += f" AND dispositivo_id = {dispositivo_id}"
    if fecha_inicio:
        query += f" AND DATE(timestamp) >= '{fecha_inicio}'"
    if fecha_fin:
```

```
    query += f" AND DATE(timestamp) <= '{fecha_fin}'"

try:
    bd.cursor.execute(query)
    lecturas = bd.cursor.fetchall()

    # Formatear los resultados en un JSON
    resultado = []
    for lectura in lecturas:
        resultado.append({
            'id': lectura[0],
            'dispositivo_id': lectura[1],
            'co2_ppm': lectura[2],
            'temperatura': lectura[3],
            'humedad': lectura[4],
            'nivel_bateria': lectura[5],
            'timestamp': lectura[6]
        })

    return jsonify(resultado), 200
except Exception as e:
    return jsonify({'error': str(e)}), 500
```

Resultados esperados

El código debería ser capaz de almacenar de forma segura los datos generados por el controlador, asegurando la autenticación de los datos previamente a la carga de los mismos y así evitar accesos indeseados.

Conclusión

Como se pudo apreciar en las capturas de pantalla, el servidor y la base de datos están correctamente integrados y funcionando, pudiendo observar que la información se carga correctamente mediante Workbench.

Se incluye el manual de la API para referencia de su funcionamiento y el formato de los datos a enviar para poder acceder a la base de datos.

Tambien se propone el agregado a futuro de una nueva ruta para acceder a leer los datos cargados a través de la API.