






Tecnicatura Superior en Telecomunicaciones Modulo: Programador

PROYECTO INTEGRADOR 2 Cuatrimestre Entrega N.º 3

Alumnos:

-  Zalazar, Joaquin
-  Márquez, José
-  Durigutti Vittorio

Profesor: Ing. Lanfranco Lisandro

Fecha: 25/10/2024

Consignas solicitadas:

Escribir un programa que permita al microcontrolador recopilar datos del sensor y prepararlos para su envío. Para esto, debe considerar la capacidad de comunicación del Sistema embebido elegido.

- Realizar la programación del comportamiento del actuador.
- Probar la comunicación completa entre el dispositivo IoT y la PC. Realizar las correcciones necesarias y la optimización del sistema que consideres.
- Proporcionar el código fuente y documentación de los programas que se han escrito.
- Documentar las pruebas realizadas, las correcciones y cambios realizados a lo largo del proyecto.

Realizar la programación del comportamiento del actuador:

Este código está diseñado para controlar un sistema de detección de gases utilizando un sensor MQ-9, un buzzer (para alerta sonora), y un extractor de aire (simulado con un LED) en una placa ESP32.

El sistema incluye una función de alarma crítica que se activa mediante una interrupción en un pin específico o si se supera un umbral de gas crítico (simulado con un pulsador).

La fracción del código que corresponde al actuador es:

```
void loop() {  
  int gasLevel = analogRead(MQ9_PIN); // Leer el nivel de gas del sensor (simulado)  
  gasLevel = map(gasLevel, 0, 4095, 0, 200); // Mapea a un rango de 0 a 200  
  
  // Alerta crítica de gas detectada  
  if (alertaCritica || gasLevel >= GAS_THRESHOLD_CRITICAL) {  
    Serial.println("¡Alerta crítica de gas detectada!");  
    activarBuzzer(true, 100); // Buzzer rápido  
    digitalWrite(EXTRACTOR_PIN, HIGH); // Encender extractor  
    alertaCritica = false; // Reiniciar alerta crítica  
  }  
  // Gas moderado
```

```
else if (gasLevel > GAS_THRESHOLD_MODERATE) {  
    Serial.println("Nivel moderado de gas.");  
  
    activarBuzzer(true, 1000);    // Buzzer lento  
    digitalWrite(EXTRACTOR_PIN, HIGH); // Encender extractor  
}  
// Gas en nivel seguro  
else {  
    desactivarAlarma();          // Apagar buzzer y extractor  
}  
  
delay(1000); // Espera antes de la próxima lectura  
}
```

Explicación del Comportamiento

- **Alerta Crítica:** Cuando el nivel de gas supera el umbral crítico (**GAS_THRESHOLD_CRITICAL**) o si se ha activado una alerta crítica, el buzzer emite una señal rápida, y el extractor de aire se enciende.
- **Nivel Moderado:** Si el nivel de gas es moderado (entre los umbrales moderado y crítico), el buzzer emite una señal lenta y el extractor también se enciende.
- **Nivel Seguro:** Cuando el gas está en un nivel seguro (por debajo del umbral moderado), tanto el buzzer como el extractor se apagan mediante la función **desactivarAlarma()**.

Este comportamiento permite una respuesta escalonada dependiendo del nivel de gas detectado.

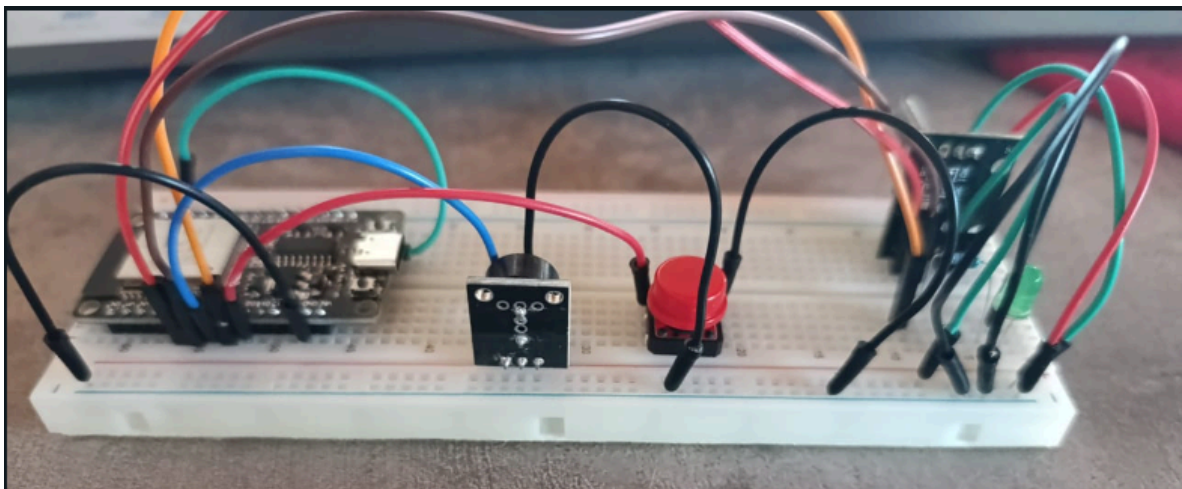
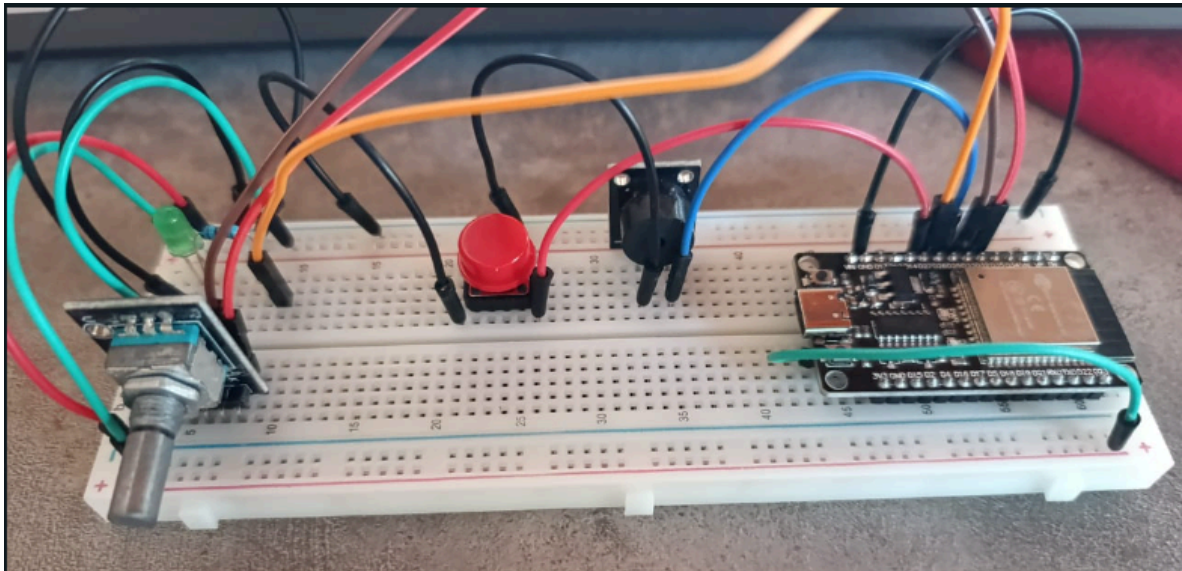
En los datos adjuntos se presentará el código completo.

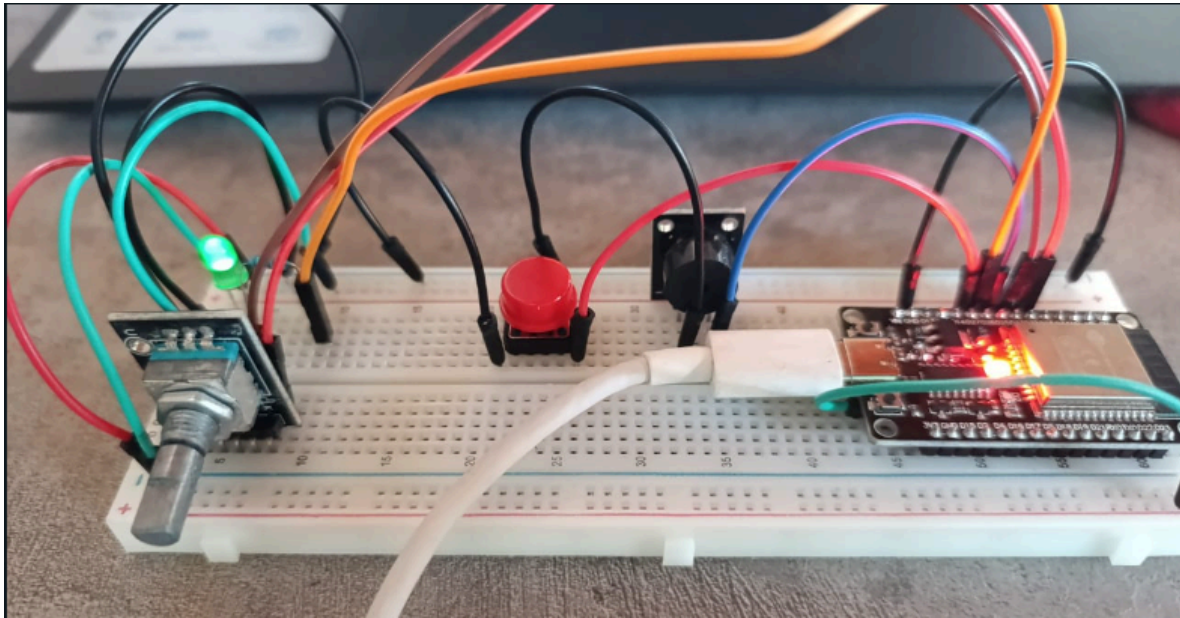
Probar la comunicación completa entre el dispositivo IoT y la PC.

Realizar las correcciones necesarias y la optimización del sistema que consideres.

.....

Se realizó en armado en físico del prototipo, y se probó la funcionalidad, se verifica que se compila correctamente el programa en el microcontrolador, se realizaron algunos cambios al sketch como por ejemplo el mapeo se retiró por que se utilizó un potenciómetro digital y se modificaron los umbrales de detección pasaron de moderate 120 a 12 y el critical de 150 a 15.





Proporcionar el código fuente y documentación de los programas que se han escrito.

Sketch C++ para la simulación:

```
#define MQ9_PIN 34          // Pin analógico para el sensor MQ-9
#define BUZZER_PIN 25       // Pin digital para el buzzer (simulado con LED)
#define EXTRACTOR_PIN 26    // Pin digital para el extractor (simulado con LED)
#define INTERRUPT_PIN 27    // Pin para activar la interrupción (simulado con botón)
#define GAS_THRESHOLD_MODERATE 120
#define GAS_THRESHOLD_CRITICAL 150
```

```
bool alertaCritica = false;

void IRAM_ATTR manejarAlertaCritica() {
    alertaCritica = true; // Cambiar estado cuando ocurra interrupción
}

void setup() {
    pinMode(MQ9_PIN, INPUT);
    pinMode(BUZZER_PIN, OUTPUT);
    pinMode(EXTRACTOR_PIN, OUTPUT);
    pinMode(INTERRUPT_PIN, INPUT_PULLUP);

    attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN),
manejarAlertaCritica, RISING);

    Serial.begin(115200);
}

void loop() {
    int gasLevel = analogRead(MQ9_PIN); // Lee el nivel de gas del sensor

    // Mapea el nivel de gas a un rango más comprensible
    gasLevel = map(gasLevel, 0, 4095, 0, 200); // Ajusta según el rango de tu
sensor

    // Alerta crítica de gas detectada
    if (alertaCritica || gasLevel >= GAS_THRESHOLD_CRITICAL) {
        Serial.println("¡Alerta crítica de gas detectada!");
        activarBuzzer(true, 100); // Buzzer rápido
        digitalWrite(EXTRACTOR_PIN, HIGH); // Encender extractor
        alertaCritica = false; // Reiniciar alerta crítica
    }

    // Gas moderado
    else if (gasLevel > GAS_THRESHOLD_MODERATE) {
        Serial.println("Nivel moderado de gas.");
        activarBuzzer(true, 1000); // Buzzer lento
        digitalWrite(EXTRACTOR_PIN, HIGH); // Encender extractor
    }

    // Gas en nivel seguro
    else {
        desactivarAlarma();
    }

    // Mostrar el nivel de gas actual
    Serial.print("Nivel de gas actual: ");
    Serial.println(gasLevel);
}
```

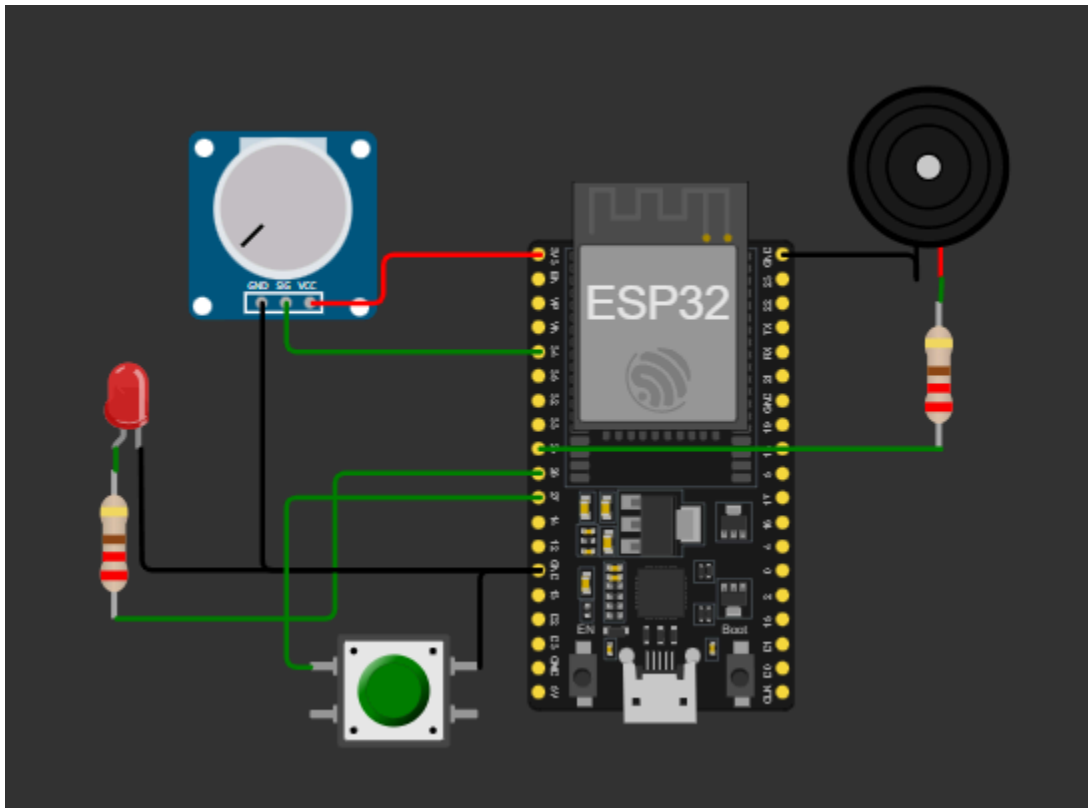


```
    delay(1000); // Espera antes de la próxima lectura
}

void activarBuzzer(bool encendido, int delayMs) {
    if (encendido) {
        digitalWrite(BUZZER_PIN, HIGH);
        delay(delayMs / 2); // Encendido por la mitad del tiempo
        digitalWrite(BUZZER_PIN, LOW);
        delay(delayMs / 2); // Apagado por la otra mitad
    } else {
        digitalWrite(BUZZER_PIN, LOW);
    }
}

void desactivarAlarma() {
    digitalWrite(BUZZER_PIN, LOW); // Apagar buzzer
    digitalWrite(EXTRACTOR_PIN, LOW); // Apagar extractor
}
```

Se adjunta vinculo a la simulación en Wokwi:



<https://wokwi.com/projects/411845954525731841>

Sketch Python, con la documentación del código.

```
# Programa en Python
# Proyecto 2
# EJECUCION DEL PROGRAMA: python menu_mysql.py
# Este programa se encarga de leer datos enviados por un ESP32 a través del
# puerto serial
# y registrar los datos en una base de datos MySQL.

#-----#
# Librerías

import mysql.connector # Para manejar la conexión con la base de datos
MySQL
from mysql.connector import Error # Para gestionar errores de MySQL
import serial # Para leer desde el puerto serial
```

```
import time # Para manejar tiempos de espera

#-----#
# Función para conectar a la base de datos
def conectar():
    try:
        # Crear conexión con la base de datos MySQL usando las credenciales
        # especificadas
        connection = mysql.connector.connect(
            host='localhost', # Servidor de la base de datos
            database='GASDETECTOR', # Nombre de la base de datos
            user='root', # Usuario de la base de datos
            password='Contrasena09081994' # Contraseña del usuario (cambiar
            si es necesario)
        )
        # Verifica si la conexión fue exitosa
        if connection.is_connected():
            print("Conexión exitosa a la base de datos")
            return connection # Devuelve la conexión para su uso posterior
        except Error as e:
            # Muestra un mensaje de error si la conexión falla
            print(f"Error al conectar a la base de datos: {e}")
            return None

#-----#
# Función para registrar la lectura del sensor en la base de datos
def registrar_lectura(connection, id_sensor, nivel_gas):
    # Crear cursor para ejecutar la consulta SQL
    cursor = connection.cursor()
    try:
        # Consulta SQL para insertar la lectura en la tabla `Lectura`
        query = """INSERT INTO Lectura (fecha_hora, nivel_gas,
            id_sensor_actuador)
            VALUES (NOW(), %s, %s)"""
        # Ejecuta la consulta con el valor de nivel de gas e ID del sensor
        cursor.execute(query, (nivel_gas, id_sensor))
        # Confirma los cambios en la base de datos
        connection.commit()
        print(f"[INFO] Lectura registrada en la BD: Sensor {id_sensor},
        Nivel de Gas: {nivel_gas}")
    except Error as e:
        # Muestra un mensaje de error si ocurre un problema al ejecutar la
        # consulta
        print(f"[ERROR] Error al registrar la lectura: {e}")
    finally:
        # Cierra el cursor después de realizar la operación
        cursor.close()
```

```
#-----#
# Función para leer datos del puerto serial y registrar en la base de datos
def leer_datos_serial(connection, puerto='COM5', baudrate=9600):
    # Inicializa variable para la conexión serial
    ser = None
    try:
        # Configura la conexión serial con el puerto y la velocidad de
        baudios especificados
        ser = serial.Serial(puerto, baudrate, timeout=3)
        ser.reset_input_buffer() # Limpia el buffer al iniciar para evitar
        lecturas residuales
        print(f"[INFO] Conectado al puerto {puerto} a {baudrate} baudios")
        time.sleep(2) # Espera breve para estabilizar la conexión

        while True:
            # Lee una línea desde el puerto serial, decodifica y elimina
            caracteres de espacio
            linea = ser.readline().decode('utf-8', errors='ignore').strip()
            if linea: # Si hay datos disponibles en la línea
                print(f"[DEBUG] Datos recibidos del ESP32: '{linea}'")
                try:
                    # Divide los datos recibidos en ID del sensor y nivel de
                    gas
                    id_sensor, nivel_gas = linea.split(":")
                    print(f"[INFO] Sensor: {id_sensor}, Nivel de Gas:
                    {nivel_gas}")

                    # Registra la lectura en la base de datos
                    registrar_lectura(connection, id_sensor,
                    float(nivel_gas))
                except ValueError as e:
                    # Muestra un mensaje si el formato de los datos es
                    incorrecto
                    print(f"[ERROR] Formato incorrecto: '{linea}' - {e}")

                    time.sleep(2) # Espera 2 segundos entre lecturas para evitar
                    saturación

            except serial.SerialException as e:
                # Muestra un mensaje si ocurre un error al abrir el puerto serial
                print(f"[ERROR] Error al abrir el puerto serial: {e}")
            except KeyboardInterrupt:
                # Interrumpe la lectura en caso de que el usuario presione Ctrl+C
                print("\n[INFO] Lectura interrumpida por el usuario.")
        finally:
            # Cierra el puerto serial si está abierto
            if ser and ser.is_open:
```

```
ser.close()
print("[INFO] Puerto serial cerrado.")

#-----#
# Menú interactivo
def menu():
    # Conectar a la base de datos MySQL
    connection = conectar()
    if connection: # Verifica si la conexión fue exitosa
        while True:
            # Muestra opciones del menú
            print("\n--- Menú de Opciones ---")
            print("1. Leer datos del monitor serial y registrar en la base de datos")
            print("2. Salir")

            # Pide al usuario que seleccione una opción
            opcion = input("Selecciona una opción: ")

            if opcion == '1':
                # Solicita el puerto serial (predeterminado COM5) y usa una
                # tasa de 9600 baudios
                puerto = input("Introduce el puerto serial (por defecto COM5): ") or 'COM5'
                baudrate = 9600
                # Inicia la lectura de datos desde el puerto serial
                leer_datos_serial(connection, puerto, int(baudrate))
            elif opcion == '2':
                # Cierra la conexión a la base de datos y termina el programa
                connection.close()
                print("[INFO] Conexión cerrada. Saliendo...")
                break
            else:
                # Mensaje de error si la opción ingresada no es válida
                print("[ERROR] Opción no válida. Inténtalo de nuevo.")
        else:
            # Muestra un mensaje si no se puede conectar a la base de datos
            print("[ERROR] No se pudo conectar a la base de datos.")

# Ejecuta el menú principal al iniciar el programa
if __name__ == "__main__":
    menu()
```


Código Fuente de la base de datos:

```
-- Tabla CLIENTE
CREATE TABLE Cliente (
  dni_cliente VARCHAR(10) PRIMARY KEY, -- DNI único
  nombre VARCHAR(100),
  email VARCHAR(100),
  telefono VARCHAR(15),
  direccion_contacto VARCHAR(255)
);

-- Tabla CONTROLADOR
CREATE TABLE Controlador (
  id_dispositivo INT AUTO_INCREMENT PRIMARY KEY, -- Clave primaria
  autoincremental
  mac_dispositivo VARCHAR(17) UNIQUE NOT NULL, -- MAC Address única y no
  nula
  nombre_dispositivo VARCHAR(100),
  dni_cliente VARCHAR(10), -- Relación con Cliente (mismo tipo que en
  Cliente)
  ubicacion_dispositivo VARCHAR(255), -- Ubicación física del dispositivo
  FOREIGN KEY (dni_cliente) REFERENCES Cliente(dni_cliente) ON DELETE
  CASCADE -- Relación con Cliente
);

-- Tabla HABITACION
CREATE TABLE Habitacion (
  id_habitacion INT AUTO_INCREMENT PRIMARY KEY, -- Clave primaria
  autoincremental
  nombre_habitacion VARCHAR(100) NOT NULL, -- Nombre no nulo
  id_dispositivo INT, -- FK hacia Controlador
  FOREIGN KEY (id_dispositivo) REFERENCES Controlador(id_dispositivo) ON
  DELETE CASCADE -- Relación con Controlador
);

-- Tabla SENSOR_ACTUADOR
CREATE TABLE Sensor_Actuador (
  id_sensor_actuador INT AUTO_INCREMENT PRIMARY KEY, -- Clave primaria
  autoincremental
  estado_sensor_actuador ENUM('activo', 'inactivo', 'en mantenimiento')
  DEFAULT 'activo',
  id_habitacion INT, -- FK hacia Habitación
  FOREIGN KEY (id_habitacion) REFERENCES Habitacion(id_habitacion) ON
  DELETE CASCADE -- Relación con Habitación
```

```
);

-- Tabla LECTURA
CREATE TABLE Lectura (
    id_lectura INT AUTO_INCREMENT PRIMARY KEY, -- Clave primaria
    autoincremental
    fecha_hora DATETIME NOT NULL, -- Fecha y hora del incidente
    nivel_gas FLOAT NOT NULL, -- Nivel de gas detectado
    id_sensor_actuador INT, -- FK hacia Sensor_Actuador
    FOREIGN KEY (id_sensor_actuador) REFERENCES
Sensor_Actuador(id_sensor_actuador) ON DELETE CASCADE -- Relación con
Sensor_Actuador
);

-- Carga de datos genericos para poder realizar consulta en las tablas.
USE GASDETECTOR;

-- Insertar clientes
INSERT INTO Cliente (dni_cliente, nombre, email, telefono,
direccion_contacto) VALUES
('20123456', 'Juan Pérez', 'juan.perez@mail.com', '1134567890', 'Calle Falsa
123, Buenos Aires, Argentina'),
('20456789', 'María López', 'maria.lopez@mail.com', '1145678901', 'Avenida
Siempre Viva 456, Rosario, Argentina'),
('30123456', 'Carlos Gómez', 'carlos.gomez@mail.com', '1156789012', 'Calle
Sol 789, Córdoba, Argentina'),
('30456789', 'Ana Martínez', 'ana.martinez@mail.com', '1167890123', 'Avenida
Libertad 321, Mendoza, Argentina'),
('40123456', 'Luis Fernández', 'luis.fernandez@mail.com', '1178901234',
'Calle Luna 654, Mar del Plata, Argentina');

-- Insertar controladores (dispositivos)
INSERT INTO Controlador (mac_dispositivo, nombre_dispositivo, dni_cliente,
ubicacion_dispositivo) VALUES
('00:1A:2B:3C:4D:5E', 'Controlador Juan', '20123456', 'Calle Falsa 123,
Buenos Aires'),
('00:1A:2B:3C:4D:5F', 'Controlador María', '20456789', 'Avenida Siempre Viva
456, Rosario'),
('00:1A:2B:3C:4D:60', 'Controlador Carlos', '30123456', 'Calle Sol 789,
Córdoba'),
('00:1A:2B:3C:4D:61', 'Controlador Ana', '30456789', 'Avenida Libertad 321,
Mendoza'),
('00:1A:2B:3C:4D:62', 'Controlador Luis', '40123456', 'Calle Luna 654, Mar
del Plata'),
('00:1A:2B:3C:4D:63', 'Controlador Juan 2', '20123456', 'Oficina Central,
Buenos Aires'),
```

```
('00:1A:2B:3C:4D:64', 'Controlador María 2', '20456789', 'Sucursal Norte,
Rosario'),
('00:1A:2B:3C:4D:65', 'Controlador Carlos 2', '30123456', 'Sucursal Centro,
Córdoba');

-- Insertar habitaciones
INSERT INTO Habitacion (nombre_habitacion, id_dispositivo) VALUES
('Sala de Estar', 1),
('Cocina', 1),
('Comedor', 2),
('Baño', 2),
('Dormitorio Principal', 3),
('Sala de Estar', 3),
('Cocina', 4),
('Comedor', 5),
('Cocina', 6),
('Oficina', 7),
('Comedor', 8);

-- Insertar sensores/actuadores
INSERT INTO Sensor_Actuador (estado_sensor_actuador, id_habitacion) VALUES
('activo', 1),
('activo', 2),
('activo', 3),
('activo', 4),
('activo', 5),
('activo', 6),
('activo', 7),
('activo', 8),
('activo', 9),
('activo', 10),
('activo', 11);

-- Insertar lecturas de ejemplo
INSERT INTO Lectura (fecha_hora, nivel_gas, id_sensor_actuador) VALUES
(NOW(), 45.6, 1),
(NOW(), 20.4, 2),
(NOW(), 18.9, 3),
(NOW(), 100.2, 4),
(NOW(), 80.6, 5),
(NOW(), 95.1, 6),
(NOW(), 70.4, 7),
(NOW(), 110.9, 8),
(NOW(), 90.3, 9),
(NOW(), 65.2, 10),
(NOW(), 40.7, 11);
```

```
-- Verificación de datos en las tablas, a fin de convalidar que ingresaron
correctamente
SELECT * FROM Cliente;
SELECT * FROM Controlador;
SELECT * FROM Habitación;
SELECT * FROM Sensor_Actuador;
SELECT * FROM Lectura;
```

Presentación del código en funcionamiento:

Se aprecia la información de las etapas cumplidas referida a la lectura.

[DEBUG]: Lo utilizamos como medio para verificar que el dato fuera en efecto recibido por programa

[INFO](1) Presenta una lectura resumida, de los datos obtenidos por el programa, en caso de la primera para, que se obtuvo lectura del sensor bajo ID 1, y dio una lectura de 6.0 en el sensor de gas, emulado con potenciómetro.

[INFO](2) Agregamos un mensaje que nos confirme que la carga de datos en la base de datos fue realizada satisfactoriamente.

Pantalla serial, al correr el programa main.py

```
[DEBUG] Datos recibidos del ESP32: '1:6'
[INFO] Sensor: 1, Nivel de Gas: 6
[INFO] Lectura registrada en la BD: Sensor 1, Nivel de Gas: 6.0
[DEBUG] Datos recibidos del ESP32: '1:7'
[INFO] Sensor: 1, Nivel de Gas: 7
[INFO] Lectura registrada en la BD: Sensor 1, Nivel de Gas: 7.0
[DEBUG] Datos recibidos del ESP32: '1:8'
[INFO] Sensor: 1, Nivel de Gas: 8
[INFO] Lectura registrada en la BD: Sensor 1, Nivel de Gas: 8.0
```

Lecturas captadas por la base de datos:

	id_lectura	fecha_hora	nivel_gas	id_sensor_actuador
	37	2024-10-25 22:01:50	14	1
	38	2024-10-25 22:01:52	15	1
	39	2024-10-25 22:01:54	14	1
	40	2024-10-25 22:01:56	13	1
	41	2024-10-25 22:01:58	12	1
	42	2024-10-25 22:02:00	11	1
	43	2024-10-25 22:02:02	10	1
	44	2024-10-25 22:02:04	9	1
●	NULL	NULL	NULL	NULL

Se adjunta un video [**Video demostracion.mp4**] donde se muestra el dispositivo y los programas corriendo en simultáneo.