

Desarrollo de Prototipo IoT con ESP32

# Proyecto: Estación meteorológica

## Introducción

Este proyecto tiene como finalidad implementar una **estación meteorológica básica** utilizando un microcontrolador **ESP32** y sensores de temperatura, humedad y presión atmosférica.

La estación muestra los valores en pantalla en tiempo real y los publica a un servidor **MQTT**, mediante el desarrollo de un software de control o **backend** obtendremos las mediciones publicadas en el servidor y las guardaremos en una base de datos.

## Objetivos Específicos

* Desarrollar la capa física del sistema en un simulador online.
* Simular mediciones de temperatura y humedad del ambiente usando un sensor DHT22.
* Simular lecturas de presión atmosférica mediante software.
* Mostrar los valores en una pantalla OLED.
* Publicar los datos a un broker MQTT de forma periódica
* Obtener estos datos a través de un backend
* Guardar los datos obtenidos en una base de datos.

Descripcion del Software utilizado

Simularemos la capa física del proyecto usando un simulador en línea: <https://wokwi.com/>   
Debido a que este simulador no incluye un componente para medir la presión atmosférica (como el BMP180), se ha implementado una **simulación de software realista**, mediante un bloque de código dentro del microcontrolador, que genera valores de presión atmosférica con variaciones graduales dentro de un rango logico.

El siguiente enlace corresponde al proyecto en el simulador Woki:  
<https://wokwi.com/projects/440483972610516993>

Como **servidor MQTT** hemos elegido **HiveMQ**, es un bróker libre que permite la publicación de datos mediante topicos a los que podemos subscribirnos para visualizar desde la versión web cliente

Para la visualización de los mensajes en el servidor MQTT se deben seguir los siguientes pasos:

1- Ingresar al enlace <https://www.hivemq.com/demos/websocket-client/>

2- Click en el botón "connect"

3- Click en "Subscriptions"

4- Click en "Add new topic subscription"

5- En topic escribir: estación-meteorologica-ESP32

6- Click en "subscribe"

7- En el apartado Messages se podrán visualizar las notificaciones el ESP32

Para la creación de la base de datos se utilizó **MySQL Server** como motor de base de datos y **MySQL Workbench** como interfaz gráfica, lo que permitió diseñar la estructura de tablas, definir usuarios y permisos, así como ejecutar consultas de prueba para verificar el correcto funcionamiento.

El software de control o **backend** se desarrolló en **Python**, utilizando **Visual Studio Code** como entorno de desarrollo. Para interactuar con el broker MQTT y con la base de datos, fue necesario instalar las siguientes librerías:

* **paho-mqtt:** permite establecer la comunicación con el broker MQTT, suscribirse a tópicos, recibir los mensajes publicados por el ESP32 y manejar reconexiones automáticas.
* **mysql-connector-python**: facilita la conexión a la base de datos MySQL, la ejecución de consultas y la inserción de datos recibidos desde el ESP32 en la tabla de mediciones.
* **json** (librería estándar de Python): se utiliza para decodificar los mensajes JSON enviados por el ESP32, extrayendo los valores de temperatura, humedad y presión para almacenarlos en la base de datos.

Este conjunto de herramientas permitió desarrollar un sistema capaz de recibir datos en tiempo real desde el ESP32 y registrarlos automáticamente en la base de datos, garantizando la integridad y disponibilidad de la información.

## 

## Descripción del Hardware Utilizado

|  |  |
| --- | --- |
| **Componente** | **Descripción** |
| **ESP32** | Microcontrolador principal del sistema. |
| **DHT22** | Sensor de temperatura y humedad digital. |
| **Pantalla OLED (SSD1306)** | Pantalla I2C de 128x64 píxeles para visualización. |
| **Cables y protoboard** | Contemplados para realizar las conexiones reales de manera física |

Simulación **de sensor BMP180**: Como Wokwi no dispone de un sensor de presión atmosférica, se simularon lecturas válidas en el rango de **980 hPa a 1030 hPa**, con cambios suaves y aleatorios cada un determinado tiempo.

## 

## Esquema de Conexión

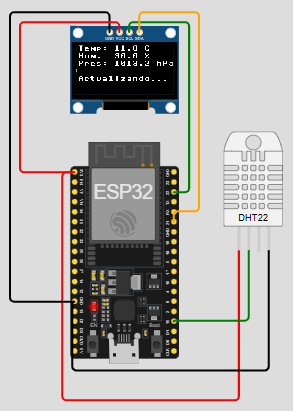


Imagen 3. Captura del simulador Wokwi, de las conexiones entre componentes

## Diagrama eléctrico

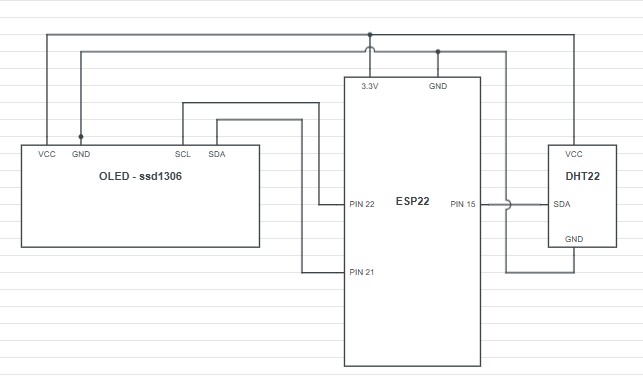


Imagen 4. Captura del diagrama eléctrico del sistema

## Conexiones principales

**ESP32:** Utilizado como microcontrolador central, es el encargado de leer los datos del sensor de temperatura y humedad, procesa estos datos y a través de una comunicación por protocolo I2C los muestra por el display OLED, además este microcontrolador permite mediante una conexión a red wifi publicar estos datos en un servidor MQTT. Los pines utilizados fueron los siguientes:

**GPIO15:** entrada digital para el DHT22.  
 **GPIO22:** salida de reloj I²C (SCL)

**GPIO21:** entrada/salida de datos I²C (SDA)

**DHT22:** Es un sensor digital de temperatura y humedad que entrega los datos de forma serial a través de un único pin de señal (**SDA**). Este módulo cuenta con 3 pines principales:

**VCC:** conectado a 3.3 V del ESP32 para su alimentación.  
**GND:** conectado a masa (GND) del ESP32.  
**SDA:** conectado al **GPIO15** del ESP32, configurado como entrada digital para recibir las lecturas de temperatura y humedad.

El sensor ya incluye la resistencia pull-up necesaria en su línea de datos, por lo que no es necesario agregar componentes adicionales para su funcionamiento.

**Pantalla OLED** (SSD1306 I2C): La pantalla OLED basada en el controlador SSD1306 utiliza el protocolo de comunicación I2C,el cual permite manejar dispositivos con solo 2 líneas de datos. En este proyecto hemos conectado los pines de la siguiente manera:

**VCC**: 3.3V del ESP32

**GND**: a masa (GND del ESP32)

**SDA**: al GPIO21 del ESP32 configurado como línea de datos I2C

**SCL**: al GPIO22 del ESP32 configurado como línea de reloj I2C

## Código Fuente en MicroPython

A continuación se muestra el código utilizado, comentado y estructurado para su correcta interpretación.

import network  
import time  
from time import sleep  
from machine import Pin, SoftI2C  
import dht  
import urandom  
from umqtt.simple import MQTTClient  
from ssd1306 import SSD1306\_I2C  
  
# Parámetros de conexión MQTT  
MQTT\_CLIENT\_ID = "micropython-luces-proyecto1"  
MQTT\_BROKER = "broker.mqttdashboard.com"  
MQTT\_TOPIC = "estacion-meteorologica-ESP32"  
  
SSID = "Wokwi-GUEST"  
PASSWORD = ""  
  
# Estado de conexión  
mqtt\_activo = False  
client = None  
  
def conectar\_wifi():  
 wlan = network.WLAN(network.STA\_IF)  
 wlan.active(True)  
 wlan.connect(SSID, PASSWORD)  
 for \_ in range(20):  
 if wlan.isconnected():  
 print("Conectado a WiFi:", wlan.ifconfig())  
 return True  
 sleep(0.5)  
 print("No se pudo conectar a WiFi. Continuando sin MQTT.")  
 return False  
  
def conectar\_servidor():  
 global client, mqtt\_activo  
 print("Conectando al servidor MQTT... ", end="")  
 try:  
 client = MQTTClient(MQTT\_CLIENT\_ID, MQTT\_BROKER)  
 client.connect()  
 mqtt\_activo = True  
 print("Conectado!")   
 except Exception as e:  
 print("Error al conectar al servidor MQTT:", e)  
 mqtt\_activo = False  
  
# Sensor DHT22 en GPIO15  
sensor = dht.DHT22(Pin(15))

# Pantalla OLED I2C  
i2c = SoftI2C(sda=Pin(21), scl=Pin(22), freq=400000)  
oled = SSD1306\_I2C(128, 64, i2c)  
  
def show\_text(lines):  
 oled.fill(0)  
 y = 0  
 for line in lines[:6]:  
 oled.text(line, 0, y)  
 y += 10  
 oled.show()  
  
# Simulación de presión  
PRS\_MIN = 980.0  
PRS\_MAX = 1030.0  
prs = 1013.2  
  
def simulate\_pressure(prev):  
 delta = (urandom.getrandbits(3) - 4) \* 0.1  
 new = prev + delta  
 if new < PRS\_MIN:  
 new = PRS\_MIN + 0.2  
 elif new > PRS\_MAX:  
 new = PRS\_MAX - 0.2  
 return new  
  
# Inicialización  
show\_text(["Iniciando...", "DHT + OLED + BMP180 (Simulado)"])  
conectar\_wifi()  
conectar\_servidor()  
  
READ\_MS = 2000  
PRESSURE\_UPDATE\_MS = 10000  
last\_read\_ms = 0  
last\_pressure\_ms = 0  
ultima\_temp = None  
ultima\_hum = None  
ultima\_prs = None  
  
while True:  
 now = time.ticks\_ms()  
  
 if time.ticks\_diff(now, last\_pressure\_ms) >= PRESSURE\_UPDATE\_MS:  
 prs = round(simulate\_pressure(prs), 1)  
 last\_pressure\_ms = now  
  
 if time.ticks\_diff(now, last\_read\_ms) >= READ\_MS:  
 last\_read\_ms = now  
 try:  
 sensor.measure()  
 t = round(sensor.temperature(), 1)  
 h = round(sensor.humidity(), 1)  
  
 if (  
 (t != ultima\_temp) or   
 (h != ultima\_hum) or   
 (ultima\_prs is None or abs(prs - ultima\_prs) >= 0.5)  
 ):  
 show\_text([  
 "Temp: {:.1f} C".format(t),  
 "Hum: {:.1f} %".format(h),  
 "Pres: {:>6.1f} hPa".format(prs),  
 " ",  
 "Actualizando..."  
 ])  
 if mqtt\_activo:  
 try:  
 mensaje = '{{"temperatura": {:.1f}, "humedad": {:.1f}, "presion": {:.1f}}}'.format(t, h, prs)  
 client.publish(MQTT\_TOPIC, mensaje)  
 print("Publicado en MQTT:", mensaje)  
 except Exception as e:  
 print("Error al publicar en MQTT:", e)  
  
 ultima\_temp = t  
 ultima\_hum = h  
 ultima\_prs = prs  
  
 except OSError as e:  
 show\_text(["Error", str(e)[:16]])  
  
 time.sleep\_ms(10)

## Explicación del Funcionamiento del Código

### **1. Conexión a WiFi y servidor MQTT**

En la etapa inicial, se simula la conexión a una red Wifi del ESP32 definiendo la red: **Wokwi-GUEST**.  
 Una vez establecida la conexión, se procede a vincular el dispositivo con un **servidor MQTT público** (broker.mqttdashboard.com).  
 Esto permite **publicar datos en la nube**, para que puedan ser consultados desde aplicaciones externas.  
 En caso de que la conexión falle, el sistema continúa funcionando en modo local.

### **2. Lectura de sensores**

El sistema utiliza un **sensor DHT22** conectado al pin GPIO15 para medir dos variables ambientales:

* **Temperatura (°C)**
* **Humedad relativa (%)**

Cada **2 segundos** se realiza una nueva lectura, garantizando una actualización frecuente y precisa de los datos.

### **3. Simulación de presión atmosférica**

Como en la simulación no se dispone de un sensor barométrico real, se implementa una función que genera valores de **presión atmosférica simulada**.  
 Estos valores se encuentran en el rango de **980 a 1030 hPa**, con pequeñas variaciones aleatorias cada **10 segundos**, lo que imita un comportamiento natural y realista.

Para la aplicación real se utiliza un sensor **BMP180**, este módulo requiere conexión vcc (3.3V), GND y posee una salida analogica (SDA) al igual que el DHT22 el cual se conectaria a un pin del ESP32 configurado como entrada analogica, ademas posee un pin de sincronización tipo I2C que se conectaria junto al PIN 22 al display OLED

### **4. Visualización en pantalla OLED**

Los datos obtenidos se muestran en una pantalla **OLED SSD1306** conectada mediante comunicación I2C (conexión bidireccional que se utiliza para sincronización del ESP32 con los componentes que así lo requieran).  
 En cada actualización se presentan los valores actuales de temperatura, humedad y presión en un formato legible, además de mensajes de inicio o error en caso de fallos.

### **5. Publicación de datos en MQTT**

Cada vez que se detecta un **cambio significativo** en los valores de temperatura, humedad o presión, se publica un mensaje en formato **JSON** dentro del servidor MQTT en el tópico:

1. estacion-meteorologica-ESP32

Ejemplo del mensaje transmitido:

1. {"temperatura": 25.3, "humedad": 55.1, "presión": 1012.8}

Esto permite que otros dispositivos o plataformas IoT reciban y procesen los datos en tiempo real.

### 

### **6. Bucle principal**

El programa funciona en un bucle infinito, en el cual se ejecutan las siguientes tareas:

1. Leer temperatura y humedad cada 2 segundos.
2. Simular la presión atmosférica cada 10 segundos.
3. Actualizar la pantalla OLED con la información más reciente.
4. Publicar en el servidor MQTT únicamente si se producen cambios relevantes.

## Lógica de Programación

1. **Entradas:**
   * Sensor DHT22 → temperatura y humedad.
   * Función simulada → presión atmosférica.
2. **Proceso:**
   * Leer valores periódicamente.
   * Comparar con la última lectura para verificar si hubo cambios significativos.
3. **Salidas:**
   * Mostrar los datos en pantalla OLED.
   * Publicar los datos en formato JSON al broker MQTT (solo si cambian).

Base de datos

Para almacenar las mediciones de temperatura, humedad y presión enviadas por el ESP32, se diseñó una base de datos relacional utilizando **MySQL Server** como motor de base de datos. La interfaz gráfica **MySQL Workbench** permitió definir la estructura de manera visual y ejecutar consultas para validar su funcionamiento.

**1.Creacion de la base de datos**

Decidimos nombrar la base de datos como **estación\_meteorologica**

El código SQL utilizado fue el siguiente:

**CREATE DATABASE** estacion\_meteorologica;

#### **2. Tabla de Mediciones**

Dentro de la base de datos, se definió una tabla llamada **mediciones** para almacenar las lecturas del sensor. La estructura de la tabla incluye:

* **id**: identificador único de cada medición, configurado como **AUTO\_INCREMENT** para generar automáticamente un valor único.
* **fecha**: timestamp que registra automáticamente la fecha y hora de la **medición** mediante CURRENT\_TIMESTAMP.
* **temperatura**: valor de temperatura en grados Celsius, tipo **FLOAT**.
* **humedad**: valor de humedad relativa en porcentaje, tipo **FLOAT**.
* **presion**: valor de presión atmosférica en hPa, tipo **FLOAT**.

El código SQL utilizado fue el siguiente:

**USE** estacion\_meteorologica;

**CREATE** **TABLE** mediciones (

id **INT AUTO\_INCREMENT PRIMARY KEY**,

fecha **TIMESTAMP DEFAULT CURRENT\_TIMESTAMP**,

temperatura **FLOAT NOT NULL**,

humedad **FLOAT NOT NULL**,

presion **FLOAT NOT NULL**

);

#### **3. Creación del Usuario de Base de Datos**

Se creó un usuario específico para el backend de la estación meteorológica, con permisos limitados a la base de datos estacion\_meteorologica.

**Usuario**: estación

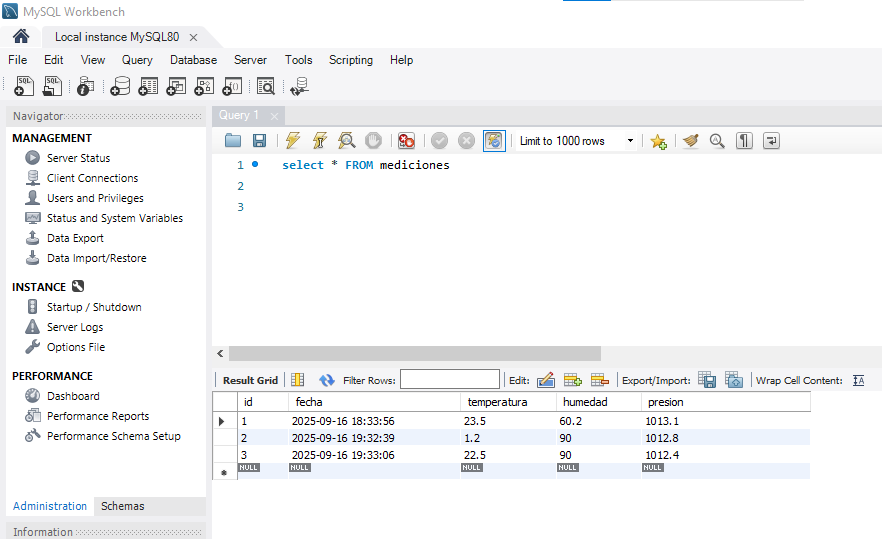
**Contraseña**: 1234abc!

El código SQL utilizado fue el siguiente:

**CREATE USER** 'estacion'@'localhost' **IDENTIFIED BY** '1234abc!';

**GRANT ALL PRIVILEGES ON** estacion\_meteorologica.\* **TO** 'estacion'@'localhost';

**FLUSH PRIVILEGES**;

De esta manera, se garantiza que el software de backend puede acceder a la base de datos para insertar las mediciones sin utilizar el usuario administrador root, aumentando la seguridad del sistema.

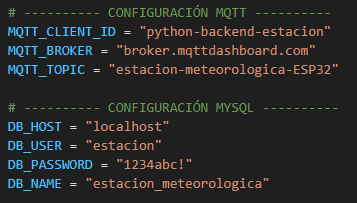
Desarrollo de Software de Control (Backend)

El backend de la estación meteorológica se desarrolló en Python con el objetivo de recibir los datos enviados por el ESP32 a través del broker MQTT y almacenarlos automáticamente en la base de datos MySQL.

El desarrollo se estructuró en distintos bloques funcionales:

**1. Configuración de Conexión**

Se definieron las configuraciones necesarias para conectarse tanto al broker MQTT como a la base de datos MySQL, incluyendo parámetros de usuario, contraseña, tópicos y nombre del cliente. Esto permite que el software pueda establecer comunicación con los dispositivos y la base de datos de manera centralizada.



**2. Funciones de Base de Datos**

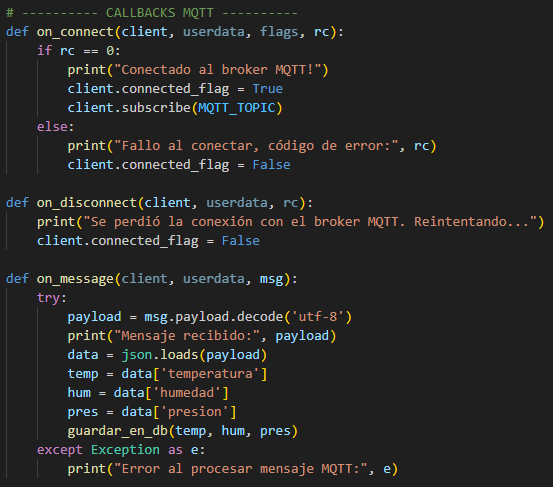
Se crearon funciones específicas para la gestión de la base de datos:

* Conexión a MySQL: se establece la conexión con la base de datos utilizando los parámetros configurados.
* Inserción de datos: cada vez que se recibe una medición válida del ESP32, se ejecuta una función que inserta la temperatura, humedad y presión en la tabla mediciones.
* Manejo de errores: se implementaron mecanismos para mostrar mensajes de error en consola si la conexión o la inserción fallan, asegurando transparencia y facilidad de depuración.

**3. Callbacks de MQTT**

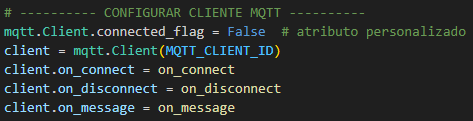
El software utiliza funciones de callback para reaccionar automáticamente a eventos de MQTT:

* **on\_connect**: se ejecuta al conectarse al broker, suscribiéndose al tópico correspondiente y mostrando un mensaje de confirmación.
* **on\_disconnect**: se ejecuta si la conexión se pierde, mostrando un mensaje de alerta y activando la reconexión automática.
* **on\_message**: se ejecuta cada vez que llega un mensaje del ESP32, decodifica el JSON recibido y llama a la función de inserción en la base de datos.



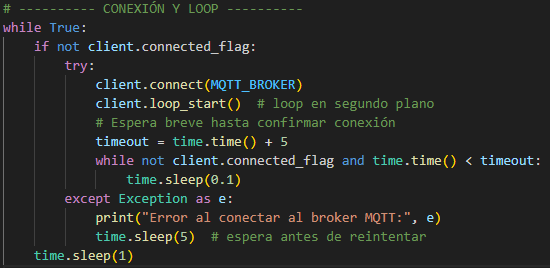
#### **4. Configuración del Cliente MQTT**

Se crea el cliente MQTT, se asignan los callbacks y se inicializa la bandera de conexión:

****

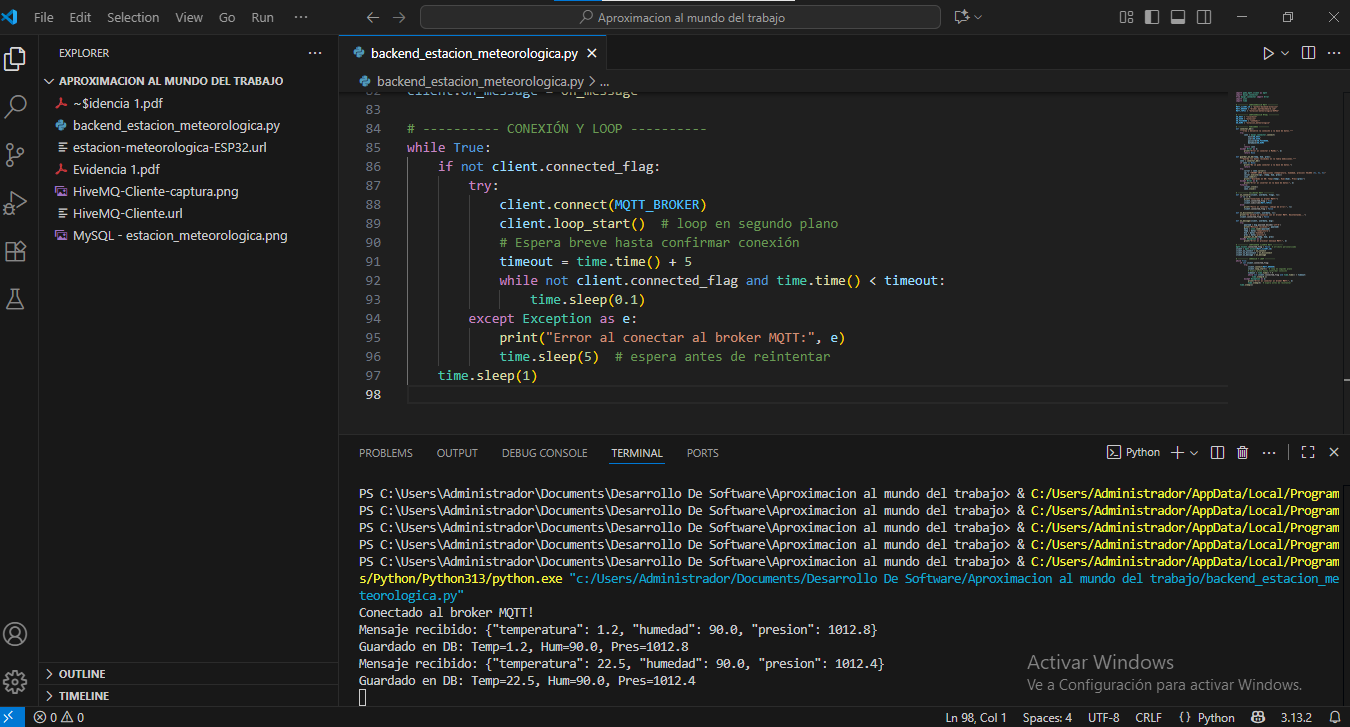
**5. Loop Principal y Reconexión**

El backend mantiene un loop principal que:

* Verifica si la conexión con el broker MQTT está activa.
* Si la conexión se pierde, intenta reconectarse automáticamente después de un breve intervalo de tiempo.
* Permite que el cliente MQTT procese los mensajes recibidos continuamente, asegurando que ninguna medición se pierda mientras la aplicación esté en ejecución.

**6. Registro de Mensajes y Depuración**

Durante la ejecución, el backend muestra en consola:

* Cuando se conecta o se reconecta al broker.
* Cada mensaje recibido desde el ESP32 con su contenido.
* Confirmación de que los datos se han guardado correctamente en la base de datos o los errores que puedan ocurrir.

## 

## Resultados Obtenidos y Pruebas Realizadas

* Las lecturas del DHT22 fueron consistentes en el simulador.
* La simulación de presión atmosférica mostró variaciones graduales dentro de un rango realista.
* El sistema publicó correctamente los datos al servidor MQTT, donde pudieron observarse los valores actualizados.
* La pantalla OLED mostró en tiempo real los parámetros atmosféricos.

## Posibles Mejoras Futuras

* Reemplazar la simulación de presión con un sensor real (BMP180/BMP280) en hardware físico.
* Agregar un sensor de luz, lluvia o viento.
* Almacenar los datos en una base de datos (como Firebase o Google Sheets).
* Añadir un botón para forzar la actualización manual de los datos.

## Conclusiones

Este proyecto integró varios elementos importantes del desarrollo embebido: sensores físicos, visualización en pantalla, simulación de sensores ausentes y comunicación en red mediante MQTT. A pesar de la limitación del simulador respecto a la medición de presión, se logró una simulación convincente, y se desarrolló un prototipo completo de una estación meteorológica básica.

El trabajo en equipo, la creatividad para superar limitaciones y el uso combinado de hardware y software fueron aspectos clave del desarrollo exitoso de este proyecto.