TP 4 A-B

Introducción

Este documento describe la implementación y configuración de un sistema MQTT utilizando los archivos mqtt_pub_r1.py, mqtt_sub_r1.py, y tabla_datos.html. Además, se incluyen detalles sobre la configuración del broker Mosquitto y la comunicación entre los componentes del sistema.

Configuración de Mosquitto

Para iniciar el broker Mosquitto en macOS:

brew services start mosquitto

El resultado esperado es el siguiente:

Successfully started `mosquitto` (label: homebrew.mxcl.mosquitto)
mariaeugeniaq@MacBook-Pro-de-Eugenia mosquitto %

1720746908: mosquitto version 2.0.18 starting
1720746908: Config loaded from /opt/homebrew/etc/mosquitto/mosquitto.conf.
1720746908: Opening ipv4 listen socket on port 1884.
1720746908: mosquitto version 2.0.18 running

Conexión de Clientes

Ejemplo de conexión y publicación de datos:

mosquitto_pub -h localhost -p 1884 -t sitio1/temperatura -m "Sitio1 Temp. = 22 C" -u your_username -P 11022004

mosquitto_pub -h localhost -p 1884 -t sitio1/temperatura -m "Sitio1 Temp. = 23 C" -u your_username -P 11022004

Suscripción a un tema:

mosquitto_sub -h localhost -p 1884 -t sitio1/temperatura -u your_username -P 11022004

Sitio1 Temp. = 22 C

Sitio1 Temp. = 23 C

Archivo mqtt_pub_r1.py

En el archivo mqtt_pub_r1.py, se implementa un servidor que maneja las peticiones relacionadas con la base de datos y la publicación de datos en un tema MQTT.

Configuración de la Base de Datos

El servidor utiliza SQLite para almacenar los datos de los sensores. La configuración se realiza de la siguiente manera:

```
import os

import json

import sqlite3

import threading

import time

import logging

from flask import Flask, render_template, jsonify, request

import paho.mqtt.client as mqtt



logging.basicConfig(level=logging.DEBUG, format='%(asctime)s
%(levelname)s: %(message)s')
```

```
app = Flask(__name__)



# Configuración de la base de datos

db_path = os.path.join(os.getcwd(), 'datos_sensores.db')

app.config['SQLALCHEMY_DATABASE_URI'] = f"sqlite:///{db_path}"

app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
```

Configuración de MQTT

El servidor está configurado para conectarse a un broker MQTT en el localhost:

```
MQTT_BROKER = "localhost"

MQTT_PORT = 1884

MQTT_TOPIC = "sensores/datos"

MQTT_USER = "your_username"

MQTT_PASSWORD = "11022004"



client = mqtt.Client()

client.username_pw_set(MQTT_USER, MQTT_PASSWORD)
```

Rutas del Servidor

Ruta Principal

La ruta principal sirve el archivo tabla_datos.html:

```
@app.route('/')
```

```
def index():

    return open('tabla_datos.html').read()
```

Ruta para Obtener Datos

Esta ruta obtiene todos los datos de la base de datos:

```
@app.route('/api/datos', methods=['GET'])

def obtener_datos():

    conn = sqlite3.connect('datos_sensores.db')

    cursor = conn.cursor()

    cursor.execute('SELECT * FROM lectura_sensores')

    datos = cursor.fetchall()

    conn.close()

    return jsonify(datos)
```

Ruta para Añadir Datos

Esta ruta permite añadir nuevos datos a la base de datos:

@app.route('/api/añadir', methods=['POST'])

```
@app.route('/api/añadir', methods=['POST'])

def añadir_dato():

    try:

        datos = request.get_json()

        co2 = datos['co2']

        temp = datos['temp']
```

```
        hum = datos['hum']

        fecha = datos['fecha']

        lugar = datos['lugar']

        altura = datos['altura']

        presion = datos['presion']

        presion_nm = datos['presion_nm']

        temp_ext = datos['temp_ext']



        # Utilizar un contexto de conexión para manejar la conexión y
cierre automáticamente

        with sqlite3.connect('datos_sensores.db') as conn:

            cursor = conn.cursor()

            cursor.execute('''

                INSERT INTO lectura_sensores (co2, temp, hum, fecha,
lugar, altura, presion, presion_nm, temp_ext)

                VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)

            ''', (co2, temp, hum, fecha, lugar, altura, presion,
presion_nm, temp_ext))

            conn.commit()  # Finaliza la transacción



        # Publicar el nuevo dato a MQTT

        data = {

            'co2': co2,

            'temp': temp,

            'hum': hum,

            'fecha': fecha,
```

```
            'lugar': lugar,
            'altura': altura,
            'presion': presion,
            'presion_nm': presion_nm,
            'temp_ext': temp_ext
        }
        client.publish(MQTT_TOPIC, json.dumps(data))
        logging.debug(f"Datos publicados: {data}")

        return jsonify({'status': 'success'})

    except Exception as e:
        logging.error(f"Error al añadir dato: {e}")
        return jsonify({'status': 'error', 'message': 'Error al
procesar la solicitud.'}), 500
```

Publicación de Datos en MQTT

El código también incluye funciones para publicar datos en el tema MQTT configurado.

Archivo mqtt_sub_r1.py

El archivo mqtt_sub_r1.py se encarga de suscribirse al tema MQTT y gestionar los mensajes recibidos.

Configuración de MQTT

La configuración para el cliente suscriptor es similar a la del publicador:

```
# Configuración de MQTT

MQTT_BROKER = "localhost"

MQTT_PORT = 1884

MQTT_TOPIC = "sensores/datos"

MQTT_USER = "your_username"

MQTT_PASSWORD = "11022004"



client = mqtt.Client()

client.username_pw_set(MQTT_USER, MQTT_PASSWORD)
```

Funciones de Gestión de Conexión

El archivo incluye funciones para gestionar la conexión y la desconexión del cliente MQTT:

```
def on_connect(client, userdata, flags, rc):

    if rc == 0:

        logging.info("Conectado al broker MQTT")

        client.subscribe(MQTT_TOPIC)

        logging.info("Suscripto al tema MQTT")

    else:
```

```
        logging.error(f"Conexión fallida con código de resultado:
{rc}")



def on_message(client, userdata, msg):

    try:

        data=json.loads(msg.payload.decode())

        conn = sqlite3.connect('datos_sensores_sub.db')

        cursor = conn.cursor()

        for row in data:

            cursor.execute('''SELECT * FROM lectura_sensores WHERE co2
= ? AND temp = ? AND hum = ? AND fecha = ? AND lugar = ? AND altura = ?
AND presion = ? AND presion_nm = ? AND temp_ext = ?''',

                           (row["co2"], row["temp"], row["hum"],
row["fecha"], row["lugar"], row["altura"], row["presion"],
row["presion_nm"], row["temp_ext"]))

            registro_existente = cursor.fetchone()



            if registro_existente:

                logging.info("Registro ya existe en la base de datos")

            else:

                cursor.execute('''

                    INSERT OR IGNORE INTO lectura_sensores (id, co2,
temp, hum, fecha, lugar, altura, presion, presion_nm, temp_ext)

                    VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

                ''', (row["id"], row["co2"], row["temp"], row["hum"],
row["fecha"], row["lugar"], row["altura"], row["presion"],
row["presion_nm"], row["temp_ext"]))

                conn.commit()
```

```
                logging.info("Nuevo registro añadido a la base de
datos")



        conn.close()



        logging.debug(f"Mensaje recibido en tópico {msg.topic}:
{msg.payload.decode()}")

    except Exception as e:

        logging.error(f"Error al leer/añadir inf publicada{e}")
```

Archivo tabla_datos.html

El archivo tabla_datos.html proporciona una interfaz web para visualizar y gestionar los datos almacenados.

Estructura HTML

La estructura del archivo HTML es la siguiente:

```
<!DOCTYPE html>

<html>

<head>

    <title>Datos de Sensores</title>

</head>

<body>

    <h1>Datos de Sensores</h1>

 <tbody id="tabla-datos">

            <!-- Aquí se insertarán las filas de la tabla -->
```

```
        </tbody>
    </table>
    <script>
      // JavaScript para gestionar la tabla de datos...
    </script>
</body>
</html>
```

Funciones JavaScript

El archivo incluye funciones JavaScript para cargar, añadir y eliminar datos:

```
function cargarDatos() {
            fetch('/api/datos')
                .then(response => response.json())
                .then(data => {
                    var tabla = document.getElementById('tabla-datos');
                    tabla.innerHTML = ''; // Limpiar tabla antes de
actualizar

                    data.forEach(dato => {
                        var row = tabla.insertRow();
                        row.innerHTML = `
                            <td>${dato[0]}</td>
                            <td>${dato[1]}</td>
                            <td>${dato[2]}</td>
```

```
                            <td>${dato[3]}</td>
                            <td>${dato[4]}</td>
                            <td>${dato[5]}</td>
                            <td>${dato[6]}</td>
                            <td>${dato[7]}</td>
                            <td>${dato[8]}</td>
                            <td>${dato[9]}</td>
                            <td><button
onclick="eliminarDato(${dato[0]})">Eliminar</button></td>
                        `;
                    });
                })
                .catch(error => {
                    console.error('Error al cargar datos:', error);
                    alert('Hubo un error al cargar los datos. Por
favor, revisa la consola para más detalles.');
                });
        }

        // Función para agregar un nuevo dato
        function agregarDato() {
            var id = document.getElementById('id').value;
            var co2 = document.getElementById('co2').value;
            var temp = document.getElementById('temp').value;
            var hum = document.getElementById('hum').value;
            var fecha = document.getElementById('fecha').value;
```

```
            var lugar = document.getElementById('lugar').value;
            var altura = document.getElementById('altura').value;
            var presion = document.getElementById('presion').value;
            var presion_nm =
document.getElementById('presion_nm').value;
            var temp_ext = document.getElementById('temp_ext').value;
            fetch('/api/añadir', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json',
                },
                body: JSON.stringify({
                    id: id, co2: co2, temp: temp, hum: hum, fecha:
fecha,
                    lugar: lugar, altura: altura, presion: presion,
                    presion_nm: presion_nm, temp_ext: temp_ext
                }),
            }).then(response => response.json())
              .then(data => {
                  console.log('Success:', data);
                  cargarDatos(); // Recargar la tabla de datos
              })
              .catch((error) => {
                  console.error('Error al agregar dato:', error);
                  alert('Hubo un error al agregar el dato. Por favor,
revisa la consola para más detalles.');
```

```
            });
        }

        // Función para eliminar un dato
        function eliminarDato(id) {
            fetch('/api/eliminar', {
                method: 'DELETE',
                headers: {
                    'Content-Type': 'application/json',
                },
                body: JSON.stringify({ id: id }),
            }).then(response => response.json())
              .then(data => {
                  console.log('Success:', data);
                  cargarDatos(); // Recargar la tabla de datos
              })
              .catch((error) => {
                  console.error('Error al eliminar dato:', error);
                  alert('Hubo un error al eliminar el dato. Por favor, revisa la consola para más detalles.');
              });
        }
        // Cargar datos al cargar la página
        cargarDatos();
```

Conclusión

En este informe, hemos descrito la implementación de un sistema MQTT utilizando Flask para el manejo de peticiones y SQLite para el almacenamiento de datos. La interfaz web desarrollada permite la visualización y gestión de los datos de sensores de manera eficiente.