PDI - TP 3 - A - RestAPI - Servidores sin control de estado - Gestión de la Base de Datos (Entrega 10/06/2024)

Instalar SQLite.

Instalar Heidi SQL en Windows.

Para Linux y Windows se puede usar DBeaver Community Edition.

También se puede instalar DB Browser for SQLite.

Desde una terminal crear la base de datos, si no existe SQLite la crea automáticamente.

La idea es crear un repositorio de datos para recibir datos de sensores.

Conectarse a la base de datos desde el gestor elegido.

Crear una tabla para recibir los datos de los sensores.

Verificar con el gestor o desde el SQLite desde una terminal que esté todo bien.

Verificar que Python y pip esté instalado.

Acceder a la base de datos de Python para ingresar nuevos valores.

Desde el programa en Python importo las dependencias.

Adaptar el código en github de acuerdo al criterio de diseño elegido.

Notas:

- Hubo varias consultas referidas a que no se desplegaban correctamente los datos.
- La idea de la versión r1 era que para el TP se adapte el código resolviendo las inconsistencias.
- Ahora podán encontrar las versiones r2 con algunas modificaciones
    - sensores_r1 genera una db llamada datos_sendores.db
    - sensor_editar_tabla_r2 agrega nuevas rutas y mensajes.

@app.route('/api/prueba')

@app.route('/')

@app.route('/api/todos-los-datos')

@app.route('/api/primer-registro')

@app.route('/api/directorio-db')

@app.route('/api/insertar-dato')

Verificar y mejorar.

La estructura de los archivos fue:
templates
— tabla_sensores_para_editar.html
funciones.py
main.py

tabla_sensores_para_editar.html:

```
<!DOCTYPE html>
<html>
<head>
 <title>Sensores de prueba</title>
 <link href="https://unpkg.com/gridjs/dist/theme/mermaid.min.css"
rel="stylesheet" />
 <style>
   body {
     font-family: Sans-Serif;
   }
 </style>
</head>
<body>
 <div>
   <h1>Sensores de prueba</h1>
   <hr>
   <!-- Table Container -->
   <div id="table"></div>
 </div>
 <script src="https://unpkg.com/gridjs/dist/gridjs.umd.js"></script>
 <script>
   const tableDiv = document.getElementById('table');

   new gridjs.Grid({
     columns: [
       { id: 'id', name: 'ID' },
       { id: 'co2', name: 'CO2' },
       { id: 'temp', name: 'Temp' },
       { id: 'hum', name: 'Hum' },
       { id: 'fecha', name: 'Fecha' },
```

```
        { id: 'lugar', name: 'Lugar' },
        { id: 'altura', name: 'Altura' },
        { id: 'presion', name: 'Presión' },
        { id: 'presion_nm', name: 'Presión NM' },
        { id: 'temp_ext', name: 'Temp Ext' },
      ],
      server: {
        url: '/api/todos-los-datos',
        then: data => data.map(row => [
          row.id, row.co2, row.temp, row.hum, row.fecha, row.lugar,
row.altura, row.presion, row.presion_nm, row.temp_ext
        ]),
        total: data => data.length,
      },
      search: {
        enabled: true
      },
      sort: {
        enabled: true,
        multiColumn: true
      },
      pagination: {
        enabled: true,
        limit: 10
      }
    }).render(tableDiv);
  </script>
</body>
</html>
```

funciones.py:

```
import requests
import json

def get_location():
    try:
        response = requests.get('https://ipinfo.io/json')
        response.raise_for_status()
        data = response.json()
        lat, lon = map(float, data['loc'].split(','))
```

```
        print(f"Lat = {lat}, Lon = {lon}")
        return lat, lon
    except requests.exceptions.RequestException as e:
        print(f"Error al obtener la ubicación: {e}")
        return None, None

def geo_latlon():
    lat, lon = get_location()
    if lat is None or lon is None:
        return

    api_key = "2f66bd561ebc7e4bde0d2a8951df0098"
    base_url = "http://api.openweathermap.org/data/2.5/weather?"

    opcion_elegida = input("Ciudad o Geo? (escriba 'ciudad' o 'geo'):
").strip().lower()

    if opcion_elegida == 'ciudad':
        city_name = input("Ciudad: ")
        complete_url =
f"{base_url}appid={api_key}&q={city_name}&units=metric&lang=es"
    elif opcion_elegida == 'geo':
        complete_url =
f"{base_url}lat={lat}&lon={lon}&appid={api_key}&units=metric&lang=es"
    else:
        print("Opción no válida.")
        return

    try:
        response = requests.get(complete_url)
        response.raise_for_status()
        x = response.json()

        if x["cod"] != "404" and x["cod"] != "400":
            y = x["main"]
            temp_ext = y["temp"]
            presion = y["pressure"]
            humedad_ext = y["humidity"]
            z = x["weather"]
            descripcion_clima = z[0]["description"]

            print(f"Temperatura = {temp_ext} C\nPresión Atmosférica =
{presion} hPa\nHumedad = {humedad_ext} %\nCielo = {descripcion_clima}")
```

```
            return temp_ext, presion, humedad_ext, descripcion_clima
        else:
            print("Ciudad no encontrada")
    except requests.exceptions.RequestException as e:
        print(f"Error al obtener datos del clima: {e}")
```

main.py:

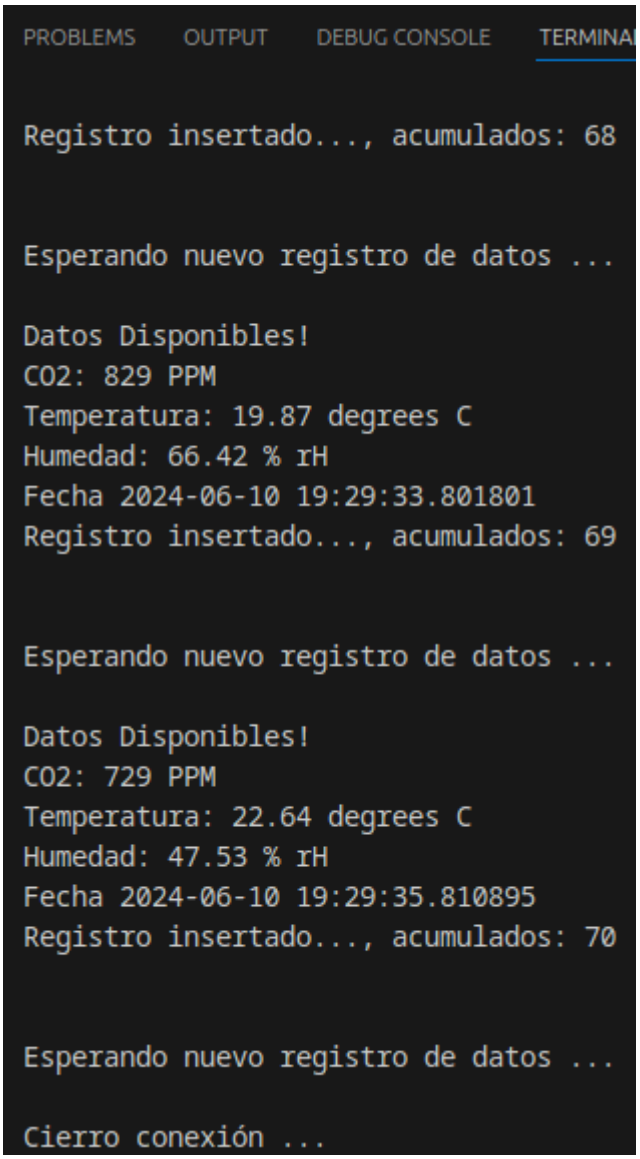
```
import os
import time
import random
import sqlite3
from flask import Flask, render_template, jsonify
from datetime import datetime
from funciones import geo_latlon

app = Flask(__name__)

DB_PATH = '/home/pjcdz/tp3.db'  # Cambia la ruta a la nueva ruta

def create_table():
    if not os.path.exists(DB_PATH):
        try:
            conn = sqlite3.connect(DB_PATH)
            cursor = conn.cursor()
            cursor.execute('''CREATE TABLE lectura_sensores (
                                id INTEGER PRIMARY KEY,
                                co2 REAL,
                                temp REAL,
                                hum REAL,
                                fecha TEXT,
                                lugar TEXT,
                                altura REAL,
                                presion REAL,
                                presion_nm REAL,
                                temp_ext REAL
                            )''')
            conn.commit()
        except sqlite3.Error as e:
            print(f"Error creating table: {e}")
        finally:
            conn.close()
```

```
def get_db_records():
    try:
        conn = sqlite3.connect(DB_PATH)
        cursor = conn.cursor()
        cursor.execute('SELECT * FROM lectura_sensores')
        records = cursor.fetchall()
    except sqlite3.Error as e:
        print(f"Error fetching records: {e}")
        records = []
    finally:
        conn.close()
    return records

@app.route('/')
def index():
    create_table()  # Asegura que la tabla exista antes de realizar la
consulta
    records = get_db_records()
    return render_template('tabla_sensores_para_editar.html',
records=records)

@app.route('/datos')
def datos():
    create_table()  # Asegura que la tabla exista antes de realizar la
consulta
    records = get_db_records()
    return jsonify([{
        'co2': record[1],
        'temp': record[2],
        'hum': record[3],
        'fecha': record[4],
        'lugar': record[5],
        'altura': record[6],
        'presion': record[7],
        'presion_nm': record[8],
        'temp_ext': record[9]
    } for record in records])

@app.route('/api/todos-los-datos', methods=['GET'])
def get_data():
    create_table()  # Asegura que la tabla exista antes de realizar la
consulta
```

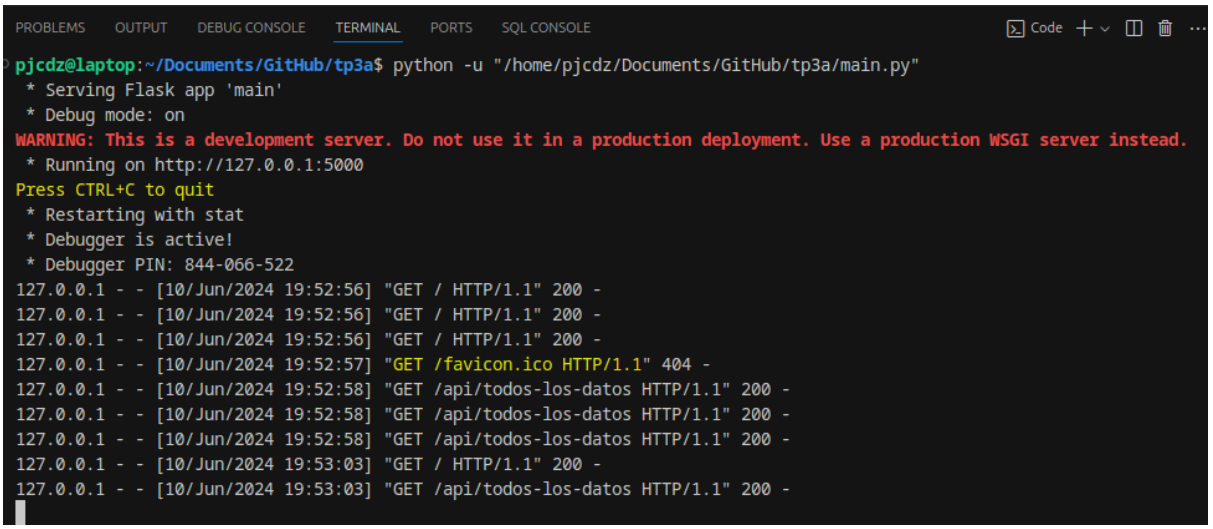
```
    records = get_db_records()
    data = [{
        'id': record[0],
        'co2': record[1],
        'temp': record[2],
        'hum': record[3],
        'fecha': record[4],
        'lugar': record[5],
        'altura': record[6],
        'presion': record[7],
        'presion_nm': record[8],
        'temp_ext': record[9]
    } for record in records]
    return jsonify(data)

def capturar_datos():
    temp_ext, presion, humedad_ext, descripcion_clima = geo_latlon()
    print("Resultados= ", temp_ext, presion, humedad_ext,
descripcion_clima)

    while True:
        try:
            lugar = input("Lugar de la captura de los datos: ")
            tipo_lugar = input("Tipo de lugar [au=abierto urbano]
[an=abierto no urbano] [c=cerrado] ")
            superficie = int(input("Superficie aproximada del lugar
[m2]: "))
            altura = int(input("Altura aproximada del lugar [m]: "))
            presion_nm = presion
            cant_capturas = int(input("Cantidad de capturas: "))
            delta_t_capturas = int(input("Tiempo entre capturas (segs) :
"))
        except ValueError:
            print("Error al ingresar datos...")
            continue
        else:
            break

    cont = 0
    while cont < cant_capturas:
        cont += 1
        verdadero = 1
        if verdadero == 1:
```

```
            print("Datos Disponibles!")
            CO2_medido = random.uniform(250, 1100)
            temp_sensor = random.uniform(temp_ext, temp_ext + 10)
            humedad_relativa = random.uniform(40, 80)
            print("CO2: %d PPM" % CO2_medido)
            print("Temperatura: %0.2f degrees C" % temp_sensor)
            print("Humedad: %0.2f %% rH" % humedad_relativa)

            d = datetime.now()
            print("Fecha", d)
            timestampStr = d.strftime("%d-%b-%Y (%H:%M:%S.%f)")

            conn = sqlite3.connect(DB_PATH)
            cursor = conn.cursor()
            cursor.execute('''INSERT INTO lectura_sensores (co2, temp,
hum, fecha, lugar, altura, presion, presion_nm, temp_ext)
                              VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)''',
                           (CO2_medido, temp_sensor, humedad_relativa,
timestampStr, lugar, altura, presion, presion_nm, temp_ext))
            conn.commit()
            conn.close()

            print("Registro insertado..., acumulados:", cont, "\n")
            time.sleep(delta_t_capturas)
            print("\nEsperando nuevo registro de datos ...\n")

    print("Cierro conexión ...")

if __name__ == '__main__':
    create_table()
    # capturar_datos()
    app.run(debug=True)
```

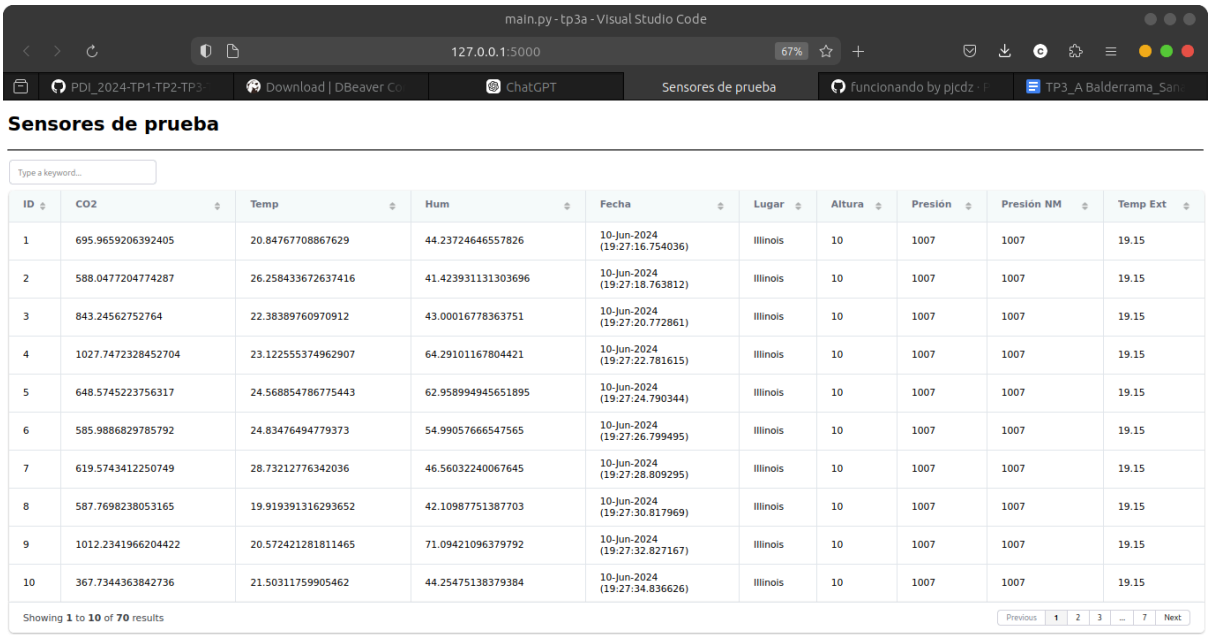
Registro de datos:



Ejecucion main.py



Print del html en http://127.0.0.1:5000/:



ID	CO2	Temp	Hum	Fecha	Lugar	Altura	Presión	Presión NM	Temp Ext
1	695.9659206392405	20.84767708867629	44.23724646557826	10-Jun-2024 (19:27:16.754036)	Illinois	10	1007	1007	19.15
2	588.0477204774287	26.258433672637416	41.423931131303696	10-Jun-2024 (19:27:18.763812)	Illinois	10	1007	1007	19.15
3	843.24562752764	22.38389760970912	43.00016778363751	10-Jun-2024 (19:27:20.772861)	Illinois	10	1007	1007	19.15
4	1027.7472328452704	23.122555374962907	64.29101167804421	10-Jun-2024 (19:27:22.781615)	Illinois	10	1007	1007	19.15
5	648.5745223756317	24.568854786775443	62.958994945651895	10-Jun-2024 (19:27:24.790344)	Illinois	10	1007	1007	19.15
6	585.9886829785792	24.83476494779373	54.99057666547565	10-Jun-2024 (19:27:26.799495)	Illinois	10	1007	1007	19.15
7	619.5743412250749	28.73212776342036	46.56032240067645	10-Jun-2024 (19:27:28.809295)	Illinois	10	1007	1007	19.15
8	587.7698238053165	19.919391316293652	42.10987751387703	10-Jun-2024 (19:27:30.817969)	Illinois	10	1007	1007	19.15
9	1012.2341966204422	20.572421281811465	71.09421096379792	10-Jun-2024 (19:27:32.827167)	Illinois	10	1007	1007	19.15
10	367.7344363842736	21.50311759905462	44.25475138379384	10-Jun-2024 (19:27:34.836626)	Illinois	10	1007	1007	19.15