TP 3-A

Consignas:

Instalar Heidi SQL en Windows
Para Linux y Windows se puede usar DBeaver Community Edition.
Tambien se puede instalar DB Browser for SQLite.
Desde una terminal crear la base de datos, si no exite 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 de Python importo las dependencias.
Adaptar el codigo 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 version r1 era que para el TP se adapte el codigo resolviendo las inconsistencias.
-Ahora puede encontrar las versiones r2 con algunas modificaciones
 sensores_r1 genera una db llamada datos_sensores.db
 sensores_editar_tabla_r2 agrega nuevas rutas y mensajes

@app.route('/api/prueba')
@app.route('/')
@app.route('/api/primer-registro')
@app.route('/api/directorio-db')
@app.route('/api/insertar-dato')
Verificar y mejorar

sensores.py:

```python
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__)

def create_table():
    conn = sqlite3.connect('sensores.db')
    cursor = conn.cursor()
    cursor.execute('''CREATE TABLE IF NOT EXISTS 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()
    conn.close()

if __name__ == '__main__':
    create_table()

    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('sensores.db')
            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 ...")
```

Se modificó el código para funcionar como un programa para crear la tabla lectura_sensores, añadiendo valores a la tabla mediante el uso de funciones que ya estan hechas en funciones.py

tabla_datos.html:

```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>
      <h2>
        <form action="/api/añadir" method="POST">
          <input type="number" name="CO2" placeholder="CO2">
          <input type="number" name="Temp" placeholder="Temp">
          <input type="number" name="Hum" placeholder="Hum">
          <input type="text" name="Lugar" placeholder="Lugar">
          <input type="number" name="Altura" placeholder="Altura">
          <input type="number" name="Presion" placeholder="Presion">
          <input type="number" name="Presion_nm" placeholder="Presion nm">
          <input type="number" name="Temp_ext" placeholder="Temp ext">

          <input type="submit" value="Añadir">
        </form>
      </h2>
      <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');

      const updateUrl = (prev, query) => {
        return prev + (prev.indexOf('?') >= 0 ? '&' : '?') + new URLSearchParams(query).toString();
```

```
<!--Table Container-->
    <script>
    };

    const editableCellAttributes = (data, row, col) => {
      if (row) {
        return {contentEditable: 'true', 'data-element-id': row.cells[0].data};
      }
      else {
        return {};
      }
    };

    new gridjs.Grid({
      columns: [
        { id: 'id', name: 'id', 'attributes' : editableCellAttributes },
        { id: 'co2', name: 'CO2', 'attributes': editableCellAttributes },
        { id: 'temp', name: 'Temp', 'attributes': editableCellAttributes },
        { id: 'hum', name: 'Hum', 'attributes': editableCellAttributes },
        { id: 'fecha', name: 'Fecha', 'attributes': editableCellAttributes },
        { id: 'lugar', name: 'Lugar', 'attributes': editableCellAttributes },
        { id: 'altura', name: 'Altura', 'attributes': editableCellAttributes },
        { id: 'presion', name: 'Presion', 'attributes': editableCellAttributes },
        { id: 'presion_nm', name: 'Presion nm', 'attributes': editableCellAttributes },
        { id: 'temp_ext', name: 'Temp ext', 'attributes': editableCellAttributes },
        { id: 'eliminar', name: 'Eliminar', formatter: (_, row) => gridjs.html(`<button onclick="eliminar(${row.cells[0].data})">Eliminar</button>`) },
      ],
      server: {
        url: '/api/datos',
        then: results => results.data,
        total: results => results.total,
      },
      search: {
        enabled: true,
        server: {
          url: (prev, search) => {
            return updateUrl(prev, {search});
```

```
<!--Table Container-->
    <script>
      new gridjs.Grid({
        search: {
          server: {
            return updateUrl(prev, {search});
          },
        },
        sort: {
          enabled: true,
          multiColumn: true,
          server: {
            url: (prev, columns) => {
              const columnIds = ['id', 'co2', 'temp', 'hum', 'fecha', 'lugar', 'altura', 'presion', 'presion_nm', 'temp_ext'];
              const sort = columns.map(col => (col.direction === 1 ? '+' : '-') + columnIds[col.index]);
              return updateUrl(prev, {sort});
            },
          },
        },
        pagination: {
          enabled: true,
          server: {
            url: (prev, page, limit) => {
              return updateUrl(prev, {start: page * limit, length: limit});
            },
          },
        },
      }).render(tableDiv);

      let savedValue;

      tableDiv.addEventListener('focusin', ev => {
        if (ev.target.tagName === 'TD') {
          savedValue = ev.target.textContent;
        }
      });
```

```
<!--Table Container-->
    <script>
      });

      function eliminar(id) {
        fetch('/api/eliminar', {
          method: 'DELETE',
          headers: {'Content-Type': 'application/json'},
          body: JSON.stringify({
            id: id.toString(),
          }),
        }).then(location.reload());
      }
    </script>
  </body>
</html>
```

Se añadió un form arriba de la tabla, para que el usuario pueda añadir una nueva entrada a la tabla. Se arregló el código de la grid para que pueda mostrar el id de cada entrada de la tabla. Se agregó una nueva columna a la tabla (‘Eliminar’), dónde hay un botón que llama a la función eliminar() que manda un pedido al servidor, mediante el método DELETE, con el id de la entrada que se desea eliminar. Luego de hacer el pedido, la función eliminar refresca la ventana del usuario para poder observar los nuevos cambios a la tabla.

sensor_editar_tabla.py:

```python
from flask import Flask, render_template, request, abort, jsonify, rec
import time
import random
import sqlite3
from datetime import datetime
from funciones import geo_latlon

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('tabla_sensores_para_editar.html')

@app.route('/api/test')
def test():
    return "Prueba de ruta API..."

@app.route('/api/datos')
def datos():
    conn = sqlite3.connect('sensores.db')
    cursor = conn.cursor()
    cursor.execute('SELECT * FROM lectura_sensores')
    records = cursor.fetchall()
    conn.close()
    data = []
    for record in records:
        data.append(
        {
        'id': record[0],
        'co2': record[1],
        'temp': record[2],
        'hum': record[3],
        'fecha': record[4],
```

```python
def datos():
        'fecha': record[4],
        'lugar': record[5],
        'altura': record[6],
        'presion': record[7],
        'presion_nm': record[8],
        'temp_ext': record[9]
        })

    return jsonify({
        'data': data,
        'total': len(data),
    })

@app.route('/api/primer-registro')
def primerRegistro():
    try:
        conn = sqlite3.connect('sensores.db')
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM lectura_sensores")
        primero = cursor.fetchone()
        conn.close()
        if not primero:
            return jsonify({'mensaje': 'No se encontraron datos'}), 404
        return jsonify(primero)
    except Exception as e:
        return jsonify({'error': 'Error al buscar datos', 'detalle': str(e)}), 500

@app.route('/api/añadir', methods=["POST"])
def añadir():
    try:
        CO2 = request.form.get('CO2')
        Temp = request.form.get('Temp')
        Hum = request.form.get('Hum')
        Lugar = request.form.get('Lugar')
        Altura = request.form.get('Altura')
        Presion = request.form.get('Presion')
```

```python
def añadir():
        Altura = request.form.get('Altura')
        Presion = request.form.get('Presion')
        Presion_nm = request.form.get('Presion_nm')
        Temp_ext = request.form.get('Temp_ext')

        if CO2 or Temp or Hum or Lugar or Altura or Presion or Presion_nm or Temp_ext == None:
            abort(400)

        current_datetime = datetime.now().strftime("%d-%B-%Y (%H:%M:%S)")

        conn = sqlite3.connect('sensores.db')
        cursor = conn.cursor()
        cursor.execute("INSERT INTO lectura_sensores (co2, temp, hum, fecha, lugar, altura, presion, presion_nm, temp_ext) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)",
                       (CO2, Temp, Hum, current_datetime, Lugar, Altura, Presion, Presion_nm, Temp_ext))
        conn.commit()
        conn.close()


        return redirect("/")
    except Exception as e:
        return jsonify({'error': 'Error al añadir datos', 'detalle': str(e)}), 500

@app.route('/api/eliminar', methods=['DELETE'])
def eliminar():
    try:
        data = request.get_json()
        if 'id' not in data:
            abort(400)

        conn = sqlite3.connect('sensores.db')
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM lectura_sensores WHERE id = ?", (data['id']))
        record = cursor.fetchone()
        conn.close()
```

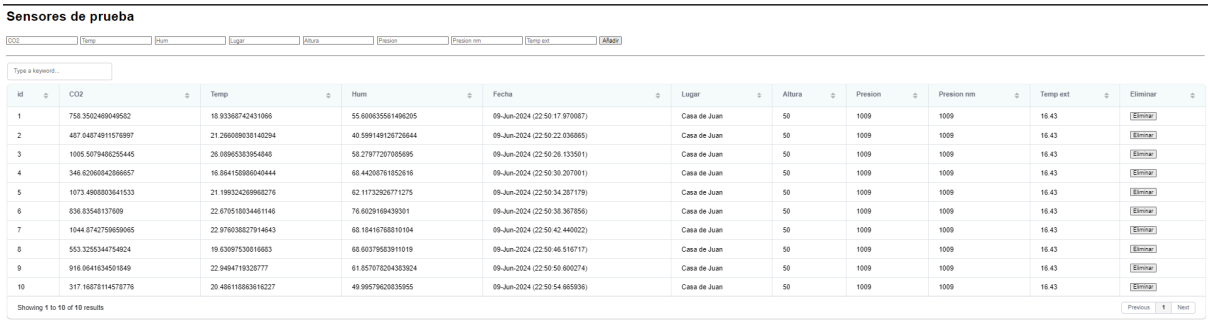
```python
def eliminar():
        if not record:
            abort(404)

        conn = sqlite3.connect('sensores.db')
        cursor = conn.cursor()
        cursor.execute("DELETE FROM lectura_sensores WHERE id = ?", (data['id']))
        conn.commit()
        conn.close()
        return '', 204
    except Exception as e:
        return jsonify({'error': 'Error al buscar datos', 'detalle': str(e)}), 500

@app.route('/api/data', methods=['PUT'])
def update():
    try:
        data = request.get_json()
        if 'id' not in data:
            abort(400)

        conn = sqlite3.connect('sensores.db')
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM lectura_sensores WHERE id = ?", (data['id']))
        record = cursor.fetchone()
        conn.close()

        if not record:
            abort(404)

        if 'lugar' in data:
            new_lugar = data['lugar']
            conn = sqlite3.connect('sensores.db')
            cursor = conn.cursor()
            cursor.execute("UPDATE lectura_sensores SET lugar = ? WHERE id = ?", (new_lugar, data['id']))
            conn.commit()
            conn.close()
```

```python
def update():
    elif 'co2' in data:
        new_co2 = data['co2']
        conn = sqlite3.connect('sensores.db')
        cursor = conn.cursor()
        cursor.execute("UPDATE lectura_sensores SET co2 = ? WHERE id = ?", (new_co2, data['id']))
        conn.commit()
        conn.close()
    elif 'temp' in data:
        new_temp = data['temp']
        conn = sqlite3.connect('sensores.db')
        cursor = conn.cursor()
        cursor.execute("UPDATE lectura_sensores SET temp = ? WHERE id = ?", (new_temp, data['id']))
        conn.commit()
        conn.close()
    elif 'hum' in data:
        new_hum = data['hum']
        conn = sqlite3.connect('sensores.db')
        cursor = conn.cursor()
        cursor.execute("UPDATE lectura_sensores SET hum = ? WHERE id = ?", (new_hum, data['id']))
        conn.commit()
        conn.close()
    elif 'altura' in data:
        altura = data['altura']
        conn = sqlite3.connect('sensores.db')
        cursor = conn.cursor()
        cursor.execute("UPDATE lectura_sensores SET altura = ? WHERE id = ?", (altura, data['id']))
        conn.commit()
        conn.close()
    elif 'presion' in data:
        presion = data['presion']
        conn = sqlite3.connect('sensores.db')
        cursor = conn.cursor()
        cursor.execute("UPDATE lectura_sensores SET presion = ? WHERE id = ?", (presion, data['id']))
        conn.commit()
        conn.close()
```

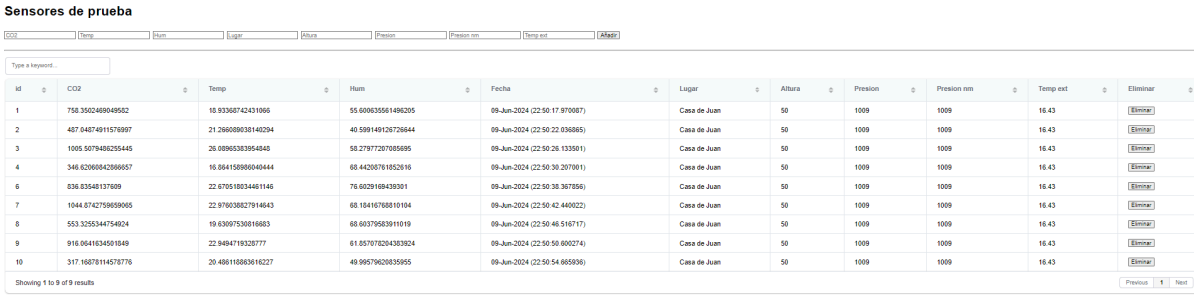
```python
def update():
        cursor = conn.cursor()
        cursor.execute("UPDATE lectura_sensores SET presion = ? WHERE id = ?", (presion, data['id']))
        conn.commit()
        conn.close()
    elif 'presion_nm' in data:
        presion_nm = data['presion_nm']
        conn = sqlite3.connect('sensores.db')
        cursor = conn.cursor()
        cursor.execute("UPDATE lectura_sensores SET presion_nm = ? WHERE id = ?", (presion_nm, data['id']))
        conn.commit()
        conn.close()
    elif 'temp_ext' in data:
        temp_ext = data['temp_ext']
        conn = sqlite3.connect('sensores.db')
        cursor = conn.cursor()
        cursor.execute("UPDATE lectura_sensores SET temp_ext = ? WHERE id = ?", (temp_ext, data['id']))
        conn.commit()
        conn.close()

    return '', 204
except Exception as e:
    return jsonify({'error': 'Error al actualizar datos', 'detalle': str(e)}), 500

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

Añadiendo la libreria sqlite3, se cambiaron y se añadieron algunas rutas.
“/”, muestra el html al usuario.
“/api/datos”, el servidor envia al cliente todas las entradas de la tabla lectura_sensores para que se agreguen a la tabla del html.
“/api/primer-registro”, se devuelve al cliente la primera entrada de la tabla.
“/api/añadir”, con el metodo POST, el usuario puede añadir una nueva entrada a la BD mediante el form.
“/api/eliminar”, con el metodo DELETE, el usuario puede eliminar una entrada de la tabla.
“/api/data”, con PUT, el usuario puede editar los campos de la tabla.

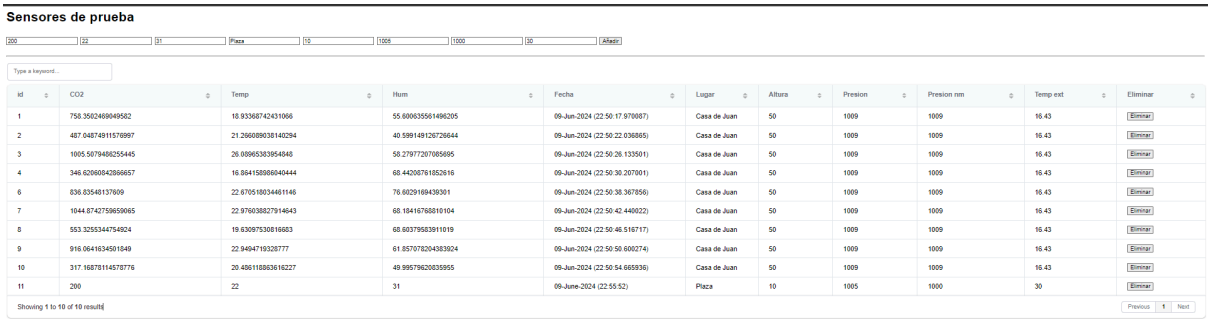
Ejemplo de eliminar con id = 5:



Resultado:



Ejemplo del form para añadir entrada:



Resultado: