Para este informe se realiza el desarrollo de un aplicación web utilizando Flash para la gestión y visualización de datos de sensores ambientales. Permite registrar y consultar lecturas de sensores y varios de los datos que registran estos como el CO2,temperatura,humedad,entre otros. Esta aplicación tiene su base de datos respaldada en SQLite donde se almacenan los datos de los sensores.

Durante este informe se explicara el desarrollo de esta aplicación donde incluirá la estructura de la base de datos, las funcionalidades implementadas en Flask y el manejo de estos datos ya sea agregando más datos, modificarlos o eliminarlos de la base.Para este caso se analizaron, se modificaron y se mejoraron los códigos disponibles en el repositorio de github.

Códigos:

Sensores.py:

```
else:
    break

cont = 0

while cont < cant_capturas:
    cont + 1

verdadero = 1

if verdadero = 1:

print("Datos Disponibles!")

COZ_medido = random.uniform(250, 1100)

temp_sensor = random.uniform(40, 80)

print("Temperatura: Wo.2f degrees C" % temp_sensor)

print("Temperatura: Wo.2f degrees C" % temp_sensor)

print("Humedad: %0.2f %% rH" % humedad_relativa)

d = datetime.now()

print("Fecha", d)

timestampStr = d.strftime("%d-%0-%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)

(CO2_medido, temp_sensor, humedad_relativa, timestampStr, lugar, altura, presion, presion_nm, temp_ext)

conn.commit()

print("Registro insertado..., acumulados:", cont, "\n")

time.sleep(delta_t_capturas)

print("Cierro conexión ...")
```

Este código, se modificó para funcionar como un programa para crear la tabla lectura_sensores en un principio, añadiendo valores a la tabla mediante el uso de las funciones ya establecidas en funciones.py

tabla datos.html:

```
d(--Table Container--)

ccript)

ccript

f(row) {
        if (row) {
            return (contentEditable: 'true', 'data-element-id': row.cells[@].data);
        }
        else {
            return (contentEditable: 'true', 'data-element-id': row.cells[@].data);
        else {
            return (contentEditable: 'true', 'attributes': editable: ellement-id': row.cells[@].data];
        else {
            return (contentEditable: 'true', 'attributes': editable: ellement-id': row.cells[@].data];
        else {
            return (contentEditable: 'true', 'attributes': editable: ellement-id': row.cells[@].data];
        else {
            return (contentEditable: 'true', 'attributes': editable: ellement-id': row.cells[@].data];
        else {
            return (contentEditable: 'true', 'attributes': editable: ellement-id': row.cells[@].data];
        else {
            return (contentEditable: 'true', 'attributes': editable: ellement-id': row.cells[@].data];
        else {
            return (contentEditable: 'true', 'attributes': editable: ellement-id': row.cells[@].data];
        else {
            return (contentEditable: 'true', 'attributes': editable: ellement-id': row.cells[@].data];
        else {
            return (contentEditable: 'true', 'attributes': editable: ellement-id': row.cells[@].data];
        else 'true', 'ame: 'true', 'attributes': editable: ellement-id': row.cells[@].data];
        else 'true', 'ame: 'true', 'attributes': editable: ellement-id': row.cells[@].data];
        else 'true', 'ame: 'true', 'attributes': editable: ellement-id': row.cells[@].data];
```

```
new gridjs.Grid({
         return updateUrl(prev, {search});
  sort: {
    enabled: true,
    multiColumn: true,
     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;
```

El primer cambio observable en este html, es que se añadió un form encima de la tabla, completando ese form, el usuario puede añadir una nueva entrada a la tabla lectura sensores.

Luego, se corrigieron algunos errores sintácticos, algún espacio que sobraba, y se arregló el código de la grid para que pueda mostrar el id de cada entrada de la tabla.

También se añadió una nueva columna a la tabla, llamada "Eliminar", en ella hay un botón, que al ser clickeado por el usuario, llama a la función eliminar() que envía un pedido al servidor, al endpoint /api/eliminar 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:

```
from flask import Flask, render_template, request, abort, jsonify, red
   import time
 3 import random
   import sqlite3
   from datetime import datetime
 6 from funciones import geo_latlon
 8 app = Flask(__name__)
9
10 @app.route('/')
   def index():
       return render_template('tabla_sensores_para_editar.html')
14 @app.route('/api/test')
15 def test():
        return "Prueba de ruta API..."
18 @app.route('/api/datos')
19 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],
```

```
23 def datos():
             'fecha': record[4],
              'altura': record[6],
              'presion_nm': record[8],
              'temp_ext': record[9]
        return jsonify({
              'data': data,
              'total': len(data),
50 @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)
             return jsonify({'error': 'Error al buscar datos', 'detalle': str(e)}), 500
64 @app.route('/api/añadir', methods=["POST"])
65 def añadir():
             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')
```

```
94 def eliminar():
106
            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
118 @app.route('/api/data', methods=['PUT'])
119 def update():
            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()
```

```
new co2 = data['co2']
    cursor = conn.cursor()
cursor.execute("UPDATE lectura_sensores SET co2 = ? WHERE id = ?", (new_co2, data['id']))
     conn.close()
     new_temp = data['temp']
    conn = salite3.connect('sensores.db')
     cursor.execute("UPDATE lectura_sensores SET temp = ? WHERE id = ?", (new_temp, data['id']))
    conn.commit()
     conn.close(
   new_hum = data['hum']
    cursor = conn.cursor()
cursor.execute("UPDATE lectura_sensores SET hum = ? WHERE id = ?", (new_hum, data['id']))
     conn.commit()
    conn.close()
    altura = data['altura']
conn = sqlite3.connect('sensores.db')
    cursor.execute("UPDATE lectura sensores SET altura = ? WHERE id = ?". (altura. data['id']))
elif 'presion' in data:
presion = data['presion']
     conn = salite3.connect('sensores.db')
    cursor = conn.cursor()
     cursor.execute("UPDATE lectura_sensores SET presion = ? WHERE id = ?", (presion, data['id']))
     conn.commit()
```

En el código de sensor_editar_tabla.py se agregaron y modificaron ciertas cosas. Primero, se utilizó la librería sqlite3, luego, se crearon o cambiaron varias rutas.

En la ruta "/", se le mostrará al usuario el html.

En la ruta /api/datos, el servidor envía al cliente todas las entradas de la tabla lectura_sensores para que se agreguen a la tabla del html.

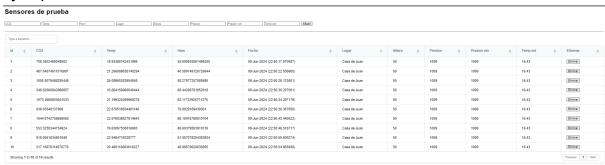
En la ruta /api/primer-registro, se devuelve al cliente en formato json la primera entrada de la tabla.

Mediante la ruta /api/añadir y con el método POST, el usuario puede añadir una nueva entrada a la base de datos a través del form de html que ya hemos visto. Después de añadir esta información el usuario será redirigido a "/" para poder observar la tabla actualizada. En la ruta /api/eliminar, con el método DELETE, que el usuario accede a través del botón

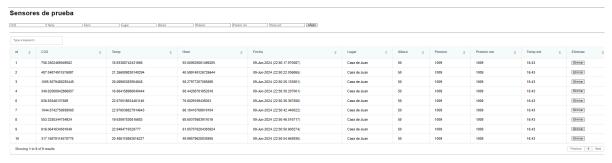
que fue mostrado en el html, se puede eliminar una entrada de la tabla.

Por último, en /api/data con PUT, el usuario puede editar los campos de la tabla, y al hacer esos cambios en el navegador, el servidor actualizará la tabla con la nueva data.

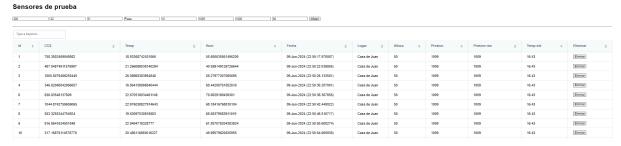
Ejemplo de utilizar el botón de eliminar en la entrada de id 5:



Resultado:



Ejemplo de usar el form para añadir una nueva entrada a la base de datos:



Resultado:

