Para comenzar, en el archivo *mqtt_sub.py* colocamos una función definida como reset_disconnect_timer y disconnect_client que nos permite gestionar una desconexión automática luego de un intervalo de tiempo predefinido en caso de no recibirse mensajes, y a su vez, reiniciar el temporizador que se encarga de desconectar al cliente MQTT.

```python
def reset_disconnect_timer(client):
    global disconnect_timer
    if disconnect_timer is not None:
        disconnect_timer.cancel()
    disconnect_timer = Timer(DISCONNECT_TIMEOUT, disconnect_client, [client])
    disconnect_timer.start()

def disconnect_client(client):
    logging.info("No se recibieron mensajes en el tiempo especificado. Desconectando...")
    client.disconnect()
```

Por otra parte, unificamos los archivos *mqtt_pub.py* y *sensores.py*, creando un servidor que maneje todas las peticiones. Dentro de él, incorporamos diferentes rutas que se utilizan para agregar información con los métodos de PUT y POST, o para eliminar información utilizando el método DELETE. Para lograr esto, creamos funciones auxiliares que nos permiten manejar los datos de forma acorde a lo requerido por la ruta.

```python
def publish_data(records):
    try:
        client.publish(MQTT_TOPIC, "\n\n\n")
        logging.debug("\n\n\n")
        for record in records:
            data = {
                'co2': float(record[1]),
                'temp': float(record[2]),
                'hum': float(record[3]),
                'fecha': record[4],
                'lugar': record[5],
                'altura': float(record[6]),
                'presion': float(record[7]),
                'presion_nm': float(record[8]),
                'temp_ext': float(record[9])
            }
            client.publish(MQTT_TOPIC, json.dumps(data))
            logging.debug(f"Datos publicados: {data}")
            time.sleep(0.1)  # Publicar cada 5 segundos
        client.publish(MQTT_TOPIC, "\n\n\n")
        logging.debug("\n\n\n")
    except Exception as e:
        logging.error(f"Error al publicar datos: {e}")
        time.sleep(0.1)
```

```python
@app.route('/')
def index():
    conn = sqlite3.connect('datos_sensores.db')
    cursor = conn.cursor()
    cursor.execute('SELECT * FROM lectura_sensores')
    records = cursor.fetchall()
    conn.close()
    publish_data(records)
    return render_template('basic_table.html', records=records)
```

La siguiente ruta se utiliza para mostrar los datos almacenados en la base de datos, posteriormente de haber sido publicados en un tema MQTT, en una página web, la cual corresponde a un archivo .html.

A continuación se adjuntan capturas del archivo .html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Datos de Sensores</title>
    <style>
        .button {
            display: inline-block;
            padding: 10px 20px;
            font-size: 16px;
            cursor: pointer;
            text-align: center;
            text-decoration: none;
            color: #fff;
            background-color: #007bff;
            border: none;
            border-radius: 5px;
            box-shadow: 0 2px 4px rgba(0,0,0,0.1);
        }

        table {
            font-family: Arial, sans-serif;
            border-collapse: collapse;
            width: 100%;
        }

        th, td {
            border: 1px solid #dddddd;
            text-align: left;
            padding: 8px;
        }

        th {
            background-color: #f2f2f2;
        }
    </style>
    <script>
        function eliminarRegistro() {
            const idInput = document.getElementById("idReg");
            const id = idInput.value;
            fetch(`/eliminacion/${id}`, {
                method: 'DELETE',
            })
            .then(response => response.json())
            .then(data => {
                if (data.error) {
                    console.log("Error1: " + data.error);
                } else {
                    console.log("Success: " + data.message);
                }
            })
            .catch(error => {
                console.error("Error2:", error);
            });
            idInput.value = '';
        }
    </script>
</head>
```

```
<body>
    <h1>Datos de Sensores</h1>
    <table border="1">
        <thead>
            <tr>
                <th>ID</th>
                <th>CO2</th>
                <th>Temperature</th>
                <th>Humidity</th>
                <th>Date</th>
                <th>Location</th>
                <th>Altitude</th>
                <th>Pressure</th>
                <th>Pressure NM</th>
                <th>External Temp</th>
            </tr>
        </thead>
        <tbody>
            {% for record in records %}
            <tr>
                <td>{{ record[0] }}</td>
                <td>{{ record[1] }}</td>
                <td>{{ record[2] }}</td>
                <td>{{ record[3] }}</td>
                <td>{{ record[4] }}</td>
                <td>{{ record[5] }}</td>
                <td>{{ record[6] }}</td>
                <td>{{ record[7] }}</td>
                <td>{{ record[8] }}</td>
                <td>{{ record[9] }}</td>
            </tr>
            {% endfor %}
        </tbody>
    </table>
    <br>
    <input type="text" id="idReg" placeholder="ID del registro">
    <a href="/" class="button" onclick="eliminarRegistro()">Eliminar</a>
    <br>
    <a href="/filtro" class="button">Filtrar</a>
    <a href="/formulario" class="button">Añadir Datos</a>
</body>
</html>
```

En este archivo, se ha creado una función llamada *eliminarRegistro()* que se encarga de otorgarle funcionalidad al botón de Eliminar. Similarmente, se hace referencia a dos rutas que se encargan del funcionamiento de los botones de Filtrar y Añadir datos. Ambas rutas se encargan de ejecutar un archivo .html, permitiendo mostrar sus resultados.

A continuación se incluyen capturas de ambas rutas, junto con sus respectivos archivos .html:

```python
@app.route('/filtro')
def filtro():
    conn = sqlite3.connect('datos_sensores.db')
    cursor = conn.cursor()
    cursor.execute('SELECT DISTINCT lugar FROM lectura_sensores')
    records = cursor.fetchall()
    conn.close()
    return render_template('listado.html', records=records)
```

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Opciones disponibles</title>
    <style>
        table {
            font-family: Arial, sans-serif;
            border-collapse: collapse;
            width: 100%;
        }

        th, td {
            border: 1px solid #dddddd;
            text-align: left;
            padding: 8px;
        }

        th {
            background-color: #f2f2f2;
        }
    </style>
</head>

<body>
    <h1>Opciones disponibles</h1>
    <table border="1">
        <thead>
            <tr>
                <th>Lugar</th>
            </tr>
        </thead>
        <tbody>
            {% for record in records %}
            <tr>
                <td>{{ record[0] }}</td>
            </tr>
            {% endfor %}
        </tbody>
    </table>
    <h2>Elegir opcion</h2>
    <form action="/resultado" method="post">
        <label for="lugar">Lugar:</label>
        <input type="text" id="lugar" name="lugar"><br><br>
        <input type="submit" value="Enviar">
    </form>
</body>
</html>
```

```python
@app.route('/formulario')
def formulario():
    return render_template("formulario.html")
```

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Sensor Nuevo Dato</title>
    <style>
        .button {
            display: inline-block;
            padding: 10px 20px;
            font-size: 16px;
            cursor: pointer;
            text-align: center;
            text-decoration: none;
            color: #fff;
            background-color: #007bff;
            border: none;
            border-radius: 5px;
            box-shadow: 0 2px 4px rgba(0,0,0,0.1);
        }

        table {
            font-family: Arial, sans-serif;
            border-collapse: collapse;
            width: 100%;
        }

        th, td {
            border: 1px solid #dddddd;
            text-align: left;
            padding: 8px;
        }

        th {
            background-color: #f2f2f2;
        }
    </style>
```

```
    <script>
        function agregarRegistro() {
            const data = {
                co2: document.getElementById('co2').value,
                temp: document.getElementById('temp').value,
                hum: document.getElementById('hum').value,
                lugar: document.getElementById('lugar').value,
                altura: document.getElementById('altura').value,
                presion: document.getElementById('presion').value,
                presion_nm: document.getElementById('presion_nm').value,
                temp_ext: document.getElementById('temp_ext').value
            };
            fetch(`/incorporacion`, {
                method: 'PUT',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify(data)
            })
            .then(response => response.json())
            .then(data => {
                if (data.error) {
                    console.log("Error1: " + data.error);
                } else {
                    console.log("Success: " + data.message);
                }
            })
            .catch(error => {
                console.error("Error2:", error);
            });

        }
    </script>
</head>

<body>
    <h1>Sensor Nuevo Dato</h1>
    <h3>Ingrese:</h3><br>
    <input type="text" id="co2" name="co2" placeholder="CO2"><br>
    <input type="text" id="temp" name="temp" placeholder="Temperatura"><br>
    <input type="text" id="hum" name="hum" placeholder="Humedad"><br>
    <input type="text" id="lugar" name="lugar" placeholder="Lugar"><br>
    <input type="text" id="altura" name="altura" placeholder="Altura"><br>
    <input type="text" id="presion" name="presion" placeholder="Presion"><br>
    <input type="text" id="presion_nm" name="presion_nm" placeholder="Presion NM"><br>
    <input type="text" id="temp_ext" name="temp_ext" placeholder="Temperatura Externa"><br><br>
    <a href="/" class="button" onclick="agregarRegistro()">Enviar</a>
</body>
</html>
```

```python
def publish_data_uno(record):
    try:
        data = {
            'co2': float(record[1]),
            'temp': float(record[2]),
            'hum': float(record[3]),
            'fecha': record[4],
            'lugar': record[5],
            'altura': float(record[6]),
            'presion': float(record[7]),
            'presion_nm': float(record[8]),
            'temp_ext': float(record[9])
        }
        client.publish(MQTT_TOPIC, json.dumps(data))
        logging.debug(f"Datos a eliminar: {data}")
        time.sleep(0.1)  # Publicar cada 0.1 segundos
    except Exception as e:
        logging.error(f"Error al publicar datos: {e}")
        time.sleep(0.1)
```

```python
@app.route('/eliminacion/<int:idReg>', methods=['DELETE'])
def eliminacion(idReg):
    conn = sqlite3.connect('datos_sensores.db')
    cursor = conn.cursor()
    query = 'SELECT * FROM lectura_sensores WHERE id=' + str(idReg)
    cursor.execute(query)
    registro = cursor.fetchone()
    if registro is None:
        conn.close()
        return jsonify({"error": "Reading not found"}), 404

    query = 'DELETE FROM lectura_sensores WHERE id=' + str(idReg)
    cursor.execute(query)
    conn.commit()
    conn.close()
    client.publish(MQTT_TOPIC, "Se hizo un DELETE")
    logging.debug("Se hizo un DELETE")
    client.publish(MQTT_TOPIC, "Datos a ser eliminados: ")
    publish_data_uno(registro)
    return jsonify({"message": "Reading deleted successfully"}), 200
```

La ruta se utiliza para eliminar un registro por id, utilizando la función *publish_data_uno()*. La función publica los datos a eliminar en un tema MQTT y luego se los elimina de la base de datos.

```python
def publish_data_dos(records):
    try:
        for record in records:
            data = {
                'co2': float(record[1]),
                'temp': float(record[2]),
                'hum': float(record[3]),
                'fecha': record[4],
                'lugar': record[5],
                'altura': float(record[6]),
                'presion': float(record[7]),
                'presion_nm': float(record[8]),
                'temp_ext': float(record[9])
            }
            client.publish(MQTT_TOPIC, json.dumps(data))
            logging.debug(f"Datos del lugar '{record[5]}': {data}")
            time.sleep(0.1)  # Publicar cada 0.1 segundos
    except Exception as e:
        logging.error(f"Error al publicar datos: {e}")
        time.sleep(0.1)
```

```python
@app.route('/resultado', methods=['POST'])
def resultado():
    conn = sqlite3.connect('datos_sensores.db')
    cursor = conn.cursor()
    clave = request.form['lugar']
    query = "SELECT * FROM lectura_sensores WHERE lugar = '" + clave + "'"
    cursor.execute(query)
    records = cursor.fetchall()
    conn.close()
    client.publish(MQTT_TOPIC, "Se hizo un POST")
    logging.debug("Se hizo un POST")
    publish_data_dos(records)
    return render_template('basic_table.html', records=records)
```

La función de publicar datos dos se encarga de tomar una lista de registros convirtiendolo al formato que corresponden respectivamente y publicarlos en el MQTT definido.
La integración de esta función en la ruta se utiliza para procesar una solicitud POST, consulta la base de datos para registros que coincidan con la clave lugar y luego los publica al tema MQTT.

```python
def publish_data_tres(datos):
    try:
        client.publish(MQTT_TOPIC, json.dumps(datos))
        logging.debug(f"Datos aniadidos: {datos}")
        time.sleep(0.1)  # Publicar cada 0.1 segundos
    except Exception as e:
        logging.error(f"Error al publicar datos: {e}")
        time.sleep(0.1)
```

```python
@app.route('/incorporacion', methods=['PUT'])
def incorporacion():
    data = request.get_json()
    # Procesar los datos recibidos aquí
    # Por ejemplo, guardar en la base de datos o realizar alguna lógica
    if data is None:
        return jsonify({"error": "Data not sent"}), 404

    co2 = data.get('co2')
    temp = data.get('temp')
    hum = data.get('hum')
    date = datetime.now()
    fecha = date.strftime("%d-%b-%Y (%H:%M:%S.%f)")
    lugar = data.get('lugar')
    altura = data.get('altura')
    presion = data.get('presion')
    presion_nm = data.get('presion_nm')
    temp_ext = data.get('temp_ext')
    conn = sqlite3.connect('datos_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, fecha, lugar, altura, presion, presion_nm, temp_ext))
    conn.commit()
    conn.close()
    client.publish(MQTT_TOPIC, "Se hizo un PUT")
    logging.debug("Se hizo un PUT")
    publish_data_tres(data)
    return jsonify({'message': 'Datos recibidos correctamente', 'data': data}), 200
```

La funcion publish_data_tres publica los datos al MQTT. En la ruta se manejan solicitudes PUT para agregar nuevos datos, guardarlos en la base de datos, y finalmente publicarlos en el tema MQTT.