Proyecto #4 Interfaces Gráficas
INFO 1128 By Alberto Caro

Gabriel Muñoz, Benjamín Sanhueza
gmunoz2021@alu.uct.cl, bsanhueza2021@alu.uct.cl

INRTODUCCION

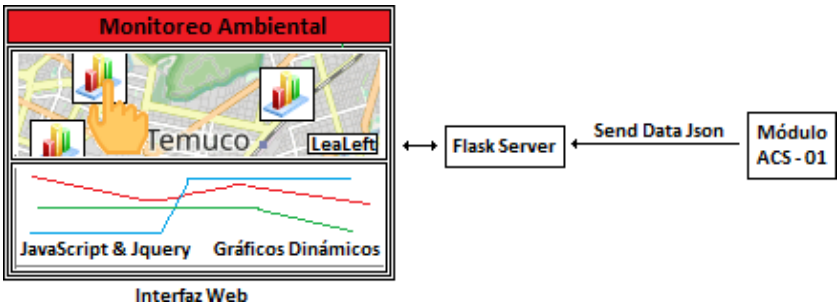
En esta entrega tenemos el objetivo de implementar los conceptos de interfaces graficas en cuanto a la conexión usuario-pagina.
Para esto se deberán desarrollarse los conocimientos de desarrollo de un proyecto, tanto en como identificar los requisitos, las herramientas a utilizar como, el operarlas, las investigaciones de nuevos procedimientos, las pruebas del software creado, optimización entre otros.
Definido por el autor, Saenz et al. quien referencia a la guía ISO 31000 del 2009, en este sentido, gestionar proyectos, exige de la implementación efectiva de procesos de planificación, ejecución y control. La precisión de cumplimientos en tiempos específicos estipulados y el hacer uso racional de recursos, es esencial en estos procesos de gestión, donde se apuesta a costos racionales a nivel de las operaciones desarrolladas y riesgos impredecibles.
Aplicando los procedimientos con las problemáticas planteadas se deberá realizar procesos de traficación de un set de datos del sensor de material particulado, lo cual debe ser dinámico, ya que se actualizará cada cierto tiempo, se debe implementar estos gráficos a un mapa donde tendrán distintos puntos de ubicación, los cuales en su función deberán mostrar graficas; siendo por ultimo recrear una ruta desde un csv que tendrá que conectarse a una base de datos la cual al tener los datos deberá poder graficarse las rutas de los vehículos en un mapa. A los cuales se utilizarán herramientas como csv, sqlite, flask, entre otros.

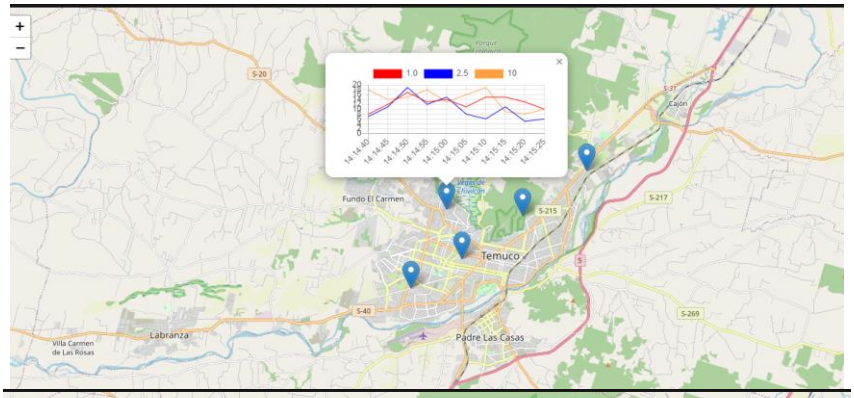
MARCO TEORICO

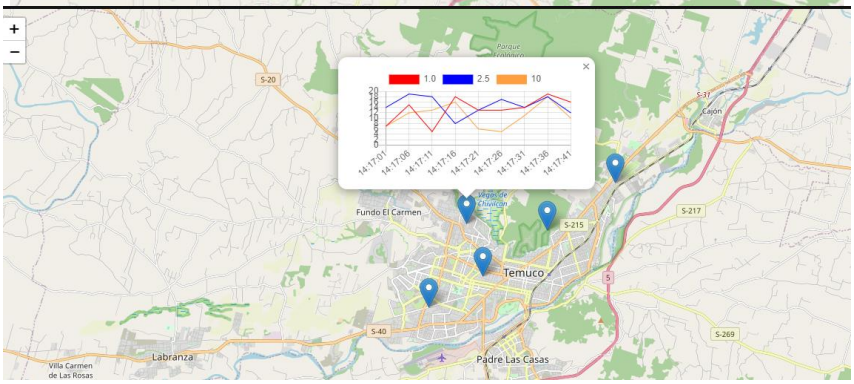
- **Folium**: facilita la visualización de datos que han sido manipulados en Python en un mapa de folleto interactivo. Permite vincular datos a un mapa para choroplethvisualizaciones, así como pasar visualizaciones ricas en vector/raster/HTML como marcadores en el mapa.
- **SQLite:** SQLite es una biblioteca en lenguaje C que implementa un motor de base de datos SQL pequeño, rápido, autónomo, de alta confiabilidad y con todas las funciones.
- **Time:** Este módulo proporciona varias funciones relacionadas con el tiempo, proporciona un conjunto de funciones para trabajar con fechas y/o horas. Además de estas funciones hay otras relacionadas en los módulos datetime y calendar.
- **Json:** librería para manipular archivos json asi como codificar, decodificar; El uso de objetos JavaScript (JSON) es una poderosa herramienta para intercambiar datos rápidamente entre varias plataformas de programación. Ya sea para almacenar datos o crear una API, convertir sus datos en JSON los hace reutilizables, independientemente de la tecnología que acceda a ellos.
- **Pandas:** librería de Python especializada en la manipulación y el análisis de datos. Ofrece estructuras de datos y operaciones para manipular tablas numéricas y series temporales.
- **Flask:** un framework minimalista escrito en Python que permite crear aplicaciones web rápidamente y con un mínimo número de líneas de código. Está basado en la especificación WSGI de Werkzeug y el motor de templates Jinja2.

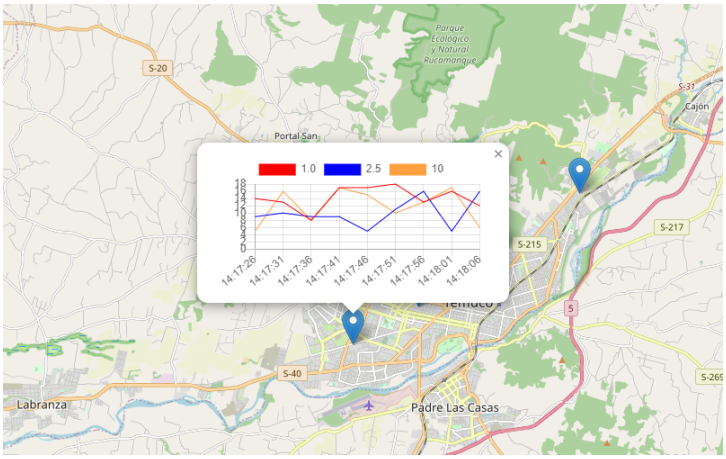
DESARROLLO

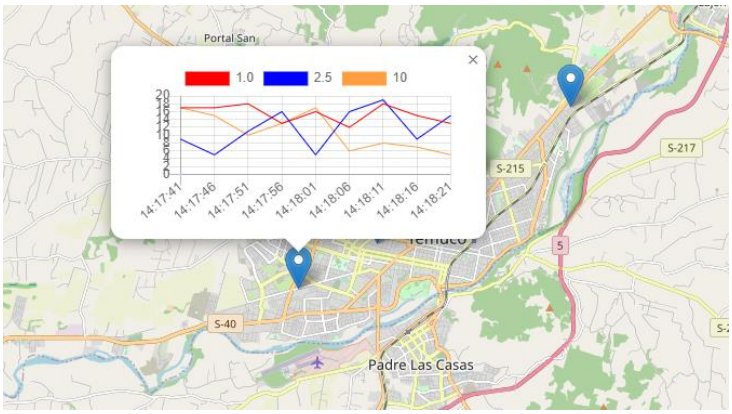
3.- Desarrolle una aplicación web que muestre el mapa de Temuco con **5** estaciones ambientales (**marcadores**). Haciendo click sobre cualquiera de ellos se deben graficar los niveles de contaminación de manera dinámica asociado a ese marcador. Sea creativo y haga los cambios que considere necesario. Implementeun **Web Server Flask** que haga lo pedido. **[35 puntos]** . Utilice **JSON**, **Jquery** y **Java Script**.











MODULO ACS-01

Para el módulo se utilizó el script facilitado con modificaciones, como al módulo utilizado para el ejercicio 3 el cual se añadió un id como también que genera 5 datas para los 5 marcadores que serán utilizados posteriormente.

```python
import requests as req, time as ti, random as ra

sURL = 'http://127.0.0.1:5000/data2'

def Generate(id):

    for i in range(5):
        dData = {
        '01': ra.randint(+5,+20),
        '25': ra.randint(+5,+20),
        '10': ra.randint(+5,+20),
        'id': id
        }
    return dData

while 1:
    for i in range(5):
        dData = Generate(i)
        print(dData)
        rq =  req.post(sURL,data=dData)
    ti.sleep(5)
rq.close()

print(rq.txt)
```

RUTA DE DATA

Para que el módulo acs-01 este habilitado, se genera una ruta en la aplicación principal, deacuerdo a los métodos get y post, se obtiene la data generada la cual se almacena en una variable global para que sea accesible en la aplicación.

```python
@app.route("/data2",methods=['GET', 'POST'])
def data2():

    if request.method == 'POST':
        if len(Dtime)> 10:
            Dtime.pop(0)
            sc[0].pop(0)
            sc[1].pop(0)
            sc[2].pop(0)
            sc[3].pop(0)
            sc[4].pop(0)
        sc[int(request.form["id"])].append(request.form)
        if len(Dtime) != 0:
            if (Dtime[len(Dtime)-1]["time"] != time.strftime("%H:%M:%S")):
                Dtime.append({
                "time": time.strftime("%H:%M:%S")
                })
        else:
            Dtime.append({
                "time": time.strftime("%H:%M:%S")
            })

        return {"success": True}
```

RUTA DATOS

Como antes se definió la ruta de data se debe tener una ruta de acceso de datos, por lo cual con los datos almacenados en los data se utilizan como un script de json que nos entrega el id de los data del módulo acs-01.

```python
@app.route("/getdata2",methods=['GET', 'POST'])
def getdata2():

    if request.method == 'POST':
        return [sc[json.loads(request.data)["id"]],Dtime]
    else:
        return render_template('test.html')
```

RUTA PRINCIPAL

Como antes se definió la ruta de datos se debe tener una ruta de acceso para visualizar el grafico, por lo cual con los datos proporcionados por las rutas de data y datos se utilizan, al ser renderizados por el archivo HTML.

```python
@app.route("/mapa",methods=['GET', 'POST'])
def mapa():

    if request.method == 'POST':
        return [data, Dtime]
    else:
        return render_template('test2.html')
```

HTML

Primeramente, generamos un script donde primeramente se genera el mapa en las coordenadas iniciadas, luego, se definen los puntos de los marcadores en el mapa, así como al clickear en este se abre un popup, donde procesa el obtener el data de un id correspondiente.

```
<script>

    const map = L.map('map').setView([-38.739596, -72.601606], 12);

    const tiles = L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png', {
        maxZoom: 30,
        attribution: '&copy; <a href="http://www.openstreetmap.org/copyright">OpenStreetMap</a>'
    }).addTo(map);

    const marker1 = L.marker([-38.721183, -72.579891]).addTo(map)
        .bindPopup("<canvas id='grafica' onclick='getdata(0)'></canvas>", {maxWidth: 500});
    const marker2 = L.marker([-38.703984, -72.548047]).addTo(map)
        .bindPopup("<canvas id='grafica' onclick='getdata(1)'></canvas>");
    const marker3 = L.marker([-38.749035, -72.634651]).addTo(map)
        .bindPopup("<canvas id='grafica' onclick='getdata(2)'></canvas>");
    const marker4 = L.marker([-38.737788, -72.609674]).addTo(map)
        .bindPopup("<canvas id='grafica' onclick='getdata(3)'></canvas>");
    const marker5 = L.marker([-38.718705, -72.617141]).addTo(map)
        .bindPopup("<canvas id='grafica' onclick='getdata(4)'></canvas>");

</script>
```

En segundo lugar, procesamos los ids al abrir el popup, asi enrutándola a la obtención de datos en función al id proporcionado, en la ruta datos donde este nos devuelve esos datos específicos condicionados por el id.

```
async function getdata(st){
console.log(JSON.stringify({"id": st}))
  fetch('/getdata2',{
    method : 'POST',
    body: JSON.stringify({"id": st})
  }).then(response =>{
    return response.json();
  }).then( data =>{
    console.log(data)
    graficar(data);
  }).catch(error =>console.error(error));

}
```

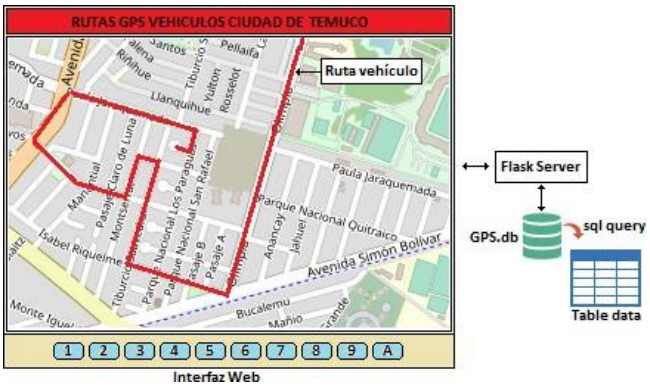
En cuanto a lo anteriormente definido, obtenemos los datos anteriormente generados y enrutados, verificamos y realizamos la gráfica, así separando los datos obtenidos en variables de almacenado, a lo cual de acuerdo a la estructura del char se realiza, para ser mostrado por pantalla; todo esto en función al marcador clickeado se genera estableciendo su id, en consiguiente el procesar los datos y finalmente graficar.

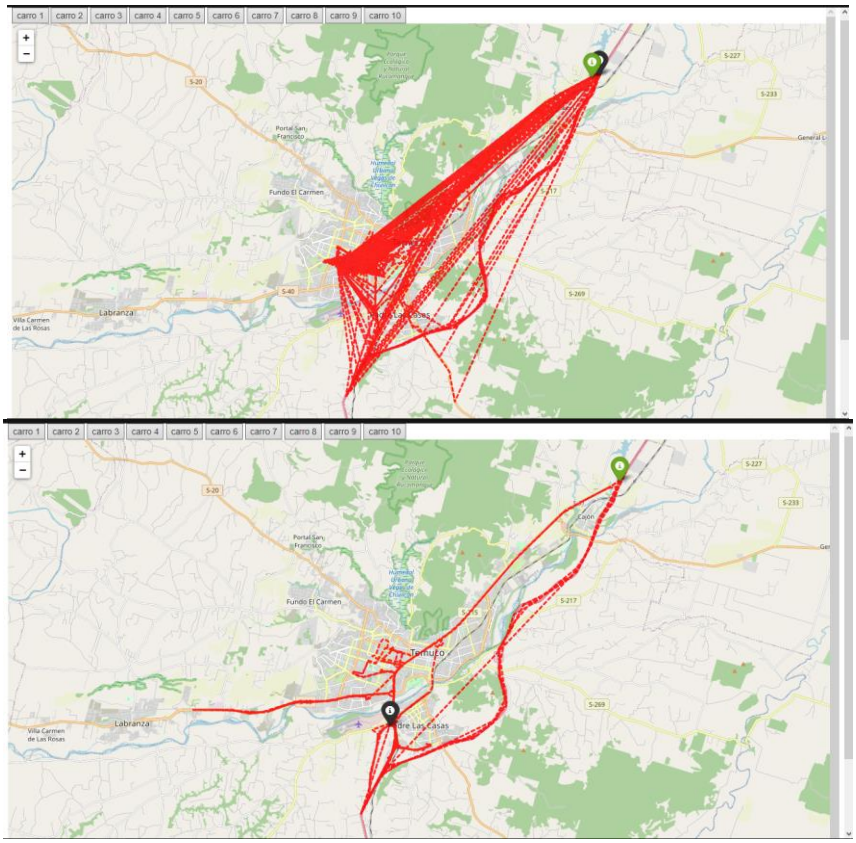
```
function graficar(data){

    var backgroundColor = 'white';
    Chart.plugins.register({
    beforeDraw: function(c) {
        var ctx = c.chart.ctx;
        ctx.fillStyle = backgroundColor;
        ctx.fillRect(0, 0, c.chart.width, c.chart.height);
    }
    });

    const $grafica = document.querySelector("#grafica");
    console.log($grafica);
    const $temperatura = document.querySelector("#temperatura");
    let etiquetas = [];
    let d01 = [];
    let d25 = [];
    let d10 = [];
    let dte = [];
    for (i in data[0]) {
        etiquetas.push(data[1][i]["time"].toString());
        d01.push(parseInt(data[0][i]["01"]));
        d25.push(parseInt(data[0][i]["25"]));
        d10.push(parseInt(data[0][i]["10"]));
        dte.push(parseInt(data[0][i]["te"]));
    }
    const P01 = {
        label: "1.0",
        data: d01,
        backgroundColor: 'transparent', // Color de fondo
        borderColor: 'rgba(255, 0, 0, 1)', // Color del borde
        borderWidth: 1,// Ancho del borde
        lineTension: 0,
        pointRadius: 0,
        pointBorderWidth: 2,
        fill: false,

    };
    const P25 = {
        label: "2.5",
        data: d25, // La data es un arreglo que debe tener la misma cantidad de valores que el eje X
        backgroundColor: 'transparent',// Color de fondo
        borderColor: 'rgba(0, 0, 255, 1)',// Color del borde
        borderWidth: 1,// Ancho del borde
        lineTension: 0,
        pointRadius: 0,
        fill: false,
    };
    const P10 = {
        label: "10",
        data: d10, // La data es un arreglo que debe tener la misma cantidad de valores que el eje X
        backgroundColor: 'transparent',// Color de fondo
        borderColor: 'rgba(255, 159, 64, 1)',// Color del borde
        borderWidth: 1,// Ancho del borde
        lineTension: 0,
        pointRadius: 0,
        fill: false,
    };
```

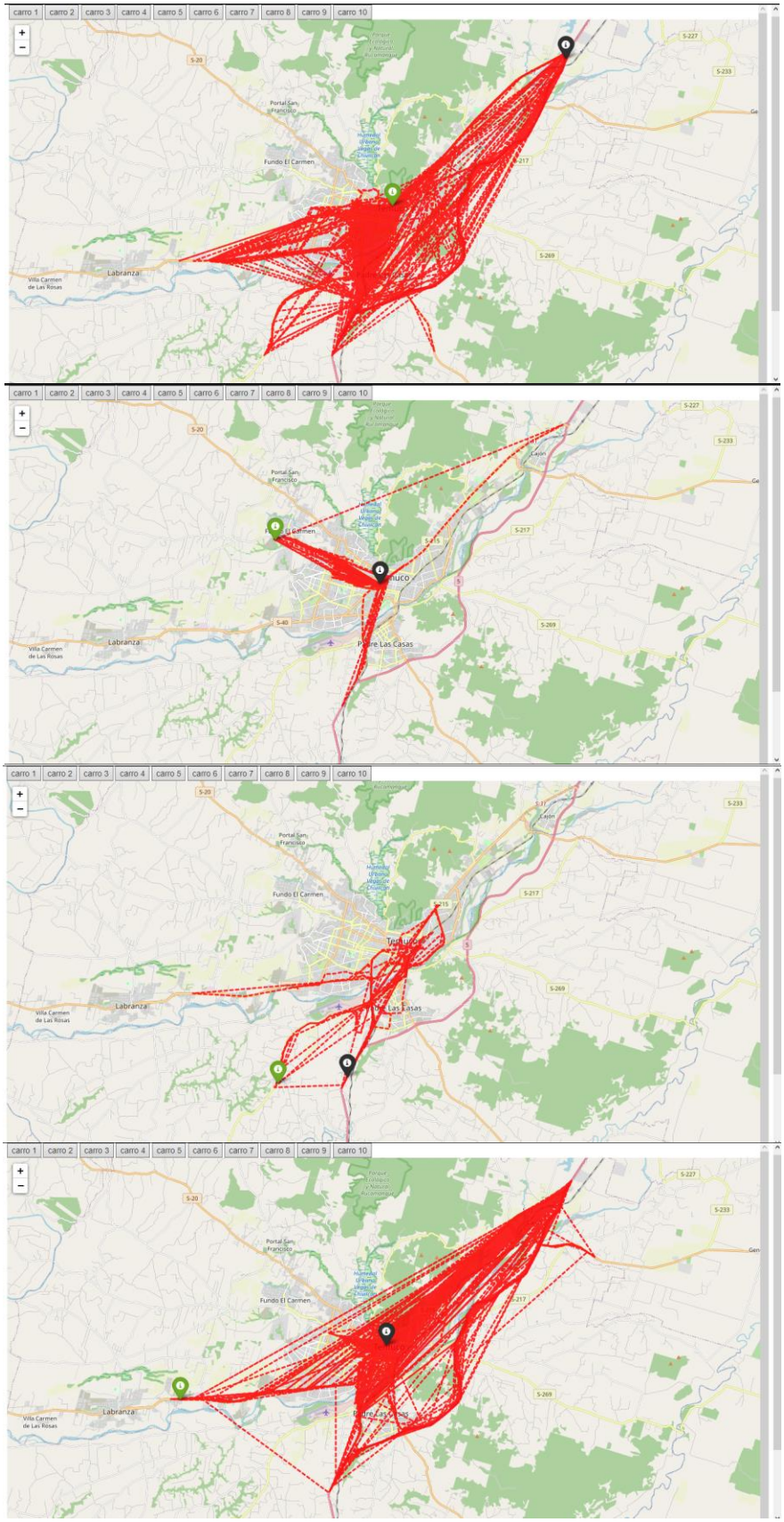
```
    const chart = new Chart($grafica, {
    type: 'line',
    data: {
        labels: etiquetas,
        datasets: [
            P01,
            P25,
            P10,

        ]
    },
    options: {
        scales: {
            yAxes: [{
                ticks: {
                    beginAtZero: true,
                }
            }],
        },
    }
});


}
```

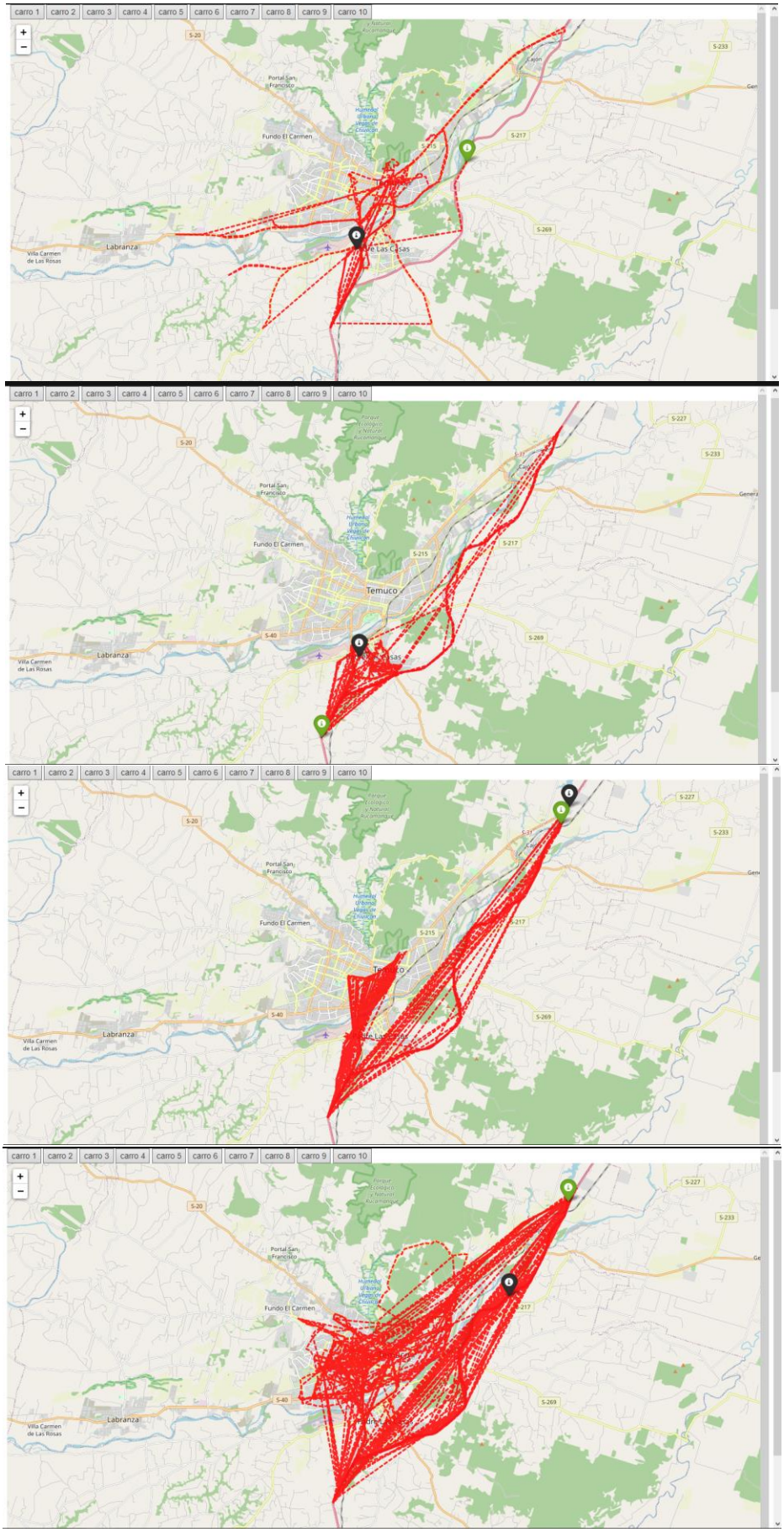
4.- Desarrolle una aplicación web que muestre el mapa de **Temuco** con las rutas de los vehículos en la base de dato **gps.db** de **SQLite**. Haciendo click sobre los botones se graficará la ruta del vehículo asociado en el mapa. En la tabla **data** esta toda la información de los **10** vehículos. Implemente un **Web Server Flask**, el cual enviará toda la información **JSON** a **Jquery Javascript** de la página **html** asociada. Por el lado del **Web Server Flas**k, se debe recibir la clave del vehiculo a buscar en la tabla **data**. Analice la base de datos y saque sus propias conclusiones. Todos los datos se encuentran en el archivo **rutas.csv**, los cuales deben ser importados a la t**abla data** de **gps.db**. **[35 puntos]**

TABLE [data](
[npk] integer PRIMARY KEY AUTOINCREMENT,
[id] VARCHAR(20), // Clave del vehículo!
[lat] double, // Latitud GPS
[lon] double, // Longitud GPS
[velo] int, // Velocidad del vehículo
[angu] int, // Orientación del vehículo
[fecha] varchar(10), // Fecha tracking del vehículo
[hora] varchar(10), // Hora tracking del vehículo
[onoff] int, // Motor encendido/apagado del vehículo
[nsat] int); // Número de Satélites vísibles






carro 1
carro 2
carro 3
carro 4
carro 5
carro 6
carro 7
carro 8
carro 9
carro 10
Portal San Francisco
Fundo El Carmen
Labranza
Villa Carmen de Las Rosas
Padre Las Casas
Las Casas
S-20
S-227
S-233
S-217
S-269
S-40



CONECCION Y RELLENAR BASE DE DATOS

En cuanto a las conexiones a las base de datos primeramente tenemos que instalar sqlite, así como importar csv para manipular los datos del archivo csv, a lo cual posteriormente de abrirlo en modo lectura, establecemos la conexión a la base de datos asi como también para poder hacer las consultas, luego generamos la consulta de crear la base de

datos con la tabla si esta no esta creada; asi para el archivo csv leemos y almacenamos los datos en una variable la cual al recorrerla en conjunto con una consulta serán ingresados los datos a la base, por lo cual para comprobarlo solo se realiza la consulta para mostrar los datos dentro de la tabla de la base de datos.

```
import sqlite3
import csv

file=open("rutas.csv","r")
reader=csv.reader(file)

conex = sqlite3.connect('gps.db')
c=conex.cursor()

c.execute( """CREATE TABLE IF NOT EXISTS data (
 npk integer PRIMARY KEY AUTOINCREMENT,
 id VARCHAR(20),
 lat double,
 lon double,
 velo int,
 angu int,
 fecha varchar(10),
 hora varchar(10),
 onoff int,
 nsat int)""")


listas=[]
for row in reader:
    listas.append(row)

for row in listas:
    c.execute("INSERT INTO data VALUES (?,?,?,?,?,?,?,?,?,?)",
    (row[0],row[1],row[2],row[3],row[4],row[5],row[6],row[7],row[8],row[9]))
    conex.commit()
c.execute("SELECT * FROM data")
data = c.fetchall()
print(data)
conex.commit()
c.close()
conex.close()
```

LIBRERIAS

Para comenzar importamos las librerías correspondientes como lo son flask para crear la aplicación, json para trabajar con métodos como archivos json, así también folium con quien se crea y manipula el mapa, así como pandas para manejar dataframe y sqlite para hacer consultas a la base de datos.

```
from flask import Flask, render_template, request
import json
import folium
import sqlite3
import pandas as pd
```

SQL

Luego, se define la conexión a la base de datos y se genera la consulta de todos los datos que hay en la tabla, para lo cual se utilizara panda creando un dataframe y sean almacenados los datos; de este extraemos la latitud, longitud e Id de los datos los cuales se seleccionas dándole el índice de la columna del dato solicitado.

```
conex = sqlite3.connect('gps.db')
c=conex.cursor()
c.execute("SELECT * FROM data ORDER BY fecha")
dt = c.fetchall()

data = pd.DataFrame.from_records(dt[1:],columns=dt[0])

place_lat = data.iloc[:,2].values.astype(float).tolist()
place_lng = data.iloc[:,3].values.astype(float).tolist()
ids = data.iloc[:,1].values.astype(int).tolist()
vel = data.iloc[:,5].values.astype(int).tolist()
```

RUTA INICIAL

En tercer lugar, definir la ruta para que ejecute el código de inicio, el cual al usuario se muestra el mapa a utilizar como también los botones, los cuales nos ejecutan un script json donde posterior a un delay, se obtiene el dato de cual botón se clickeo, para lo cual se establece un id el cual es enviado al siguiente script de trabajo, para definir que ruta se debe mostrar correctamente por pantalla.

```python
@app.route("/carro")
def home():
    map = folium.Map(
        location=[-38.739596, -72.601606],
        zoom_start=12
    )
    map.get_root().html.add_child(folium.Element("""
    <input type=button onClick="enviar(1)" value='carro 1'>
    <input type=button onClick="enviar(2)" value='carro 2'>
    <input type=button onClick="enviar(3)" value='carro 3'>
    <input type=button onClick="enviar(4)" value='carro 4'>
    <input type=button onClick="enviar(5)" value='carro 5'>
    <input type=button onClick="enviar(6)" value='carro 6'>
    <input type=button onClick="enviar(7)" value='carro 7'>
    <input type=button onClick="enviar(8)" value='carro 8'>
    <input type=button onClick="enviar(9)" value='carro 9'>
    <input type=button onClick="enviar(10)" value='carro 10'>

    <script>
    function delay(milliseconds){
    return new Promise(resolve => {
      setTimeout(resolve, milliseconds);
    });
    }
    async function enviar(data){
        console.log(JSON.stringify({'id': data}));
      fetch('/Dcarro',{
        method : 'POST',
        body: JSON.stringify({"id": data})
      }).then(response =>{
        return response.json();
      }).catch(error =>console.error(error));
      await delay(1000);
      console.log("redireccionando");
      location.replace("http://127.0.0.1:5000/Dcarro")

    }
    </script>
    """))
    map.save("templates/out.html")
    return map._repr_html_()
```

RUTA PRINCIPAL

Finalmente, se crea la conexión a los datos de json, a los cuales se obtiene el id, los datos anteriormente definidos, los utilizamos con algunas restricciones, como tomar la cantidad de longitudes y latitudes correctas para un id de vehículo especifico, las cuales se almacenan en una variable para luego poder realizar la ruta; así también definimos las coordenadas donde generar la visualización del mapa, luego definimos marcadores de donde comenzó la ruta y donde finalizo, y en consiguiente, de acuerdo a los puntos obtenidos generamos la ruta que conecta los puntos obtenidos, añadiendo estos visualmente al mapa, retornando la función todo lo anteriormente definida, así también se reutiliza el condigo de los botones de selección de carro para así obtener el id para nuevamente mostrar la ruta del vehículo en el mapa.

```python
@app.route("/Dcarro",methods=['POST', 'GET'])
def carro():
    if request.method == 'POST':
        id = json.loads(request.data)["id"]
        points = []
        for i in range(len(ids)):
            if ids[i]==id:
                points.append([place_lat[i], place_lng[i]])
        map = folium.Map(
            location=[-38.739596, -72.601606],
            zoom_start=12
        )
        folium.Marker(
            location=points[0],
            tooltip="Punto de partida",
            icon=folium.Icon(color='green')
        ).add_to(map)
        folium.Marker(
            location=points[-1],
            tooltip="Destino",
            icon=folium.Icon(color='black')
        ).add_to(map)

        folium.PolyLine(points, color='red',dash_array='5',opacity ='.85',
            tooltip ='Trayectoria'
            ).add_to(map)
```

```
    map.get_root().html.add_child(folium.Element("""
    <input type=button onClick="enviar(1)" value='carro 1'>
<input type=button onClick="enviar(2)" value='carro 2'>
<input type=button onClick="enviar(3)" value='carro 3'>
<input type=button onClick="enviar(4)" value='carro 4'>
<input type=button onClick="enviar(5)" value='carro 5'>
<input type=button onClick="enviar(6)" value='carro 6'>
<input type=button onClick="enviar(7)" value='carro 7'>
<input type=button onClick="enviar(8)" value='carro 8'>
<input type=button onClick="enviar(9)" value='carro 9'>
<input type=button onClick="enviar(10)" value='carro 10'>

<script>
function delay(milliseconds){
return new Promise(resolve => {
  setTimeout(resolve, milliseconds);
});
}
async function enviar(data){
    console.log(JSON.stringify({'id': data}));
  fetch('/Dcarro',{
    method : 'POST',
    body: JSON.stringify({"id": data})
  }).then(response =>{
    return response.json();
  }).catch(error =>console.error(error));
  await delay(1000);
  console.log("redireccionando");
  location.replace("http://127.0.0.1:5000/Dcarro")

}
</script>
"""))
    map.save("templates/out.html")
    return {"success": True}
if request.method == 'GET':
    return render_template('out.html')
```

CONCLUSION

Por lo anteriormente expuesto, se puede concluir que el utilizar herramientas como json, pandas, request entre otros, facilitaron los enrutamientos, así como generar funciones en base a algún parámetro; flask por otro lado resulto ser un entorno minimalista ya que no necesita una estructura compleja para realizar una pagina web, permitiendo en codificación tener más libertad para realizar procedimientos; sqlite, es pequeño, rápido, autónomo, de alta confiabilidad y con todas las funciones, por lo cual con las funciones correspondientes no se dificulta la conexión de flask a una base de datos; para finalmente la herramienta folium nos provee de una librería de mapa, la cual es muy intuitiva a la hora de establecer un lugar, como marcadores, bastando en la información primordial de la latitud como longitud de un punto, para lo cual graficar una ruta se basaba en un arreglo con los datos listados de las coordenadas.

Finalizando, con lo anteriormente desarrollado, la curva de aprendizaje es favorable ya que, en la capacidad de desarrollar aplicaciones, flask es un framewrok que proporciona una amplia gama de posibilidades para el desarrollo de alguna problemática.