PROYECTO # 2

INTERFACES GRÁFICAS ⋆ INFO 1128

AUTOR: FELIPE ANDRÉS ESPINOZA SÁNCHEZ

1. Desarrollo

1.1. Problema 1,

Para resolver el primer enunciado del proyecto, necesitamos un script el cual llamamos cliente, ya que será el que enviará las imágenes a un servidor de flask. Se crea un diccionario el cual contendrá el nombre de todas las imágenes a transferir por POST. Luego declaramos una variable URL a la cual le haremos la petición POST. Se crea un bucle For el cual irá recorriendo las imágenes del diccionario, también la iremos asignando a una estructura de file (Librería de request), abrimos la imagen en modo 'rb', luego de eso haremos el POST..

```python
import requests as r

dd = {'img': ['a.jpg', 'b.jpg','c.jpg', 'd.jpg' , 'e.jpg']}
url = 'http://127.0.0.1:4000/im_data'

for i in dd['img']:
    files = {'img': open(f'figuras/{i}', 'rb')}
    res = r.post(url, files=files)
    print('Status: ',res.status_code)
```

Por el lado del servidor, desde flask, importamos FLask, request entre las más importantes por el lado del servidor, luego desde PIL importaremos Image, este módulo nos ayudará a volver a convertir el file enviado desde el cliente a una imagen. Preguntamos si se recibe una petición por POST, de ser así almacenaremos el file recibido en una variable para luego abrirlo con PIL y hacerle una conversión a blancos y negros y un $FLIP-TOP\_Bottom$ lo cual hace que la imagen quede inversa en vertical. Luego y finalmente guardamos las imagenes en una nueva carpeta con el nombre recibido.

```python
from flask import Flask, request, jsonify
from PIL import Image

app = Flask(__name__)

@app.route('/im_data', methods=['POST'])
def index():
    if request.method == "POST":
        files = request.files['img']
        img = Image.open(files).convert('L')
        img = img.transpose(method=Image.FLIP_TOP_BOTTOM)
        img.save('static/img/'+ files.filename)
    return jsonify({ 'msg': 'FILE SAVE SUCCESS'})

if __name__ == '__main__':
    app.run(debug=True, port=4000)
```

1.2. Problem 2.

Se nos pidió que con la simulacion del sensor material particulado enviaramos la información entregada a un servidor de Flask cada 5 segundos. Declaramos la URL del servidor de flask. Luego crearemos un While True ya que siempre este enviando la información del material particulado con un tiempo de espera de 5 segundos, eso lo logramos con Sleep(5), luego con la libreria Request su metodo Post enviamos la información a flask.

```
import random as RA,requests as r
from time import sleep
#"Funciones Data_MP25 MP10 y Get_DD, dadas en proyecto"
def Data_MP25(nType):
    ...
def Data_MP10(nType):
    ...
def Get_DD():
    ...

URL = "http://127.0.0.1:4000/data"

while 1:
    res = r.post(URL, json =  Get_DD())
    print('Status: ',res.status_code)
    sleep(5)
```

Por parte del Servidor, creamos el servidor de flask, importamos librerias requeridas, para renderizar html, etc. Declaré una variable dataDD, la cual contendrá un diccionario con toda la información recibida desde el cliente (Simulador de PM 25 y PM 10), creo una ruta a la cual se puede acceder por metodo GET, para en ella poder retornar nuestro diccionario con la informacion requerida, Renderizamos nuestro index, para poder trabajar las graficas con Js y otras librerias requeridas.

```
from flask import Flask, request, jsonify, render_template

app = Flask(__name__)

dataDD = {'EA': [], '25': [], '10': []}

@app.route('/data', methods=['POST'])
def index():
    if request.method == "POST":
        data = request.get_json(force=True)
        dataDD['EA'] = data['EA']
        dataDD['25'] = data['25']
        dataDD['10'] = data['10']
    return jsonify({ 'msg': 'FILE SAVE SUCCESS'})

@app.route('/get_data', methods=['GET'])
def data_get():
    if request.method == "GET": return dataDD

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

if __name__ == '__main__':
    app.run(debug=True, port=4000)
```

Por parte del BackEnd Usamos JS, para poder obtener la información a través de una petición GET usaremos la librería Jquery para obtenerla como JSON como se ve en la función a continuación, luego esa data la guardamos en array que contienen objetos(ya estructurados para poder graficarlos con la Liberia CHART.JS.) de los 2 Material Particulado (2.5 y 10).

```
1
2  var data = [
3      MP_25 = {
4          label: "MP 2.5 ug/m3",
5          data: [],
6          lineTension: 0,
7          borderColor: 'green',
8          steppedLine: true
9      },
10     MP_10 = {
11         label: "MP 10 ug/m3",
12         data: [],
13         lineTension: 0,
14         borderColor: 'red',
15         steppedLine: true
16     }
17 ]
18
19
20 const get_Data = (resp) => {
21     data[0].data = resp[25];
22     data[1].data = resp[10];
23     data_set_25.datasets = [data[0]];
24     data_set_10.datasets = [data[1]];
25 }
26 const GetJson = () => {
27     $.getJSON("http://127.0.0.1:4000/get_data", function (data) {
28         get_Data(data);
29     });
30 };
```

Para Graficar luego nuestros datos de Material Particulado, crearemos una nueva instancia de Chart la cual construirá nuestros gráficos a partir de los parámetros y estilos que nosotros le demos.

```
1  const Grafico_PM_25 = () => {
2      grafico_PM_25 = new Chart(canvas_MP_25, {
3          type: 'line',
4          data: data_set_25,
5          options: { legend: { display: true, position: 'top', labels: { boxWidth: 75, fontColor: 'black' } } }
6      });
7      grafico_PM_25.render();
8  };
9
10
11 const Grafico_PM_10 = () => {
12     grafico_PM_10 = new Chart(canvas_MP_10, {
13         type: 'line',
14         data: data_set_10,
15         options: { legend: { display: true, position: 'top', labels: { boxWidth: 75, fontColor: 'black' } } }
16     });
17     grafico_PM_10.render();
18 };
```

A continuacion ya vemos los 2 graficos renderizados, uno de PM 2.5 y el otro del PM 10.

1.3. Problem 3.

Nos piden generar 10 subplot de Velocidad y Aceleración de un csv con datos de GPS con distintos Autos. Para ello leeremos el CSV con la librería pandas, luego debemos filtrar solo los vehículos con el motor encendido (onOff = 1), para ello usaremos pandas para filtrar los datos con una función que creamos llamada $filter\_ON()$, recibe por parámetros el dataframe y el estado en el cual queremos el motor, en este caso encendido (1) . Por otro lado nos piden 10 subplot, entonces son 10 vehiculos diferente, crearemos una funcion la cual nos filtre por ID, para asi obtener los 10 ID de vehículos diferentes, esta funcion recibe el dataframe y el id. Retorna un nuevo dataframe.

```
import pandas as pd, pylab as plt, numpy as np

def read_to_csv(path):
    return pd.read_csv(path)

def filter_ON(df,on = 1):
    it_on = df['onoff'] == on
    new_df = df[it_on]
    return new_df

def filter_ID(df,id):
    id = df['id'] == id
    new_df = df[id]
    return new_df
```

Ahora para obtener solo la columna de Velocidad, crearemos una función que reciba por parámetro el dataframe y nos retorna una lista con las primeras 300 velocidades del vehículo seleccionado, con la cual podremos luego subplotear las velocidades.

```
def get_data_velo(df):
    data_Array = list(df['velo'].head(300))
    return data_Array
```

La siguiente función nos hará el cálculo de velocidad a aceleración con la fórmula de $a = \frac{v_f - v_i}{t}$ así nos retornará una nueva lista con las 300 aceleraciones del vehículo seleccionado.

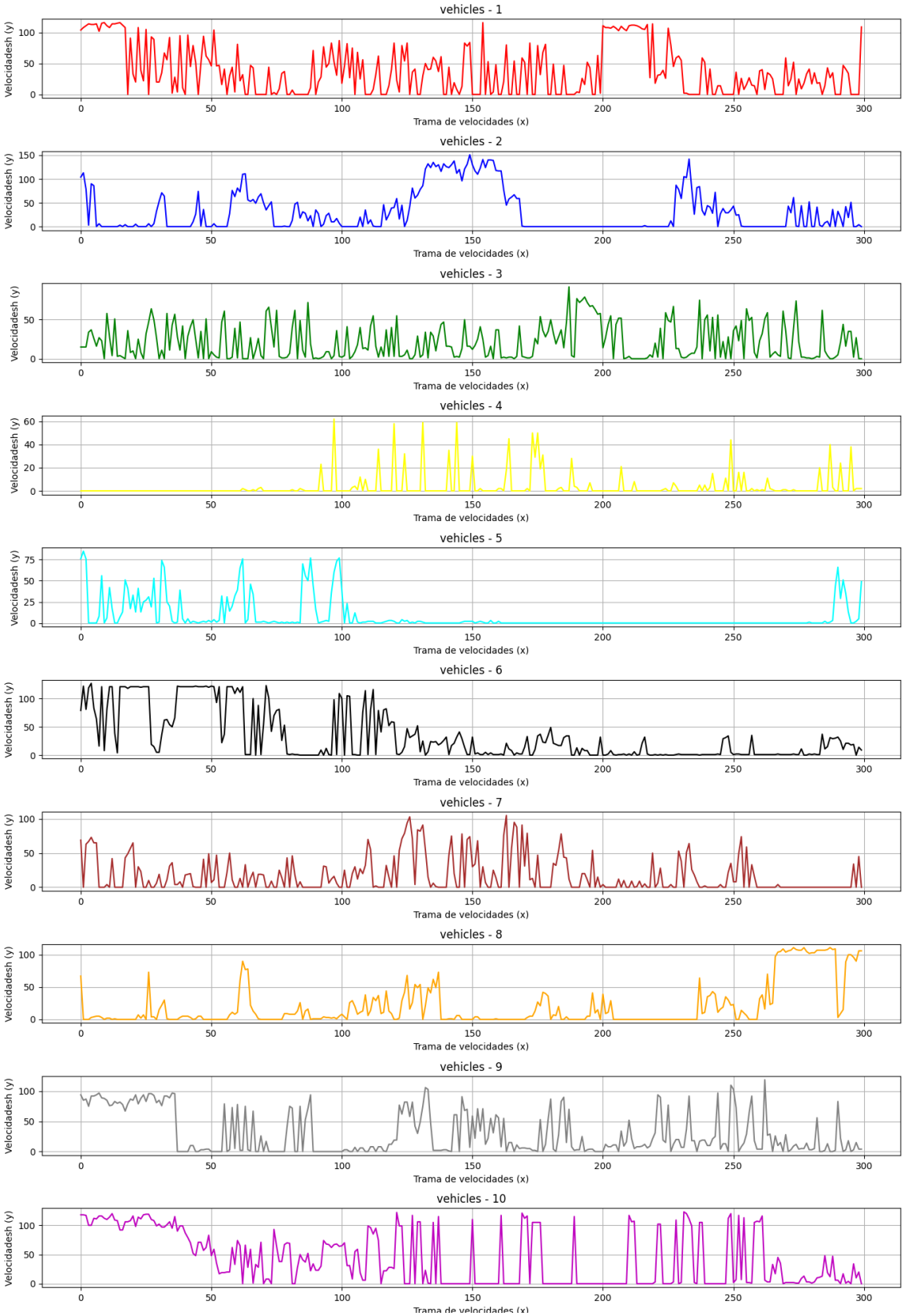
```
def get_data_acel(df):
    data_Array = list(df['velo'].head(300))
    new_Array = [int((i-0)/60) for i in data_Array]
    #"(velocidad final - velocidad incicial )/ tiempo (60 min) = aceleracion"
    return new_Array
```

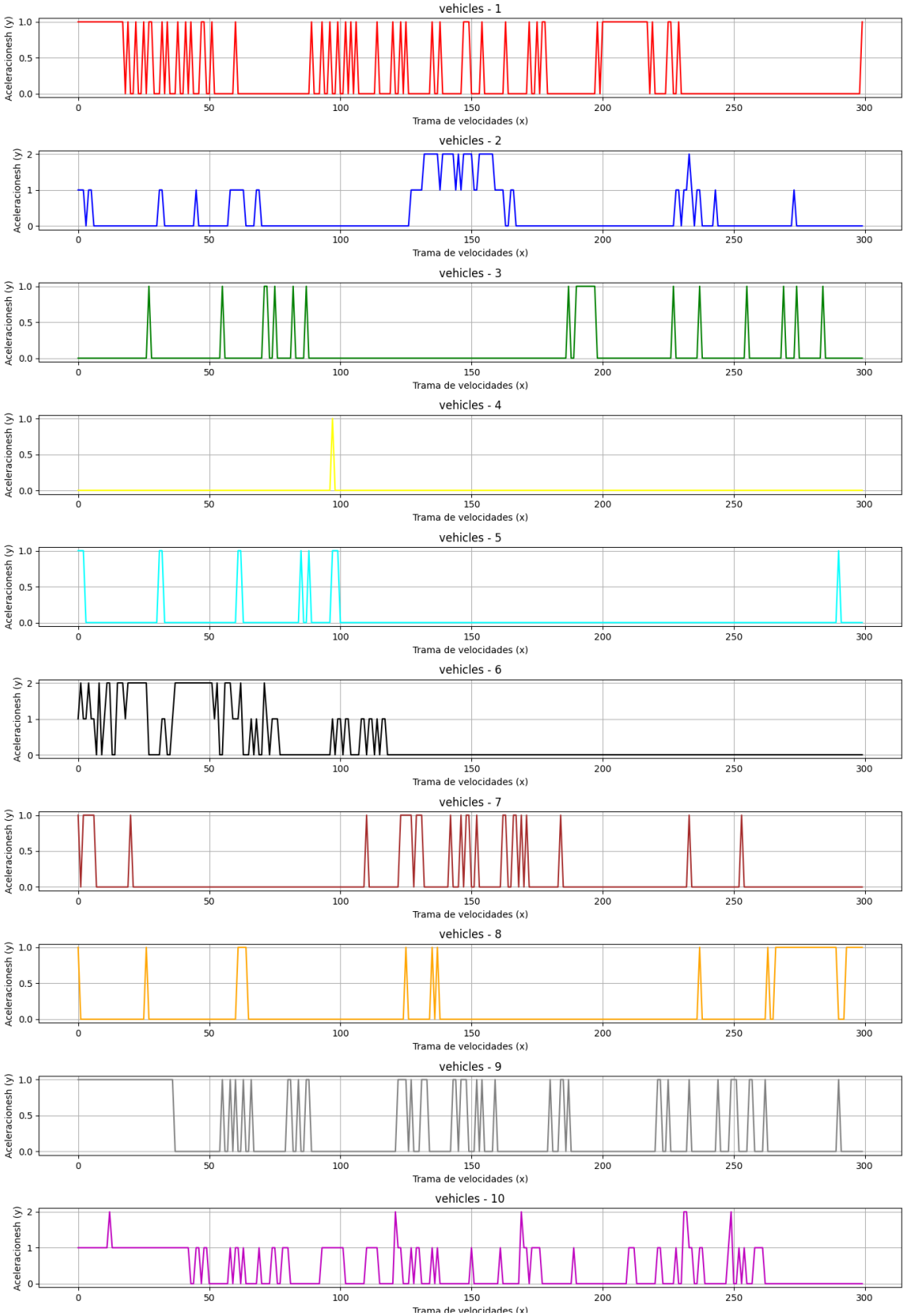
Las siguiente funciones se harán cargo de obtener los datos de los 10 vehículos añadirlas a un diccionario y luego graficarlas.

```
def Velocidad(df):
    new_dict = {1:[],2:[],3:[],4:[],5:[],6:[],7:[],8:[],9:[],10:[]}
    for i in range(1,11):
        temp = filter_ID(df,i)
        new_dict[i] = get_data_velo(temp)
    grafica(new_dict,'Velocidades')

def Aceleracion(df):
    new_dict = {1:[],2:[],3:[],4:[],5:[],6:[],7:[],8:[],9:[],10:[]}
    for i in range(1,11):
        temp = filter_ID(df,i)
        new_dict[i] = get_data_acel(temp)
    grafica(new_dict,'Aceleraciones')
```

A continuacion las imagenes de los 2 graficos con sus subplot correspondientes.




vehicles - 1
Aceleracionesh (y)
Trama de velocidades (x)
vehicles - 2
Aceleracionesh (y)
Trama de velocidades (x)
vehicles - 3
Aceleracionesh (y)
Trama de velocidades (x)
vehicles - 4
Aceleracionesh (y)
Trama de velocidades (x)
vehicles - 5
Aceleracionesh (y)
Trama de velocidades (x)
vehicles - 6
Aceleracionesh (y)
Trama de velocidades (x)
vehicles - 7
Aceleracionesh (y)
Trama de velocidades (x)
vehicles - 8
Aceleracionesh (y)
Trama de velocidades (x)
vehicles - 9
Aceleracionesh (y)
Trama de velocidades (x)
vehicles - 10
Aceleracionesh (y)
Trama de velocidades (x)

1.4. Problem 4.

Se nos pide que realicemos lo mismo que en la problemática número 1, pero en vez de un mismo cliente en Python, debemos enviar las imágenes por lazarus hacia un servidor de flask. Creamos un nuevo proyecto de lazarus, en la cual declaramos constantes que contengan las rutas de las imágenes, y una variable para poder recibir de alguna manera las imágenes en el servidor flask previamente.

```
unit Unit1;
{$mode objfpc}{$H+}
interface

uses
  Classes, SysUtils, Forms, Controls, Graphics, Dialogs,
  StdCtrls, ExtCtrls, fphttpclient, fpjson;

const
  inputfield = 'imgs';
  Img1 = '../figuras/a.jpg';
  Img2 = '../figuras/b.jpg';
  Img3 = '../figuras/c.jpg';
  Img4 = '../figuras/d.jpg';
  Img5 = '../figuras/e.jpg';
```

Al Form le agregamos 5 Botones, y también un TMemo en el cual almacenaremos la respuesta recibida desde el servidor cuando hayamos realizado la petición POST,.

```
type
  { TForm1 }
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    Button5: TButton;
    Label1: TLabel;
    Memo1: TMemo;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private

  public

  end;
```

Para Cada uno de los botones realizaremos un procedimiento cuando sea clickeado tal boton, mediante la librería FPHttpClient haremos una petición post hacia el servidor de flask, enviando con ella en un Formulario de tipo archivo la imagen respectiva al botón clickeado, luego él response recibido desde el servidor la almacenamos en el TMemo para luego en la interfaz poder visualizar lo ocurrido.

```
Procedure TForm1.Button1Click(Sender: TObject);
Var
  Respo: TStringStream;
  S : String;
Begin
  With TFPHttpClient.Create(Nil) do
    try
      Respo := TStringStream.Create('');
      FileFormPost('http://127.0.0.1:4000/data',
                   inputfield,
                   Img1,
                   Respo);
      S := Respo.DataString;
      Memo1.Append('IMG A: ->'+ S);
      Respo.Destroy;
    finally
      Free;
    end;
end;
```

Por parte del servidor tenemos casi el mismo código que en la primera problemática, lo unico que cambia es la ruta en la cual guardaremos las imagenes recibidas.

```
from flask import Flask, request, jsonify
from PIL import Image

app = Flask(__name__)

@app.route('/data', methods=['POST'])
def index():
    if request.method == "POST":
        files = request.files['imgs']
        img = Image.open(files)
        img = img.convert('L')
        img = img.transpose(method=Image.FLIP_TOP_BOTTOM)
        img.save('static/img_from_lazarus/'+ files.filename)
    return jsonify({ 'msg': 'FILE SAVE SUCCESS'})

if __name__ == '__main__':
    app.run(debug=True, port=4000)
```