



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Laboratorio de Temas Selectos de
Programación: Ciencia de Datos

Práctica 02

Implementación de una Interfaz de Programación de Aplicaciones (API)

Profesor: Dr. Miguel Serrano Reyes
Grupo: 02

Alumno: Flores Rivas Eduardo

Semestre: 2024-1
Entrega: 16/11/2023



Índice

Práctica 02: Implementación de una Interfaz de Programación de Aplicaciones (API)	3
Objetivo	3
Descripción	3
Requisitos	3
Condiciones Adicionales	3
Desarrollo de la práctica	3
Implementación del servidor Flask	3
Implementación del Script de prueba	5
Resultados y análisis	6
Conclusiones	6
Anexo de código fuente	7
Código del servidor Flask	7
Código del Script de pruebas	8

Práctica 02: Implementación de una Interfaz de Programación de Aplicaciones (API)

Objetivo

Familiarizarse con la creación de una Interfaz de Programación de Aplicaciones (API) básica utilizando el *microframework* *Flask* de Python y realizar pruebas a los *endpoints* utilizando la biblioteca *'requests'*.

Descripción

En esta tarea, desarrollarás una API simple con dos endpoints usando Flask. El primer endpoint aceptará datos a través de un método POST y responderá con un mensaje de confirmación. El segundo endpoint responderá a una solicitud GET con un saludo personalizado. Además, crearás un script de pruebas para validar el funcionamiento de ambos endpoints.

Requisitos

- Desarrollar un servidor Flask con dos endpoints:
- Un endpoint POST `"/nombre"` que reciba un dato y responda con un mensaje de confirmación.
- Un endpoint GET `"/mensaje"` que responda con un saludo personalizado basado en un dato recibido.
- Crear un script de pruebas utilizando la biblioteca *'requests'* que verifique el funcionamiento correcto de ambos endpoints.

Condiciones Adicionales

- Asegurarse de que el servidor Flask se ejecute correctamente sin errores.
- El script de pruebas debe mostrar claramente si las respuestas a las solicitudes son las esperadas.
- Manejar adecuadamente los tipos de contenido (Content-Type) en las solicitudes y respuestas.

Desarrollo de la práctica

Implementación del servidor Flask

El servidor Flask se ejecuta como un Script cualquiera de python, al hacerlo se mostrará en la terminal un mensaje con el nombre del servidor Flask (en este caso P2), también se incluirá la dirección IP y el puerto en la que se está ejecutando el servidor, para fines de esta práctica el servidor iniciará en la dirección 127.0.0.1 y el puerto 5000.

```
* Serving Flask app 'P2'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with watchdog (windowsapi)
* Debugger is active!
* Debugger PIN: 987-124-821
```

Además, se indica si el modo DEBUG está activo y el pin para acceder a esa función. Las letras en rojo muestran una advertencia relacionada con el tipo de servidor, y que éste no debe ser usado más que para desarrollo, pues no es seguro para la producción.

Se desarrolló con 3 endpoints distintos, los cuales tienen las siguientes funciones:

- Endpoint “/nombre”: es un método POST que se encarga de recibir el dato “nombre” desde el Script de prueba y responde con un mensaje de confirmación.

```
@app.route("/nombre", methods=["POST"])
def setName():
    if request.method=='POST':
        dato_posteado = request.get_json()
        dato = dato_posteado['dato']
        return jsonify(str("Almacenado correctamente " + str(dato)))
```

- Endpoint “/mensaje”: es un método GET que recibe un dato desde el Script de prueba y responde con un mensaje personalizado de saludo.

```
@app.route("/mensaje", methods=["GET"])
def message():
    dato_posteado = request.get_json()
    nombre = dato_posteado['nombre']
    return jsonify("Saludos cordiales, " + nombre)
```

- Endpoint “/raíces”: es un método GET que recibe tres datos que representan los coeficientes de un polinomio de segundo grado (a, b, c) desde el Script de prueba y responde con las raíces del polinomio en caso de que estas sean reales.

```
@app.route("/raices", methods=["GET"])
def calcularRaices():
    # Obtén los valores a, b y c de la URL
    coefa = float(request.args.get("a"))
    coefb = float(request.args.get("b"))
    coefc = float(request.args.get("c"))

    # Calcula las raíces de la ecuación cuadrática
    discriminante = coefb**2 - 4*coefa*coefc
    if discriminante > 0:
        x1 = (-coefb + math.sqrt(discriminante)) / (2*coefa)
        x2 = (-coefb - math.sqrt(discriminante)) / (2*coefa)
        return jsonify({"raices": [x1, x2]})
    elif discriminante == 0:
        x = -coefb / (2*coefa)
        return jsonify({"raices": [x]})
    else:
        return jsonify({"raices": []}) # No hay raíces reales
```

Implementación del Script de prueba

Este Script se encarga de probar la API establecida con el servidor Flask, para ello se establece el URL del servidor y se prueba cada endpoint del servidor por separado, a través de los pasos siguientes:

- Establecer el URL para cada método

```
# La URL del servidor, por ejemplo, localhost si estás ejecutándolo
localmente con el puerto predeterminado que Flask utiliza.
BASE_URL = 'http://127.0.0.1:5000'

# Probando el endpoint POST /nombre
url_nombre = f'{BASE_URL}/nombre'
```

- Cargar los datos del mensaje

```
# Datos que quieres enviar, reemplaza 'tu_nombre' con un valor real.
datos_para_enviar = {
    'dato': 'Eduardo Flores'
}
headers = {'Content-Type': 'application/json'}
```

- Enviar la solicitud del método (solo si estos son POST)


```
# Enviar la solicitud POST
response = requests.post(url=url_nombre, headers=headers,
data=json.dumps(datos_para_enviar))
```

- Verificar el estado de la solicitud

```
# Verificar si la solicitud fue exitosa
if response.status_code == 200:
    print("Respuesta del servidor:", response.json())
else:
    print(f"Error al hacer la solicitud: {response.status_code}")
```

El Script de prueba se ejecuta desde la libreta de Jupyter llamada test_script.ipynb, este archivo contiene el código necesario para llevar los pasos descritos anteriormente para los diferentes endpoints programados en el servidor Flask. Solo se debe ejecutar este archivo mientras el servidor se encuentra en ejecución, o de otra manera se obtendrá un error de conexión.

Resultados y análisis

Tras ejecutar el servidor Flask desde el símbolo del sistema ingresando `Python P2.py`, como se comentó en su implementación, se deberá ejecutar el Script de prueba desde su respectiva libreta de Jupyter presionando el signo  a un costado de su celda.

El resultado obtenido en la terminal donde se ejecutó el servidor será el siguiente:

```
127.0.0.1 - - [07/Nov/2023 15:41:38] "POST /nombre HTTP/1.1" 200 -  
127.0.0.1 - - [07/Nov/2023 15:41:38] "GET /mensaje HTTP/1.1" 200 -  
127.0.0.1 - - [07/Nov/2023 15:41:38] "GET /raices?a=1&b=-3&c=2 HTTP/1.1" 200 -
```

Esto representa la respuesta del servidor ante las solicitudes realizadas, observamos que se la primer parte es la dirección IP en la que se aloja el servidor, luego se coloca una estampa de tiempo; también se añade un identificador para cada solicitud con el tipo del método (POST o GET) y el nombre específico del método (/nombre, /mensaje o /raices); por último el código de respuesta que, en este caso, es 200 para todos, lo que indica que la acción solicitada por el cliente se llevó a cabo con éxito.

Por otro lado, dentro de la libreta de Jupyter se observa la respuesta obtenida por el Script de prueba con los datos cargados para la solicitud realizada al servidor. La respuesta fue la siguiente

```
Respuesta del servidor: Almacenado correctamente Eduardo Flores  
Respuesta del servidor: Saludos cordiales, Eduardo Flores  
Respuesta del servidor: {'raices': [2.0, 1.0]}
```

Estos mensajes se obtuvieron directamente de lo que se programó desde la libreta y corresponden a los métodos /nombre, /mensaje y /raices, respectivamente. El primero de ellos confirma que el dato enviado se almacenó correctamente con el método POST; el segundo responde a la solicitud con un saludo personalizado; y el tercero envía las raíces del polinomio de 2º grado para los coeficientes a, b y c proporcionados.

Conclusiones

En esta práctica se logró con éxito la implementación de una Interfaz de Programación de Aplicaciones (API) básica utilizando Flask de Python. Se cumplieron los objetivos establecidos, al desarrollar una API con tres endpoints: uno para recibir datos a través de un método POST y dos para responder a través de una solicitud GET (el primero para un saludo personalizado y el segundo para la obtención de las raíces de un polinomio de segundo grado). Los resultados obtenidos fueron consistentes, ya que las respuestas del servidor a las solicitudes arrojaron un código de respuesta 200 en todos los casos, indicando que las acciones se realizaron sin errores.

Además, el script de pruebas utilizando la biblioteca 'requests' demostró ser eficaz al verificar el funcionamiento correcto de ambas rutas, confirmando que los datos se almacenan correctamente y que el servidor responde con los mensajes esperados. Esta práctica permitió familiarizarse con la creación de una API básica y su validación, lo que representa un paso importante en el desarrollo de habilidades en el campo de la programación y la interacción con servicios web mediante Python y Flask.

Anexo de código fuente

Código del servidor Flask

```
from flask import Flask, jsonify, request
import math
app= Flask(__name__)

@app.route("/nombre", methods=["POST"])
def setName():
    if request.method=='POST':
        dato_posteado = request.get_json()
        dato = dato_posteado['dato']
        return jsonify(str("Almacenado correctamente " + str(dato)))

@app.route("/mensaje", methods=["GET"])
def message():
    dato_posteado = request.get_json()
    nombre = dato_posteado['nombre']
    return jsonify("Saludos cordiales, " + nombre)

@app.route("/raices", methods=["GET"])
def calcularRaices():
    # Obtén los valores a, b y c de la URL
    coefa = float(request.args.get("a"))
    coefb = float(request.args.get("b"))
    coefc = float(request.args.get("c"))

    # Calcula las raíces de la ecuación cuadrática
    discriminante = coefb**2 - 4*coefa*coefc
    if discriminante > 0:
        x1 = (-coefb + math.sqrt(discriminante)) / (2*coefa)
        x2 = (-coefb - math.sqrt(discriminante)) / (2*coefa)
        return jsonify({"raices": [x1, x2]})
    elif discriminante == 0:
        x = -coefb / (2*coefa)
        return jsonify({"raices": [x]})
    else:
        return jsonify({"raices": []}) # No hay raíces reales

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

Código del Script de pruebas

```
import requests
import json

# La URL del servidor, por ejemplo, localhost si estás ejecutándolo
localmente con el puerto predeterminado que Flask utiliza.
BASE_URL = 'http://127.0.0.1:5000'

# Probando el endpoint POST /nombre
url_nombre = f'{BASE_URL}/nombre'

# Datos que quieres enviar, reemplaza 'tu_nombre' con un valor real.
datos_para_enviar = {
    'dato': 'Eduardo Flores'
}
headers = {'Content-Type': 'application/json'}

# Enviar la solicitud POST
response = requests.post(url=url_nombre, headers=headers,
data=json.dumps(datos_para_enviar))

# Verificar si la solicitud fue exitosa
if response.status_code == 200:
    print("Respuesta del servidor:", response.json())
else:
    print(f"Error al hacer la solicitud: {response.status_code}")

# Probando el endpoint GET /mensaje
url_mensaje = f'{BASE_URL}/mensaje'

# Aquí, asumimos que el servidor espera un JSON con la clave 'nombre'.
# Esto podría no ser necesario si el método GET no requiere un cuerpo en la
solicitud.
datos_para_mensaje = {
    'nombre': 'Eduardo Flores'
}

response = requests.get(url=url_mensaje, headers=headers,
data=json.dumps(datos_para_mensaje))

# Verificar si la solicitud fue exitosa
if response.status_code == 200:
    print("Respuesta del servidor:", response.json())
else:
    print(f"Error al hacer la solicitud: {response.status_code}")
```



```
# Probando el endpoint GET /raices
url_raices = f'{BASE_URL}/raices'

# Datos que quieres enviar, escribe los coeficientes.
coeficientes = {
    'a': '1',
    'b': '-3',
    'c': '2'
}

response = requests.get(url=url_raices, params=coeficientes,
headers=headers, data=json.dumps(datos_para_mensaje))

# Verificar si la solicitud fue exitosa
if response.status_code == 200:
    print("Respuesta del servidor:", response.json())
else:
    print(f"Error al hacer la solicitud: {response.status_code}")
```