

**EJERCICIO 1**

- Sigue la GUIA DE LABORATORIO para construir tu API con Flask
- **Define una ruta para eliminar empleados:**
  - Crea una ruta en la API de Flask que escuche las solicitudes DELETE en la URL `/employees/<int:id>`, donde `<int:id>` representa el ID del empleado a eliminar.
  - La función asociada debe aceptar el parámetro `id`, que será el ID del empleado a eliminar.
- **Implementa la lógica de eliminación:**
  - Utiliza una función para eliminar al empleado con el ID especificado de la lista de empleados existente.
  - Si el empleado no existe en la lista, devuelve un mensaje de error con un código de estado 404 (Not Found).
  - Si se elimina correctamente, devolver los detalles del empleado eliminado con un código de estado 200 (OK).
- **Actualizar la lista de empleados**
  - Utiliza una lista global llamada `employees` que contenga los detalles de los empleados.
  - Actualiza la lista después de eliminar un empleado para reflejar los cambios.

## Creando el proyecto base

Para crear el proyecto base, lo primero que haremos será crear una carpeta con el nombre python-flask-api en el directorio elegido.

Con la nueva carpeta creada, abriremos una terminal en la raíz de la carpeta para que se puedan ejecutar comandos para construir y ejecutar nuestro proyecto Python. Una vez que su terminal apunte al directorio raíz de su proyecto, ejecute los siguientes comandos para que pueda inicializar el proyecto Python Rest API Flask y administrar las dependencias.

Primero, usaremos pip para instalar Flask en el directorio del proyecto. Para hacer esto, ejecute el siguiente comando.

```
pip install Flask
```

## Escribiendo el código

En nuestra primera línea de código en el app.py archivo, importaremos los módulos para json, Flask, jsonify y request.

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

A continuación, crearemos una nueva aplicación Flask agregando el siguiente código justo debajo de nuestras declaraciones de importación.

```
app = Flask(__name__)
```

A continuación, para darle a nuestra API algunos datos con los que trabajar, definiremos una matriz de objetos de empleado con un ID y un nombre.

```
employees = [ { 'id': 1, 'name': 'Ashley' }, { 'id': 2, 'name': 'Kate' }, { 'id': 3, 'name': 'Joe' } ]
```

Para definir nuestro punto final API, ahora agregaremos código para definir una ruta para solicitudes GET al punto final '/employees'. Esto devolverá todos los empleados (de nuestra matriz de empleados definida anteriormente) en formato JSON.

```
@app.route('/employees', methods=['GET'])  
def get_employees():  
    return jsonify(employees)
```

Además de nuestro método GET, también definiremos una ruta para los métodos POST. Estas funciones se pueden utilizar para crear un nuevo empleado y actualizar o eliminar al empleado según su identificación proporcionada.

```
@app.route('/employees', methods=['POST'])
def create_employee():
    global nextEmployeeId
    employee = json.loads(request.data)
    if not employee_is_valid(employee):
        return jsonify({'error': 'Invalid employee properties.'}), 400
    employee['id'] = nextEmployeeId
    nextEmployeeId += 1
    employees.append(employee)
    return "", 201, {'location': f'/employees/{employee["id"]}'}
@app.route('/employees/<int:id>', methods=['PUT'])
def update_employee(id: int):
    employee = get_employee(id)
    if employee is None:
        return jsonify({'error': 'Employee does not exist.'}), 404
    updated_employee = json.loads(request.data)
    if not employee_is_valid(updated_employee):
        return jsonify({'error': 'Invalid employee properties.'}), 400
    employee.update(updated_employee)
    return jsonify(employee)
```

Una vez que nuestro código esté completo, debería verse así:

```
import json
from flask import Flask, jsonify, request
app = Flask(__name__)
employees = [
    {'id': 1, 'name': 'Ashley'},
    {'id': 2, 'name': 'Kate'},
    {'id': 3, 'name': 'Joe'}
]
nextEmployeeId = 4
3
@app.route('/employees', methods=['GET'])
def get_employees():
    return jsonify(employees)
@app.route('/employees/<int:id>', methods=['GET'])
def get_employee_by_id(id: int):
    employee = get_employee(id)
    if employee is None:
        return jsonify({'error': 'Employee does not exist'}), 404
    return jsonify(employee)
def get_employee(id):
```

```
return next((e for e in employees if e['id'] == id), None)
def employee_is_valid(employee):
    for key in employee.keys():
        if key != 'name':
            return False
    return True
@app.route('/employees', methods=['POST'])
def create_employee():
    global nextEmployeeId
    employee = json.loads(request.data)
    if not employee_is_valid(employee):
        return jsonify({'error': 'Invalid employee properties.'}), 400
    employee['id'] = nextEmployeeId
    nextEmployeeId += 1
    employees.append(employee)
    return "", 201, {'location': f'/employees/{employee["id"]}'}
@app.route('/employees/<int:id>', methods=['PUT'])
def update_employee(id: int):
    employee = get_employee(id)
    if employee is None:
        return jsonify({'error': 'Employee does not exist.'}), 404
    updated_employee = json.loads(request.data)
    if not employee_is_valid(updated_employee):
        return jsonify({'error': 'Invalid employee properties.'}), 400
    employee.update(updated_employee)
    return jsonify(employee)
if __name__ == '__main__':
    app.run(port=5000)
```

Por último, agregaremos una línea de código para ejecutar nuestra aplicación Flask. Como puede ver, llamamos al `run` método y ejecutamos la aplicación Flask en el puerto 5000.

```
if __name__ == '__main__':
    app.run(port=5000)
```

## Ejecutar y probar el código

Con nuestro código escrito y guardado, podemos iniciar la aplicación. Para ejecutar la aplicación, en la terminal ejecutaremos el siguiente pip comando.

```
pip api.py
```

Ahora nuestra API está en funcionamiento. Puede enviar una solicitud HTTP de prueba a través de Postman. Enviando una solicitud a `localhost:5000/employees`.

Después de enviada la solicitud. Debería ver un 200 OK código de estado en la respuesta junto con una serie de empleados devueltos.

Para esta prueba, no se necesita ningún cuerpo de solicitud para la solicitud entrante.

http://localhost:5000/employees Save Send

GET Send http://localhost:5000/employees

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	*** Bulk Edit
Key	Value	Description	

Body Cookies Headers (5) Test Results Status: 200 OK Time: 11 ms Size: 237 B Save Response

Pretty Raw Preview Visualize JSON ≡

```

1 {
2   {
3     "id": 1,
4     "name": "Ashley"
5   },
6   {
7     "id": 2,
8     "name": "Kate"
9   },
10  {
11    "id": 3,
12    "name": "Joe"
13  }
14 }
```