GUÍA DE CLASE

Presentacion previa API Gateway

Fundación Universitaria de Popayán
Mag. Luis Vejarano | luis.vejarano@docente.fup.edu.co

Presentación Previa

Una vez ya están implementados los backends académico y de seguridad ahora es el momento de implementar el api Gateway, tal como se diseñó en el diagrama arquitectónico que se planteó desde el inicio del proyecto.

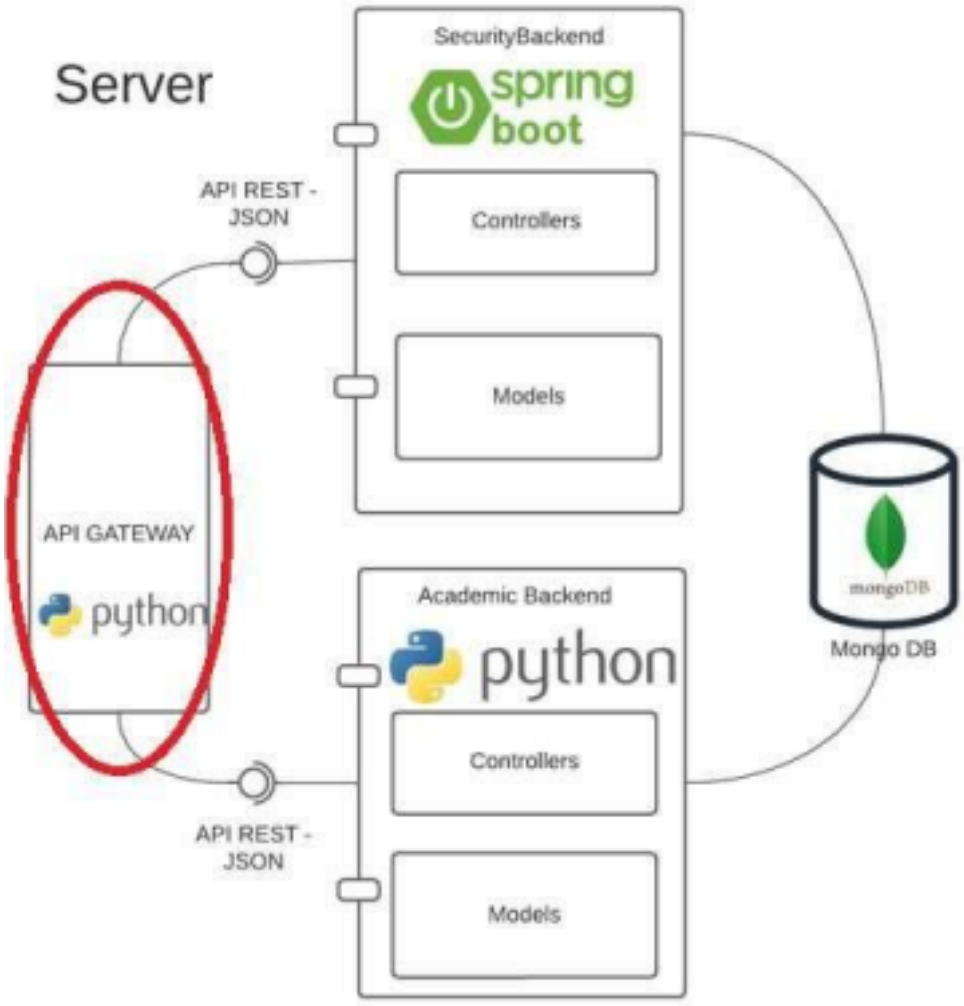
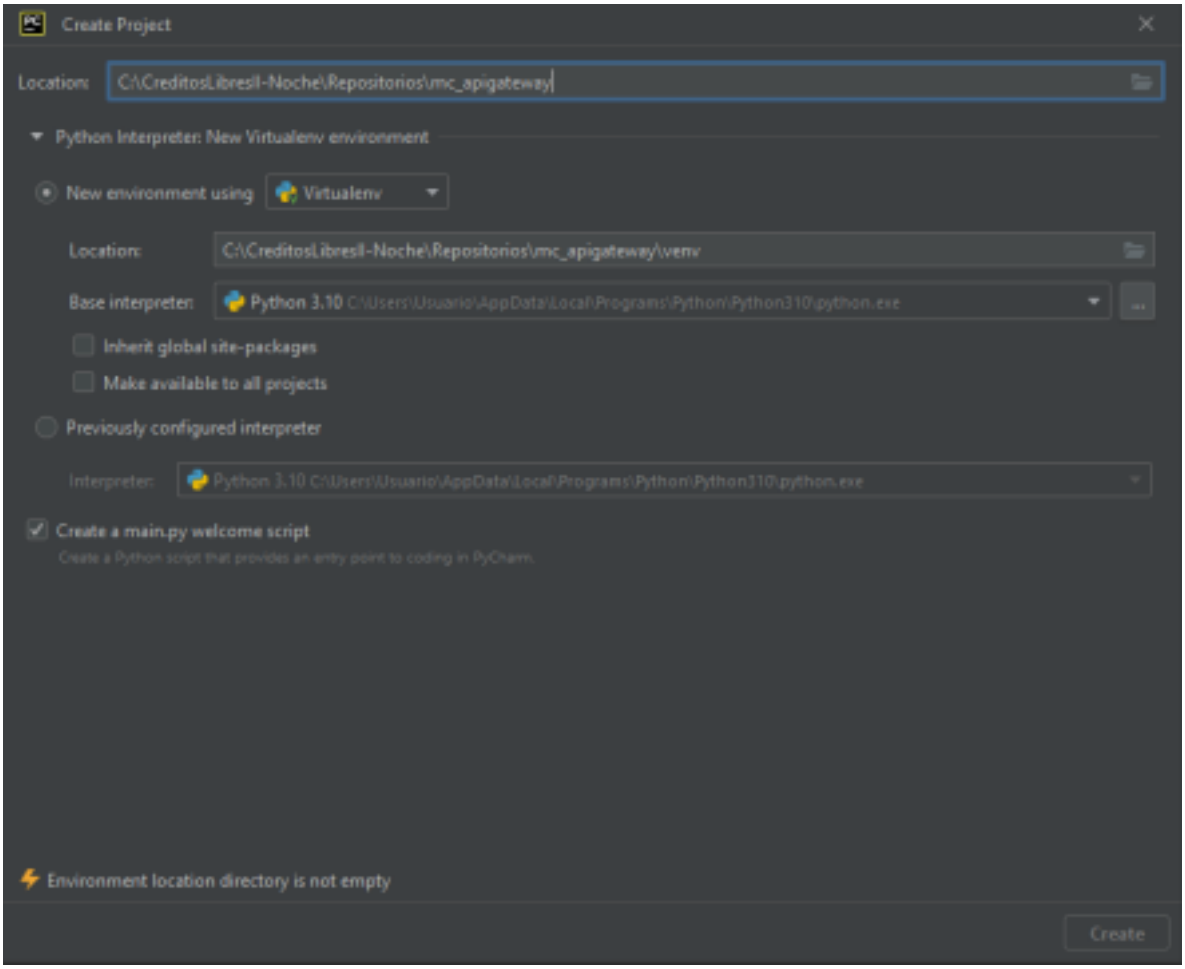


Imagen 1. Arquitectura del sistema

Por definición un api Gateway es "*es el gestor de tráfico que interactúa con los*

datos o el servicio backend real y aplica políticas, autenticación y control de acceso general para las llamadas de una API para proteger datos valiosos." [1]. Este concepto posee una gran variedad de ventajas asociadas con temas tales como la escalabilidad de las aplicaciones, análisis, supervisión y monitoreo de los servicios que ofrecen la plataforma, además la integración y llamados a múltiples microservicios ante una misma solicitud por parte del cliente [2]. Para el presente proyecto se implementará un api Gateway utilizando Python y Flask, como se ve en el diagrama arquitectónico, este será la puerta de entrada para manipular los servicios de los backends de seguridad y académicos. Este se encargará de procesar en un primer lugar las peticiones del cliente, validar si el usuario tiene los permisos suficientes para acceder a determinados servicios y además se podría programar otras funcionalidades para llevar a cabo estadísticas de funcionamiento, etc.

Creación del Proyecto



Luego de esto es necesario copiar las siguientes líneas de código, las cuales permiten llevar a cabo las importaciones de las librerías que se utilizarán en el

proyecto:

```
from flask import Flask
from flask import jsonify
from flask import request
from flask_cors import CORS
import json
from waitress import serve
import datetime
import requests
import re

app = Flask(__name__)
cors = CORS(app)

@app.route("/", methods=['GET'])
def test():
 json = {}
 json["message"] = "Server running ..."
 return jsonify(json)

def loadFileConfig():
 with open('config.json') as f:
 data = json.load(f)
 return data

if __name__ == '__main__':
 dataConfig = loadFileConfig()
 print("Server running : " + "http://" + dataConfig["url-backend"] + ":"
+ str(dataConfig["port"]))
 serve(app, host=dataConfig["url-backend"], port=dataConfig["port"])
```

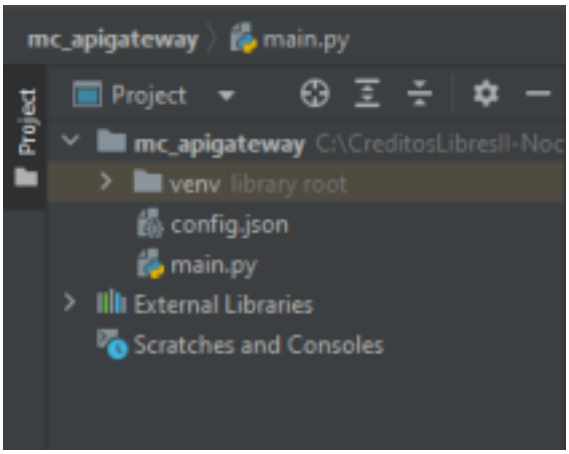
Es importante recordar que como se creó un nuevo entorno de Python, se deben de importar de nuevo las librerías tal como se explicó en el tutorial “2.2 PythonFlask-Creación y configuración del Proyecto “

Luego es necesario crear un archivo en la raíz del proyecto llamado “config.json”el cual debe tener el siguiente contenido:

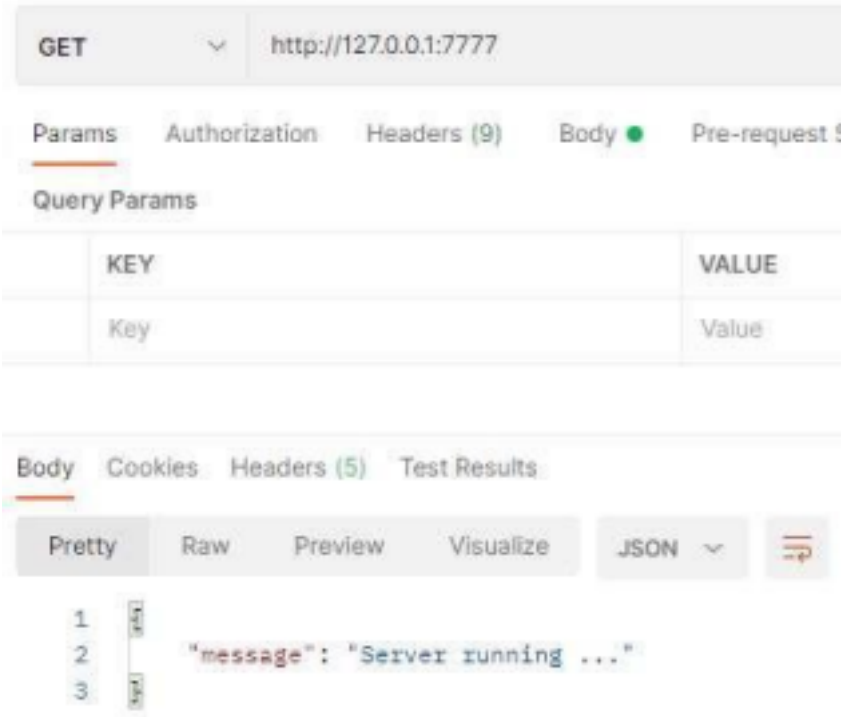
```
{
    "url-backend": "127.0.0.1",
    "port": 7777,
    "url-backend-security": "http://127.0.0.1:8080",
    "url-backend-academic": "http://127.0.0.1:9999"
```

}

En el archivo “config.json” se configura la información tal como la url del backend como el puerto que se utilizará para el presente proyecto, a su vez se define allí la ruta y puerto en la cual se encuentra corriendo el backend de seguridad y académico, los cuales han sido implementados en los tutoriales anteriores.



Luego de esto se puede ejecutar la main del proyecto y luego hacer una primera prueba utilizando postman para verificar el correcto funcionamiento hasta el momento del proyecto. Tal como se muestra a continuación.



Referencias

TIBCO, «¿Qué es API Gateway?,» [En línea]. Available: https://www.tibco.com/es/reference-center/what-is-an-api gateway#:~:text=Un%20API%20Gateway%20es%20el,API%20para%20proteger% 20datos% 20valiosos.. [Último acceso: 09 05 2022].

RedHat, «¿Cuál es la función de una puerta de enlace de API?,» [En línea]. Available: https://www.redhat.com/es/topics/api/what-does-an-api-gateway-do. [Último acceso: 0905 2022].