**Estructura de Evaluación**

Tiempo recomendado: 24 horas

Nivel de exigencia: Media

Nivel del problema: Media

Nota promedio recomendada: 14.6

Nombre: Catherine Lucero Varas Padilla

------------------

**Se adjunta**

* palabras.txt: Archivo de 500mil de filas
* palabras.100.txt: Archivo muestra con 100 filas

**El problema**

**Link del repositorio github:** [**https://github.com/CateHey/PythonPrueba-Puerto4000-Varas**](https://github.com/CateHey/PythonPrueba-Puerto4000-Varas)

Importante: Los comodines en python son diferentes **.+** equivale a \* y . equivale a ?

Es importante contar con **python instalado y la extensión de python**correspondiente de vs code, además de ejecutar el comando: **pip install flask** en el terminal para poder usar los objetos flask que nos permiten conectar con el servidor.

Ejemplo: http://localhost:4000/datos/M.NTES o <http://localhost:4000/datos/.+M>.+

1. Alimentar una base de datos en memoria con el archivo "palabras.txt". Puede utilizar el formato, técnica o artificio que considere mas eficiente para resolver este problema **(sustentar)**, tenga en cuenta que lo sustentado debe coincidir con el código del método que utilice.

import urllib, json

#nombre del archivo

data = []

with open("palabras.txt") as f:

        for line in f:

                line = line.split()

                data.append({"nombre": line[0]})

data = {"datos": data}

out\_file = open("listacompleta.json", "w")

json.dump(data, out\_file, indent = 1)

out\_file.close()

Se uso un JSON para almacenar los datos de los archivos txt

Se adjunta un pequeño fragmento del json generado, se puede visualizar completo en el archivo **listacompleta.json**:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

1. Construir un servicio web de tal manera que permita consultar los datos que se encuentran en la base de datos.

**Se uso la líbreria Flask de Python para la implementación del servidor**

**Es importantes resaltar que en Python los comodines cambian. Los equivalente de los comodines son para "\*" = “,+” y para "? = “.". Por ejemplo: Al poner “.+sa.+” , debe devolver los valores: "sa", "santo", "mesa", "casaca". Y al poner ".so" , devolverá: "eso".**

Librerias: Json, flask(para el servidor), jsonify, re (para buscar el parametro)

Controlador para lista 500 mil:

import json

from flask import Flask, jsonify

import re

#app es la aplicacion de servidor

app = Flask (\_\_name\_\_)

#abrimos el archivo json generado con el txt

f = open ('listacompleta.json',"r")

c= f.read()

#retorna un json como diccionario

#cerramos el archivo json

f.close()

data = json.loads(c)

@app.route('/datos/<string:query>', methods=['GET'])

def datos(query):

    resultado= []

    for i in range(0, 99):

        if (re.search(query, data["datos"][i]['nombre'])):

            resultado.append(data["datos"][i])

    return jsonify(resultado)

#si se ejecuta como main o archivo principal

if \_\_name\_\_ == '\_\_main\_\_':

#ejecutamos como debug

    app.run(debug=True, port=4000)

Texto

Descripción generada automáticamente

Para ejecutar el servidor, click derecho al final del app.py y click en Run Python File in Terminal. La dirección para acceder al servidor es, el puerto puede cambiarse desde le app.run, el port por default es: 4000. El query debe tener un valor.

<http://localhost:4000/datos/query>

Ejemplo para el primer comodín:

Lista 500 mil: <http://localhost:4000/datos/.+M>.+

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza baja

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Ejemplo 2 para segundo comodin:

Lista 500 mil: [localhost:4000/datos/M.NTES](http://localhost:4000/datos/M.NTES)

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

3, Este servicio web puede tener la arquitectura que crea más conveniente **(sustentar)**, tenga en cuenta que lo sustentado debe coincidir con el código de su servicio.

**Estructura:**

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

* El **lector.py** lee archivos y los convierte en json, usa librerías como json y urllib.
* En el proyecto se encuentran dos listas json, una listacompleta.json de palabras.txt y otra lista.json de palabras.100.txt
* El **app.py** es el rest api con flask, que nos permite enviar un parametro para buscar dentro del json de la data y nos envía los elementos en los que hay coincidencias
* Se puede cambiar el puerto desde app.py, después de cualquier cambio debe volver a ejecutar el app.py.

Texto

Descripción generada automáticamente

**Las consideraciones**

* Se harán 5 pruebas al servicio con Postman (las pruebas serán iguales para todos los participantes). Se tomará el promedio del tiempo de respuesta en milisegundos y se comparará con el tiempo promedio de todos los participantes. El ganador obtendrá 3 puntos adicionales, en caso de empate, todos los empatados obtendrá +3 puntos.
* El endpoint debe tener un parámetro llamado "query" y recibirá una cadena que puede tener comodines. Los comodines son "\*" y "?". Por ejemplo: Al poner "\*sa\*", debe devolver los valores: "sa", "santo", "mesa", "casaca". Y al poner "?so", devolverá: "eso".
* Cada prueba estará compuesta de la siguiente forma
  + elementos: 10, prueba: http://<URL>:[PORT]/api?cs=1&top=100&query=\*ru??a
    - Esta prueba debe devolver 10 elementos
    - cs: La consulta es case sensitive solo si el valor es 1 [opcional]
    - top: Devolverá los primeros resultados que indique este parámetro [opcional]
    - query: Cadena que se buscará [obligatorio]
* **No deberá utilizar ningún motor de base datos. Su uso invalida la prueba**
* Para esta prueba no hay penalidad por usar librerías
* Subir su proyecto a un repositorio git como <https://github.com/>

**La calificación**

* Se harán 5 pruebas con postman al servicio (3.4 pts por cada consulta exitosa)
* La nota solo llega hasta 17. El ganador de las pruebas obtiene 3 puntos adicionales
* No sustentar adecuadamente el punto 1 del problema costará -0.25 por cada parte de la sustentación que no coincida con el método utilizado y -0.5 por cada concepto mal aplicado.
* No sustentar adecuadamente la arquitectura utilizada le costará -0.25 por cada parte de la sustentación que no coincida con el servicio, y -0.5 por cada concepto mal aplicado.
* Las sustentaciones de los puntos 1 y 3 deben ser enviadas por correo en un archivo word.
* La claridad y organización del proyecto serán evaluadas: -0.25 por errores menores, -0.5 por errores medios, -1 por cada error grave