Implementación funcionamiento: Desafio_Uno_Luis_Gonzalez

Para que la implementacion pueda funcionar de manera correcta se debe configurar de la siguente forma:

Servidor de Aplicaciones WildFly V 10.x.x

- La carpeta WildFly contiene dos archivos *.properties que hace que el servicio rest funcione correctamente ' wildfly-10.1.0.Final/standalone/configuration ' que son:

```
DateGenerateWS.properties
DateGenerateEJB.properties
```

Opciones de Deploy de componentes war y ear

- Para poder deployar puede utilizar dos maneras para poder hacerlo efectivo.

```
1- Se puede deployar los componentes en la carpeta deployments que se
encuentra en la carpeta wildfly ' wildfly-
10.1.0.Final/standalone/deployments ', los componentes se encuentran
compilados en la carpeta target.
      - war:
/Desafio_Uno/Proyect/DateGenerate/DateGenerateWS/target/DateGenerateWS-
0.0.1-SNAPSHOT

      - ear:
/Desafio_Uno/Proyect/DateGenerate/DateGenerateEJBEAR/target/DateGenerateEJB
EAR-0.0.1-SNAPSHOT

2- La segunda opción es desde el IDE de JAVA ya sea Eclipse, Netbeans,
JBoss Developer Studio, importando el proyecto como "maven proyect",
generar el comando 'mvn clean compile package' y configurar el servidor de
aplicaciones wildfly 10 indicando la ruta donde se encuentra la carpeta
wildfly-10.1.0.Final
```

Prueba de servicio

- Para poder probar el servicio rest, se debe realizar la siguiente configuracion probando desde un plugin desde el navegador.

```
PLUGIN REST
  - RESTer
```

```
    url:
https://chrome.google.com/webstore/detail/rester/eejfoncpjfgmeleakejdcanedm
efagga
  -Postman
    url:
https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcddcbnc
dddomop

CONFIG HEADERS

Method:POST
URL: localhost:8080/DateGenerateWS/api/dateGenerateProyect/DateGenerate
Name: Content-Type
Value: application/json
```

```
BODY:
{
    "id": 6,
    "fechaCreacion": "2000-03-01",
    "fechaFin": "2010-01-01",
    "fechas": [
      "1969-03-01",
      "1969-05-01",
      "1969-09-01",
      "1970-01-01"]
}
```

Desafío 1: Periodos perdidos

El desafío consiste en lo siguiente:

- Existe un servicio REST que llamaremos Generador De Datos o GDD.
- El servicio responde con una lista de fechas generadas aleatoriamente. Estas fechas se encuentran en un lapso definidos por dos valores: fechaCreacion y fechaFin.
- Cada fecha generada corresponde al primer día de un mes.
- La respuesta contienen un máximo de 100 fechas.
- El servicio no entrega todas las fechas dentro del periodo, omite algunas de forma también aleatoria.
- El objetivo de este ejercicio es que determines cuáles son los periodos que faltan.

Este es un ejemplo de la respuesta que entrega este servicio:

```
{
    "id": 6,
    "fechaCreacion": "1968-08-01",
    "fechaFin": "1971-06-01",
```

```
    "fechas": [
      "1969-03-01",
      "1969-05-01",
      "1969-09-01",
      "1971-05-01"]
}
```

Acá se puede apreciar que el servicio generó fechas entre el 1 de agosto de 1968 y el 1 de junio de 1971. Sólo se generaron 4 fechas en este caso. De acuerdo a esto, faltarían 5 fechas de 1968, 9 fechas de 1969 y 5 fechas de 1971. Una versión del GDD se encuentra en este repositorio en GitHub: https://github.com/previred/Generador_Datos_Desafio_Uno

El desafío puede ser resuelto de tres maneras distintas. Tú eliges cuál es la que más te acomoda entre estos tres niveles:

Nivel 1:

```
Crear un programa que recibe, a través de la entrada estándar, un archivo
en formato Json con la estructura de la respuesta de servicio (como el
ejemplo de arriba) y que entrega a través de la salida estándar, como
respuesta, un archivo Json con las fechas faltantes.
```

Ejemplo: Se entrega un archivo con este contenido:

```
{
    "id": 6,
    "fechaCreacion": "1969-03-01",
    "fechaFin": "1970-01-01",
    "fechas": [
      "1969-03-01",
      "1969-05-01",
      "1969-09-01",
      "1970-01-01"]
}
```

El programa debe responder con archivo con este contenido:

```
{
    "id": 6,
    "fechaCreacion": "1968-08-01",
    "fechaFin": "1971-06-01",
    "fechasFaltantes": [
      "1969-04-01",
      "1969-06-01",
      "1969-07-01",
      "1969-08-01",
```

```
        "1969-10-01",
        "1969-11-01",
        "1969-12-01"]
}
```

El programa se debe ejecutar de la siguiente manera: $ mi_solucion < nombre_archivo_entrada > nombre_archivo_salida

Nivel 2:

Construir un programa que invoque al servicio REST GDD y escriba como salida un archivo con las fechas, los periodos recibidos y la lista de periodos faltantes. Ejemplo:

```
INVOCACION:
        $ mi-solucion
SALIDA (un archivo con el siguiente contenido) :
          fecha creación: 2018-10-01
         fecha fin: 2019-04-01
         fechas recibidas: 2018-10-01, 2018-12-01, 2019-01-01, 2019-04-01
            fechas faltantes: 2018-11-01, 2019-02-01, 2019-03-01
```

Nivel 3:

Implementar un nuevo servicio REST. Este servicio REST debe invocar al servicio GDD y entregar la respuesta en formato JSON con las fechas recibidas y las fechas faltantes. Ejemplo de la respuesta que debería entregar:

```
{
    "id": 6,
    "fechaCreacion": "1969-03-01",
    "fechaFin": "1970-01-01",
    "fechas": [
      "1969-03-01",
      "1969-05-01",
      "1969-09-01",
      "1970-01-01"],
    "fechasFaltantes": [
      "1969-04-01",
      "1969-06-01",
      "1969-07-01",
      "1969-08-01",
      "1969-10-01",
      "1969-11-01",
      "1969-12-01"]

}
```

REQUISITOS:

- Se pueden implementar las soluciones en cualquier lenguaje y framework. Aunque recomendamos usar: Java(con Spring Boot), Go y Python.
- La solución debe ser enviada vía un pull request a este repositorio.
- La solución debe contener un README con las instrucciones para compilar e instalar.
- Puedes implementar cualquiera de los 3 niveles, no es necesario implementar los 3.
- Hay bonus si usas SWAGGER.
- Junto con la solución debes entregar un archivo con la entrada y con la salida en formato JSON.