Parcial 2do Tercio -Ejercicios de Diseño

Enunciado

Diseñe, construya y despliegue una aplicación web para investigar el problema matemático asignado. El programa debe estar desplegado en AWS. Las tecnologías usadas en la solución deben ser Maven, Git, GitHub, Spring, HTML5 y js. No use liberías adicionales.

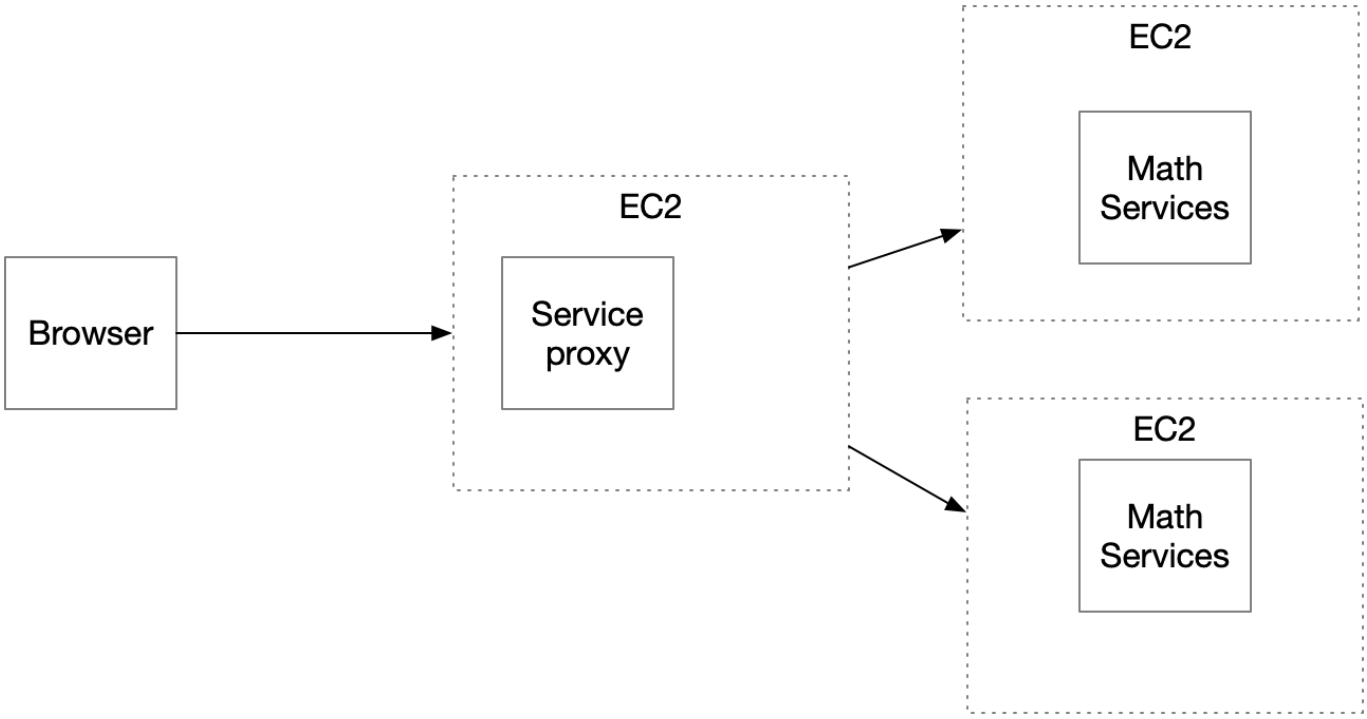
Desactive cualquier agente de IA de su IDE. Solo use la documentación autorizada por el profesor. No copie código existente ni suyo ni de otras fuentes ni generado por IA. Todo el entregable debe ser original, creado en la clase por usted.

Documentación autorizada:

https://docs.oracle.com/javase/8/docs/api/ https://spring.io/guides/gs/rest-service
https://docs.aws.amazon.com/corretto/latest/corretto-17-ug/amazon-linux-install.html AWS Academy AWS

Problema:

Diseñe un prototipo de sistema de microservicios que tenga un servicio (En la figura se representa con el nombre Math Services) para computar las funciones numéricas. El servicio de las funciones numéricas debe estar desplegado en al menos dos instancias virtuales de EC2. Adicionalmente, debe implementar un service proxy que reciba las solicitudes de llamado desde los clientes y se las delegue a las dos instancias del servicio numérico usando un algoritmo de round-robin. El proxy deberá estar desplegado en otra máquina EC2. Asegúrese de poder configurar las direcciones y puertos de las instancias del servicio en el proxy usando variables de entorno del sistema operativo. Finalmente, construya un cliente Web mínimo con un formulario que reciba el valor y de manera asíncrona invoque el servicio en el PROXY. Puede hacer un formulario para cada una de las funciones. El cliente debe ser escrito en HTML y JS.



Detalles adicionales de la arquitectura y del API

Construya una aplicación web para investigar este problema. La aplicación debe tener esta arquitectura:

- Cliente asíncrono que corra en el browser escrito en HTML5 y JS (No use librerías, solo html JS básico).
- El cliente NO COMPUTA LA SECUENCIA DIRECTAMENTE, sino que la delega a un servicio REST corriendo en AWS.
- El servicio REST puede ser GET o POST.
- Se debe construir la aplicación usando Spring Boot y desplegarla en un contenedor corriendo en AWS.
- Cree un solo repositorio en github para toda la aplicación.
- Use los mejores estándares de diseño, arquitectura y programación.

Entregable:

1. Proyecto actualizado en github (un solo repositorio)
2. Descripción del proyecto en el README con pantallazos que muestren el funcionamiento.
3. Descripción de cómo correrlo en EC2.
4. Video de menos de un minuto del funcionamiento (lo puede tomar con el celular una vez funcione).

En el campo de texto escriba la dirección de su repositorio GITHUB.

Ayuda

Para invocar un servicios get desde java puede hacerlo de manera fácil con:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.net.URL;

public class HttpConnectionExample {

    private static final String USER_AGENT = "Mozilla/5.0";
    private static final String GET_URL = "https://www.alphavantage.co/query?
function=TIME_SERIES_DAILY&symbol=fb&apikey=Q1QZFVJQ21K7C6XM";

    public static void main(String[] args) throws IOException {

        URL obj = new URL(GET_URL);
        HttpURLConnection con = (HttpURLConnection) obj.openConnection();
        con.setRequestMethod("GET");
        con.setRequestProperty("User-Agent", USER_AGENT);

        //The following invocation perform the connection implicitly before
getting the code
        int responseCode = con.getResponseCode();
        System.out.println("GET Response Code :: " + responseCode);
```

```
        if (responseCode == HttpURLConnection.HTTP_OK) { // success
            BufferedReader in = new BufferedReader(new InputStreamReader(
                    con.getInputStream()));
            String inputLine;
            StringBuffer response = new StringBuffer();

            while ((inputLine = in.readLine()) != null) {
                response.append(inputLine);
            }
            in.close();

            // print result
            System.out.println(response.toString());
        } else {
            System.out.println("GET request not worked");
        }
        System.out.println("GET DONE");
    }

}
```

Para invocar servicios rest de forma asíncrona desde un cliente JS

```
<!DOCTYPE html>
<html>
  <head>
    <title>Form Example</title>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  </head>
  <body>
    <h1>Form with GET</h1>
    <form action="/hello">
      <label for="name">Name:</label><br />
      <input type="text" id="name" name="name" value="John" /><br /><br />
      <input type="button" value="Submit" onclick="loadGetMsg()" />
    </form>
    <div id="getrespmsg"></div>
    <script>
      function loadGetMsg() {
        let nameVar = document.getElementById("name").value;
        const xhttp = new XMLHttpRequest();
        xhttp.onload = function () {
          document.getElementById("getrespmsg").innerHTML = this.responseText;
        };
        xhttp.open("GET", "/hello?name=" + nameVar);
        xhttp.send();
      }
    </script>
```

```
    <h1>Form with POST</h1>
    <form action="/hellopost">
      <label for="postname">Name:</label><br />
      <input type="text" id="postname" name="name" value="John" /><br /><br />
      <input type="button" value="Submit" onclick="loadPostMsg(postname)" />
    </form>
    <div id="postrespmsg"></div>
    <script>
      function loadPostMsg(name) {
        let url = "/hellopost?name=" + name.value;
        fetch(url, { method: "POST" })
          .then((x) => x.text())
          .then((y) => (document.getElementById("postrespmsg").innerHTML = y));
      }
    </script>
  </body>
</html>
```

Aplicación Spring mínima

Controller

```
package co.edu.eci.lambda.springrest;

import java.util.concurrent.atomic.AtomicLong;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class GreetingController {

    private static final String template = "Hello, %s!";
    private final AtomicLong counter = new AtomicLong();

    @GetMapping("/greeting")
    public Greeting greeting(@RequestParam(value = "name", defaultValue = "World")
String name) {
        return new Greeting(counter.incrementAndGet(), String.format(template,
name));
    }
}
```

Main

```java
package co.edu.eci.lambda.springrest;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class RestServiceApplication {

    public static void main(String[] args) {
        SpringApplication.run(RestServiceApplication.class, args);
    }

}
```

Record

```java
package co.edu.eci.lambda.springrest;

public record Greeting(long id, String content) { }
```

Ejemplo pom.xml de Spring con plugins

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.3.0</version>
        <relativePath/>
    </parent>
    <groupId>com.example</groupId>
    <artifactId>rest-service-complete</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>rest-service-complete</name>
    <description>Demo project for Spring Boot</description>
    <properties>
        <java.version>17</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
```

```
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

Ejercicios de diseño posibles

1. Conjetura de Collatz

La conjetura de Collatz dice que si usted crea una secuencia de números, a partir de cualquier entero positivo, siguiendo las reglas descritas abajo, siempre la secuencia terminará en el número 1. Esta conjetura aún no se ha demostrado.

Las reglas son: **f(n)=n/2** si n es par. **f(n)=3n+1** si n es impar.

La secuencia se construye a partir de un número dado **k** así: $a_0 = k$ $a_i = f(a_{i-1})$

Por ejemplo, dato el número **k=13** la secuencia sería:

13→40→20→10→5→16→8→4→2→1

Detalles adicionales de la arquitectura y del API

Servicio REST para construcción de la secuencia de Collatz, el servicio puede ser GET o POST. El servicio recibe el número inicial en la variable del query con el nombre `"value"`.

Ejemplo de un llamado:

```
http://amazonxxx.x.xxx.x.xxx:{port}/collatzsequence?value=13
```

Salida. El formato de la salida y la respuesta debe ser un JSON con el siguiente formato:

```
{
 "operation": "collatzsequence",
 "input":  13,
 "output":  "13 -> 40 -> 20 -> 10 -> 5 -> 16 -> 8 -> 4 -> 2 -> 1"
}
```

---

2. BÚSQUEDA BINARIA

La búsqueda binaria es un método más eficiente que la búsqueda lineal, pero requiere que el conjunto de datos esté ordenado previamente. Su proceso se describe de la siguiente manera:

Inicio

Determinar los índices de inicio y fin del conjunto de datos, que inicialmente son el primer y último elemento, respectivamente.

División

Calcular el índice medio del conjunto de datos actual y comparar el elemento en esta posición con el valor buscado.

Comparación

- Si el elemento medio es igual al valor buscado, se retorna la posición de este elemento, indicando que se ha encontrado.
- Si el elemento medio es mayor que el valor buscado, se descarta la mitad superior del conjunto y se repite el proceso con la mitad inferior.
- Si el elemento medio es menor que el valor buscado, se descarta la mitad inferior del conjunto y se repite el proceso con la mitad superior.

Iteración

El proceso se repite, reduciendo a la mitad el tamaño del conjunto de datos en cada paso.

Finalización

Este proceso continúa hasta que se encuentra el valor o el subconjunto se reduce a cero. La búsqueda binaria es muy eficiente en conjuntos de datos grandes, ya que reduce significativamente el número de comparaciones necesarias en comparación con la búsqueda lineal, logrando una complejidad temporal de:

`O(log n)`

donde `n` es el número de elementos en el conjunto de datos.

Detalles adicionales de la arquitectura y del API

Implemente los servicios para responder al método de solicitud HTTP GET. Deben usar el nombre de la función especificado, los parámetros deben ser pasados en variables de query con nombres **'list'** y **'value'**. El

proxy debe delegar el llamado a los servicios de backend. El proxy y los servicios se deben implementar en Java usando Spark.

Ejemplo 1 de un llamado:

```
https://amazonxxx.x.xxx.x:xxx/port[/linearSearch?list=10,20,13,40,60&value=13
```

Salida Formato JSON:

```
{
  "operation": "linearSearch",
  "inputList": "10,20,13,40,60",
  "value": "13",
  "output": "2"
}
```

Ejemplo 2 de un llamado:

```
https://amazonxxx.x.xxx.x:xxx/port[/linearSearch?list=10,20,13,40,60&value=99
```

Salida Formato JSON:

```
{
  "operation": "linearSearch",
  "inputList": "10,20,13,40,60",
  "value": "99",
  "output": "-1"
}
```

Ejemplo 3 de un llamado:

```
https://amazonxxx.x.xxx.x:xxx/port[/binarysearch?list=10,20,13,40,60&value=13
```

Salida Formato JSON:

```
{
  "operation": "binarySearch",
  "inputList": "10,20,13,40,60",
  "value": "13",
```

```
  "output": "2"
}
```

Entregable

- Proyecto actualizado en github (un solo repositorio)
- Descripción del proyecto en el README con pantallazos que muestren el funcionamiento.
- Descripción de como correrlo en EC2.
- Video de menos de un minuto del funcionamiento (lo puede tomar con el celular una vez funcione)