Taller Clientes y Servicios (MicroSpark)

Julián Eduardo Arias Barrera

Septiembre 2020

1 Introducción

Esta aplicación web permite colocar servicios/funciones por path (o paths y query) y también retornar recursos estáticos a través de funciones simples usando lambda.

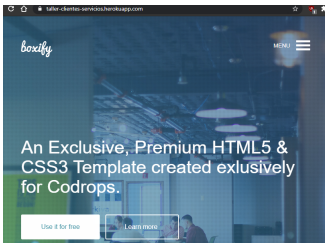

Figure 1: Resultado de Respuesta de HTML , js , css y png/jpg

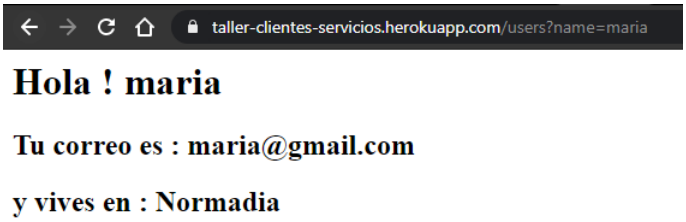

Figure 2: Resultado de funcion de consulta en una base de datos

Se creo una implementación propia de get sobre un "MicroSpark"

2 Instalación y Ejecución

Con la terminal debemos ejecutar los siguientes comandos:

- Debemos clonar el repositorio con la instrucción:

  ```
  > git clone https://github.com/AriasAEnima/Taller-Clientes-Servicios.git
  ```

  Podremos usarlo en local directamente con java con el siguiente comando

- En linux:

```
> java $JAVA_OPTS -cp target/classes:target/dependency/*

edu.escuelaing.arep.tallerClientesServicios.microSpark.MicroSparkServer
```

- En windows:

```
> java -cp target/classes:target/dependency/*
edu.escuelaing.arep.tallerClientesServicios.microSpark.MicroSparkServer
```

- O con heroku CLI si se tiene instalado con

```
> heroku local web
```

3 Diseño

En cuanto al diseño de la capa lógica:

Las interfaz interna definida como FuntionResponse seria una Interfaz Funcional que recibe un path y un request por medio de notación lambda como clases anónimas al utilizar el metodo getResponse(path,func) . Podríamos decir que un acercamiento a un patrón de comando [1]. Se utilizo un ResourceChooser para determinar el tipo de recurso estático que se va a responder y devolver el writer concreto que pueda responder adecuadamente.

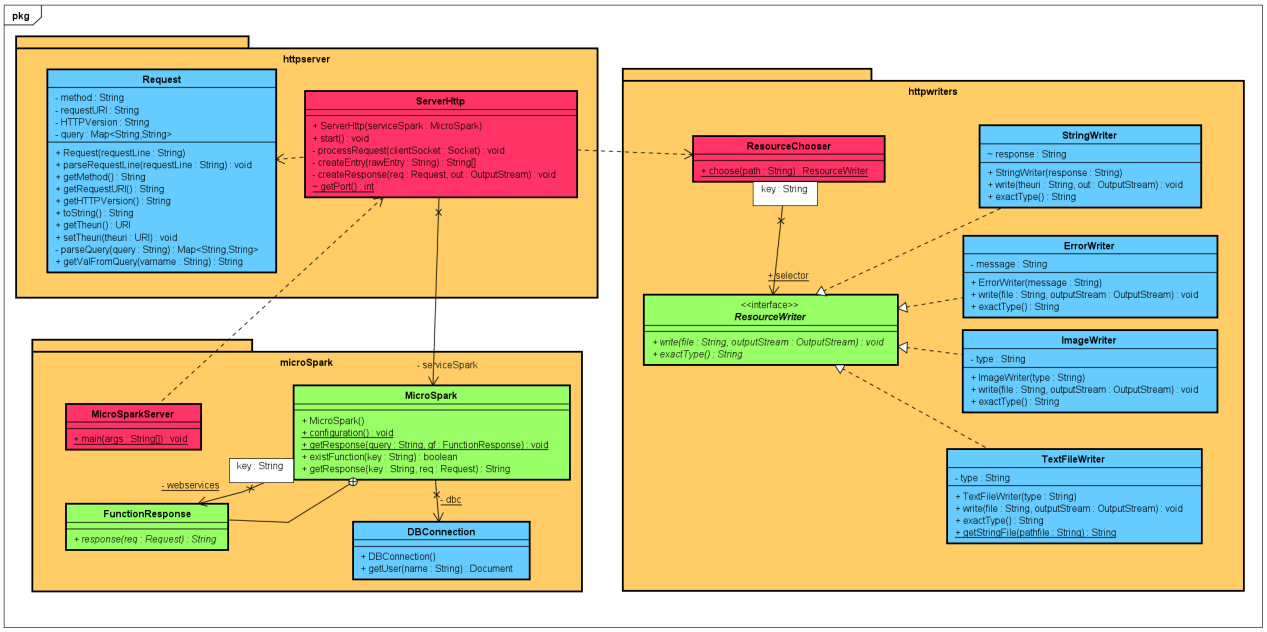

Figure 3: Modelo

La implementación de MicroSpark nos permite agregar funciones fácilmente haciendo mucho mas facil el trabajo.

```java
public MicroSpark() {
    webservices=new HashMap<>();
    dbc=new DBConnection();
    configuration();
}


/**
 * Se puede añadir servicios usando el metodo getResponse el constructor llamara esta funcion
 */
public static void configuration(){
    getResponse("/hello",(req)->
        "<h1> Hola ! "+ req.getValFromQuery("name") +"</h1>"
    );
    getResponse("/users",(req)->{
             Document doc=dbc.getUser(req.getValFromQuery("name"));
            return "<h1> Hola ! "+ doc.getString("name")
                    +"</h1> <h2> Tu correo es : "+doc.getString("correo")
                    + "</h2> <h2> y vives en : "+doc.getString("dir") +"</h2>";
        }
    );
    getResponse("/potencia", (req) -> {
            int x = Integer.parseInt(req.getValFromQuery("x"));
            int y = Integer.parseInt(req.getValFromQuery("y"));
            return "<h1> El calculo de " + x + "^" + y + " da como resultado: " + Math.pow(x, y);
        }
    );
    getResponse("/", (req) -> {
           return TextFileWriter.getStringFile("examples/index.html");
        }
    );

}
```

Figure 4: MicroSpark Configuracion

Aquí podemos ver la separación y utilización de los servicios de MicroSpark con la respuesta estándar de servicios estáticos.

```java
/**
 * Verifica si el microSpark tiene asignada alguna funcion relacionado con el path
 * de lo contrario tratara de buscar el recurso estatico relacionado.
 * @param req
 * @param out
 */
private void createResponse(Request req, OutputStream out) {
    String finalresource;
    URI theuri = req.getTheuri();
    ResourceWriter rw;
    try {
        if (serviceSpark.existFunction(theuri.getPath())) {
            finalresource=theuri.getPath();
            rw = new StringWriter(serviceSpark.getResponse(finalresource, req));
        } else {
            finalresource = "examples"+theuri.getPath();
            rw = ResourceChooser.choose(finalresource);
        }
        System.out.println("Path final: "+finalresource);
        rw.write(finalresource, out);
    } catch (Exception ex) {
        Logger.getLogger(ServerHttp.class.getName()).log(Level.SEVERE, null, ex);
    }

}
```

Figure 5: ServerHttp Response

```java
/**
 * Añadira una funcion de Respuesta al HashMap
 * @param path la ruta que se quiere colocar el servicio
 * @param gf la funcion de respuesta
 */
public static void getResponse(String path,FunctionResponse gf){
    webservices.put(path, gf);
}


/**
 * Verifica si esiste una funcion relacionada con el path
 * @param key el path del posible servicio
 * @return si existe o no
 */
public boolean existFunction(String key){
    return webservices.containsKey(key);
}


/**
 * Ejecuta la respuesta con el Request relacionado (si se necesita)
 * @param key el path del servicio
 * @param req el request completo
 * @return la ejecucion de la funcion de respuesta
 */
public String getResponse(String key,Request req){
    return webservices.get(key).response(req);
}

/**
 * Interfaz que nos permite declarar funciones de respuesta tipo string y
 * tener a mano un Request para utilizar elementos como el Query
 */
public interface FunctionResponse{
    public String response(Request req);
}
```

Figure 6: Interfaz Funcional en MicroSpark

```java
 * @author J. Eduardo Arias
 */
public class ResourceChooser {
    // Permite a partir de un string devolver un Writer
    public static Map<String,ResourceWriter> selector=new HashMap<String,ResourceWriter>(){{
            put("html",new TextFileWriter("html"));
            put("png",new ImageWriter("png"));
            put("jpg",new ImageWriter("jpg"));
            put("js",new TextFileWriter("javascript"));
            put("css",new TextFileWriter("css"));
            put("err",new ErrorWriter("501 Ese tipo/servicio no encontrado"));
        }
    };

    /**
     * A partir de un string devuelve un Writer
     * @param path el path del recurso web
     * @return Devuelve el ResourceWriter para el tipo de archivo especifico.
     * @throws Exception si el path no esta bien formado o si no es un archivo
     */
    public static ResourceWriter choose(String path) throws Exception{
        String resource="";
        try{
            String[] s=path.split("\\.");
            resource=s[s.length-1].toLowerCase();           // Existen archivos como owl.min.js
        }catch(ArrayIndexOutOfBoundsException e){
            throw new Exception(" No es una peticion de Recurso Especifico/ Peticion mal formada");
        }
        if (!selector.containsKey(resource)){
            return selector.get("err");
        }else{
            return selector.get(resource);
        }

    }
```

Figure 7: Resource Chooser

4 Pruebas

Se probo que respondiera específicamente una imagen, html con css js , funciones con parámetros y finalmente una con base de datos.

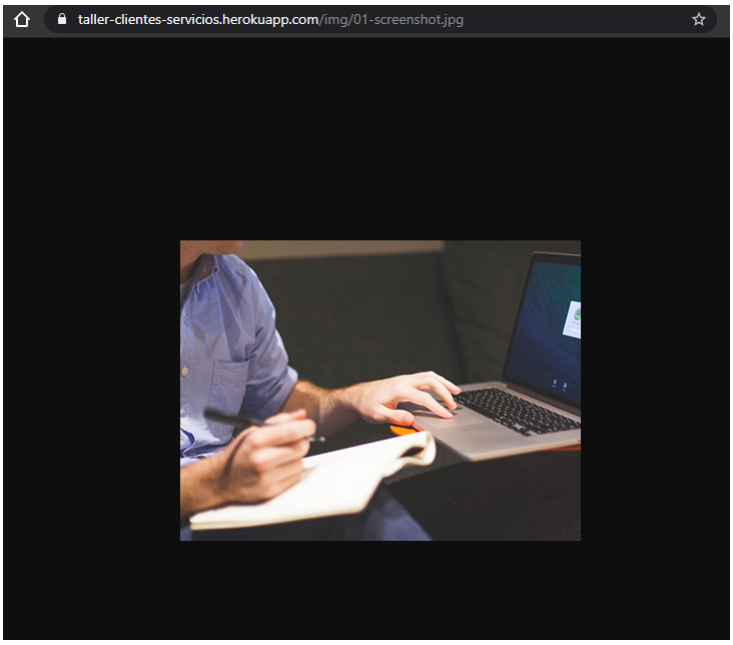

Figure 8: Respuesta correcta de una imagen

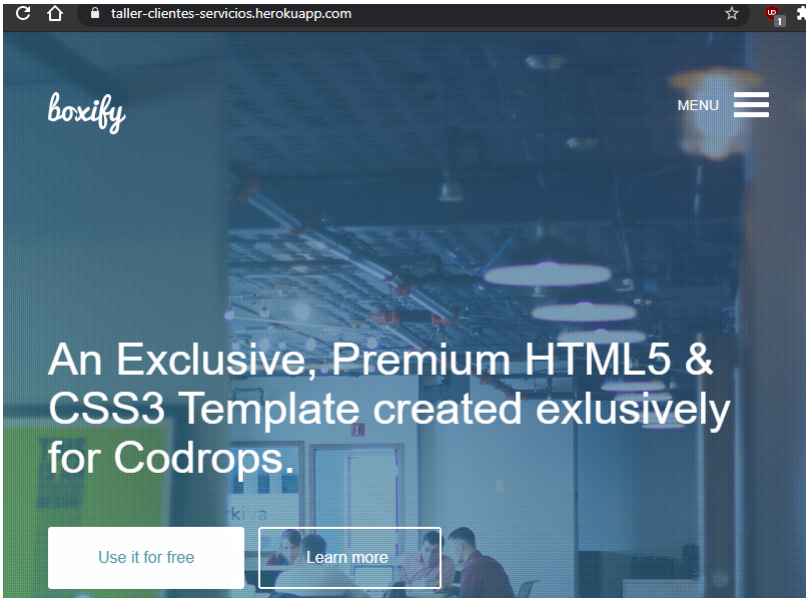

Figure 9: Prueba con html, css, js e imagenes

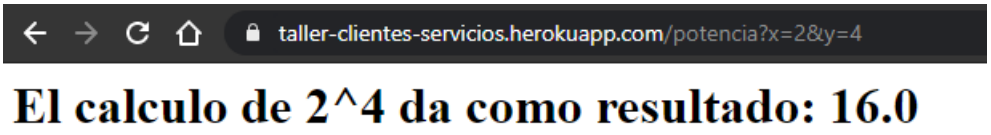

Figure 10: Potencia con dos parametros en query

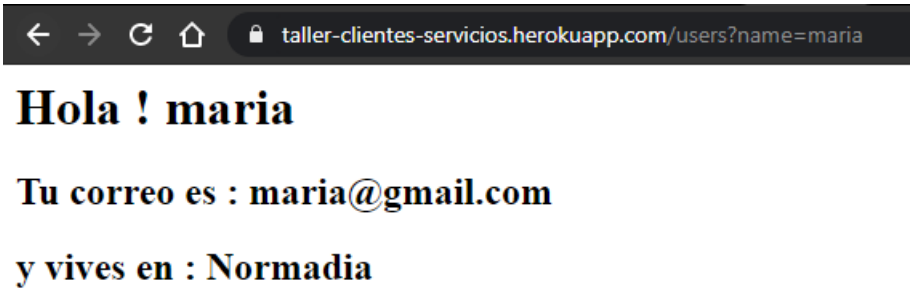

Figure 11: Pruebas con base de datos

Figure 12: Valores de la base datos

5 Conclusiones

Utilizar interfaces genéricas y una correcto diseño de abstracciones permite crear componentes que parecen muy fáciles de configurar y tremendamente poderosos como lo puede ser la función get de Spark. Este programa es un buen ejercicio permite intuir o dar idea de como podría estar implementado algo tan robusto como Spark.

References

[1] Refactoring. *Command*. URL: `https://refactoring.guru/design-patterns/command`. (accessed: 12.08.2020).