ARQUITECTURA EMPRESARIAL
TALLER 3
CLIENTES Y SERVICIOS

Andres Ricardo Martinez Diaz

Septiembre 2020

1. Introducción

La situacion es completar dos retos. El primer reto consiste en escribir un servidor web que soporte múltiples solicitudes seguidas (no concurrentes). El servidor debe retornar todos los archivos solicitados, incluyendo páginas html e imágenes y debe ser hecho sin usar frameworks web como Spark o Spring, solo Java y las librerías para manejo de la red.

El segundo reto consiste en usar el servidor del reto 1 e intentar escribir un framework similar a Spark, que permita publicar servicios web 'get' con funciones lambda. Para probar su funcionalidad se debe crear una base de datos e intentar conectar con el servidor para probar la solución.

Para realizar esta tarea primero se explicara lo más importante del diseño planeado, después se hablara un poco de cómo se evaluaron sus funciones y al final unas conclusiones respecto al resultado final.

2. Diseño

Para construir el servidor que reciba las solicitudes y devuelva los archivos. Primero se debe crear una clase que actúe como servidor web y atienda las solicitudes del cliente.

2.1. HttpServer

Esta clase actuara como servidor, primero tendrá un puerto como atributo así podrá tener un constructor que lo creara con un numero de puerto que reciba como parámetro y un constructor que no reciba nada y cree el servidor con el puerto por defecto. También un atributo booleano para saber si se está ejecutando. Los métodos mas importantes que debe tener son:

Método Start: Este método hará uso de las librerías para el manejo de la red, será el que se mantenga escuchando por solicitudes, y tratando de mantener la conexión con el cliente. Cuando reciba la solicitud se la pasara a otro método.

Método processRequest: Este método se encargará de procesar la solicitud del cliente y de organizar la información de esta para que otro método busque por lo que este necesitando el cliente.

Método generateResponse: Este método recibirá la solicitud ya procesada y organizada. Si lo que pide el cliente es un archivo que existe en la carpeta pública de archivos o datos que solicite del api entonces este método le pedirá a uno de dos métodos, según sea el caso, que obtenga recursos estáticos o que obtenga la información directo del api construido. En el caso de que no encuentre nada con la ruta, generara una respuesta de error 404. Una vez obtenga la respuesta sea cual sea, se la escribirá al cliente.

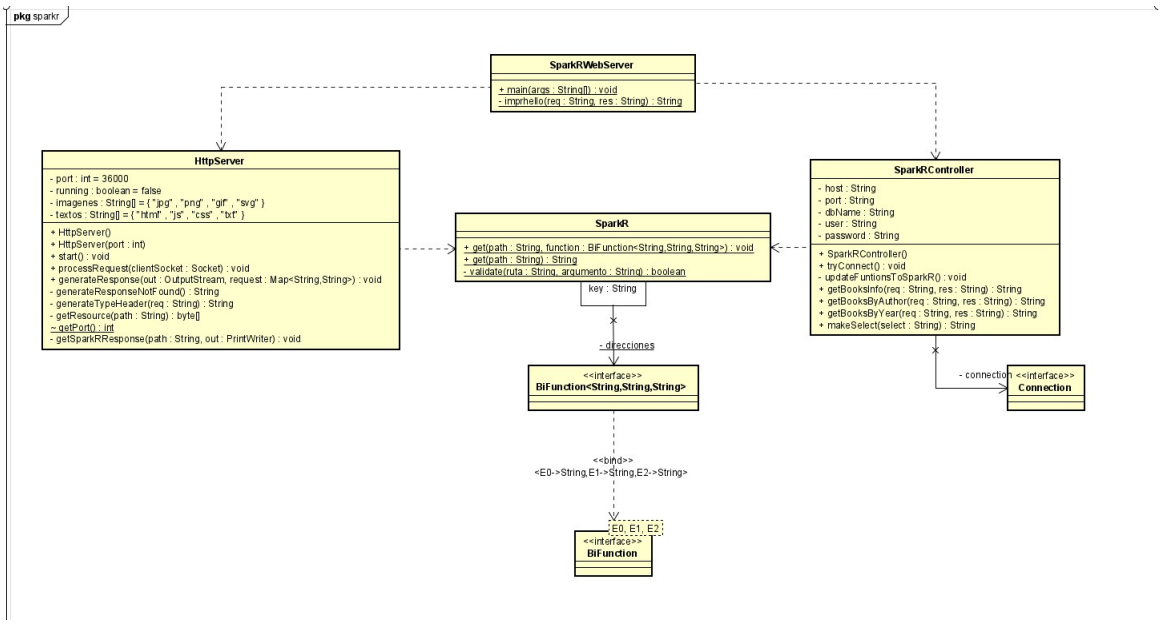
Con esta clase ya quedaria completado el primer reto. para el segundo reto se necesitaran dos clases más, una que funcionará como la api, se llamará SparkR y la otra que se conectara a la base de datos, y creara metodos para incluir en el api, se llamará SparkRController.

2.2. SparkR

Esta clase es quien asociara rutas con métodos, para hacer esto tendrá de atributo un diccionario que asocie rutas "Cadenas" con los métodos "Bifunciones", para que funcione debe tener tres métodos. Metodo get 1: Este método recibirá la cadena y la bifunción después las agregará al diccionario. Metodo get 2: Este método recibirá una cadena, que será la ruta que se busca en el api, si a partir de esta ruta puede obtener una función y un resultado entonces lo devolverá, de lo contrario devolverá un valor nulo. Método validate: Este método validara si la ruta es válida, solo es valida si puede obtener un único resultado, si descubre que no hay funciones que se asocien a la ruta o que hay varias que se podrían obtener de la ruta entonces dirá que no es válida.

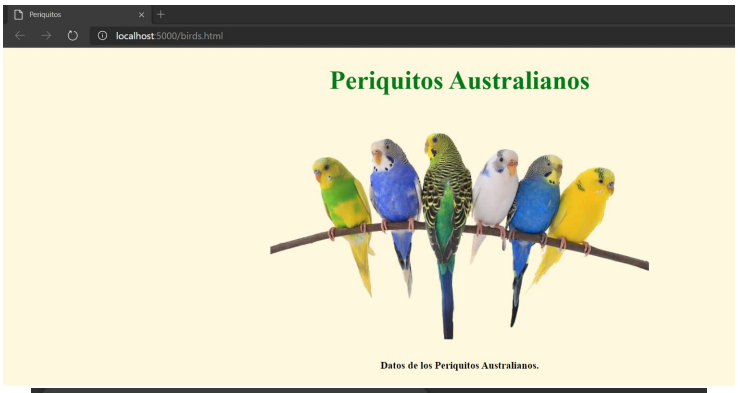
2.3. SparkRController

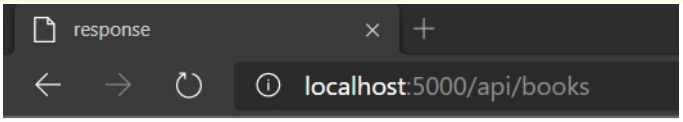
Esta clase tendrá como atributos todo lo necesario para conectarse a la base de datos y la conexión al crearse lo que hará será intentar crear la conexión a la base de datos y luego intentará subir las funciones al api de SparkR, para hacer esto necesita de tres métodos importantes. Método tryConnect: Este método intentara dos cosas, primero comprobar que pueda obtener el controlador de conexión a postgresql, si lo encuentra entonces intentara crear la conexión y guardarla. Método makeSelect: Este método recibirá una consulta en modo de cadena e intentara ejecutar la consulta en la conexión a la base de datos, si la consulta es exitosa entonces procesara la consulta en una cadena que devolverá a la función que lo haya llamado, en el caso de que la consulta no se realice intentara reconectarse y avisara del error de conexión. Método updateFuntionsToSparkR: Este método se encargará de cargar las funciones al api de SparkR, estas funciones serán muy simples, primero tendrán una cadena que almacenara la consulta, luego la enviaran a makeSelect y al final la devolverán a quien las haya pedido.



3. Evaluación

Antes de probar con la conexión a la base de datos se probó con un método simple que debía retornar un hello, después de varias modificaciones se consiguió que este método funcionara escribiendo rutas como “/hello/Person” y que devolviera al cliente la cadena “Hello Person!” o funcionara también escribiendo “/hello” y retornara “Hello” en el navegador, esto fue bastante útil para que el api pudiera funcionar de una manera más dinámica. Cuando esto funciono y ya el servidor devolvía todos los recursos estáticos se paso a intentar traer valores de la base de datos, se creó una tabla básica de libros. Y un método que realizara la consulta de esta tabla e imprimiera en consola los datos organizados, una vez se logró esto en vez de imprimir en consola se almacenaron los datos en una cadena y de devolvía el valor de esa cadena al servidor. Esto resulto bie, pero por estética en el servidor se cambian los saltos de línea en la consulta por un salto de línea html “¡br¿”. Se subió a heroku para probar si aun devolvía los recursos estáticos y los datos desde el api y todas las consultas las devolvió exitosamente.





0, Divina Comedia, Dante Alighieri, 1265
1, Orgullo y prejuicio, Jane Austen, 1813
2, Don Quijote de la Mancha, Miguel de Cervantes, 1605
3, Los hermanos Karamazov, Fiódor Dostoievski, 1880
4, Moby-Dick, Herman Melville, 1851
5, 1984, George Orwell, 1949
6, Hamlet, William Shakespeare, 1603
7, Otelo, William Shakespeare, 1609
8, El rey Lear, William Shakespeare, 1608
9, Rebelión en la granja, George Orwell, 1945

4. Conclusiones

1. Al principio fue un poco desafiante hacer que el servidor devolviera correctamente los archivos, la solución pensada para esto fue intentar leer los archivos, si existían entonces los convertía en un arreglo de bits que más tarde eran escritos usando un OutputStream del cliente, esto funcionaba bien, pero hubo problemas con algunos archivos debido a que los navegadores no podían leer bien los encabezados. Para eso se creó un método que genera encabezados por tipo de archivo y para el objetivo del taller funciono bastante bien la solución.

2. El API resulto sencillo de hacer, cuando se hace una solicitud al api el servidor le pasa la solicitud a SparkR y este como ya tiene guardadas las funciones entonces las ejecuta según lo que se quiera hacer o lo que deba recibir cuando se definió. Como funcionaba desde que se comenzó la implementación no hubo conflictos a medida que crecía el código y se hacía fácil de usar.