MicroSpark

Julián David Gutierrez Vanegas

September 2020

1. Introducción

En el presente documento explicaremos el diseño e implementación de un servidor HTTP, inspirado en Spark, el cual responderá únicamente a solicitudes de tipo "Getçon archivos estáticos (como lo son: imágenes, javascripts, html y css) y dinámicos. Estos últimos, se construyen a partir de información de una base de datos Postgres, ubicada en los servidores de Heroku, la cual se escribe en formato json para luego ser enviada. Vale aclarar que el servidor HTTP también se encuentra desplegado en Heroku.

La descripción de nuestro diseño arrancará por la capa de persistencia, explicando de manera clara dónde y de qué manera están almacenados los datos. Después, se explicará en detalle cómo estos datos son accedidos por nuestro programa, y luego, cómo son manejados y almacenados por este.

A continuación, entraremos en detalle con la capa del controlador. Explicaremos en detalle cómo nuestro servidor HTTP fue construido, mostrando las clases que fueron necesarias para su implementación, los métodos más relevantes de estas clases y qué tipo de solicitudes puede soportar en este momento nuestro servidor.

Posteriormente, daremos un breve repaso por los recursos con los que cuenta nuestro servidor y que pueden ser accedidos actualmente mediante el sitio web de nuestro programa.

Para finalizar, daremos unas conclusiones breves acerca de la experiencia que nos dejó la construcción de este programa.

2. Objetivo

Implementar un pequeño servidor web capaz de responder, con diferentes tipos de recursos e información (ya sea que esté almacenada localmente o en una base de datos externa), a solicitudes de tipo "Get". Se busca implementar este servidor sin utilizar ninguna librería externa a Java, como Spark, que implemente este tipo de servidor automaticamente.

3. Marco Teórico

- Socket: Es un tipo de software que sirve como un punto final para establecer una conexión bidireccional de un punto de red a otro. Normalmente son usados por los servidores para establecer una comunicación con un cliente en aplicaciones de tipo çliente-servidor"
- Http: Es el protocolo en el cual se basan todas las comunicaciones web existentes. Este protocolo nos permite realizar una petición de o enviar recursos a otra máquina a través de la web.
- MyBatis: Es una librería de Java que nos permite implementar capas de persistencia mediante el mapeo de sentencias SQL a objetos. Para realizar este mapeo se pueden utilizar tanto ficheros escritos en XML, como interfaces de Java.

4. Explicación del diseño

A continuación, presentamos cómo el programa "MicroSpark"fue construido.

4.1. Datos

Para comenzar, abordaremos un poco los datos qué utiliza el programa. Al ser nuestro programa nada más que un prototipo, de un pequeño servidor web, se optó por construir una base de datos pequeña, almacenada en Heroku, que apenas nos sirva como prueba de su correcto funcionamiento. Nuestra pequeña base de datos está conformada por una única tabla, llamada Üsuario", en la que se almacenan apenas dos registros de dos usuarios de prueba. Estos registros cuentan con: un nombre, apellido, documento y teléfono, donde la llave primaria es el número de documento.

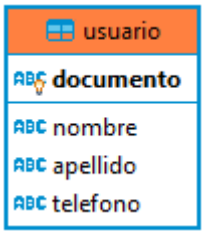

Figura 1: Base de datos de la aplicación

4.2. Acceso a los datos

Para acceder a la información de nuestra base de datos, que se encuentra en los servidores de Heroku, hicimos uso de la librería "MyBatis", junto con una

parte de la librería "Spring". Estas dos librerías nos permiten, no solo la lectura de la información de la base de datos, sino que nos transforma inmediatamente la información en objetos de Java para que podamos manipularla fácilmente. Para implementar estas dos librerías, fue necesario crear las siguientes clases:

- Una clase de configuración que nos permita crear una sesión SQL a nuestra base de datos. Esta clase la llamamos .ªppConfigz se encuentra en nuestro subdirectorio çonfig".
- Una clase "POJO"denominada Üsuario"que nos ayuda a definir cómo esa información de la base de datos se va a trasformar en un objeto.
- Una interfaz, llamada ÜsuarioMapper", en la que definimos las consultas SQL con las cuales accederemos a la información de la tabla.

Estas clases son instanciadas o utilizadas por nuestra clase ÜsuarioServiceMyBatis", la cual es la clase encargada de recuperar la información de nuestra base de datos usando la librería MyBatis y retornarla según sea solicitada, Para asegurarnos que nuestro programa sea flexible, en el sentido del acceso a los datos, hicimos que la clase ÜsuarioServiceMyBatis"no se conecté directamente con las clases inferiores, sino que exista una interfaz de por medio. Esta interfaz la llamamos ÜsuarioServicez es la que define qué información necesita la aplicación de su capa de persistencia para funcionar correctamente. Gracias a esta interfaz, hacemos que las capas superiores estén desacopladas de la capa de persistencia, y que así, se pueda cambiar la manera en que los datos son accedidos sin necesidad de cambiar el resto de la aplicación.

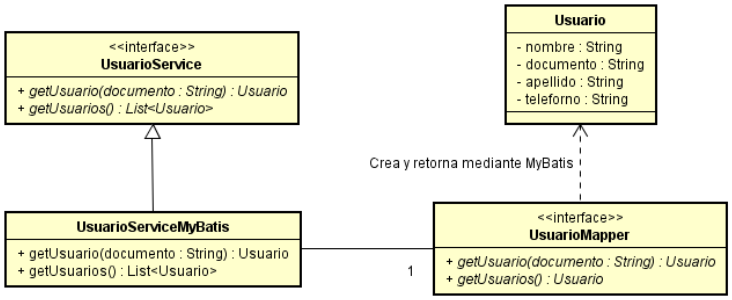

Figura 2: Diagramas de acceso a datos

4.3. Servidor HTTP

Continuamos con la parte central de nuestro programa, ya que esta era el objetivo centrar de todo nuestro desarrollo, que es la explicación de cómo implementamos nuestro servidor web sin utilizar ninguna librería externa a las

que nos proporciona Java por defecto. Para implementar nuestro servidor, fue necesario dividirlo en dos parates: una para que se encargue exclusivamente de acceder y retornar recursos, ya sean de nuestra base de datos o estáticos, y, una segunda parte, que se encargue de la gestión del protocolo HTTP, esto incluye: abrir los sockets, recibir y leer la información de los encabezados HTTP que entren, preparar los propios encabezados HTTP según el tipo de recurso a retornar y enviar las respuestas. Para la primera parte, creamos una clase llamada "SparkJ", este nombre se debe a la librería en que está inspirada, cuya función es acceder tanto a los recursos estáticos, como a los dinámicos. Para esto, se crearon dos métodos, uno que se encargue de retornar los archivos en formato texto plano y otro en bytes. El primer método lo usamos para retornar archivos de tipo: html, javascript, json y css; mientras que el segundo, se utilizó para retornar archivos que necesitan ser enviados en bytes, como lo son las imágenes (y, en caso de que se quiere implementar después, los audios). Y así, nos aseguramos de retornar nuestros archivos en los formatos idóneos para ser leídos por cualquier browser de manera correcta. Adicional a este par de métodos, esta clase también cuenta con otro que se encarga exclusivamente de retornar la página de error (404.html) en caso de que haya un tipo de excpeción a la hora de acceder a un archivo o recurso. Para la segunda parte, creamos una clase llamada "ServidorHTTP". Esta clase es la encargada de: primero, abrir los sockets, tanto de cliente como de servidor; segundo, leer los datos que se ingresan y, con estos, solicitar el recurso a la clase "SparkJ"; y tercero, enviar el recurso solicitado con el encabezado HTTP correspondiente al tipo de recurso.

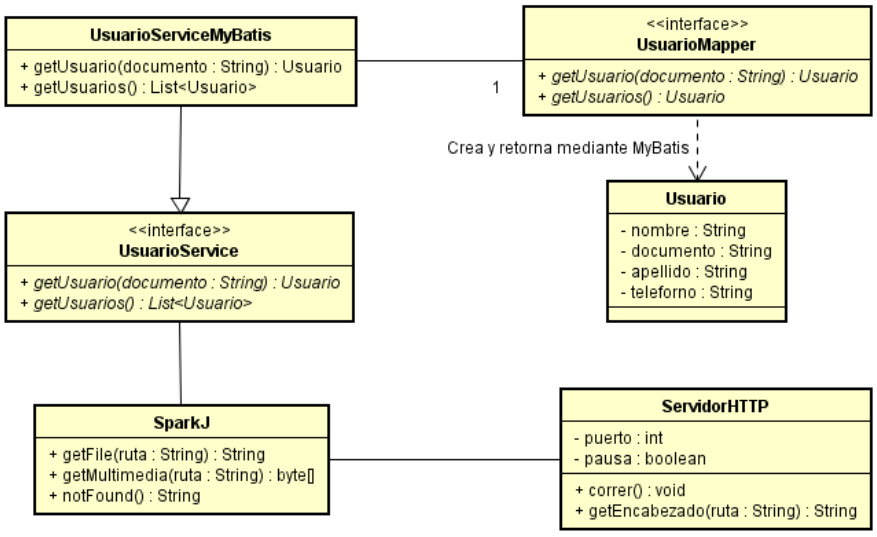

Figura 3: Diagramas de clases completo

4.4. Recursos

Finalmente, procederemos a explicar con qué recursos cuenta nuestro programa y cómo acceder a ellos. Primero, encontramos los recursos estáticos, a los cuales nuestro programa accede directamente, ya que se encuentran almacenados en el servidor. Estos recursos son:

- /Index.html (página principal de nuestro programa)
- /js/api.js
- /mystyle.css
- /usuarios.png
- /radiohead.jpg

Por otra parte, están los recursos dinámicos. Estos son los recursos que la aplicación crea mediante la información que se encuentran en la base de datos. Para accede a ellos directamente, se puede hacer utilizando el path /App/", seguido por la función que se desea utilizar. Ya que lo que diseñamos fue un pequeño prototipo, solo hay dos funciones:

- getUsuarios
- getUsuario?id=¡documento de un usuario¿

Al utilizar cualquiera de estas dos funciones, se podrán acceder a dos archivos tipo "json"que fueron construidos por la aplicación con a la información encontrada en la base de datos en ese momento. Para finalizar, vale resaltar que estas dos funciones pueden ser accedidas más fácilmente utilizando los botones "Buscar todosz "Buscar"que se encuentran en nuestra página web de inicio.

5. Conclusiones

Para poder implementar un servidor web funcional es necesario tener en cuenta varias cosas: es necesario utilizar sockets, ya que son las herramientas que nos facilitan en tráfico de mensajes; es importante conocer bien los encabezados HTTP, porque estos suelen variar entre tipos de archivo a enviar (el encabezado de una imagen, no es el mismo que del de un documento html); los archivos multimedia necesitan obligatoriamente ser enviados en bytes, de otra manera, no podrán ser procesados al otro lado; y, para finalizar, hay que tener en cuenta que el usuario no siempre se va a comportar de la manera correcta en nuestra aplicación, es por esto, que las páginas de error siempre deben estar presentes para evitar que nuestra aplicación colapse.

6. Bibliografía

- SpeedCheck. (2016, 13 febrero). ¿Qué es un socket? Recuperado 3 de septiembre de 2020, de https://www.speedcheck.org/es/wiki/socket/
- MyBatis. (2020, 5 junio). MyBatis – MyBatis 3 — Primeros pasos. Recuperado 3 de septiembre de 2020, de https://mybatis.org/mybatis-3/es/getting-started.html
- Mozilla. (2020, 7 agosto). Generalidades del protocolo HTTP. Recuperado 3 de septiembre de 2020, de https://developer.mozilla.org/es/docs/Web/HTTP/Overview