



## INGENIERÍA DE SISTEMAS

ARQUITECTURAS EMPRESARIALES  
LABORATORIO No.3 (RETO 2)  
SERVIDOR WEB SIN FRAMEWORKS (CONEXIÓN A  
DB)

*Diego Alejandro Puerto Gómez*  
*diego.puerto@mail.escuelaing.edu.co*

Bogotá  
Agosto 2020

## 1 Introducción

Se quiso desarrollar un servidor web el cual no contara con frameworks ya desarrollados, con el propósito de implementar y entender cierta parte de las arquitecturas las cuales estas nos proveen habitualmente, especialmente el funcionamiento e interacción que existe desde que se solicitan recursos por medio del buscador para luego ser retornados al cliente, en esta ocasión, por medio de su visualización en el buscador.

## 2 Resumen

Los frameworks como Spark o Spring son marcos de trabajo que están diseñados para facilitar prácticas que son comúnmente utilizadas en el desarrollo de aplicaciones web, pero en muchas ocasiones se pasa por alto la arquitectura implementada detrás de ella y los procedimientos que conlleva el llamado de cada uno de sus componentes.(1)

En torno al llamado de archivos de un servidor web por medio de la ruta especificada en un buscador, se quiere entender, diseñando la arquitectura web básica, y los componentes que tienen que estar presentes en cada una de las interacciones para que este se pueda comunicar con sus clientes y viceversa.

Se quiso implementar una conexión a una base de datos PostgreSQL por medio de únicamente código Java, de la cual se extraerán datos que serán llamados y mostrados en un archivo html por uno de los recursos solicitados directamente en el path.

## 3 Diseño

Como primer paso del servidor web desarrollado, se crea una variable para almacenar objetos de tipo *ServerSocket*, luego le asigna a esa variable un *Socket* en el puerto detectado en el ambiente, que en este caso es el provisto por Heroku o el 4567 por defecto. Cuando alguien se quiere conectar, la aplicación creará un *ClientSocket* y este quien realmente va a interactuar con el cliente. Al no tener una conexión el programa para hasta recibir una, aceptarla y asignarla a la variable.

Se crea un *Stream* de salida obtenido del *ClientSocket*, crea un *Stream* de entrada obtenido de la misma fuente y va a imprimir en la consola del servidor todos los datos de entrada, esto dentro de un ciclo infinito para que el cliente pueda consumir en múltiples ocasiones los recursos solicitados. Hasta aquí llega la parte en que el código actúa como simulador de buscador, deja de pedir cosas e inicia la parte en que el código (servidor) entrega recursos.

Se identifica como tal el valor solicitado y se pasa como parámetro al método *returnRequest*, se cierran los *Stream* de entrada y salida junto con el *ClientSocket* para iniciar de nuevo su creación. Ya dentro del método *returnRequest*, se identifica si el recurso solicitado es una imagen ya que estas están dentro de otra carpeta para definir la ruta del recurso a buscar. Y por último se generan

dos formatos de cabeceras de salida correspondientes a los que son y no son imágenes.

La implementación de una función lambda para enviar los datos obtenidos desde el path hacia el método que devuelve los recursos al cliente, brinda más modularidad a la solución, queriendo con esto imitar el llamado de funciones *get* que son provistas por frameworks como Spark.

Desde la clase principal se hace el llamado a una función que hará una conexión a una base de datos PostgreSQL la cual se encuentra almacenada en *Heroku*. Para hacer la lectura de los datos en Java, es necesario que se cree una clase representativa de la tabla a ser llamada, la cual debe contener los métodos *getter* que serán usados para poner la información en el archivo html que será mostrado al cliente. (2)

En la figura 2, se muestra el resultado de consultar el recurso llamado *Tabla.html*, escribiendo el nombre de ese recurso en el path del buscador.

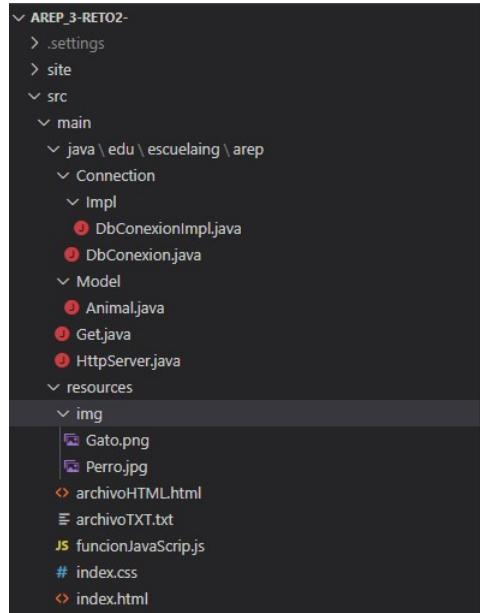


Figure 1: Organización

## 4 Conclusiones

- Los sockets actúan como puntos clave para transmitir la información entre el cliente y el servidor
- Las cabeceras de respuesta de imágenes y demás archivos que responde el servidor web, poseen un formato específico el cual debe ser seguido para lograr una correcta transmisión de la información

The screenshot shows a web browser window with the URL [mighty-dusk-38640.herokuapp.com/Tabla.html](http://mighty-dusk-38640.herokuapp.com/Tabla.html). The title bar includes standard navigation icons and the URL. Below the title bar is a toolbar with 'Aplicaciones' and 'Bookmarks'. The main content area has a heading 'Tabla Animales' followed by a table with the following data:

Id	Animal	Nombre
1	Perro	Kila
2	Loro	Paco
3	Gato	Luis
4	Iguana	Rene
5	Serpiente	Devora
6	Hamster	Pinky

Figure 2: Solicitud de recurso almacenado en la Base de Datos

The screenshot shows a web browser window with the URL [mighty-dusk-38640.herokuapp.com/ValorLambda?FuncionamientoLambda](http://mighty-dusk-38640.herokuapp.com/ValorLambda?FuncionamientoLambda). The title bar includes standard navigation icons and the URL. Below the title bar is a toolbar with 'Aplicaciones' and 'Bookmarks'. The main content area displays the text: 'Bienvenido al programa lambda, el valor que ingreso fue: FuncionamientoLambda'

Figure 3: Solicitud del recurso Lambda construido

- Cada vez que se hace una petición desde el buscador hacia el servidor web, se envía cierta información la cual puede ser interpretada, separada y usada para el funcionamiento interno.
- Las funciones lambda modularizan el comportamiento del programa ya que permiten la declaración de su funcionamiento según la necesidad inmediata.
- Las conexiones a la base de datos se permiten de manera no tan óptima pero funcional, haciendo uso de únicamente código Java, es decir, sin librerías como Spark o Spring.

## References

- [1] ORIX. ¿Qué es un framework y para qué se utiliza?  
<https://www.orix.es/que-es-un-framework-y-para-que-se-utiliza>
- [2] Conectar con PostgreSQL utilizando el driver JDBC – Java PostgreSQL JDBC. Código Xules  
<http://codigoxules.org/conectar-postgresql-utilizando-driver-jdbc-java-postgresql-jdbc/a>