FRAMEWORK JAVA  
(Septiembre 2018)

Brayam Estevan Vargas Pinzón, Arquitecturas Empresariales,

*Este artículo tiene como fin presentar la arquitectura, implementación y pruebas pertenecientes a un framework java creado a partir de un servidor web utilizando sockets y que responde a varias solicitudes concurrentemente.*

*Dicho servidor web se encuentra en la dirección web:* [*https://shrouded-escarpment-88677.herokuapp.com/index*](https://shrouded-escarpment-88677.herokuapp.com/index)

# INTRODUCCIÓN

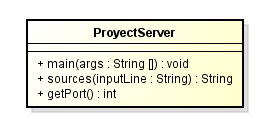
El framework que se describe en este documento es una herramienta que sirve a desarrolladores y usuarios que crean aplicaciones java y desean desplegarlas en internet, el framework se despliega en un servidor web que reponde múltiples solicitudes al mismo tiempo.

En este caso el servidor web escucha por medio de un puerto a que un cliente se conecta, cuando esto ocurre el servidor está en condición de soportar aplicaciones web de java, el framework utiliza el patrón de inversión de control para cargar las clases y los métodos que la aplicación web requiera, este framework provee anotaciones para poder leer y cargar las clases y los métodos requeridos.

# ARQUITECTURA

El servidor web tiene varios componentes, primero existe un browser que se quiere conectar a través de un socket, este socket es el mismo por el que el servidor está escuchando, en el servidor web varios componentes interactúan, el *Buffered Reader*, el método *sources()* y el *Print Writer*.

# implementación

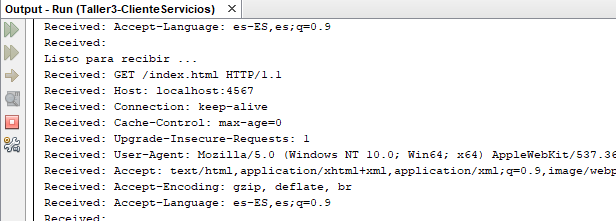


Este servidor web está construido en el lenguaje Java, cuenta con estructura Maven y presenta modularidad para atender los diferentes procesos que tiene.

El servidor cuenta con una función *main(),* la cual se encarga de crear y abrir el socket con el número de socket que obtiene de la función *getPort()*, en el main también se crea un *Print Writer* que es el que se encargará de lanzar los datos de salida que el servidor retorna, también cuenta con un *Buffered Reader* que es quien capta y guarda las solicitudes que está recibiendo el servidor, a partir de este *Buffered Reader* se obtiene un String llamado *inputLine*, este se envía como parámetro al método *sources()*, en este método se evalúa si *inputLine* contiene una petición de un index.html o de un jpg y retorna un string con formato html con texto plano o con una imagen jpg dependiendo de la petición que contiene inputLine, en el método *main()* se imprime por consola cada petición que recibe el servidor y finalmente se cierra el *Print Writer*, el *Buffered Reader* y el *Socket*.

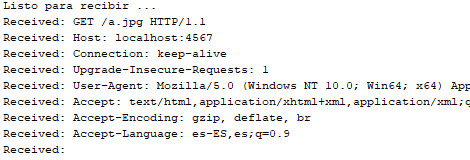
# Pruebas

Al ejecutar el servidor web desde un IDE de java (en este caso NetBeans) el servidor muestra por consola que se encuentra “**listo para recibir**”, el servidor web se prueba primero localmente a través de *localhost:4567,* el número 4567 es el número de puerto que retorna el método *getPort(),* al ir a la URL *localhost:4567/****index.html*** el servidor retorna por consola que recibe una petición GET index.html HTTP/1.1 y el método *source()* reconoce que es una petición de tipo index y retorna un html.

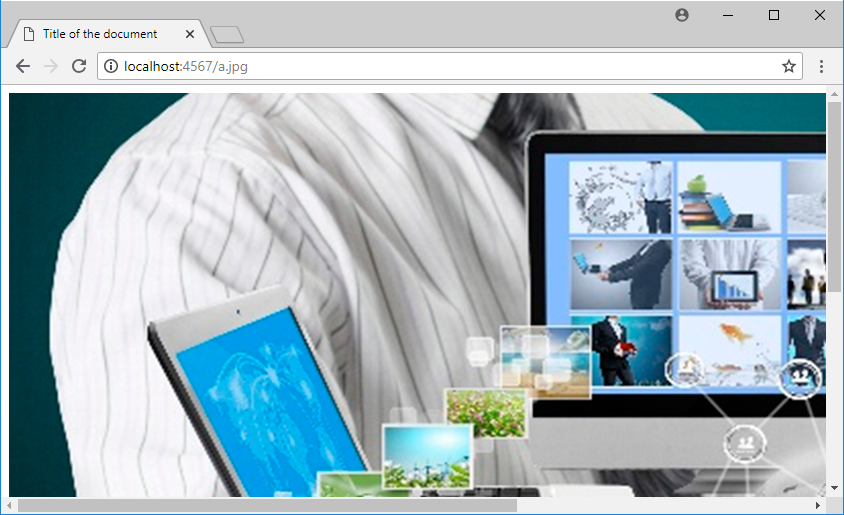


En el browser servidor retorna una página web html, además en esta página está incluida una imagen de tipo jpg.

Al ir a la URL *localhost:4567/****a.jpg*** el servidor retorna por consola que recibe una petición GET a.jpg HTTP/1.1 y el método *source()* reconoce que es una petición de tipo .jpg y retorna una imagen.



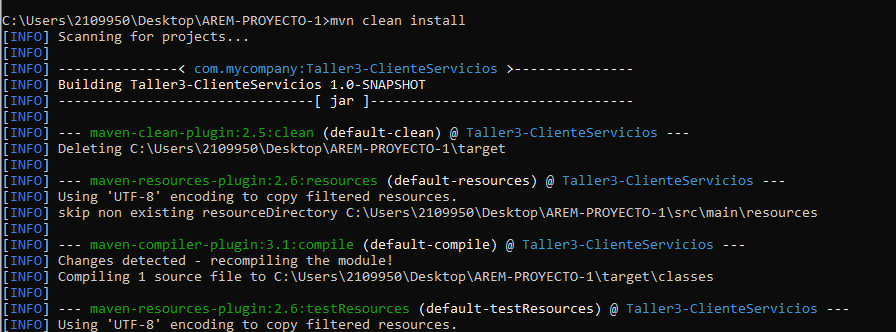
En el browser servidor retorna una imagen.



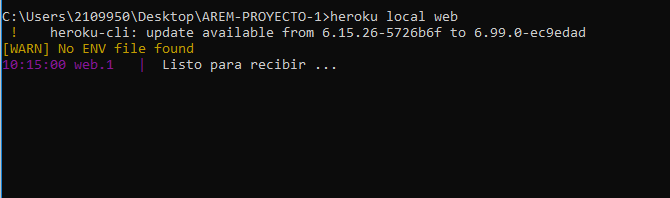
Después de cada petición respondida el servidor queda de nuevo disponible y en consola muestra: “**listo para recibir**”.

También se prueba desplegándolo en heroku localmente con los comandos:

**> maven clean install**

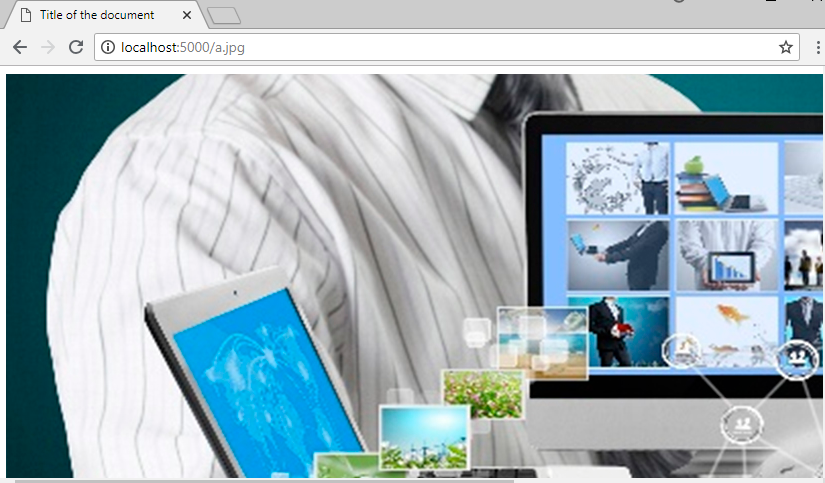


**> heroku local web**



Al desplegarlo en heroku localmente podemos probar por medio del browser en la dirección *localhost:5000/index.html* o *localhost:5000/a.jpg.*





# conclusiones

Un servidor web necesita siempre de un socket para que los “clientes” puedan conectarse y accede a él, además tiene que soportar múltiples peticiones y responderlas una tras otra, se necesita un correcto manejo de las URL para acceder a los recursos que se desean y el browser debe ser capaz de interpretar y mostrar los datos html que el servidor le retorna.