ARTICULO PROYECTO #1 AREM

First A. Author, *Andrés Vásquez, PROYECTO #1*

1. INTRODUCCIÓN

En este documento se hablara sobre cómo fue la construcción de un servidor Web (tipo Apache) en Java. El servidor debe ser capaz de entregar páginas HTML e imágenes tipo PNG. Primero se expondrá como ejecutar el código y se presentará también el diseño general y arquitectura del código y el porqué de la implementación, luego se presentarán los resultados de las pruebas que se le hicieron al código probando todo su funcionamiento y también probando casos en los que el servidor botaría un error en particular y por último se expondrá las conclusiones sobre la realización de este proyecto. También se utilizó una herramienta que permitió montar y tener nuestra aplicación alojada en la web llamada HEROKU y el link del aplicativo es el siguiente: https://proyecto1-arem.herokuapp.com/index.html (podrá ver en funcionamiento la aplicación vía web sin necesidad de descargar el proyecto y correrlo localmente).

2. PASOS PARA LA PREPARACION Y EJECUCION DEL CODIGO

Para compilar el **PROYECTO #1 AREM** se deben seguir los siguientes pasos en orden:

A. *Empezar*

- Visite el repositorio del proyecto del siguiente enlace https://github.com/vashigo/PROYECTO1-AREM copie el link **SSH** del proyecto y usando Linux haga una clonación en su carpeta preferida o copie la siguiente línea de código y péguela en su consola para alojar el proyecto:

```
git clone https://github.com/vashigo/PROYECTO1-AREM.git
```

- Una vez ubicado en la raíz del proyecto, ejecutar el siguiente comando en la terminal:

```
mvn package
```

- Si requiere la documentación del proyecto (JAVADOC), por favor generala con el siguiente comando:

```
mvn javadoc:javadoc
```

B. *Compilando*

- compile el proyecto en terminal desde la carpeta raíz ejecutando la siguiente línea:

```
java -cp target/HttpServer-1.0.0-jar-with-dependencies.jar
co.edu.escuelaing.arem.HttpServer.HttpServer
```

- una vez compilado el programa se ejecuta en el puerto 4567, para probarlo vaya a esta dirección desde su navegador preferido:
  - para ver el HTML:

    ```
    http://localhost:4567/index.html
    ```

  - para ver solo la imagen PNG:

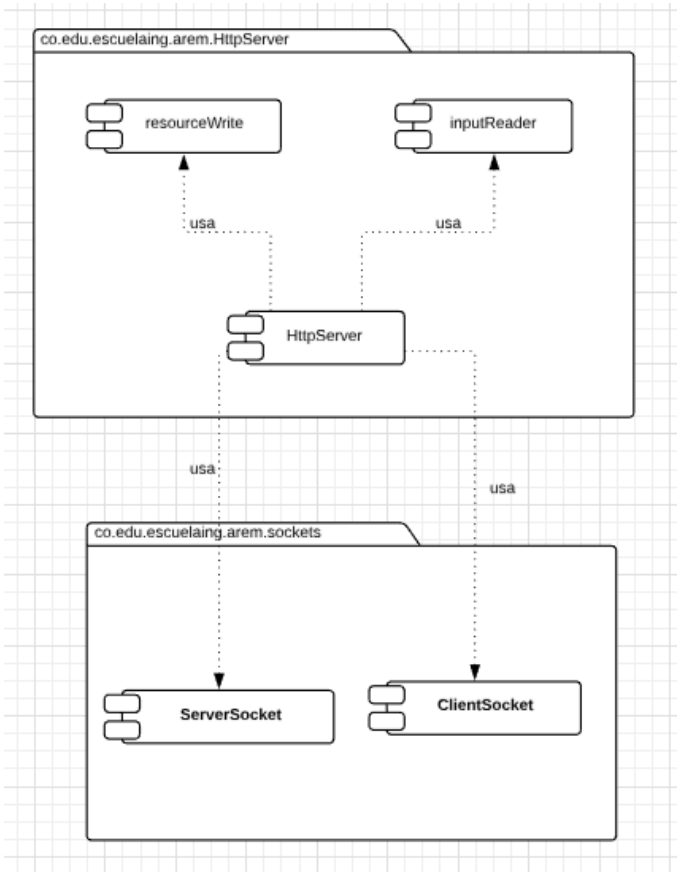
    ```
    http://localhost:4567/imagen.png
    ```

3. CONTEXTO DEL APLICATIVO

Para este proyecto se consideró a solucionar según los requerimientos de nuestro profesor que eran los siguientes:

A. Se deberá construir un servidor Web (tipo Apache) en Java.
B. El servidor debe ser capaz de entregar páginas html e imágenes tipo PNG.
C. Usando el servidor se debe construir un sitio Web de ejemplo y desplegarlo en Heroku.
D. El servidor debe atender múltiples solicitudes no concurrentes.

4. DISEÑO DEL APLICATIVO



[1]

Fig. 1. Modelo representativo tipo MVC (Modelo Vista Controlador). Usando conceptos básicos de modularidad por medio de clientes y servicios. Patron utilizado Patrón 1: Remote procedure call (RPC), Aunque todo se hace localmente, pero se podría decir que los recursos alojados son una mini base de datos alojada también localmente.

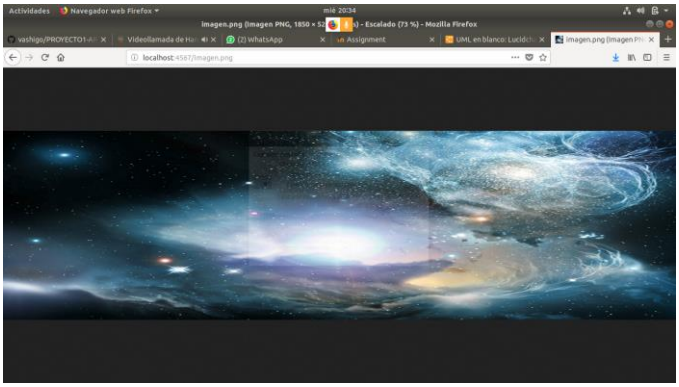
Como se aprecia en el modelo de la figura 1, se cuenta con 5 clases java donde la clase principal (Main) es HttpServer.
HttpServer usa a ServerSocket, ClientSocket, inputReader y ResourceWrite.
InputReader se encargará de procesar la información importante de la solicitud para que ResourceWrite en base a esa respuesta se encargue del envió según la solicitud del InputReader.
ServerSocket se encarga de crear y retornar el socket necesario para la recepción de solicitudes (Servidor) y ClientSocket para en el envió de esas respuestas.

5. PRUEBAS

Se presentarán a continuación 4 pruebas diferentes del aplicativo:

- **Probando solo un .PNG:**

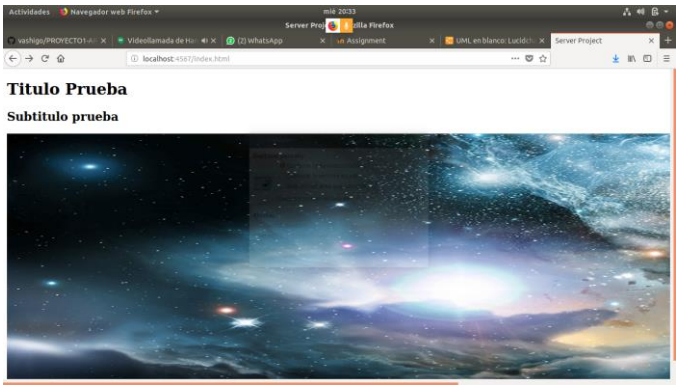
Visualización prueba 1 en HEROKU:
- https://proyecto1-arem.herokuapp.com/imagen.png



[2]Fig. 2.
Aquí simplemente el servidor en funcionamiento localmente se prosigue a abrir un archivo imagen en este caso **IMAGEN.PNG,** se hace la solicitud al navegador ingresando a http://localhost:4567/imagen.png por consiguiente la respuesta que nos manda es el archivo que esta previamente creado y alojado en la carpeta de resources sin ningún problema alguno.

- **Probando solo un .HTML:**

Visualización prueba 2 en HEROKU
- https://proyecto1-arem.herokuapp.com/index.html

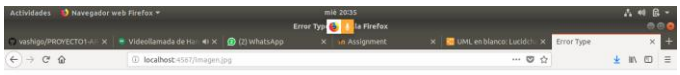


[3]Fig. 3.
Aquí simplemente el servidor en funcionamiento localmente se prosigue a abrir un archivo .HTML en este caso **index.html,** se hace la solicitud al navegador ingresando a http://localhost:4567/index.html por consiguiente la respuesta que nos manda es el archivo que esta previamente creado y alojado en la carpeta de resources sin ningún problema alguno.

[1] Modelo hecho en la web https://www.lucidchart.com/
[2] Pantallazo Prueba 1
[3] Pantallazo Prueba 2

La respuesta es una página muy básica en html que contiene un título y un subtítulo previamente después una imagen que por consiguiente prueba que todo lo proceso bien.

- **Probando con un formato diferente:**

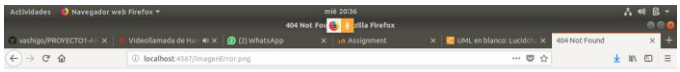


erroType: solo se procesa archivos con .html o .png

[4]Fig. 4.
Aquí simplemente probamos que nuestro diseño funciona correctamente intentando ahora en este caso mandar la petición de que nos abra un archivo de otro formato (no .html o no .jpg), en este caso solicitamos un archivo jpg "imagen.jpg", por ende, nos manda el error que también implementamos "errorType"

- **Probando un formato valido, pero que no existe o su contenido es erróneo:**

Visualización prueba 4 en HEROKU:
- https://proyecto1-arem.herokuapp.com/imagen2.png



Error 404: Contenido del archivo no correto o no encontrado

[5]Fig. 5.
Aquí simplemente probamos que nuestro diseño funciona correctamente intentando ahora en este caso mandar la petición de que nos abra un archivo del formato valido (.html o .jpg), pero ahora el archivo por dentro no está o no funciona como debería, en este caso solicitamos un archivo .png "imagen2.png", por ende, nos manda el error que también implementamos "Error 404".

6. CONCLUSIONES

- Realmente fue muy caótico trabajar el modelo localmente por el patron RPC porque requiere mucho de modularidad.
- Pero la ventaja es que la única forma de interactuar es por medio de mensajes al servidor.
- Aunque esta estrategia permite crear sistemas modulares, tolerantes a fallas y seguros. Así mismo como pide complejidad y dificultad, recibimos a cambio completa seguridad.

REFERENCIAS

Luis Daniel Benavides Navarro, "Integración empresarial" Diapositiva. Ciudad de Publicación, (only Bogotá), Ciudad: Colombia.

Jorge V. (2011). Sockets en Java (cliente y servidor): Codigoprogramacion. Recuperado de http://codigoprogramacion.com/cursos/java/103-sockets-en-java-con-cliente-y-servidor.html#.W5W3YM5KjIU

NA. (2017). Java Socket Server Examples (TCP/IP): codejava. Recuperado de https://www.codejava.net/java-se/networking/java-socket-server-examples-tcp-ip

[4] Pantallazo Prueba 3

[5] Pantallazo Prueba 4