Modularización con virtualización en AWS

Juan Nicolás Nontoa Caballero, Escuela Colombiana de Ingeniería, Servidor Web Concurrente

***Abstract*— The objective of this document is to define the architecture of the laboratory realized of modularization with virtualization of the course enterprise architectures. In this case, a web server was developed that would be able to receive and display web requests or images according to their file types, in this case "html" and "PNG" concurrently. The aim was to create a concurrent server that could interpret the two most basic types of requests that we can execute when we use a web server.**

I. INTRODUCCION

Este documento se toma como referencia ayudando a entender la implementación realizada en el laboratorio, así ayudaría a comprender mejor en el entorno de diseño. Todo esto tomando como base los requisitos planteados al momento de iniciar este proyecto. Se definirá todos sus componentes detalladamente, entre los cuales se pueden observar a grandes rasgos de su arquitectura, diseño, implementación, documentación y variables definidas.

II. DISEÑO

Las capas de diseño se implementaron con base a los requerimientos y/o requisitos y necesidades sobre la marcha de la implementación. Siempre buscando generar aspectos de extensibilidad y estabilidad. Dicho lo anterior en diseño para el diseño que se implementó al laboratorio se implementaron 5 capas que se presentan a continuación.

A. Socket:

Se encarga meramente de crear e inicializar los Sockets, tanto el del Cliente, como el del servidor. Estos dos en sus clases correspondientes. Dichas clases son llamadas desde la clase principal, esta corre el servidor crenado instancias de los sockets de estos.

B. HttpServer:

Esta capa construye la clase principal del proyecto que en este caso es "HttpServer" desde aquí corre el servidor llamando a los sockets y se empiezan a pedir solicitudes del cliente hacia el servidor. También se recorren los POJOS dentro de la capa de framework y extrae los métodos de estos alojándolos en un atributo de tipo HashMap y extrae los parámetros de algún tipo de solicitud si este el caso.

C. Paginas:

Esta capa es la que genera la visualización de todas las solicitudes de tipo HTML, png y los POJOS dentro del servidor con anotaciones de tipo @web. es acá donde tanto se leen, cómo se interpretan, visualizan y publican, los resultados de cualquier solicitud en el servidor.

D. Resourses:

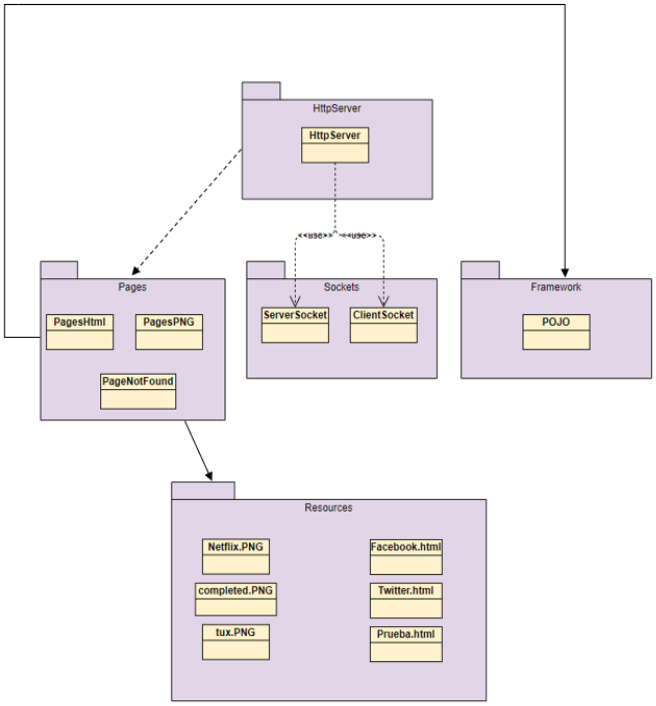
En esta capa del proyecto, podemos hacer una referencia a la abstracción de memoria donde se encuentran alojadas los recursos HTML y PNG con los que cuenta el servidor para responder las múltiples solicitudes no concurrentes por parte de cualquier cliente.

E. FrameWork:

En esta capa del proyecto se encuentran alojados los diferentes Pojos que se utilizaran para la construcción de aplicaciones web si el cliente lo solicita. Este será el único paquete en donde se podrán adicionar tantos Pojos como se quiera. Adicionalmente deberán de almacenar las diferentes anotaciones que se quieran implementar.

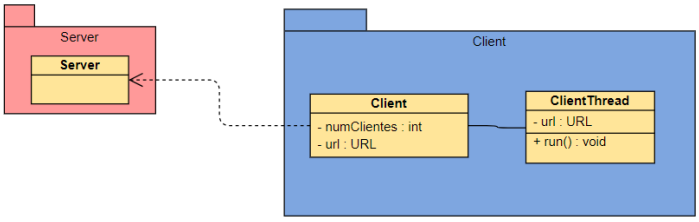
III. ESTRUCTURA DEL SERVIDOR WEB

Este compuesto por los diferentes recursos, los cuales se podrán acceder directamente desde el navegador. De aquí se generan todos los vínculos a las demás dependencias del servidor, Incluyendo tanto las páginas HTML, como las imágenes usadas para visualizarlas. Para acceder a la visualización de los diferentes métodos con notación web (Pojos) se debe de colocar la extensión del método a llamar con su respectivo parámetro, como ejemplo https://proyectoarep.herokuapp.com/pojo/param:Nicolas.



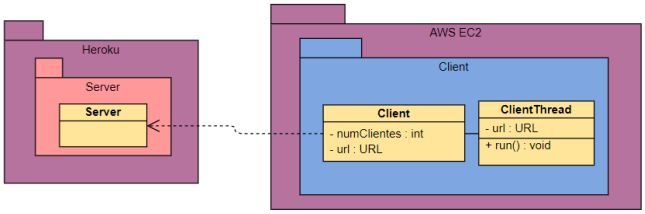
IV. ESTRUCTURA DEL CLIENTE

El cliente está compuesto por dos clases, una de ellas se encarga de administrar la cantidad de solicitudes que se realizarán en paralelo como también de a que URL se harán las solicitudes. La otra clase se encarga de leer los datos de la URL y mostrar su contenido.



V. ESTRUCTURA DEL SERVIDOR

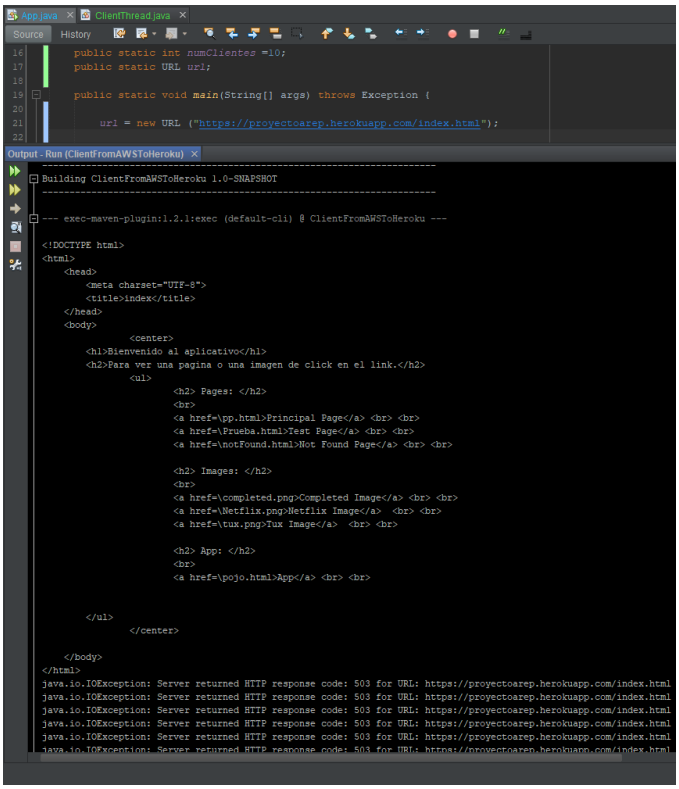
El cliente realizará solicitudes desde una máquina virtual en AWS a un servidor desplegado en Heroku el cual estará en capacidad de recibir múltiples solicitudes.



VI. EXPERIMENTO

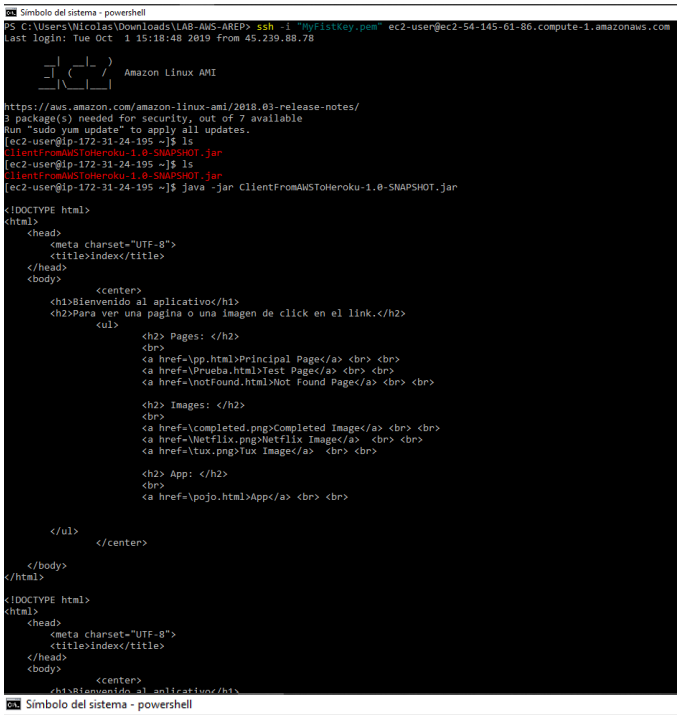
Servidor no concurrente vs Cliente concurrente:

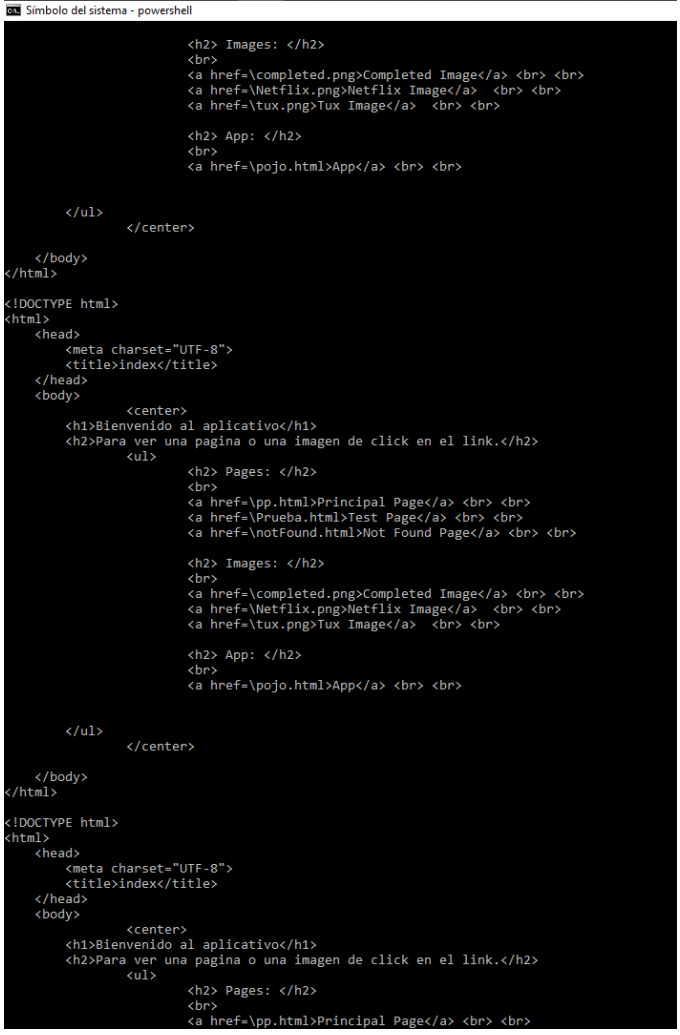
En este caso como el servidor no es concurrente no tiene la capacidad de recibir múltiples solicitudes por lo cual solo responde la primera y las siguientes sin importar la cantidad responde con un error 503.



Servidor concurrente vs Cliente concurrente:

En este caso como el servidor es capaz de recibir múltiples solicitudes responderá a cada una con el recurso solicitado. El cliente esta desplegado en una máquina virtual de AWS (EC2).





VII. CONCLUSION

Gracias al Desarrollo de este este laboratorio se pudo apreciar de un modo más claro la importancia de delegar, diseñar y estructurar de una forma eficiente en las diferentes capas la implementación del servidor, dado que resultaría demasiado complejo la comunicación entre los diferentes sockets en solo una clase y la implementación de las diferentes solicitudes que se hagan. Se pudo observar la gran importancia que tienen los sockets a la hora de implementar y/o desarrollar un servidor web que, aunque si bien se sabe son de bajo nivel, ayudan notablemente en la comunicación Cliente-Servidor.

Juan Nicolás Nontoa Caballero nació en Tunja, Boyacá, Colombia en 1999. Graduado del Colegio San Jerónimo Emiliani en el 2015, inició sus estudios de Ingeniería de Sistemas en 2016 en la Escuela Colombiana de Ingeniería Julio Garavito.