Construcción de un servidor web concurrente en Java (septiembre 2019)

Duran Vivas, Karen Paola

***Resumen*—En el siguiente artículo se evidencia la construcción de un servidor Web (tipo Apache) concurrente en Java. El servidor debe ser capaz de entregar páginas html e imágenes tipo PNG. Igualmente, el servidor debe proveer un framework IoC para la construcción de aplicaciones web a partir de POJOS. Usando el servidor se debe construir una aplicación Web de ejemplo y desplegarlo en Heroku. El servidor debe atender múltiples solicitudes no concurrentes.**

***Palabras Clave*—Servidor web, POJOS**

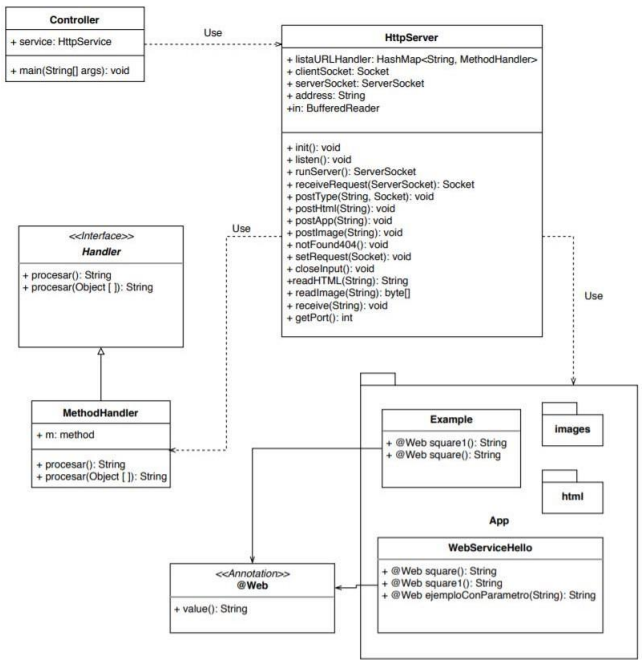
I. INTRODUCCIÓN

Para dar un contexto del proyecto a desarrollar es necesario abarcar algunos de los conceptos que se tratarán a continuación, por ejemplo, el concepto de POJO, POJO son las iniciales de «Plain Old Java Object», que puede interpretarse como «Un objeto Java Plano Antiguo». Un POJO es una instancia de una clase que no extiende ni implementa nada en especial. Para los programadores Java sirve para enfatizar el uso de clases simples y que no dependen de un framework en especial. Este concepto surge en oposición al modelo planteado por los estándares EJB anteriores al 3.0, en los que los Enterprise JavaBeans (EJB) debían implementar interfaces especiales. Otro concepto importante es el de framework, por tanto, cuando se habla de que el proyecto deberá tener un framework se hace referencia a que este deberá proveer un esquema, esqueleto o patrón para la implementación de la aplicación.

El siguiente artículo describe la arquitectura de la implementación de un servidor no concurrente desarrollado en Java y desplegado en Heroku, el cual es capaz de entregar páginas HTML e imágenes en formato PNG, además cuenta con un framework para la construcción de aplicaciones web a partir de POJOS (Plain Old Java Object). La usabilidad de este software está dada por peticiones HTTP/GET, es decir, se le solicitarán recursos ya predefinidos, y en dado caso que este recurso no exista se le notificará al usuario con una página de error. El ciclo de vida de este proyecto está estructurado y manejado en Maven. En la primera sección de este articulo se encuentra el diagrama de clases de la solución propuesta para este proyecto, después su respectivo diagrama de componentes, diagrama de arquitectura y el diagrama de despliegue, luego de esto se encuentra la sección de resultados en donde se prueban las diferentes URLs y se adjuntan las imágenes de los resultados obtenidos.

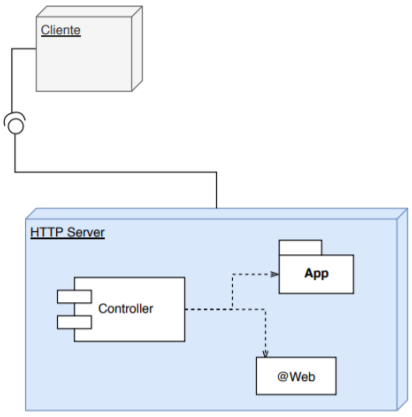
II. DIAGRAMA DE CLASES

En el diagrama de clases podemos observar las relaciones que tienen unas clases con otras, para este ejercicio se creó una clase controlador que es la que invoca los métodos de inicialización y escucha al servidor HTTP, este se ejecuta por el puerto 4567 (localhost). Dentro de la clase del servidor HTTP, se tienen métodos específicos de lectura hacia cada una de las clases .java que existen en el paquete /app, esto se hace con el fin de encontrar cada uno de los métodos de estas clases que contengan la anotación @Web y de esta forma poder cargar todos los POJOS. Para poder acceder a cada uno de los recursos ubicados en la carpeta /app, es necesario conocer el nombre del recurso que se quiere obtener; en caso de que no sea ni .HTML, ni .PNG, será necesario escribir en la URL /app/recurso, en caso de que tenga algún parámetro se escribirá de la siguiente manera .../app/recurso/parámetro.



[1] Diagrama de clases.

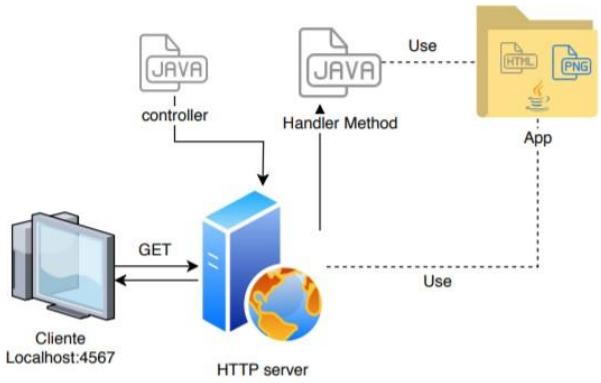
III. DIAGRAMA DE COMPONENTES



[2] Diagrama de componentes.

En el diagrama de componentes se puede apreciar como el cliente interactúa con el servidor a través de la interfaz que este provee por medio de peticiones HTTP.

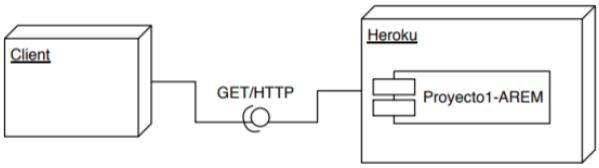
IV. DIAGRAMA DE ARQUITECTURA



[3] Diagrama de arquitectura.

En este diagrama podemos observar varios de los componentes anteriormente mencionados y la interacción entre ellos. Tenemos un servidor HTTP que es quien se encarga de gestionar las peticiones que el cliente le realiza solicitándole recursos ya predefinidos, la carpeta app que es donde se encuentran almacenados todos los recursos para este ejercicio, en esta carpeta las clases .java que se almacenan utilizan la anotación @Web en sus métodos de tal forma que se pueda acceder a ellas mediante /app/recurso, y por último también se tiene un manejador de recursos.

V. DIAGRAMA DE DESPLIEGUE

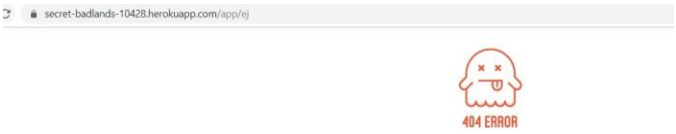


[4] Diagrama de despliegue.

El despliegue se realizó en Heroku cumpliendo con uno de los requisitos de la entrega del proyecto, en este diagrama se tienen sólo dos componentes, uno de ellos es el servidor de Heroku que es en donde se encuentra almacenado el servicio web que se desarrolló y el otro elemento es el cliente que mediante peticiones GET/HTTP solicita recursos.

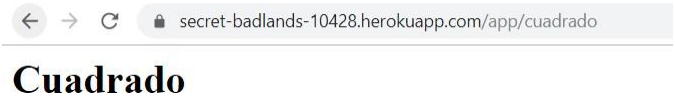
VI. RESULTADOS

Las imágenes que se observan a continuación son ejemplos de los resultados que usted puede obtener utilizando distintas rutas ya predefinidas, ya sea para obtener un archivo con extensión HTML o PNG o simplemente recursos de /app.



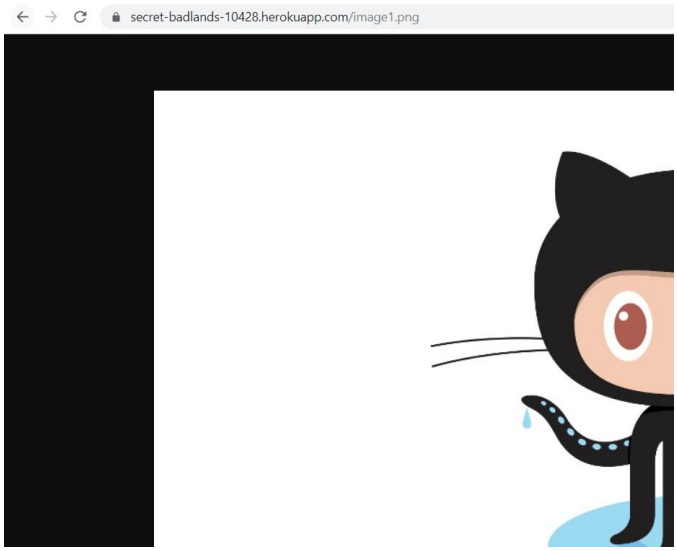
[5] Resultado de solicitar un recurso inexistente.

[6]Resultado del recurso ../app/ejemplo:karen, Karen es un parámetro y por ende puede cambiarse por cualquier otro.



[7] Resultado de solicitar el recurso ../app/cuadrado.

[8]Resultado de solicitar un HTML.



[9] Resultado de solicitar una imagen en formato PNG.

[10]Resultado de la URL framework

[11]Resultado de la URL framework

[12]Resultado de la URL framework

En las anteriores 3 imágenes se observa que se accede a la ruta: …/app/example, la cual proporcionará una interfaz en donde el usuario deberá poner su nombre para obtener algún resultado, ese input lo que hace es redireccionar el dato obtenido a otro componente encargado de escribir el texto final.

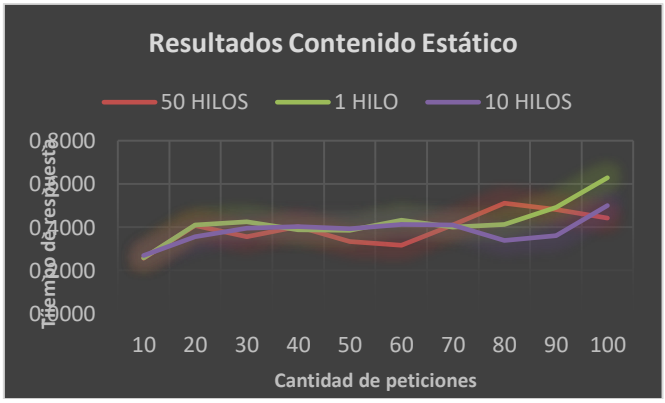
VII. PRUEBAS DE LA CONCURRENCA

El objetivo de las siguientes pruebas es medir el desempeño del servidor web variando el número de peticiones y el número de hilos que soporta solicitando tanto recursos estáticos como dinámicos.

La siguiente gráfica muestra los resultados obtenidos haciendo solicitudes a un contenido estático, variando la cantidad de hilos, primero con 1 hilo, luego 10 hilos, y por último 50 hilos.

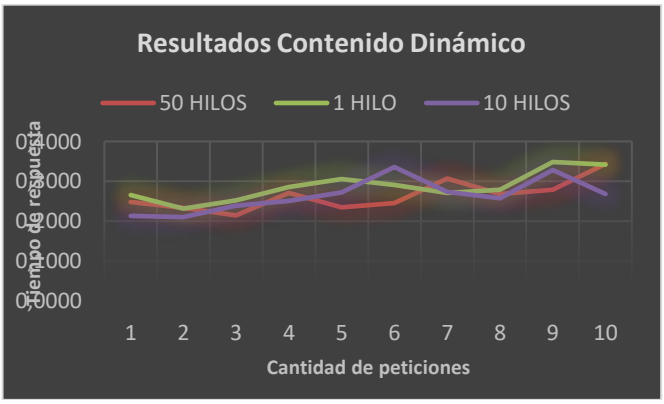
[1] Tabla contenido estático con tiempo en segundos

En una escala más grande, es decir no haciendo peticiones de 1 a 10, sino de 10 en 10, los resultados obtenidos fueron los evidenciados en la siguiente tabla.



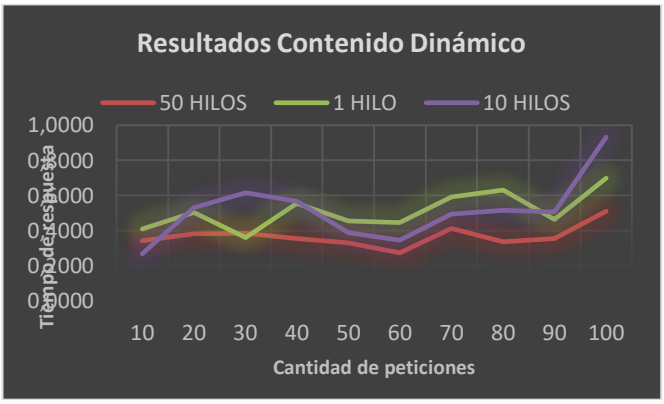
[2] Tabla contenido estático a gran escala con tiempo en segundos

La siguiente gráfica muestra los resultados obtenidos haciendo solicitudes a un contenido dinámico, variando la cantidad de hilos, primero con 1 hilo, luego 10 hilos, y por último 50 hilos.



[3] Tabla contenido dinámico con tiempo en segundos

En una escala más grande, es decir no haciendo peticiones de 1 a 10, sino de 10 en 10, los resultados obtenidos fueron los evidenciados en la siguiente tabla.



[4] Tabla contenido dinámico a gran escala con tiempo en segundos

VIII. CONCLUSIONES

Este proyecto fue útil para aprender a construir un servidor web en Java y además desplegarlo en Heroku, se tuvo un acercamiento hacia los que son los POJOS y se aprendió a usarlos durante la construcción del proyecto. También se utilizaron anotaciones, estas anotaciones las utilizan mucho las bibliotecas como Spring e incluso las propias Java como, por ejemplo: @Override, y son una forma de añadir metadatos al código fuente de Java que están disponibles para la aplicación el tiempo de ejecución o de compilación, muchas veces se usan como alternativa de la tecnología XML, pueden añadirse a elementos del programa como las clases, métodos, metadatos, campos, parámetros, variables locales y paquetes. Las ventajas que tiene el uso de anotaciones son que le permiten al programador declarar en su código cómo debe comportarse el software, son un método muy útil para añadir y obtener información de los elementos que se encuentren anotados, nos permite acceder a estos elementos en tiempo de ejecución por lo que se pueden llegar a usar como variables internas para modificar el comportamiento del flujo del programa, sin embargo, también tiene desventajas como la sobrecarga adicional de memoria.

IX. REFERENCIAS

- ✓ *Contenido temático de la asignatura de arquitecturas empresariales en la Escuela Colombiana de Ingeniería Julio Garavito.*
- ✓ *https://www.adictosaltrabajo.com/2015/08/10/crear-anotaciones-propias-en-java/*
- ✓ *https://jarroba.com/annotations-anotaciones-en-java/*
- ✓ *https://blog.infranetworking.com/servidor-web/*
- ✓ *http://carlospesquera.com/que-es-un-pojo-ejb-y-un-bean/*