**--------------APLICACIONES EN COMUNICACIONES EN RED------------**

**PRÁCTICA 6:**

Servidor HTTP no bloqueantes

**Alumno:**

Meza Vargas Brandon David

**Grupo:**

3CM16

**Profesor:**

Moreno Cervantes Axel Ernesto

**Índice**

[**Introducción** 4](#_Toc103766284)

[**Desarrollo** 5](#_Toc103766285)

[**Server** 5](#_Toc103766286)

[**initServer** 5](#_Toc103766287)

[**HTTPServer** 6](#_Toc103766288)

[**connection** 6](#_Toc103766289)

[**getFile** 9](#_Toc103766290)

[**sendFileHead** 10](#_Toc103766291)

[**sendFile** 11](#_Toc103766292)

[**delete** 13](#_Toc103766293)

[**put** 13](#_Toc103766294)

[**response** 14](#_Toc103766295)

[**Constants** 15](#_Toc103766296)

[**Pruebas de funcionamiento** 15](#_Toc103766297)

[**Probando método GET** 15](#_Toc103766298)

[**Probando método HEAD** 18](#_Toc103766299)

[**Probando método POST** 19](#_Toc103766300)

[**Probando método PUT** 20](#_Toc103766301)

[**Probando método DELETE** 22](#_Toc103766302)

[**Conclusiones** 24](#_Toc103766303)

[**Bibliografía** 25](#_Toc103766304)

**Índice de ilustraciones**

[Ilustración 1. Método initServer() 6](#_Toc103766251)

[Ilustración 2. Recibimiento de línea y método HEAD. 7](#_Toc103766252)

[Ilustración 3. Método POST 7](#_Toc103766253)

[Ilustración 4. Petición PUT 8](#_Toc103766254)

[Ilustración 5. Petición delete. 8](#_Toc103766255)

[Ilustración 6. Petición GET 9](#_Toc103766256)

[Ilustración 7. Método getFile 10](#_Toc103766257)

[Ilustración 8. Método sendFileHead 11](#_Toc103766258)

[Ilustración 9. Método sendFile 12](#_Toc103766259)

[Ilustración 10. Método delete. 13](#_Toc103766260)

[Ilustración 11. Método put. 14](#_Toc103766261)

[Ilustración 12. Método response. 14](#_Toc103766262)

[Ilustración 13. Clase Constants. 15](#_Toc103766263)

[Ilustración 14. Corriendo server 15](#_Toc103766264)

[Ilustración 15. Formulario 16](#_Toc103766265)

[Ilustración 16. Formulario lleno. 16](#_Toc103766266)

[Ilustración 17. Parámetros obtenidos usando GET 16](#_Toc103766267)

[Ilustración 18. Respuesta servidor 17](file:///C:\Users\BrandMV\Documents\ESCOM\Sexto%20Semestre%20ESCOM\Aplicaciones%20para%20comunicaciones%20en%20red\Prácticas\Pràctica%206%20-%20HTTP%20Server%20sockets%20NB\Reporte\Meza_Vargas_Practica6%20-%20Servidor%20HTTPSNB.docx#_Toc103766268)

[Ilustración 19. Pidiendo una imagen 18](#_Toc103766269)

[Ilustración 20. Respuesta del servidor 18](#_Toc103766270)

[Ilustración 21. Petición HEAD desde postman 18](#_Toc103766271)

[Ilustración 22. Respuesta HEAD 19](#_Toc103766272)

[Ilustración 23. Formulario para método post. 19](#_Toc103766273)

[Ilustración 24. Obteniendo parámetros 19](#_Toc103766274)

[Ilustración 25. Respuesta servidor. 20](#_Toc103766275)

[Ilustración 26. PUT con postman 20](#_Toc103766276)

[Ilustración 27. Respuesta servidor. 21](#_Toc103766277)

[Ilustración 28. Respuesta de postman 21](#_Toc103766278)

[Ilustración 29. Actualizando archivo 21](#_Toc103766279)

[Ilustración 30. Respuesta del servidor. 22](#_Toc103766280)

[Ilustración 31. Respuesta de postman 22](#_Toc103766281)

[Ilustración 32. Archivo borrado con postman 23](#_Toc103766282)

[Ilustración 33. Respuesta servidor método DELETE 23](#_Toc103766283)

# **Introducción**

Un servidor web es un software que forma parte del servidor y tiene como misión principal devolver información cuando recibe peticiones por parte de los usuarios. Para que un servidor web funciones correctamente necesitamos un cliente web que realice una petición http a través de un navegador como Chrome, Firefox o Safari y un servidor donde esté almacenada la información, también estas peticiones se pueden realizar usando distintas herramientas, entre ellas Postman, la cual usamos en esta práctica para probar las peticiones, HEAD, PUT y DELETE.

La comunicación entre un servidor y sus clientes se basa en HTTP y la principal función del servidor es mostrar el contenido de un sitio web almacenando, procesando y entregando las páginas web a los usuarios.

En la presente práctica se realizará un servidor HTTP, con el que podamos interactuar con clientes que hagan peticiones HTTP, entre ellas se implementarán los métodos GET, POST, HEAD, DELETE y PUT, todo esto usando sockets no bloqueantes.

Los sockets en modo bloqueante, las llamadas de eventos del sistema se esperan hasta que una respuesta apropiada llegue, los no bloqueantes, a diferencia de estos, continúan con su ejecución incluso si las llamadas al sistema han sido invocadas y trata con ellas de manera apropiada más tarde incluso si las llamadas no han sido completadas y cada llamada se trata de manera separada, por esta razón, usando este tipo de sockets no es necesario el uso de hilos.

# **Desarrollo**

A continuación, se muestra el desarrollo de la práctica con las capturas de todo el código utilizado

# **Server**

Esta es la clase que inicia el servidor usando sockets no bloqueantes, dentro de esta clase tenemos un método llamado initServer donde se inicia la conexión.

## **initServer**

Primeramente, se crea un selector que es un multiplexor para el objeto SelectableChannel, un selector se crea invocando a su método open, la cual usa el selector por defecto del sistema para crear uno nuevo. Posteriormente creamos un canal usando la clase ServerSocketChannel y su método open(), una instancia de esta clase la usamos para establecer una conexión entre el cliente y el servidor como la clase Socket, después configuramos este canal como no bloqueante y hacemos el bind con un puerto establecido y registramos el canal creado con el selector igualmente creado, retornándonos una llave de selección.

Después dentro de un ciclo infinito donde seleccionamos un grupo de llaves correspondientes a los canales listos para operaciones de E/S.

Si la llave es aceptable, haremos la aceptación del cliente y configuramos al cliente como no bloqueante al igual que el servidor. Si la llave puede ser leída es donde lanzamos nuestro servidor HTTP.

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Ilustración 1. Método initServer()

# **HTTPServer**

En el servidor tenemos toda la lógica implementada de los métodos HEAD, GET, POST, DELETE y PUT.

## **connection**

Primeramente, establecemos nuestro ByteBuffer para asignarle memoria, posteriormente leemos lo que nos viene del socket, esto será nuestra petición HTTP, en caso de que sea nula indicamos que nos llegó una petición vacía. Para comenzar a estructurar nuestra respuesta lo token izamos a partir de cada salto de línea, esta línea nos indicará el tipo de petición http que será.

Primeramente, preguntamos si la línea viene con parámetros, en caso de que no podremos descartar que se traté de una petición GET ya que en esta se mandan parámetros visibles separados por un ‘?’. En primer lugar, tenemos al método HEAD, este es un método idéntico a GET pero sin el cuerpo de la petición, por lo que solo mandamos los headers con la información del archivo, si no se indica el archivo se mandará por defecto un archivo index.html.

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Ilustración 2. Recibimiento de línea y método HEAD.

Si la línea nos indica que se trata de un método POST lo que hacemos es encontrar los parámetros que se encuentran al final de la línea, una vez esto mandamos la respuesta con los mensajes apropiados e indicando los parámetros obtenidos, como se trata de una petición POST, estos parámetros no se visualizan en la url del navegador, pero si los mostramos en pantalla para ver que se obtuvieron los parámetros correctos.



Ilustración 3. Método POST

En caso de ser una petición de tipo PUT, mandamos el nombre del archivo a ser modificado y mandamos a llamar al método put explicado más adelante.

Texto

Descripción generada automáticamente

Ilustración 4. Petición PUT

Finalmente, en caso de tratarse de una petición DELETE, se obtiene el nombre del archivo y se manda a llamar el método delete explicado más adelante.

Texto

Descripción generada automáticamente

Ilustración 5. Petición delete.

En caso de que la línea venga con parámetros en el url, sabremos que se trata de una petición GET, aquí lo que hacemos es obtener el nombre del archivo en caso de que se indique, si no se indica se manda el archivo index.htm por defecto, posteriormente obtenemos los parámetros sabiendo que están separados por un ‘?’, a partir de esto preparamos la respuesta que daremos indicando los parámetros obtenidos y los escribiéndolo en el socket obteniendo los bytes de la respuesta.

En caso de recibir una petición no implementada se mandará un mensaje indicando que no se ha implementado.

Texto

Descripción generada automáticamente



Ilustración 6. Petición GET

## **getFile**

Este método solo se encarga de obtener el nombre del archivo solicitado y determinar la extensión de este, la extensión nos servirá para determinar el content-type de las peticiones.

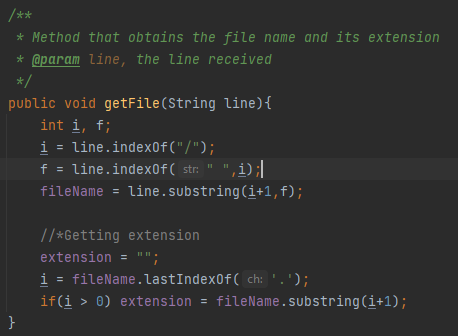


Ilustración 7. Método getFile

## **sendFileHead**

En este método se encarga de mandar el archivo solicitado cuando es una petición HEAD, recibe el nombre del archivo y se crea el cuerpo, de acuerdo con la extensión se determina el Content-Type correcto y se junta con el cuerpo para ser enviado, en este caso el cuerpo son los headers ya que se trata de una petición HEAD.

Texto

Descripción generada automáticamente

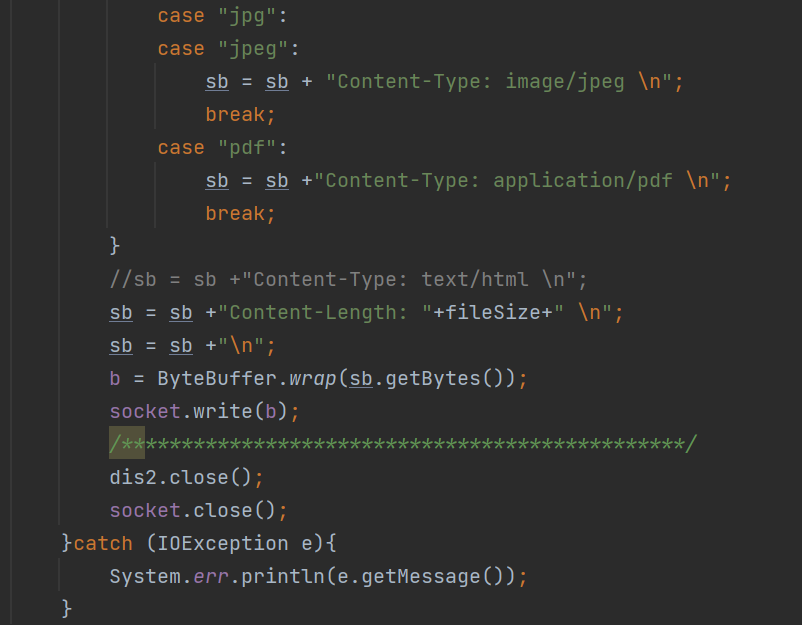


Ilustración 8. Método sendFileHead

## **sendFile**

Este método es lo mismo que lo anterior, pero aquí si mandaremos el archivo solicitado ya que se tratará de una petición GET, este archivo se envía usando un ciclo while donde se van leyendo ls bytes del archivo y se van escribiendo en el flujo de salida.

Texto

Descripción generada automáticamente



Ilustración 9. Método sendFile

## **delete**

Esté método realiza la petición delete, solo se encarga de borrar el archivo indicado en la petición, posteriormente se manda la respuesta dependiendo si fue posible borrar el archivo, si no se pudo borrar o si no se encontró el archivo.

Texto

Descripción generada automáticamente

Ilustración 10. Método delete.

## **put**

Este método se encarga de realizar la petición PUT, si el archivo existe esto quiere decir que se editará el archivo, para esto se elimina el archivo que se editará y se guardará el mismo archivo, pero con los cambios realizados, en caso de que el archivo no exista se creará uno nuevo. Cada operación manda la respuesta correspondiente de creación o edición de archivo.

Texto

Descripción generada automáticamente

Ilustración 11. Método put.

## **response**

Este método se encarga de estructurar la respuesta HTTP, indicando las cabeceras de la respuesta y un cuerpo que contiene un archivo html que mostrará un mensaje dependiendo de la petición que se haga.

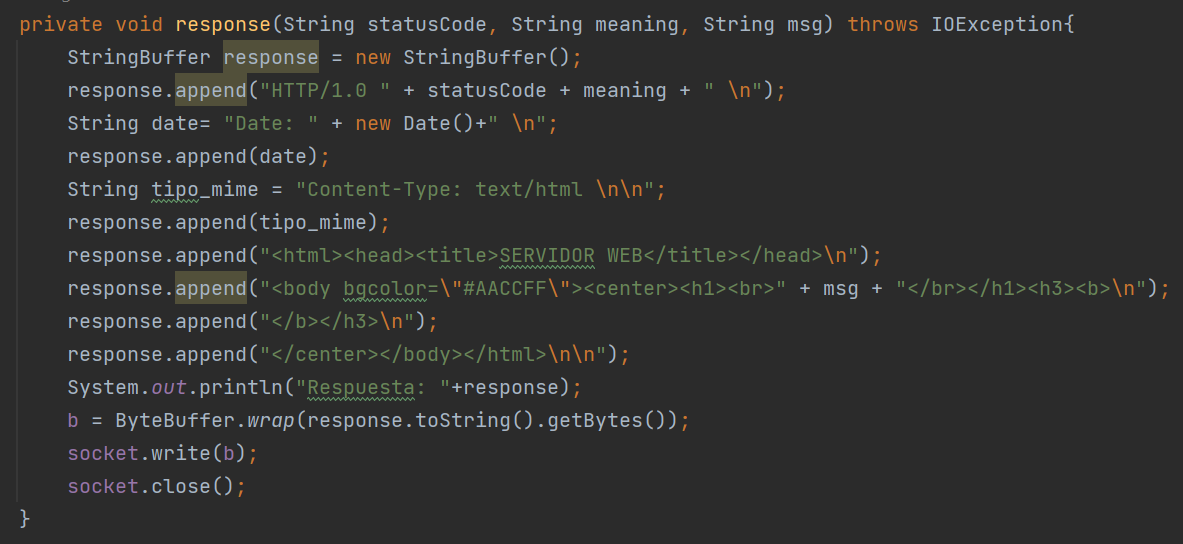


Ilustración 12. Método response.

# **Constants**

En esta clase solo se guardan los mensajes y algunas configuraciones usadas en todo el programa como constantes.

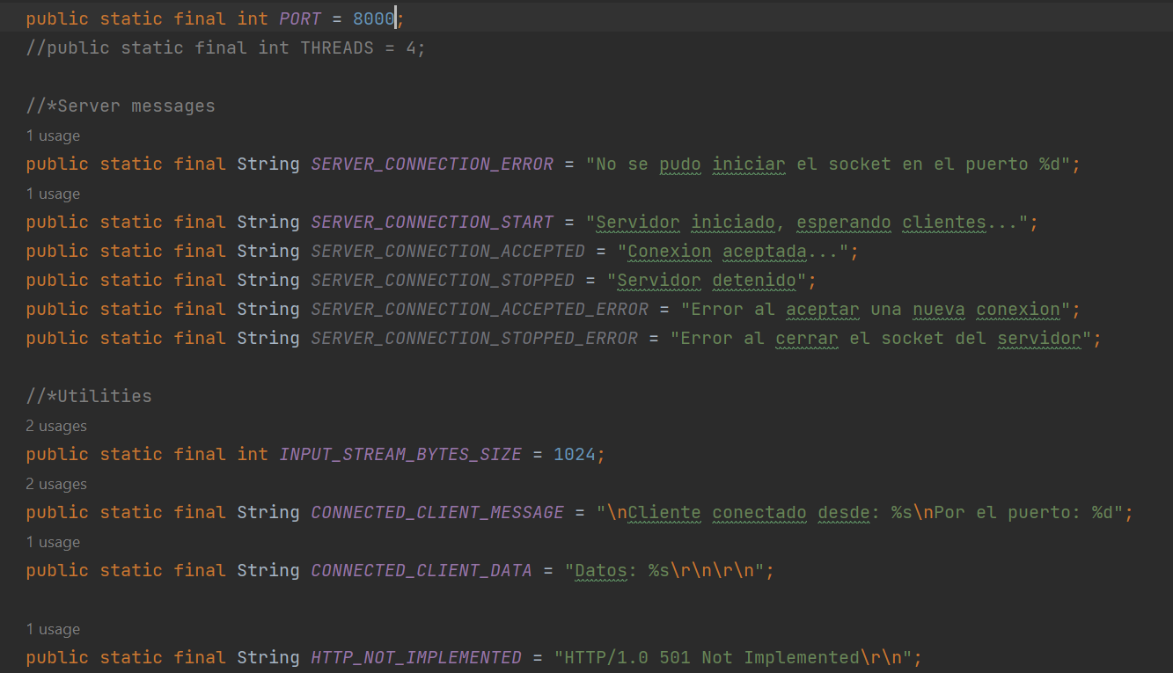
**

Ilustración 13. Clase Constants.

## **Pruebas de funcionamiento**

Primeramente, tenemos que correr el server usando la clase Server y estará a la espera de clientes como se muestra en la siguiente ilustración.

Imagen que contiene medidor, reloj, puesto

Descripción generada automáticamente

Ilustración 14. Corriendo server

## **Probando método GET**

Primeramente, probaremos el método get, para esto tenemos una página con dos formularios, uno corresponde al método get y el otro al método post.

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

Ilustración 15. Formulario

Ahora bien, para probar el método se llenará el formulario y s enviará para recibir la respuesta con los parámetros.

Imagen que contiene Diagrama

Descripción generada automáticamente

Ilustración 16. Formulario lleno.

Una vez lleno y enviado recibimos la respuesta que se ve en la siguiente ilustración.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Ilustración 17. Parámetros obtenidos usando GET

La repuesta que obtenemos en nuestro servidor es la siguiente

Texto

Descripción generada automáticamente

Ilustración 18. Respuesta servidor

De igual forma podemos obtener archivos colocando el nombre en el url.

Pantalla de un video juego

Descripción generada automáticamente con confianza media

Ilustración 19. Pidiendo una imagen

En el servidor tenemos la siguiente respuesta.

Texto

Descripción generada automáticamente

Ilustración 20. Respuesta del servidor

## **Probando método HEAD**

Ahora veremos la petición HEAD, para esto nos ayudaremos de postman como se ve a continuación.

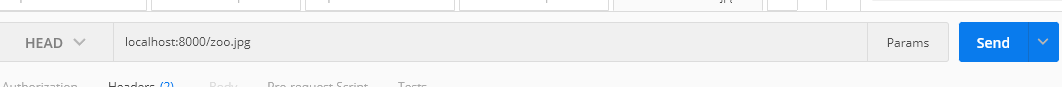


Ilustración 21. Petición HEAD desde postman

En el servidor obtenemos lo siguiente

Texto

Descripción generada automáticamente

Ilustración 22. Respuesta HEAD

## **Probando método POST**

Para este método volvemos a recurrir al formulario, mandando los datos obtenemos la siguiente respuesta.

Gráfico

Descripción generada automáticamente

Ilustración 23. Formulario para método post.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Ilustración 24. Obteniendo parámetros

Texto

Descripción generada automáticamente

Ilustración 25. Respuesta servidor.

## **Probando método PUT**

De igual forma nos ayudaremos de postman para probar este método, vamos a crear un nuevo archivo y después editarlo.

Primero lo crearemos usando postman como se ve a continuación.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Ilustración 26. PUT con postman

Una vez creado postman y el servidor nos muestran las siguientes respuestas.

Texto

Descripción generada automáticamente

Ilustración 27. Respuesta servidor.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Ilustración 28. Respuesta de postman

Ahora actualizaremos ese archivo.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Ilustración 29. Actualizando archivo

Al actualizar el archivo se nos muestran las siguientes respuestas.

Texto

Descripción generada automáticamente

Ilustración 30. Respuesta del servidor.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Ilustración 31. Respuesta de postman

## **Probando método DELETE**

Para este método usaremos de igual forma postman, eliminando el archivo creado con el método put anteriormente como se ve a continuación,

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Ilustración 32. Archivo borrado con postman

La respuesta arrojada por el servidor fue la siguiente.

Texto

Descripción generada automáticamente

*Texto

Descripción generada automáticamente*

Ilustración 33. Respuesta servidor método DELETE

# **Conclusiones**

Con la realización de la práctica pude comprender de mejor manera como se usan los sockets no bloqueantes, podemos darnos cuenta de que obtenemos la misma funcionalidad que en este caso fue la práctica del servidor HTTP, con la particularidad que al estar usando sockets no bloqueantes, no tenemos la necesidad de implementar hilos como en la práctica 4 donde usamos sockets bloqueantes.

# **Bibliografía**

1. Debnath, M, (2018). “What is non-blobking socket programming in Java”. Obtenido de: https://www.developer.com/java/data/what-is-non-blocking-socket-programming-in-java/