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

**PRÁCTICA 4:**

Servidor HTTP

**Alumno:**

Meza Vargas Brandon David

**Grupo:**

3CM16

**Profesor:**

Moreno Cervantes Axel Ernesto

**Índice**

[**Introducción** 4](#_Toc101540535)

[**Desarrollo** 5](#_Toc101540536)

[**Pool** 5](#_Toc101540537)

[**stopped** 6](#_Toc101540538)

[**stop** 6](#_Toc101540539)

[**initServer** 6](#_Toc101540540)

[**Server** 7](#_Toc101540541)

[**getFile** 10](#_Toc101540542)

[**sendFileHead** 11](#_Toc101540543)

[**sendFile** 13](#_Toc101540544)

[**delete** 15](#_Toc101540545)

[**put** 16](#_Toc101540546)

[**response** 16](#_Toc101540547)

[**Constants** 17](#_Toc101540548)

[**Pruebas de funcionamiento** 18](#_Toc101540549)

[**Probando método GET** 18](#_Toc101540550)

[**Probando método HEAD** 21](#_Toc101540551)

[**Probando método POST** 22](#_Toc101540552)

[**Probando método PUT** 23](#_Toc101540553)

[**Probando método DELETE** 25](#_Toc101540554)

[**Conclusiones** 27](#_Toc101540555)

[**Bibliografía** 28](#_Toc101540556)

**Índice de ilustraciones**

[Ilustración 1. Método run del pool de hilos. 5](#_Toc101540499)

[Ilustración 2. Método stopped. 6](#_Toc101540500)

[Ilustración 3. Método stop 6](#_Toc101540501)

[Ilustración 4. Método initServer. 7](#_Toc101540502)

[Ilustración 5. Recibimiento de línea y método HEAD. 8](#_Toc101540503)

[Ilustración 6. Mètodo POST 8](#_Toc101540504)

[Ilustración 7. Peticiòn PUT 9](#_Toc101540505)

[Ilustración 8. Petición delete. 9](#_Toc101540506)

[Ilustración 9. Petición GET 10](#_Toc101540507)

[Ilustración 10. Método getFile 11](#_Toc101540508)

[Ilustración 11. Método sendFileHead 12](#_Toc101540509)

[Ilustración 12. Método sendFile 15](#_Toc101540510)

[Ilustración 13. Método delete. 15](#_Toc101540511)

[Ilustración 14. Método put. 16](#_Toc101540512)

[Ilustración 15. Método response. 17](#_Toc101540513)

[Ilustración 16. Clase Constants. 17](#_Toc101540514)

[Ilustración 17. Corriendo server 18](#_Toc101540515)

[Ilustración 18. Formulario 18](#_Toc101540516)

[Ilustración 19. Formulario lleno. 19](#_Toc101540517)

[Ilustración 20. Parámetros obtenidos usando GET 19](#_Toc101540518)

[Ilustración 21. Respuesta servidor 20](file:///D:\PC\ESCOM\ESCOM\Sexto%20Semestre%20ESCOM\Aplicaciones%20para%20comunicaciones%20en%20red\Prácticas\Práctica%204%20-%20HTTP%20Server\Reporte\Meza_Vargas_Practica4%20-%20Servidor%20HTTP.docx#_Toc101540519)

[Ilustración 22. Pidiendo una imagen 20](#_Toc101540520)

[Ilustración 23. Respuesta del servidor 21](#_Toc101540521)

[Ilustración 24. Petición HEAD desde postman 21](#_Toc101540522)

[Ilustración 25. Respuesta HEAD 21](#_Toc101540523)

[Ilustración 26. Formulario para método post. 22](#_Toc101540524)

[Ilustración 27. Obteniendo parámetros 22](#_Toc101540525)

[Ilustración 28. Respuesta servidor. 23](#_Toc101540526)

[Ilustración 29. PUT con postman 23](#_Toc101540527)

[Ilustración 30. Respuesta servidor. 24](#_Toc101540528)

[Ilustración 31. Respuesta de postman 24](#_Toc101540529)

[Ilustración 32. Actualizando archivo 24](#_Toc101540530)

[Ilustración 33. Respuesta del servidor. 25](#_Toc101540531)

[Ilustración 34. Respuesta de postman 25](#_Toc101540532)

[Ilustración 35. Archivo borrado con postman 26](#_Toc101540533)

[Ilustración 36. Respuesta servidor método DELETE 26](#_Toc101540534)

# **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.

# **Desarrollo**

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

# **Pool**

Dentro de la clase Pool se implementó un pool de hilos que nos servirá de ayuda para tener varios servidores, cada uno conectado a un puerto y con sus propios clientes conectados a él. Primeramente se sincroniza el hilo actual para trabajar con él, para posteriormente inicializar el servidor, mientras el servidor no este detenido se estarán aceptando conexiones en el socket y se ejecutará el hilo encolado en el pool, en este caso cada hilo corresponderá a un servidor HTTP, finalmente se finaliza de manera ordenada.

Texto

Descripción generada automáticamente

Ilustración 1. Método run del pool de hilos.

## **stopped**

En esté método sincronizado lo único que se hace es retornar el valor de la bandera que indica si el servidor está detenido o no.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente con confianza media

Ilustración 2. Método stopped.

## **stop**

Este método de la clase Pool se encarga de cambiar el valor de la bandera a verdadero, esto indica que el servidor será detenido y se cerrará la conexión, este método solo lo usaremos en caso de que queramos detener el servidor, en el caso de la práctica no lo usamos pero se incluye por si se quisiera utilizar en el futuro.

Texto

Descripción generada automáticamente

Ilustración 3. Método stop

## **initServer**

Esté método inicializa el servidor creando un nuevo serverSocket conectándose al puerto que se le indique.

Texto

Descripción generada automáticamente

Ilustración 4. Método initServer.

# **Server**

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

Primeramente establecemos nuestros flujos de entrada y salida necesarios para leer y mandar las respuestas HTTP, 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

Ilustración 5. 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.

Texto

Descripción generada automáticamente

Ilustración 6. Mètodo 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 7. Peticiòn 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 8. 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 mandamos a través de un flujo de salida.

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

Texto

Descripción generada automáticamente

Ilustración 9. 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.

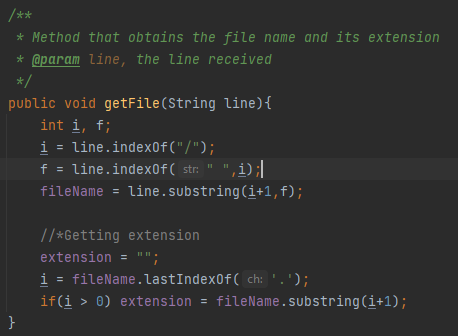


Ilustración 10. Método getFile

## **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 11. 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

Texto

Descripción generada automáticamente

Ilustración 12. 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 13. 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 14. 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.

Texto

Descripción generada automáticamente

Ilustración 15. Método response.

# **Constants**

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

*Texto

Descripción generada automáticamente*

Ilustración 16. Clase Constants.

## **Pruebas de funcionamiento**

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

Imagen que contiene medidor, reloj, puesto

Descripción generada automáticamente

Ilustración 17. 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 18. 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 19. 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 20. Parámetros obtenidos usando GET

La repuesta que obtenemos en nuestro servidor es la siguiente:

Texto

Descripción generada automáticamente

Ilustración 21. 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 . Pidiendo una imagen

En el servidor tenemos la siguiente respuesta.

Texto

Descripción generada automáticamente

Ilustración . Respuesta del servidor

## **Probando método HEAD**

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

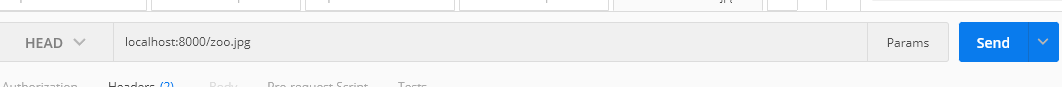


Ilustración . Petición HEAD desde postman

En el servidor obtenemos lo siguiente

Texto

Descripción generada automáticamente

Ilustración . 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 . Formulario para método post.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Ilustración . Obteniendo parámetros

Texto

Descripción generada automáticamente

Ilustración . 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 . PUT con postman

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

Texto

Descripción generada automáticamente

Ilustración . Respuesta servidor.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Ilustración . Respuesta de postman

Ahora actualizaremos ese archivo.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Ilustración . Actualizando archivo

Al actualizar el archivo se nos muestran las siguientes respuestas.

Texto

Descripción generada automáticamente

Ilustración . Respuesta del servidor.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Ilustración . 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 . Archivo borrado con postman

La respuesta arrojada por el servidor fue la siguiente.

Texto

Descripción generada automáticamente

*Texto

Descripción generada automáticamente*

Ilustración . Respuesta servidor método DELETE

# **Conclusiones**

Gracias a esta práctica pude entender de mejor forma cómo funciona un servidor HTTP real, pues se realizó uno implementando los métodos más importantes que fueron HEAD, GET, DELETE, PUT y POST. Además en esta práctica se implementó lo visto en clase sobre el pool de hilos para tener múltiples servidores en diferentes puertos y que estos tengan sus propios clientes haciendo a la aplicación una aplicación escalable.

En conclusión fue una buena práctica para comprender como funciona un servidor HTTP y la importancia del pool de hilos.

# **Bibliografía**

1. MDN, (2011). “Que es un servidor WEB?”. Obtenido de: <https://developer.mozilla.org/es/docs/Learn/Common_questions/What_is_a_web_server>