Instituto Politécnico Nacional

Escuela Superior de Cómputo

Aplicaciones para Comunicaciones en Red

Práctica 4:
Servidor HTTP

Alumnos:

Malagón Baeza Alan Adrian

Martínez Chávez Jorge Alexis

Profesor:
Moreno Cervantes Axel Ernesto

Grupo: 6CM1

1. Introducción

Un servidor web es un software que forma parte del servidor y tiene como misión principal devolver información (páginas) cuando recibe peticiones por parte de los usuarios.

En otras palabras, es el software que permite que los usuarios que quieren ver una página web en su navegador puedan hacerlo. Para el funcionamiento correcto de un servidor web necesitamos un cliente web que realice una petición http o https a través de un navegador como Chrome, Firefox o Safari y un servidor donde esté almacenada la información.

Los métodos HTTP, son el formato de comunicación entre el cliente y el servidor web. Maneja varios formatos: POST, GET, PUT, DELETE, OPTIONS, HEAD, PATCH, etc.

Método GET

Es uno de los métodos HTTP, este realiza una petición a un recurso específico. No permite el envió de datos a excepción si dichos datos se envían como parámetro en la URL que realiza la petición. Esta petición retorna tanto la cabecera como el contenido. Ahora, este método GET puede retornar una respuesta en formato HTML, JSON, XML, Imágenes o JavaScript. Semánticamente se utiliza para consultar información como una SELECT a la base de datos, se puede filtrar datos empleando los datos enviados por la URL

Método POST

Es otro de los métodos HTTP, este puede enviar datos al servidor por medio del cuerpo (body) y nada por la URL como se emplea en el método GET. El tipo de cuerpo de solicitud se define en la cabecera Content-Type. Semánticamente se utiliza
4
para registrar información, similar al INSERT de datos a nivel de base de datos. A pesar de eso se puede forzar este método de petición HTTP para otras acciones como actualización, eliminación de registro, como carga de archivos, etc. También, se emplea para acciones que no tienen relación con el registro de información se debe considerar que el método POST no es idempotente, es decir cada petición realiza un cambio diferente en el recurso del servidor web.

Método PUT

Es similar al método de petición POST, solo que el método PUT es idempotente; es decir puede ser ejecutado varias veces y tiene el mismo efecto, caso contrario a un POST que cada vez que se ejecuta realiza la agregación de un nuevo objeto, ya que semánticamente es como una inserción de un nuevo registro. Semánticamente el método HTTP PUT se utiliza para la actualización de información existente, es semejante a un UPDATE de datos a nivel de base de datos. Los requests de un PUT usualmente se envían los datos por formularios, formato JSON entre otros. Si se compara con las sentencias SQL es similar a un UPDATE.

Método HEAD

Este método de petición es similar al método HTTP GET, sin embargo no retorna ningún contenido HTTP Response. Cuando se trabaja con este método de petición no se está interesado en el contenido, solo en el código de HTTP de Respuesta y el encabezado (Headers). Este método se puede emplear en casos particulares. Por ejemplo, se tiene un sitio web con varios enlaces web y se requiere verificar si hay “enlaces rotos”; entonces se realiza la petición a todos los enlaces correspondientes y verifica el estado de dichos enlaces.

Método DELETE

Este método de petición permite eliminar un recurso específico. También es idempotente; es decir puede ser ejecutado varias veces y tiene el mismo efecto similar al PUT y GET. Semánticamente se utiliza para eliminar información existente, es semejante a un DELETE de datos a nivel de base de datos.

2. Desarrollo

Para el desarrollo de esta práctica se realizaron 2 clases: la clase Manejador y la clase WebServer, a continuación se explicará a grandes rasgos el funcionamiento de estas clases.

2.1 Clase WebServer

Esta clase permite al usuario establecer el puerto al que se conectará y el tamaño del pool de conexiones, permitiendo iniciar el servidor en el localhost. Después de iniciar la conexión, el servidor espera en un ciclo forever al cliente, luego de aceptar la conexión se crea el manejador para posteriormente ejecutarlo.

```
ExecutorService pool = Executors.newFixedThreadPool(tamPool);
System.out.println("\n\n..:: Iniciando Servidor ::.. \nPool de Conexiones = "
+ tamPool);

ServerSocket s = new ServerSocket(pto);
System.out.println("Servidor iniciado: http://localhost:" + pto + "/ --- OK");
System.out.println("Esperando cliente....");

for( ; ; ) {
   Socket cl = s.accept();
   Manejador manejador = new Manejador(cl);
   pool.execute(manejador);
}ExecutorService pool = Executors.newFixedThreadPool(tamPool);
System.out.println("\n\n..:: Iniciando Servidor ::.. \nPool de Conexiones = "
+ tamPool);

ServerSocket s = new ServerSocket(pto);
System.out.println("Servidor iniciado: http://localhost:" + pto + "/ --- OK");
System.out.println("Esperando cliente....");

for( ; ; ) {
   Socket cl = s.accept();
   Manejador manejador = new Manejador(cl);
   pool.execute(manejador);
}
```

En la siguiente captura se puede observar el puerto y tamaño del pool de conexiones que el cliente desea

```
run:
Indique el puerto: 8000
Indique el tamaño del pool de conexiones: 5


..:: Iniciando Servidor ::..
Pool de Conexiones = 5
Servidor iniciado: http://localhost:8000/ --- OK
Esperando cliente....
```

2.2 Clase Manejador

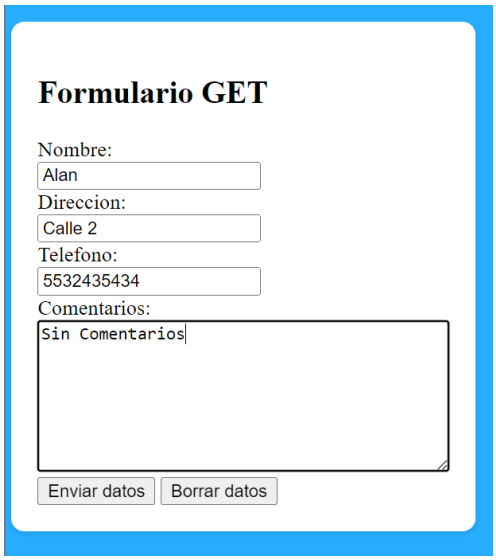
Esta clase contiene los métodos HTTP: GET, POST, PUT, HEAD y DELETE. Además contiene los MIME types para especificar el tipo de contenido de un recurso.

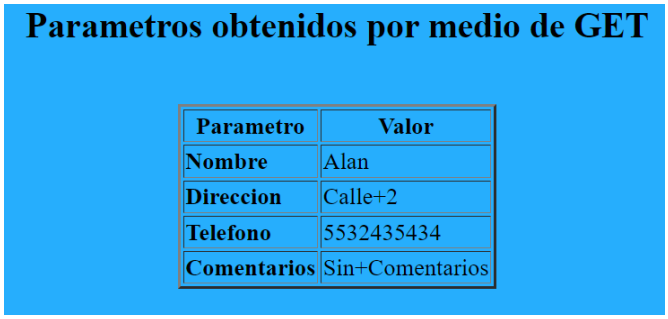
MÉTODO GET

Cuando el cliente se conecta al localhost, se le desplegará el index.html. El cual contiene una imagen y los formularios para realizar el GET y el POST . En las siguientes imágenes se puede observar las peticiones que se le hacen al servidor, y las respuestas que se envían al cliente, utilizando el método GET.

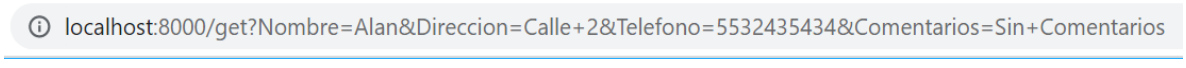
```
..:: Cliente Conectado desde /0:0:0:0:0:0:0:1 ::..
Desde el puerto: 54937
Datos: GET / HTTP/1.1


Respuesta GET:
HTTP/1.1 200 OK
Date: Mon May 22 00:44:06 CST 2023
Server: NetworkServer Server/1.0
Content-Type: text/html
Content-Length: 1464
```

El usuario puede rellenar el formulario del GET y enviar sus datos al servidor. Luego se le mostraran al cliente los parámetros obtenidos de su petición, como se muestra a continuación.





Además como se puede observar el envió de datos se envía como parámetro en la URL que realiza la petición.



Se realizaron otras pruebas con el método GET para obtener recursos con diferentes extensiones. Recursos de imágenes, pdf y archivos mp3 por ejemplo.

Para que el recurso se mostrará correctamente se recurrió a los MIME types, los cuales dependiendo de la extensión del recurso se le asigna un content-type.

GET http://localhost:8000/recursos/image.jpg Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Body Cookies Headers (4) Test Results Status: 200 OK Time: 84 ms Size: 2.27 MB Save as Example

GET http://localhost:8000/recursos/micro.pdf Send

Params Auth Headers (6) Body Pre-req. Tests Settings Cookies

Query Params

Key	Value	Description	Bulk Edit

Body 200 OK 542 ms 13.61 MB Save as Example

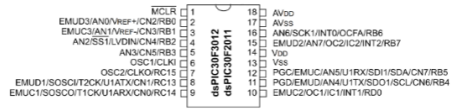
dsPIC30F2011/2012/3012/3013

dsPIC30F2011/2012/3012/3013 Sensor Family

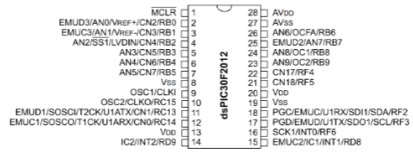
Device	Pins	Program Memory		SRAM Bytes	EEPROM Bytes	Timer 16-bit	Input Cap	Output Comp/Std PWM	A/D 12-bit 200 Ksps	UART	SPI	I²C™
		Bytes	Instructions									
dsPIC30F2011	18	12K	4K	1024	–	3	2	2	8 ch	1	1	1
dsPIC30F3012	18	24K	8K	2048	1024	3	2	2	8 ch	1	1	1
dsPIC30F2012	28	12K	4K	1024	–	3	2	2	10 ch	1	1	1
dsPIC30F3013	28	24K	8K	2048	1024	3	2	2	10 ch	2	1	1

Pin Diagrams

18-Pin PDIP and SOIC



28-Pin PDIP and SOIC



28-Pin SPDIP and SOIC

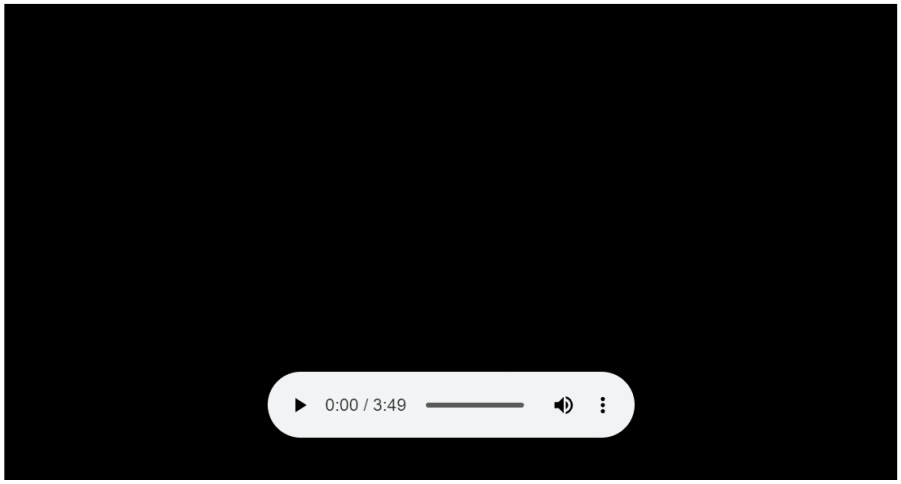
GET http://localhost:8000/recursos/fly.mp3 Send

Params Auth Headers (6) Body Pre-req. Tests Settings Cookies

Query Params

Key	Value	Description	Bulk Edit

Body 200 OK 298 ms 8.79 MB Save as Example

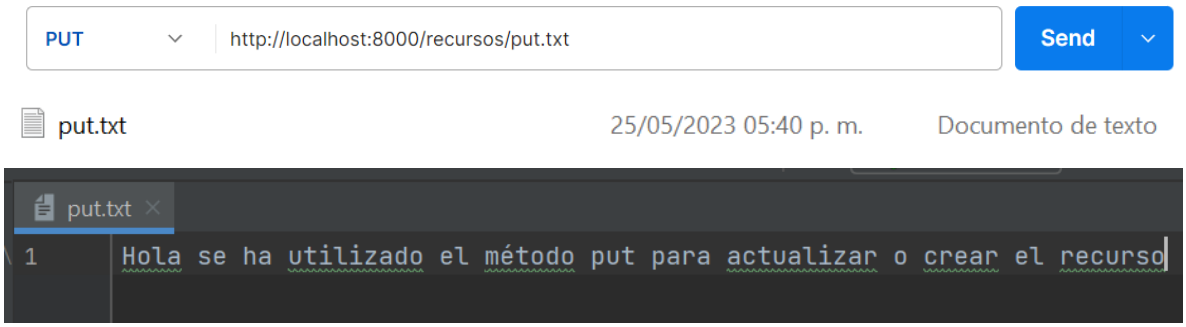


MÉTODO POST

Para el método POST el usuario podrá rellenar el formulario con datos solicitados, al enviar el formulario, se le desplegaran los parámetros que se encuentran en el body, es decir en la URL solo aparece el método post sin los datos del formulario como lo hace el método GET. En las siguientes imágenes se puede observar

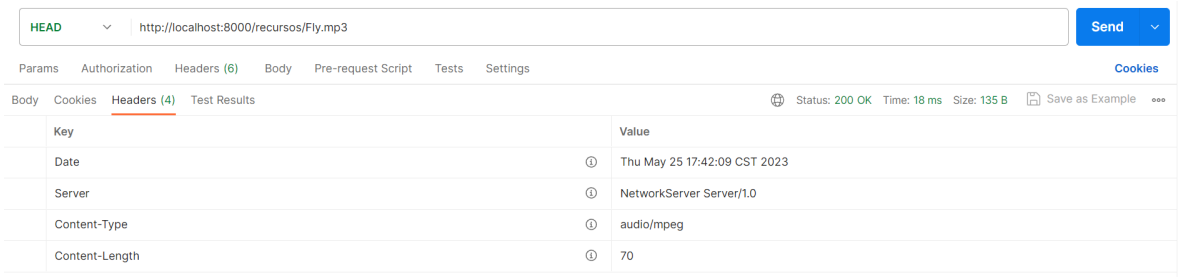
MÉTODO PUT

El método PUT permite al usuario crear o actualizar un recurso con una cadena de texto. Para ejemplificarlo se decidió crear el archivo de texto put.txt con una cadena de texto.



MÉTODO HEAD

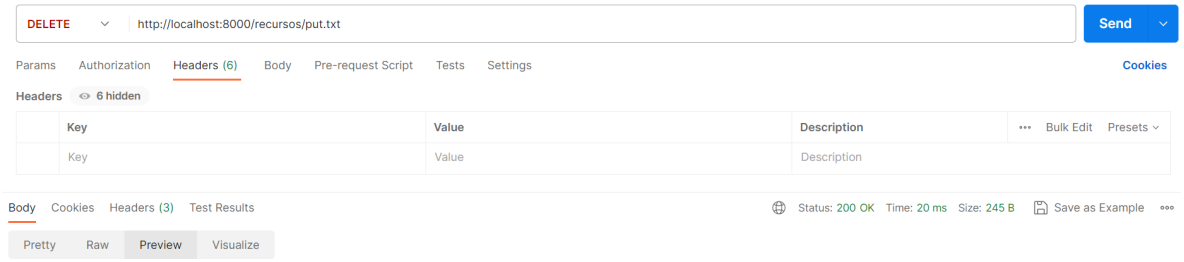
Con el método HEAD el usuario podrá revisar las cabeceras ya que cuando se trabaja con este método de petición no se está interesado en el contenido, solo en el código de HTTP de respuesta y el encabezado (Headers) como se puede ver a continuación. La respuesta del servidor contiene el tipo de método HTTP aplicado, una respuesta OK, la fecha, el nombre del servidor, el content type y el content length.




```
..:: Cliente Conectado desde /0:0:0:0:0:0:0:1 ::..
Desde el puerto: 51588
Datos: HEAD /recursos/Fly.mp3 HTTP/1.1


Respuesta HEAD:
HTTP/1.1 200 OK
Date: Thu May 25 17:42:09 CST 2023
Server: NetworkServer Server/1.0
Content-Type: audio/mpeg
Content-Length: 70
```

MÉTODO DELETE

Con el método DELETE el cliente puede eliminar el recurso que desee . En caso de que el recurso se elimine exitosamente se le desplegará el aviso de que el recurso ha sido eliminado del servidor.



202 Deleted Resource recursos/put.txt

3. Preguntas

1. ¿Qué ventajas tiene el uso del método POST vs GET o HEAD?

En lo relativo a los datos, como, por ejemplo, al rellenar formularios con nombres de usuario y contraseñas, el método POST ofrece mucha discreción. Los datos no se muestran en el caché ni tampoco en el historial de navegación. La flexibilidad del método POST también resulta muy útil: no solo se pueden enviar textos cortos, sino también otros tipos de información, como fotos o vídeos.

2. ¿Qué modificaciones harías al servidor para que fuese capaz de interpretar scriptlets JSP?

Los JSP, así como los servlets, son programas del lado del servidor que se ejecutan dentro de un servidor HTTP compatible con Java . Apache Tomcat Server es la implementación de referencia oficial (RI) para Java servlet y JSP. Por lo que para que nuestro servidor fuese capaz de interpretar scriplets JSP, se debería encapsular el código, la lógica y definir cómo se deben manejar las solicitudes y las respuestas en un servidor de Java.

Además de asignar las solicitudes del cliente al motor adecuado para su procesamiento. Estos motores están contenidos en otros elementos, como hosts y servidores, que limitarán el alcance de la búsqueda del servidor

4. Conclusiones

Con el desarrollo de esta práctica se comprendió el desarrollo de un servidor HTTP basándonos en sockets de flujo, hilos y principios de funcionamiento del protocolo HTTP. Además de poder aplicar y probar cada uno de los métodos propuestos: GET, POST, PUT, HEAD y DELETE. Se pudo comprobar que los mensajes HTTP consisten en una petición de un cliente a un servidor y una respuesta de un servidor al cliente. Las peticiones o respuestas pueden ser simples o completas, dependiendo del contenido. La diferencia radica en que una petición completa incluye encabezado y contenido, mientras que una simple solo el encabezado. Finalmente se pudo establecer un pool de conexiones para que el servidor pueda operar de forma concurrente.

Bibliografía

- Tanenbaum, Andrew. “Redes de Computadoras”. Cuarta Edición, Pearson Prentice Hall, 2003.
- Métodos HTTP - POST, GET, PUT, DELETE. Estilo Web. (2022). Retrieved 14 May 2022, from http://estilow3b.com/metodos-http-post-get-put-delete/.
- Métodos HTTP. Diego.com.es. (2022). Retrieved 15 May 2022, from https://diego.com.es/metodos-http.
- Servidor Web o Servidor HTTP. Edu4java.com. (2022). Retrieved 15 May 2022, from http://www.edu4java.com/es/web/web30.html.