



INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO



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

PRÁCTICA 4:

Servidor HTTP

Alumno:

Meza Vargas Brandon David

Grupo:

3CM16

Profesor:

Moreno Cervantes Axel Ernesto

Índice

Introducción	4
Desarrollo	5
Pool	5
stopped	6
stop	6
initServer	6
Server	7
getFile	10
sendFileHead	11
sendFile	13
delete	15
put	16
response	16
Constants	17
Pruebas de funcionamiento	18
Probando método GET	18
Probando método HEAD	21
Probando método POST	22
Probando método PUT	23
Probando método DELETE	25
Conclusiones	27
Bibliografía	28

Índice de ilustraciones

Ilustración 1. Método run del pool de hilos.....	5
Ilustración 2. Método stopped.....	6
Ilustración 3. Método stop.....	6
Ilustración 4. Método initServer.....	7
Ilustración 5. Recibimiento de línea y método HEAD.....	8
Ilustración 6. Método POST.....	8
Ilustración 7. Petición PUT.....	9
Ilustración 8. Petición delete.....	9
Ilustración 9. Petición GET.....	10
Ilustración 10. Método getFile.....	11
Ilustración 11. Método sendFileHead.....	12
Ilustración 12. Método sendFile.....	15
Ilustración 13. Método delete.....	15
Ilustración 14. Método put.....	16
Ilustración 15. Método response.....	17
Ilustración 16. Clase Constants.....	17
Ilustración 17. Corriendo server.....	18
Ilustración 18. Formulario.....	18
Ilustración 19. Formulario lleno.....	19
Ilustración 20. Parámetros obtenidos usando GET.....	19
Ilustración 21. Respuesta servidor.....	20
Ilustración 22. Pidiendo una imagen.....	20
Ilustración 23. Respuesta del servidor.....	21
Ilustración 24. Petición HEAD desde postman.....	21
Ilustración 25. Respuesta HEAD.....	21
Ilustración 26. Formulario para método post.....	22
Ilustración 27. Obteniendo parámetros.....	22
Ilustración 28. Respuesta servidor.....	23
Ilustración 29. PUT con postman.....	23
Ilustración 30. Respuesta servidor.....	24
Ilustración 31. Respuesta de postman.....	24
Ilustración 32. Actualizando archivo.....	24
Ilustración 33. Respuesta del servidor.....	25
Ilustración 34. Respuesta de postman.....	25
Ilustración 35. Archivo borrado con postman.....	26
Ilustración 36. Respuesta servidor método DELETE.....	26

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 funcione 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.

```
@Override
public void run() {
    synchronized (this){
        this.runningThread = Thread.currentThread();
    }
    initServer();
    while(!stopped()){
        Socket cl;
        try{
            /** Accepting connections
            cl = this.s.accept();
            System.out.println(Constants.SERVER_CONNECTION_ACCEPTED);
        }catch (IOException e){
            if(stopped()){
                System.out.println(Constants.SERVER_CONNECTION_STOPPED);
                break;
            }
            throw new RuntimeException(Constants.SERVER_CONNECTION_ACCEPTED_ERROR, e);
        }
        this.pool.execute(new Server(cl));
    }
    // * sorted ending
    this.pool.shutdown();
    System.out.println(Constants.SERVER_CONNECTION_STOPPED);
}
```

Ilustración 1. Método run del pool de hilos.

stopped

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

```
/**
 * Method that obtains the flag value
 * @return stopped, the flag
 */
private synchronized boolean stopped() { return this.stopped; }
```

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.

```
/**
 * Method that changes the flag value to stop the socket connection
 */
public synchronized void stop(){
    this.stopped = true;
    try {
        this.s.close();
    } catch (IOException e){
        throw new RuntimeException(Constants.SERVER_CONNECTION_STOPPED_ERROR, e);
    }
}
```

Ilustración 3. Método stop

initServer

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

```

/**
 * Method that initialize the server
 */
private void initServer(){
    try {
        this.s = new ServerSocket(this.port);
        System.out.println(Constants.SERVER_CONNECTION_START);
    } catch (IOException e){
        throw new RuntimeException(String.format(Constants.SERVER_CONNECTION_ERROR, s.getLocalPort()), e);
    }
}

```

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.

```

public void run(){
    try {
        dos = new DataOutputStream(socket.getOutputStream());
        dis = new DataInputStream(socket.getInputStream());
        byte[] b = new byte[Constants.INPUT_STREAM_BYTES_SIZE];
        int t = dis.read(b);
        String request = new String(b, 0, t);
        System.out.println("t: " + t);

        if(request == null){
            StringBuilder sb = new StringBuilder();
            sb.append("<html><head><title>Servidor WEB</title>");
            sb.append("</title><body bgcolor='\"#AACCFF\"'><br>Linea Vacia</br></body></html>");
            sb.append("</body></html>");
            dos.write(sb.toString().getBytes());
            dos.flush();
            socket.close();
            return;
        }

        System.out.println(String.format(Constants.CONNECTED_CLIENT_MESSAGE, socket.getInetAddress(), socket.getPort()));
        System.out.println(String.format(Constants.CONNECTED_CLIENT_DATA, request));

        StringTokenizer stl = new StringTokenizer(request, "\n");
        String line = stl.nextToken();

        if(line.indexOf(' ') == -1){
            if(line.toUpperCase().startsWith("HEAD")){
                getFile(line);
                if(fileName.compareTo("") == 0)
                    sendFileHead("index.htm", dos);
                else sendFileHead(fileName, dos);
            }

```

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.

```

}else if(line.toUpperCase().startsWith("POST")){
    String params = request.substring(request.lastIndexOf(' ') + 1);
    //Getting response
    StringBuffer response = new StringBuffer();
    response.append("HTTP/1.0 200 OK \n");
    String date = "Date: " + new Date() + "\n";
    response.append(date);
    String tipo_mime = "Content-Type: text/html \n\n";
    response.append(tipo_mime);
    response.append("<html><head><title>SERVIDOR WEB</title></head>");
    response.append("<body bgcolor='\"#AACCFF\"'><center><h1><br> Parametros obtenidos </br></h1><n3><b>");
    response.append(params);
    response.append("</b></n3>");
    response.append("</center></body></html>");
    System.out.println("Respuesta: " + response);
    dos.write(response.toString().getBytes());
    dos.flush();
    dos.close();
    socket.close();
}

```

Ilustración 6. 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.

```
}else if(line.toUpperCase().startsWith("PUT")){  
    getFile(line);  
    while(!line.contains("Content-Type"))  
        line = stl.nextToken();  
    stl.nextToken();  
    put(stl);  
}
```

Ilustración 7. 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.

```
}else if(line.toUpperCase().startsWith("DELETE")){  
    getFile(line);  
    delete();  
}
```

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.

```

} else if (line.toUpperCase().startsWith("GET")) {
    getFile(line);
    if (fileName.compareTo("") == 0)
        sendFile("index.htm", dos);
    else sendFile(fileName, dos);
    StringTokenizer tokens = new StringTokenizer(line, "?");
    String req_a = tokens.nextToken();
    String req = tokens.nextToken();
    System.out.println("Token1: " + req_a);
    System.out.println("Token2: " + req);
    String params = req.substring(0, req.indexOf(" ")) + "\n";
    System.out.println("Params: " + params);
    /*Getting response
    StringBuffer response = new StringBuffer();
    response.append("HTTP/1.0 200 OK \n");
    String date = "Date: " + new Date() + " \n";
    response.append(date);
    String tipo_mime = "Content-Type: text/html \n\n";
    response.append(tipo_mime);
    response.append("<html><head><title>SERVIDOR WEB</title></head>\n");
    response.append("<body bgcolor=#AACCFF><center><h1><br> Parametros obtenidos </br></h1><h3><b>\n");
    response.append(params);
    response.append("</b></h3>\n");
    response.append("</center></body></html>\n");
    System.out.println("Respuesta: " + response);
    dos.write(response.toString().getBytes());
    dos.flush();
    dos.close();
    socket.close();
    } else {
        dos.write(Constants.HTTP_NOT_IMPLEMENTED.getBytes());
        dos.flush();
        dos.close();
        socket.close();
    }
} catch (IOException e) {
    e.printStackTrace();
}

```

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.

```

/**
 * Method that obtains the file name and its extension
 * @param line, the line received
 */
public void getFile(String line){
    int i, f;
    i = line.indexOf("/");
    f = line.indexOf(" ", i);
    fileName = line.substring(i+1, f);

    /*Getting extension
    extension = "";
    i = fileName.lastIndexOf('.');
    if(i > 0) extension = fileName.substring(i+1);
    }

```

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.

```

/**
 * Method that send the file when is HEAD request
 * @param arg The file name
 * @param dos1 The data OutputStream required to write
 */
public void sendFileHead(String arg, DataOutputStream dos1){
    try{
        DataInputStream dis2 = new DataInputStream(new FileInputStream(arg));
        File ff = new File(arg);
        long fileSize = ff.length();
        /*******
        String sb = "";
        sb = sb+"HTTP/1.0 200 ok\n";
        sb = sb + "Server: BrandonMV Server/1.0 \n";
        sb = sb + "Date: " + new Date()+" \n";
        switch (extension){
            case "html":
                sb = sb + "Content-Type: text/html \n";
                break;
            case "jpg":
            case "jpeg":
                sb = sb + "Content-Type: image/jpeg \n";
                break;
            case "pdf":
                sb = sb + "Content-Type: application/pdf \n";
                break;
        }
        //sb = sb + "Content-Type: text/html \n";
        sb = sb + "Content-Length: "+fileSize+" \n";
        sb = sb + "\n";
        dos1.write(sb.getBytes());
        dos1.flush();
        /*******
        dis2.close();
        dos1.close();
    }catch (IOException e){
        System.err.println(e.getMessage());
    }
}

```

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.

```

/**
 * Method that send the file
 * @param arg The file name
 * @param dos1 The data OutputStream required to write
 */
public void sendFile(String arg, DataOutputStream dos1){
    try{
        /*
        String extension = "";
        int i = arg.lastIndexOf('.');
        if(i > 0) extension = arg.substring(i+1);
        */

        DataInputStream dis2 = new DataInputStream(new FileInputStream(arg));
        byte[] buf = new byte[Constants.INPUT_STREAM_BYTES_SIZE];
        int x = 0;
        File ff = new File(arg);
        long fileSize = ff.length(), cont = 0;
        /**
        String sb = "";
        sb = sb+"HTTP/1.0 200 ok\n";
        sb = sb + "Server: BrandonMV Server/1.0 \n";
        sb = sb +"Date: " + new Date()+" \n";
        switch (extension){
            case "html":
                sb = sb +"Content-Type: text/html \n";
                break;
            case "jpg":
            case "jpeg":
                sb = sb + "Content-Type: image/jpeg \n";
                break;
            case "pdf":
                sb = sb +"Content-Type: application/pdf \n";
                break;
        }
        //sb = sb +"Content-Type: text/html \n";
        sb = sb +"Content-Length: "+fileSize+" \n";
        sb = sb +"\n";
        dos1.write(sb.getBytes());
        dos1.flush();
        */
    }
}

```

```

    /**
    while(cont < fileSize){
        x = dis2.read(buf);
        dos1.write(buf, off: 0, x);
        cont+=x;
        dos1.flush();
    }
    dis2.close();
    dos1.close();
} catch (IOException e){
    System.err.println(e.getMessage());
}
}

```

Ilustración 12. Método sendFile

delete

Este 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.

```

/**
 * The delete HTTP method, deletes the indicated file
 * @throws IOException
 */
private void delete() throws IOException{
    File f = new File(fileName);
    if(f.exists()){
        if(f.canWrite()){
            f.delete();
            /**Getting response
            response( statusCode: "200", meaning: "OK", msg: "File deleted");
        }else{
            response( statusCode: "403", meaning: "Forbidden", msg: "403 FORBIDDEN");
        }
    }else{
        response( statusCode: "404", meaning: "Not Found", msg: "404 Not Found");
    }
}
}

```

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.

```
/**
 * Method that updates or creates a file
 * @param stl The file content
 * @throws IOException Exception thrown
 */
private void put(StringTokenizer stl) throws IOException{
    FileWriter fw;
    File f = new File(fileName); /** File to create or update
    File f2 = new File(fileName); /** File updated
    /**If the file exists and is writable means that the file will be updated
    if(f.exists()){
        if(f.canWrite()){
            f.delete();
            f2.createNewFile();
            fw = new FileWriter(fileName);
            while(stl.hasMoreTokens())
                fw.write(stl.nextToken());
            fw.close();
            response( statusCode: "200", meaning: "OK", msg: "Archivo actualizado");
        }else response( statusCode: "403", meaning: "Forbidden", msg: "403 Forbidden"); /** Sin permiso para escribir
    }else{
        /** If file does not exist we have to create it
        f.createNewFile();
        response( statusCode: "201", meaning: "Created", msg: "Archivo creado");
        fw = new FileWriter(fileName);
        while(stl.hasMoreTokens())
            fw.write(stl.nextToken());
        fw.close();
    }
}
```

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.


```

/**
 * Method that set the html response
 * @param statusCode The status code
 * @param meaning The status code meaning
 * @param msg The message that will be print
 */
private void response(String statusCode, String meaning, String msg) throws IOException{
    StringBuffer response = new StringBuffer();
    response.append("HTTP/1.0 " + statusCode + meaning + " \n");
    String date= "Date: " + new Date()+" \n";
    response.append(date);
    String tipo_mime = "Content-Type: text/html \n\n";
    response.append(tipo_mime);
    response.append("<html><head><title>SERVIDOR WEB</title></head>\n");
    response.append("<body bgcolor=\"#AACCFF\"><center><h1><br>" + msg + "</br></h1><h3><b>\n");
    response.append("</b></h3>\n");
    response.append("</center></body></html>\n\n");
    System.out.println("Respuesta: "+response);
    dos.write(response.toString().getBytes());
    dos.flush();
    dos.close();
    socket.close();
}

```

Ilustración 15. Método response.

Constants

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

```

/**Server configs
public static final int PORT = 8000;
public static final int THREADS = 100;

/**Server messages
public static final String SERVER_CONNECTION_ERROR = "No se pudo iniciar el socket en el puerto %d";
public static final String SERVER_CONNECTION_START = "Servidor iniciado, esperando clientes...";
public static final String SERVER_CONNECTION_ACCEPTED = "Conexion aceptada...";
public static final String SERVER_CONNECTION_STOPPED = "Servidor detenido";
public static final String SERVER_CONNECTION_ACCEPTED_ERROR = "Error al aceptar una nueva conexion";
public static final String SERVER_CONNECTION_STOPPED_ERROR = "Error al cerrar el socket del servidor";

/**Utilities
public static final int INPUT_STREAM_BYTES_SIZE = 1024;
public static final String CONNECTED_CLIENT_MESSAGE = "\nCliente conectado desde: %s\nPor el puerto: %d";
public static final String CONNECTED_CLIENT_DATA = "Datos: %s\r\n\r\n";

public static final String HTTP_NOT_IMPLEMENTED = "HTTP/1.0 501 Not Implemented\r\n";

```

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.

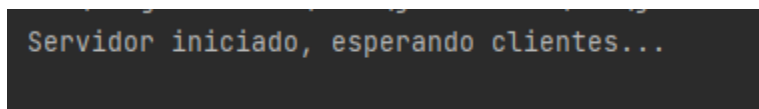


Ilustración 17. Corriendo server

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.

A screenshot of a web page with a yellow background. At the top, it says 'Ejemplo sencillo de un formulario' and 'Formulario de Registro'. Below this is a photo of a man holding a basketball. Under the photo, there is a message: 'Aquí hay un ejemplo de cómo se ve un formulario de registro.' followed by the text 'Usuarios:'. Below this, there are two sets of form fields. The first set has fields for 'Nombre', 'Apellido', 'Email', and 'Contraseña', followed by a 'Registrar' button. The second set has fields for 'Nombre', 'Apellido', 'Email', and 'Contraseña', followed by a 'Registrar' button. There are also links for 'Crear nuevo' and 'Recuperar contraseña'.

Ilustración 18. Formulario

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

Sea tan amable de rellenar el siguiente formulario.

Nombre:

Dirección:

Teléfono:

Comentarios:

Nombre:

Ilustración 19. Formulario lleno.

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

Parametros obtenidos

Apellido=brandon&Direccion=Ecatepec&Telefono=5610690179&comentario=Hola+estamos+probando+el+metodo+get

Ilustración 20. Parámetros obtenidos usando GET

La respuesta que obtenemos en nuestro servidor es la siguiente:

```

Cliente conectado desde: /127.0.0.1
Por el puerto: 54890
Datos: GET /?Apellido=brandon&Direccion=Ecatepec&Telefono=5610690179&comentario=Hola+estamos+probando+el+metodo+get HTTP/1.1
Host: 127.0.0.1:8080
Connection: keep-alive
sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="100", "Microsoft Edge";v="100"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100.0.4896.127 Safari/537.36 Edg/100.0.1185.44
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: cross-site
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Accept-Language: es-419;es;q=0.9,es-ES;q=0.8,en;q=0.7,en-GS;q=0.6,en-US;q=0.5

Token1: GET /
Token2: Apellido=brandon&Direccion=Ecatepec&Telefono=5610690179&comentario=Hola+estamos+probando+el+metodo+get HTTP/1.1
Params: Apellido=brandon&Direccion=Ecatepec&Telefono=5610690179&comentario=Hola+estamos+probando+el+metodo+get

Respuesta: HTTP/1.0 200 OK
Date: Fri Apr 22 15:19:27 CDT 2022
Content-Type: text/html

<html><head><title>SERVIDOR WEB</title></head>
<body bgcolor="#AACCFF"><center><h1><br> Parametros obtenidos </br></h1><h3><b>
Apellido=brandon&Direccion=Ecatepec&Telefono=5610690179&comentario=Hola+estamos+probando+el+metodo+get
</b></h3>
</center></body></html>

```

Ilustración 21. Respuesta servidor

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

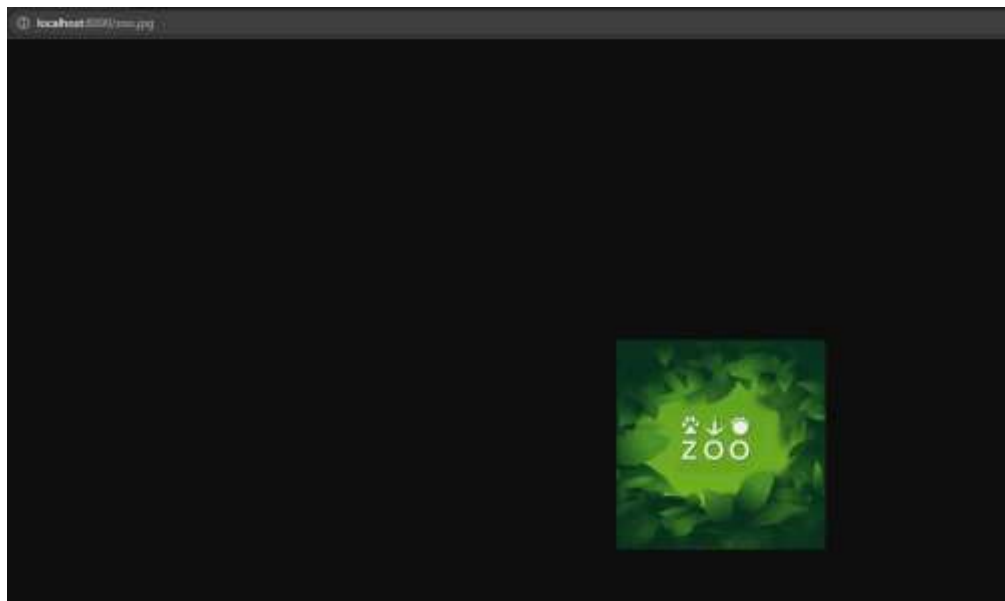


Ilustración 22. Pidiendo una imagen

En el servidor tenemos la siguiente respuesta.

```

Conexion aceptada...
t: 671

Cliente conectado desde: /0:0:0:0:0:0:1
Por el puerto: 51995
Datos: GET /zoo.jpg HTTP/1.1
Host: localhost:8000
Connection: keep-alive
sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="100", "Google Chrome";v="100"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100.0.4896.127 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Accept-Language: es-ES,es;q=0.9

```

Ilustración 23. Respuesta del servidor

Probando método HEAD

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

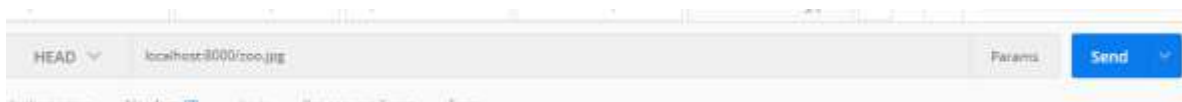


Ilustración 24. Petición HEAD desde postman

En el servidor obtenemos lo siguiente

```

Conexion aceptada...
t: 561

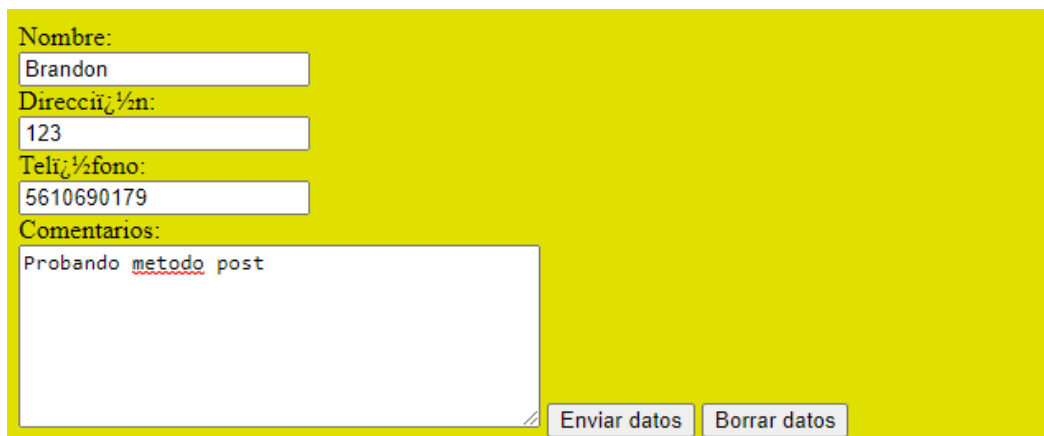
Cliente conectado desde: /0:0:0:0:0:0:1
Por el puerto: 51968
Datos: HEAD /zoo.jpg HTTP/1.1
Host: localhost:8000
Connection: keep-alive
sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="100", "Google Chrome";v="100"
Cache-Control: no-cache
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100.0.4896.127 Safari/537.36
sec-ch-ua-platform: "Windows"
Postman-Token: c0eee910-350e-8a4e-f2e9-3a3de8b8fa5a
Accept: */*
Sec-Fetch-Site: none
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Accept-Encoding: gzip, deflate, br
Accept-Language: es-ES,es;q=0.9

```

Ilustración 25. Respuesta HEAD

Probando método POST

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



Formulario para el método POST. El formulario está sobre un fondo amarillo y contiene los siguientes campos:

- Nombre:
- Dirección:
- Teléfono:
- Comentarios:

En la parte inferior derecha hay dos botones: "Enviar datos" y "Borrar datos".

Ilustración 26. Formulario para método post.



Pantalla de parámetros obtenidos. El fondo es azul claro. En el centro, el título "Parametros obtenidos" está en negrita. Debajo, se muestra la cadena de parámetros: "Apellido=Brandon&Direccion=123&Telefono=5610690179&comentario=Probando+metodo+post".

Ilustración 27. Obteniendo parámetros

```

Cliente conectado desde: /127.0.0.1
Por el puerto: 52021
Datos: POST / HTTP/1.1
Host: 127.0.0.1:8000
Connection: keep-alive
Content-Length: 82
Cache-Control: max-age=0
sec-ch-ua: "Not A;Brand";v="99", "Chromium";v="100", "Google Chrome";v="100"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: http://localhost:8000
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100.0.4896.127 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: cross-site
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost:8000/
Accept-Encoding: gzip, deflate, br
Accept-Language: es-ES,es;q=0.9

Apellido=Brandon&Direccion=123&Telefono=5610690179&comentario=Probando+metodo+post

Respuesta: HTTP/1.0 200 OK
Date: Fri Apr 22 17:05:07 CDT 2022
Content-Type: text/html

<html><head><title>SERVIDOR WEB</title></head>
<body bgcolor="#AACCFF"><center><h1><br> Parasetros obtenidos: </br></h1><h3><br>
Apellido=Brandon&Direccion=123&Telefono=5610690179&comentario=Probando+metodo+post</br></h3>
</center></body></html>

```

Ilustración 28. 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.

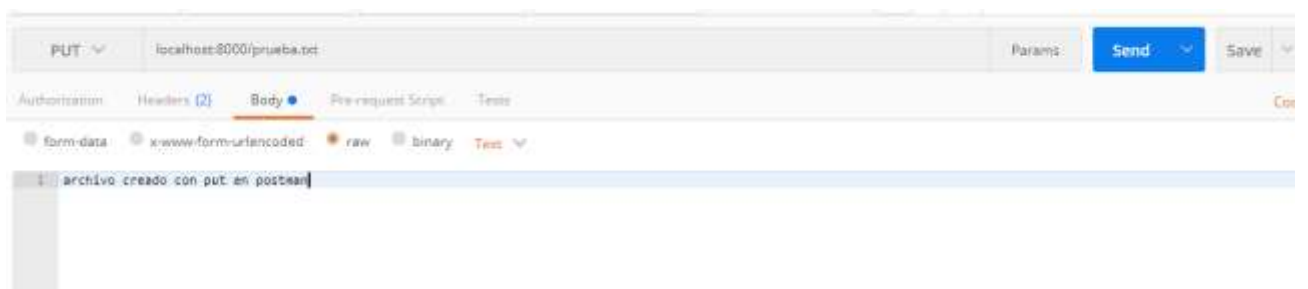


Ilustración 29. PUT con postman

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

```

Cliente conectado desde: /0:0:0:0:0:0:1
Por el puerto: 82104
Datos: PUT /prueba.txt HTTP/1.1
Host: localhost:8000
Connection: keep-alive
Content-Length: 33
sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="108", "Google Chrome";v="108"
Cache-Control: no-cache
Content-Type: text/plain;charset=UTF-8
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100.0.4896.127 Safari/537.36
sec-ch-ua-platform: "Windows"
Postman-Token: eaa4fa7c-2a2c-c164-4648-c7d2265c0a6f
Accept: */*
Origin: chrome-extension://fhbjgbiflinjbdggehcddcbncddownop
Sec-Fetch-Site: none
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Accept-Encoding: gzip, deflate, br
Accept-Language: es-ES,es;q=0.9

archivo creado con put en postman

Respuesta: HTTP/1.0 201Created
Date: Fri Apr 22 17:09:36 CDT 2022
Content-Type: text/html

<html><head><title>SERVIDOR WEB</title></head>
<body bgcolor="#AACCFF"><center><h1><br>Archivo creado</br></h1><n3><b>
</b></n3>
</center></body></html>

```

Ilustración 30. Respuesta servidor.

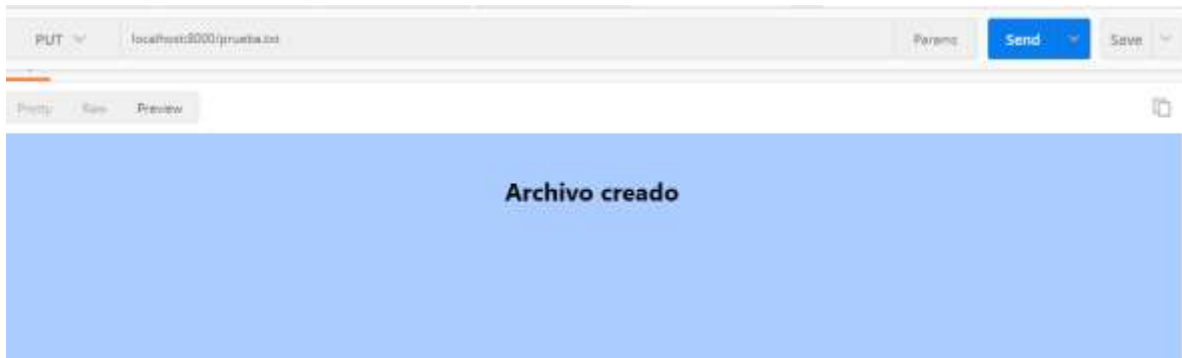


Ilustración 31. Respuesta de postman

Ahora actualizaremos ese archivo.

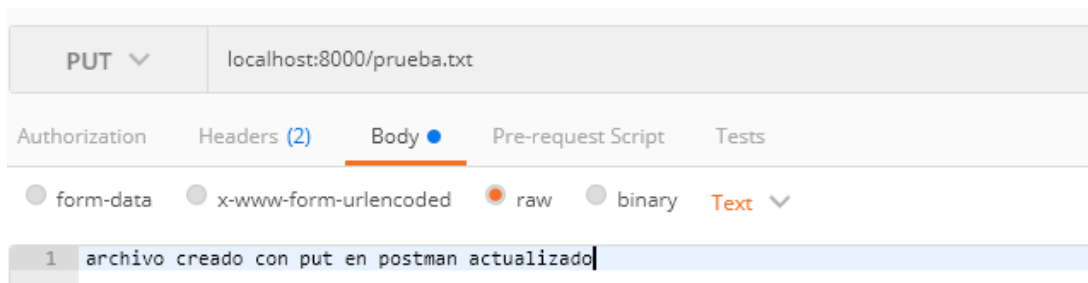


Ilustración 32. Actualizando archivo

Al actualizar el archivo se nos muestran las siguientes respuestas.

```
Cliente conectado desde: /0:0:0:0:0:0:1
Por el puerto: 52146
Datos: PUT /prueba.txt HTTP/1.1
Host: localhost:8000
Connection: keep-alive
Content-Length: 45
sec-ch-ua: * Not A;Brand";v="99", "Chromium";v="100", "Google Chrome";v="100"
Cache-Control: no-cache
Content-Type: text/plain;charset=UTF-8
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100.0.4896.127 Safari/537.36
sec-ch-ua-platform: "Windows"
Postman-Token: a49dd994-d4e1-107a-d59e-1d5fef72533e
Accept: */*
Origin: chrome-extension://fmbjgblflnjbdgqehcddcbncdddeomp
Sec-Fetch-Site: none
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Accept-Encoding: gzip, deflate, br
Accept-Language: es-ES,es;q=0.9

archivo creado con put en postman actualizado

Respuesta: HTTP/1.1 200OK
Date: Fri Apr 22 17:12:01 CDT 2022
Content-Type: text/html

<html><head><title>SERVIDOR WEB</title></head>
<body bgcolor="#AACCFF"><center><h1><br>Archivo actualizado</br></h1><h3><b>
</b></h3>
</center></body></html>
```

Ilustración 33. Respuesta del servidor.

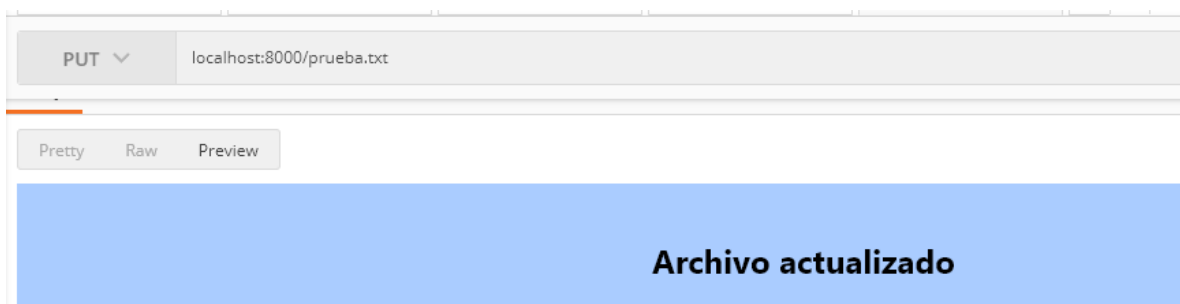


Ilustración 34. Respuesta de postman

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,

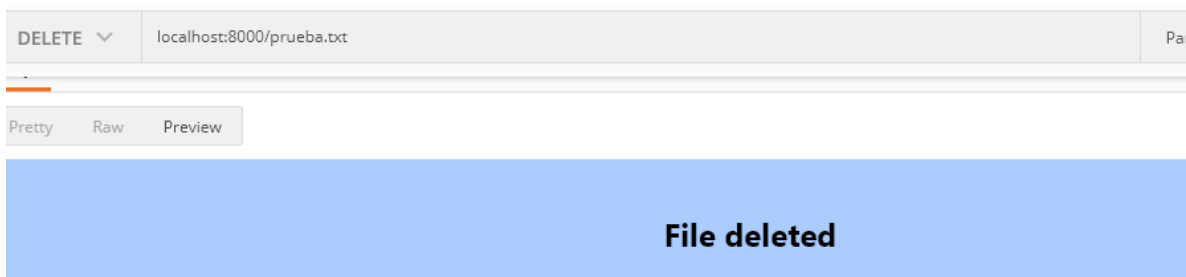


Ilustración 35. Archivo borrado con postman

La respuesta arrojada por el servidor fue la siguiente.

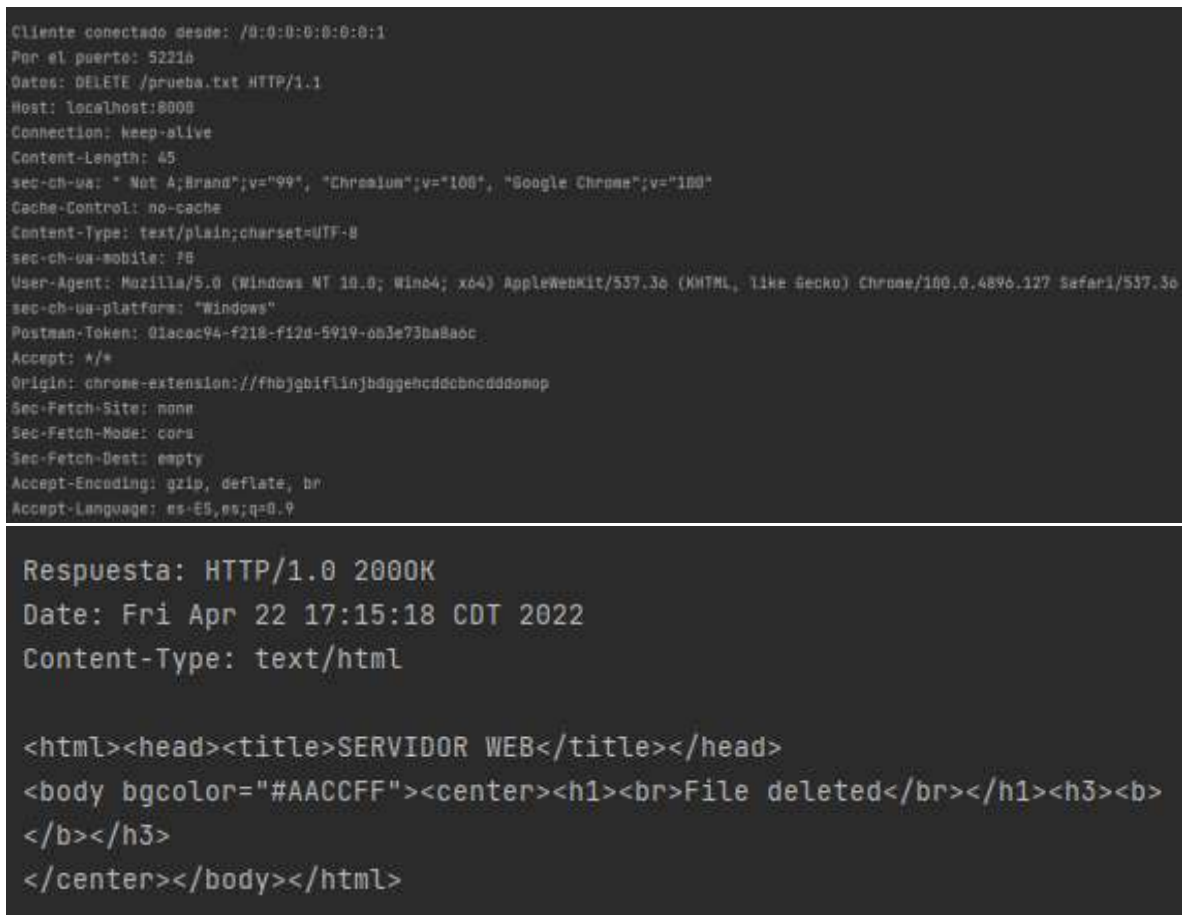


Ilustración 36. Respuesta servidor método DELETE

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