Instituto Politécnico Nacional

Escuela Superior de Cómputo

Práctica 4 - Servidor HTTP

Unidad de aprendizaje: Aplicaciones para Comunicaciones de Red

Grupo: 3CM8

Alumnos(a):
Nicolás Sayago Abigail
Ramos Díaz Enrique

Profesor(a):
Moreno Cervantes Axel

14 de Mayo 2019

Índice

1. Introducción

En ésta práctica de implementa un servidor HTTP de acuerdo con el RFC2616, que atienda peticiones GET, POST, HEAD, DELETE para distintos tipos de recursos (ver tipos MIME) y que permita al usuario definir el tamaño de pool de conexiones que se usará para limitar la cantidad de usuarios atendidos de manera concurrente:

2. Desarrollo

2.1. Servidor de conexiones

Al hacer la conexión, se pide el número de conexiones que se desean. Usamos un pool para llevar a cabo esas n conexiones.

```
1  import java.net.*;
2  import java.io.*;
3  import java.util.*;
4  import java.util.concurrent.ExecutorService;
5  import java.util.concurrent.Executors;
6
7  public class ServidorWeb {
8
9      public static void main(String[] args) {
10      int pto, tamPool;
11
12         try {
13          Scanner sc = new Scanner(System.in);
14          System.out.print("Puerto: ");
15    pto = sc.nextInt();
16    System.out.print("Tamanio del pool de conexiones: ");
17    tamPool = sc.nextInt();
18
19          // Pool de Conexiones
20          ExecutorService pool = Executors.newFixedThreadPool(tamPool);
21          System.out.println("\n\n -----> Iniciando Servidor.... Pool de
             ↪  Conexiones = " + tamPool);
22
23          ServerSocket s = new ServerSocket(pto);
24          System.out.println("Servidor iniciado: http://localhost:" + pto +
             ↪  "/ --- OK");
25          System.out.println("Esperando a Cliente....");
26
```

```
27          for( ; ; ) {
28              Socket cl = s.accept();
29              Manejador manejador = new Manejador(cl);
30              pool.execute(manejador);
31          }
32         }
33         catch(Exception e){
34          e.printStackTrace();
35         }
36     }
37 }
```

2.2. Manejador

En esta clase se crea un hilo por cada cliente dependiendo de que petición se este llevando acabo.

```
1  import java.net.*;
2  import java.io.*;
3  import java.util.*;
4  import java.util.Base64;
5
6  public class Manejador extends Thread {
7      protected Socket cl;
8      protected DataOutputStream dos;
9      protected Mime mime;
10     protected DataInputStream dis;
11
12     public Manejador(Socket cl) throws Exception {
13         this.cl = cl;
14         this.dos = new DataOutputStream(this.cl.getOutputStream());
15         this.mime = new Mime();
16         this.dis = new DataInputStream(this.cl.getInputStream());
17     }
18
19     public void eliminarRecurso(String arg, String headers){
20         try {
21             System.out.println(arg);
22             File f = new File(arg);
23
24             if(f.exists()) {
25                 if (f.delete()) {
26                     System.out.println("------> Archivo " + arg + "
                        ↪  eliminado exitosamente\n");
```

```
27
28                  String deleteOK = headers +
29                                    "<html><head><meta
                                       ↪ charset='UTF-8'><title>202 OK
                                       ↪ Recurso eliminado</title></head>"
                                       ↪ +
30                                    "<body><h1>202 OK Recurso eliminado
                                       ↪ exitosamente.</h1>" +
31                                    "<p>El recurso " + arg + " ha sido
                                       ↪ eliminado permanentemente del
                                       ↪ servidor." +
32                                    "Ya no se podra acceder más a él.</p>"
                                       ↪ +
33                                    "</body></html>";
34
35                  dos.write(deleteOK.getBytes());
36                  dos.flush();
37                  System.out.println("Respuesta DELETE: \n" + deleteOK);
38              }
39              else {
40                  System.out.println("El archivo " + arg + " no pudo ser
                     ↪ borrado\n");
41
42                  String error404 = "HTTP/1.1 404 Not Found\n" +
43                                    "Date: " + new Date() + " \n" +
44                                    "Server: EnrikeAbi Server/1.0 \n" +
45                                    "Content-Type: text/html \n\n" +
46
47                                    "<html><head><meta
                                       ↪ charset='UTF-8'><title>404 Not
                                       ↪ found</title></head>" +
48                                    "<body><h1>404 Not found</h1>" +
49                                    "<p>Archivo " + arg + " no
                                       ↪ encontrado.</p>" +
50                                    "</body></html>";
51
52                  dos.write(error404.getBytes());
53                  dos.flush();
54                  System.out.println("Respuesta DELETE - ERROR 404: \n" +
                     ↪ error404);
55              }
56          }
57      }
58      catch(Exception e) {
```

```
59              System.out.println(e.getMessage());
60          }
61      }
62
63      public void enviarRecurso(String arg, int bandera) {
64          try {
65           File f = new File(arg);
66           String sb = "HTTP/1.1 200 OK\n";
67
68           if(!f.exists()) {
69            arg = "404.html"; // Recurso no encontrado
70            sb = "HTTP/1.1 404 Not Found\n";
71           }
72           else if(f.isDirectory()) {
73            arg = "403.html"; // Recurso privado
74            sb = "HTTP/1.1 403 Forbidden\n";
75           }
76
77        DataInputStream dis2 = new DataInputStream(new FileInputStream(arg));
78        int tam = dis2.available();
79
80        // Obtenemos extension para saber el tipo de recurso
81              int pos = arg.indexOf(".");
82              String extension = arg.substring(pos + 1, arg.length());
83
84              // Enviamos las cabeceras de la respuesta HTTP - METODO HEAD
85              sb = sb + "Date: " + new Date() + " \n" +
86                  "Server: EnrikeAbi Server/1.0 \n" +
87                  //Distintos tipos MIME para distintos tipos de archivos
88                  "Content-Type: " + mime.get(extension) + " \n" +
89                  "Content-Length: " + tam + " \n\n";
90
91              dos.write(sb.getBytes());
92              dos.flush();
93
94              String metodo = "HEAD";
95              if (bandera == 1) {
96               metodo = "GET";
97               // Respuesta GET, enviamos el archivo solicitado
98               byte[] b = new byte[1024];
99               long enviados = 0;
100              int n = 0;
101
102              while(enviados < tam) {
```

```
103                 n = dis2.read(b);
104                 dos.write(b, 0, n);
105                 dos.flush();
106                 enviados += n;
107              }
108             }
109             System.out.println("Respuesta " + metodo + ": \n" + sb);
110             dis2.close();
111         }
112         catch(Exception e) {
113             System.out.println(e.getMessage());
114             //e.printStackTrace();
115         }
116     }
117 
118     public String obtenerNombreRecurso(String line) {
119      // Obtiene el nombre del recurso de la peticion HTTP
120         int i = line.indexOf("/");
121         int f = line.indexOf(" ", i);
122         String resourceName = line.substring(i + 1, f);
123 
124         // Si es vacio, entonces se trata del index
125         if(resourceName.compareTo("") == 0)
126             resourceName = "index.html";
127 
128         return resourceName;
129     }
130 
131     public String obtenerParametros(String line, String headers, int
    ↪  bandera) {
132         String metodo = "POST";
133         String request2 = line;
134 
135         if(bandera == 0) {
136             metodo = "GET";
137          // Line: GET /?Nombre=&Direccion=&Telefono=&Comentarios= HTTP/1.1
138          // Separamos los parametros de "GET"
139             System.out.println(line);
140          StringTokenizer tokens = new StringTokenizer(line, "?");
141             String request = tokens.nextToken();
142             request = tokens.nextToken();
143 
144             // Separamos los parametros de "HTTP/1.1"
145             StringTokenizer tokens2 = new StringTokenizer(request, " ");
```

```
146             request2 = tokens2.nextToken();
147         }
148 
149         System.out.println(request2);
150         // Separamos los parametros junto a su valor uno del otro
151         StringTokenizer paramsTokens = new StringTokenizer(request2, "&");
152 
153         String html = headers +
154        "<html><head><meta charset='UTF-8'><title>Metodo " + metodo + "\n" +
155                       "</title></head><body
                        ↪  bgcolor='#AACCFF'><center><h2>Parametros obtenidos
                        ↪  por medio de " + metodo + "</h2><br>\n" +
156                       "<table
                        ↪  border='2'><tr><th>Parametro</th><th>Valor</th></tr>";
157 
158        // Se recorren todos los parametros, mientras existan
159         while(paramsTokens.hasMoreTokens()) {
160          String parametros = paramsTokens.nextToken();
161          // Separamos el nombre del parametro de su valor
162          StringTokenizer paramValue = new StringTokenizer(parametros, "=");
163          String param = ""; //Nombre del parametro
164          String value = ""; //Valor del parametro
165 
166          // Hay que revisar si existen o si se enviaron parametros vacios
167          if(paramValue.hasMoreTokens())
168           param = paramValue.nextToken();
169 
170          if(paramValue.hasMoreTokens())
171           value = paramValue.nextToken();
172 
173          html = html + "<tr><td><b>" + param + "</b></td><td>" + value +
              ↪  "</td></tr>\n";
174         }
175         html = html + "</table></center></body></html>";
176         return html;
177     }
178 
179     @Override
180     public void run() {
181      // Cabeceras de respuestas HTTP
182      String headers = "HTTP/1.1 200 OK\n" +
183                  "Date: " + new Date() + " \n" +
184                 "Server: EnrikeAbi Server/1.0 \n" +
185                 "Content-Type: text/html \n\n";
```

```
186         try {
187             String line = dis.readLine(); // Lee primera linea DEPRECIADO
                ↪  !!!!
188             // Linea vacia
189             if(line == null) {
190                 String vacia = "<html><head><title>Servidor WEB</title><body
                    ↪  bgcolor='#AACCFF'>Linea Vacia</body></html>";
191                 dos.write(vacia.getBytes());
192                 dos.flush();
193             }
194             else {
195                 System.out.println("\n------> Cliente Conectado desde: " +
                    ↪  cl.getInetAddress());
196                 System.out.println("Por el puerto: " + cl.getPort());
197                 System.out.println("Datos: " + line + "\r\n\r\n");
198 
199                 // Metodo GET
200                 if(line.toUpperCase().startsWith("GET")) {
201                  if(line.indexOf("?") == -1) {
202                      // Solicita un archivo
203             String fileName = obtenerNombreRecurso(line);
204             // Bandera HEAD = 0, GET = 1
205                      enviarRecurso(fileName, 1);
206                  }
207                  else {
208                      // Envia parametros desde un formulario
209                         // Bandera GET = 0, POST = 1
210                      String respuesta = obtenerParametros(line, headers, 0);
211                      // Respuesta GET, devolvemos un HTML con los parametros
                        ↪  del formulario
212                      dos.write(respuesta.getBytes());
213                      dos.flush();
214                      System.out.println("Respuesta GET: \n" + respuesta);
215                  }
216                 } // Metodo HEAD
217                 else if(line.toUpperCase().startsWith("HEAD")) {
218                  if(line.indexOf("?") == -1) {
219                   // Solicita archivo, unicamente enviamos tipo mime y
                      ↪  longitud
220                   String fileName = obtenerNombreRecurso(line);
221                   // Bandera HEAD = 0, GET = 1
222                         enviarRecurso(fileName, 0);
223                  }
224                  else {
```

```
225                     // Respuesta HEAD, devolvemos unicamente las cabeceras
                         ↪ HTTP
226                     dos.write(headers.getBytes());
227                     dos.flush();
228                     System.out.println("Respuesta HEAD: \n" + headers);
229                  }
230                 } // Metodo POST
231                 else if(line.toUpperCase().startsWith("POST")) {
232                     // Leemos el flujo de entrada
233                     int tam = dis.available();
234                     byte[] b = new byte[tam];
235
236                     dis.read(b);
237                     //Creamos un string con los bytes leidos
238                     String request = new String(b, 0, tam);
239
240                     // Separamos los parametros del resto de los encabezados
                         ↪ HTTP
241                     String[] reqLineas = request.split("\n");
242                     //Ultima linea del request
243                     int ult = reqLineas.length - 1;
244
245                     // Bandera GET = 0, POST = 1
246                     String respuesta = obtenerParametros(reqLineas[ult],
                         ↪ headers, 1);
247
248                     // Respuesta POST, devolvemos un HTML con los parametros
                         ↪ del formulario
249                     dos.write(respuesta.getBytes());
250                     dos.flush();
251                     System.out.println("Respuesta POST: \n" + respuesta);
252                 } // Metodo DELETE
253                 else if(line.toUpperCase().startsWith("DELETE")) {
254                     String fileName = obtenerNombreRecurso(line);
255                     eliminarRecurso(fileName, headers);
256                 }
257                 else {
258                  //Metodos no implementados en el servidor
259                     String error501 = "HTTP/1.1 501 Not Implemented\n" +
260                           "Date: " + new Date() + " \n" +
261                       "Server: EnrikeAbi Server/1.0 \n" +
262                       "Content-Type: text/html \n\n" +
263
```

```
264                     "<html><head><meta charset='UTF-8'><title>Error
                         ↪  501</title></head>" +
265                     "<body><h1>Error 501: No implementado.</h1>" +
266                     "<p>El método HTTP o funcionalidad solicitada no
                         ↪  está implementada en el servidor.</p>" +
267                     "</body></html>";
268
269                        dos.write(error501.getBytes());
270                        dos.flush();
271                        System.out.println("Respuesta ERROR 501: \n" +
                            ↪  error501);
272                     }
273                 }
274                 dis.close();
275                 dos.close();
276                 cl.close();
277             }
278             catch(Exception e) {
279                 e.printStackTrace();
280             }
281         }
282 }
```

2.3. Mime

Esta clase permite agregar el nombre y extensión de archivos que serán mostrados en la página de recursos.

```
1  import java.util.*;
2
3  public class Mime {
4   public static HashMap<String, String> mimeTypes;
5
6   public Mime() {
7    mimeTypes = new HashMap<>();
8    mimeTypes.put("doc", "application/msword");
9    mimeTypes.put("pdf", "application/pdf");
10   mimeTypes.put("rar", "application/x-rar-compressed");
11   mimeTypes.put("mp3", "audio/mpeg");
12   mimeTypes.put("jpg", "image/jpeg");
13   mimeTypes.put("jpeg", "image/jpeg");
14   mimeTypes.put("png", "image/png");
15   mimeTypes.put("html", "text/html");
16   mimeTypes.put("htm", "text/html");
```

```
17    mimeTypes.put("c", "text/plain");
18    mimeTypes.put("txt", "text/plain");
19    mimeTypes.put("java", "text/plain");
20    mimeTypes.put("mp4", "video/mp4");
21   }
22
23   public String get(String extension) {
24    if(mimeTypes.containsKey(extension))
25     return mimeTypes.get(extension);
26    else
27     return "application/octet-stream";
28   }
29
30  }
```

3. Pruebas

```
C:\Users\UnADM\Documents\ESCUELA\GitHub\Servicios-en-Red\Practica4>java ServidorWeb
Puerto: 5000
Tamanio del pool de conexiones: 3


 -----> Iniciando Servidor.... Pool de Conexiones = 3
Servidor iniciado: http://localhost:5000/ --- OK
Esperando a Cliente....
```

Figura 1: Se ingresa el tamaño de conexiones

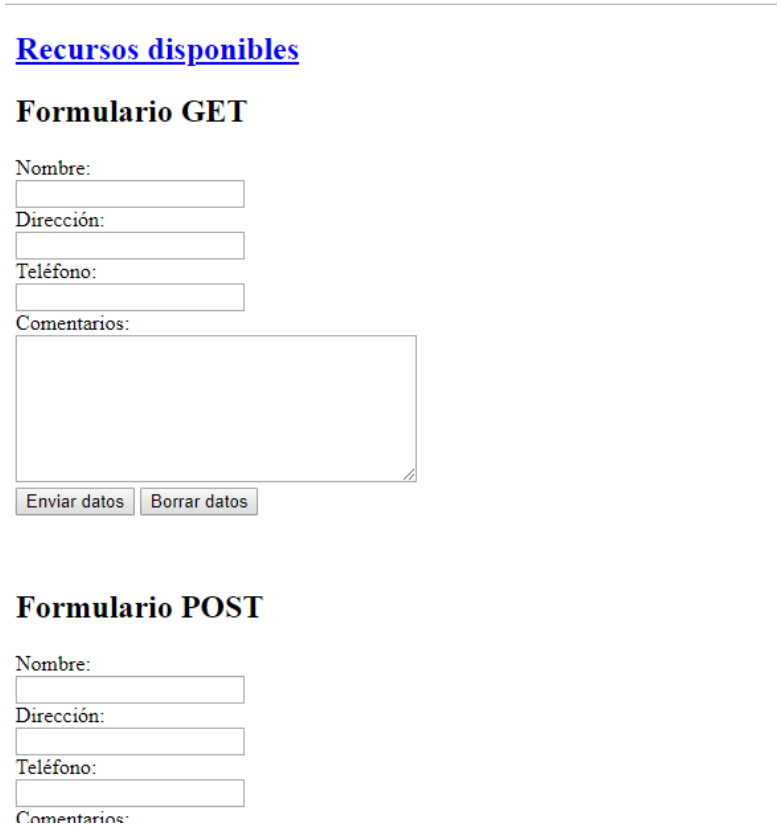

Figura 2: Página para probar métodos POST y GET

3.1. GET

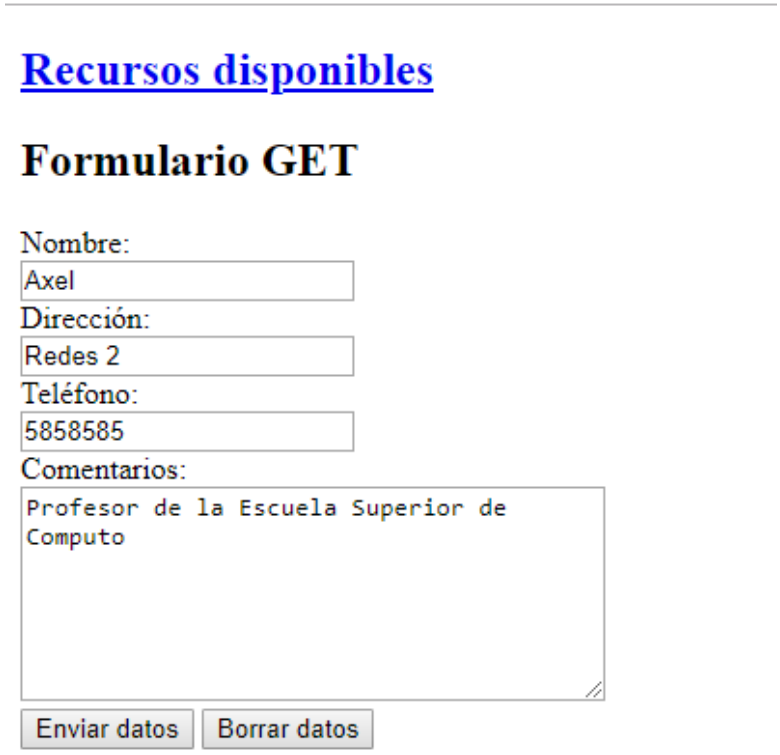

Figura 3: Ingreso de datos para prueba GET

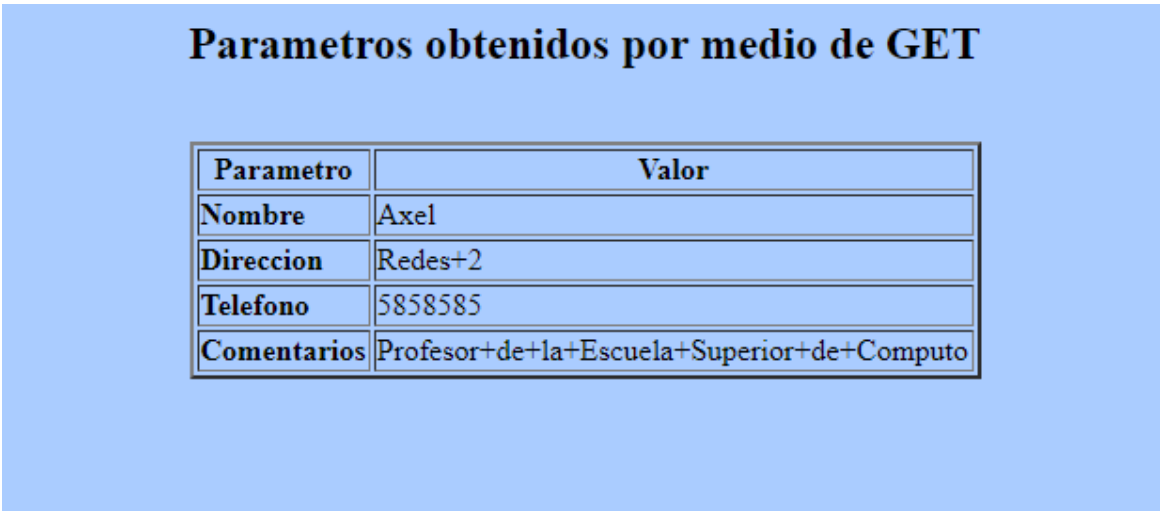

Figura 4: Respuesta del servidor

3.2. POST

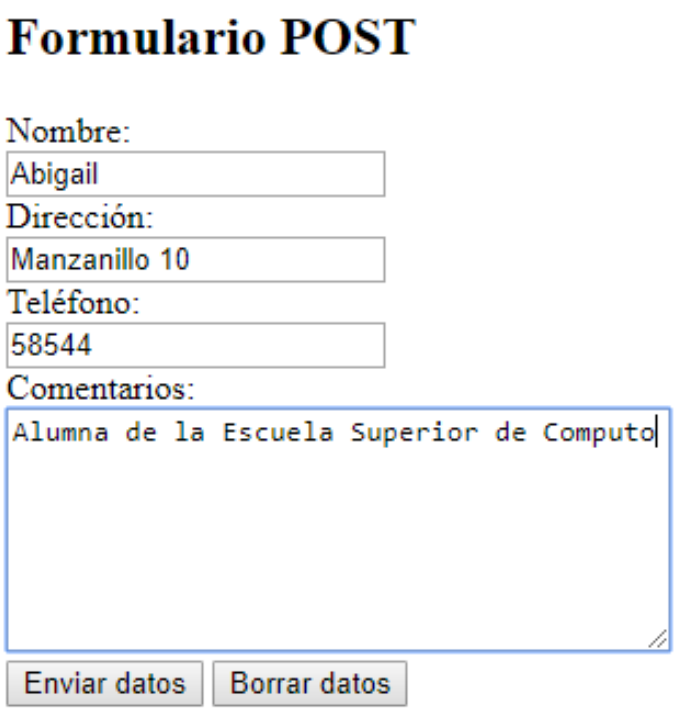

Figura 5: Formulario POST

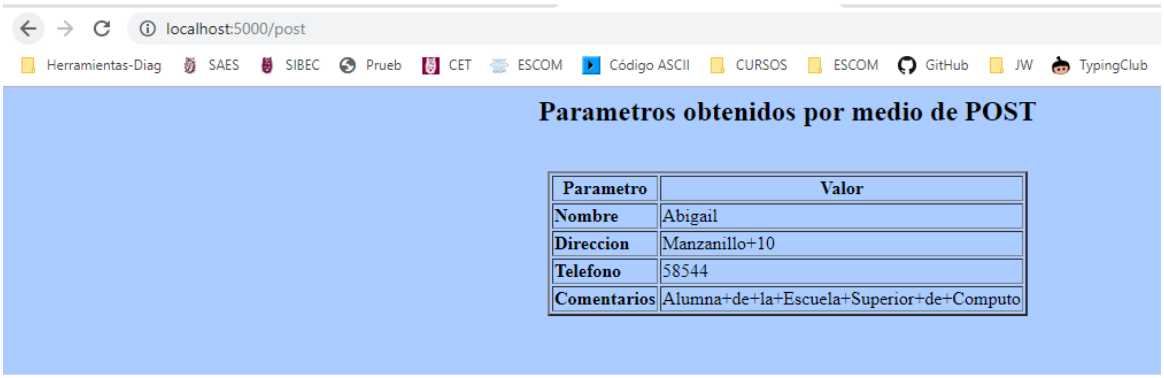

Figura 6: Respuesta del servidor

3.3. DELETE

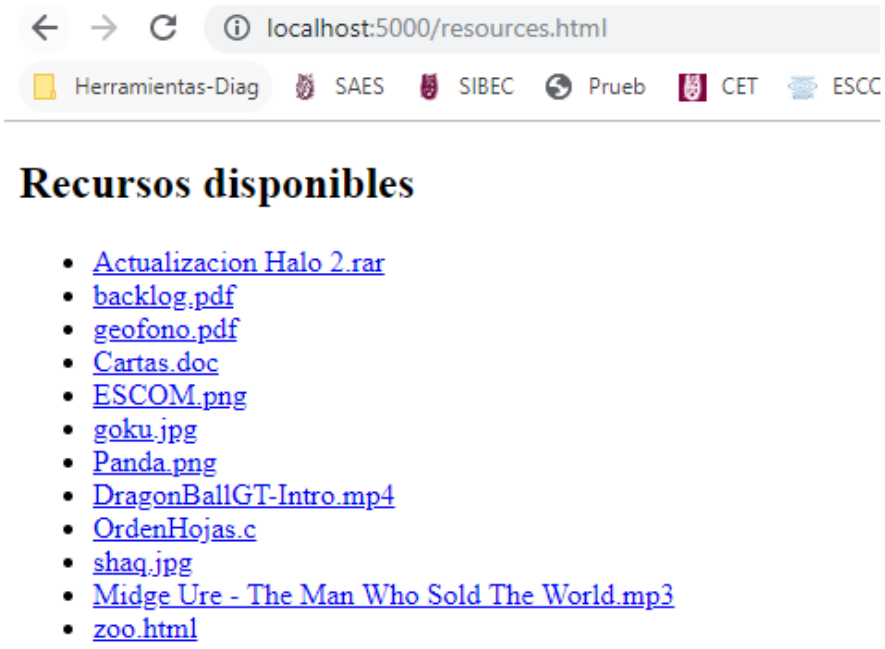

Figura 7: Recursos disponibles en el servidor

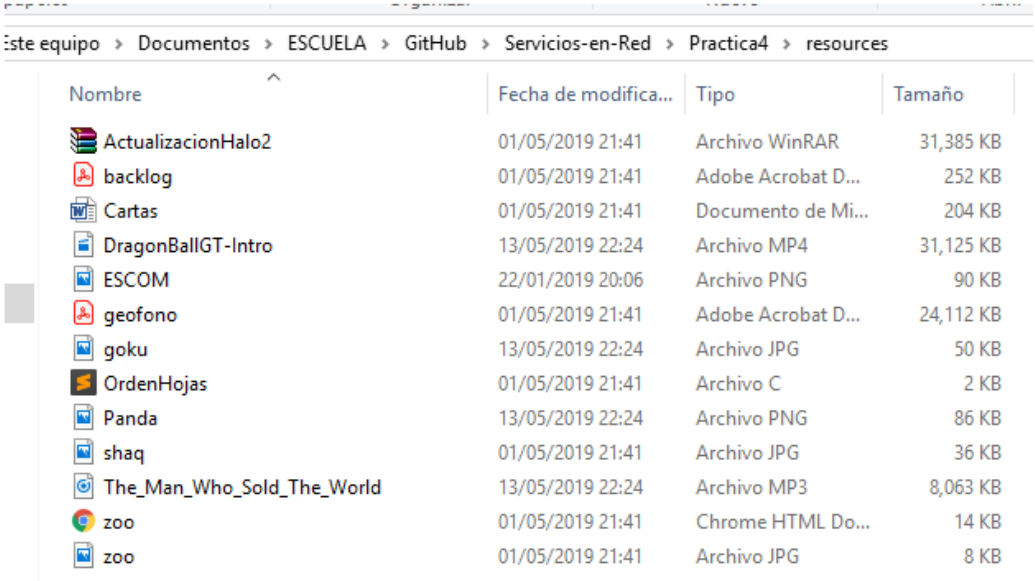

Figura 8: Archivos que existen en la carpeta de servidor

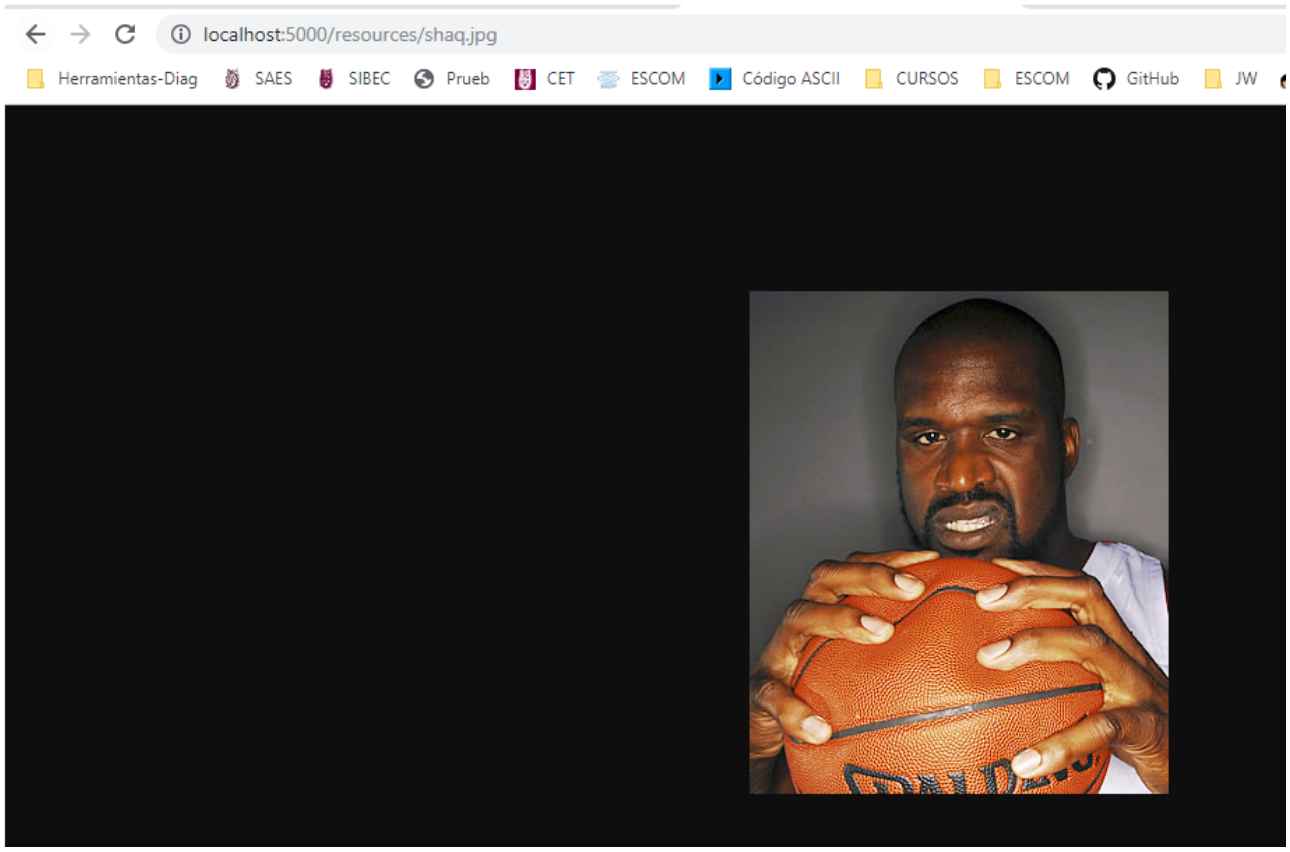

Figura 9: Imagen que existe en el servidor

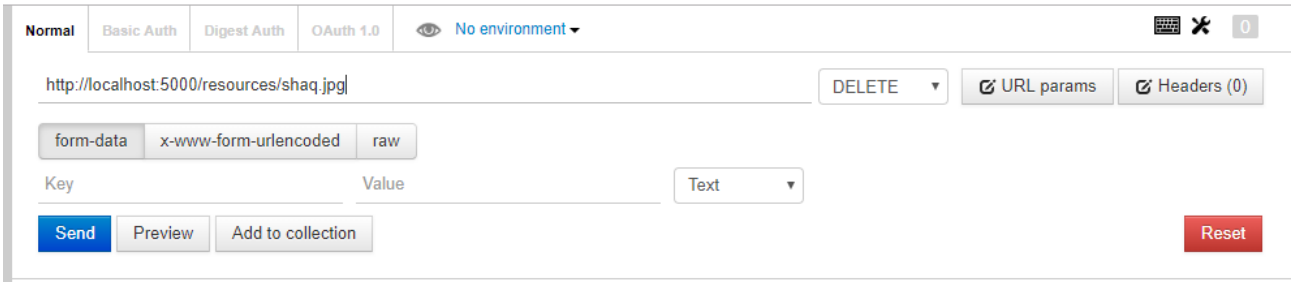

Figura 10: Uso de POSTMAN, agregando la url para eliminar

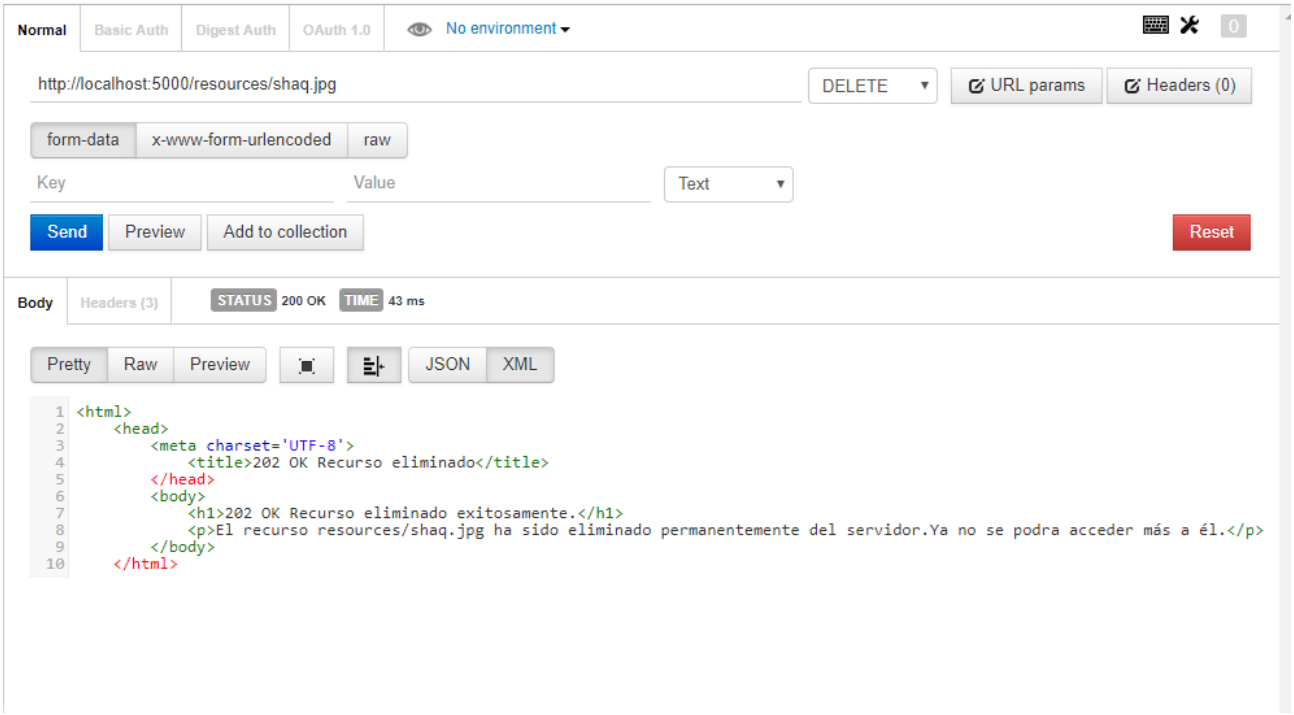

Figura 11: Muestra respuesta del servidor

Figura 12: Mensaje de error enviado por el servidor

4. Conclusiones

4.1. Nicolás Sayago Abigail

El aprendizaje adquirido en esta práctica no fue únicamente sobre sockets, también aprendimos a usar una tecnología WEB que nos ayudará para próximos proyectos. Existen implementaciones que después de haberlas hechas pensamos que trabajamos doble. Aprendimos a poner emojis, crear conversaciones privadas y públicas.

4.2. Ramos Diaz Enrique

En ésta práctica decidimos probar una nueva tecnología distinta a la que veníamos trabajando en prácticas anteriores. Con ayuda de frameworks y bibliotecas, el envío, recibo y reenvío por parte del servidor de mensajes de texto, archivos, emojis, etc. se nos facilitó de forma considerable. Tuvimos ciertas complicaciones en separar los canales públicos y privados, ya que el manejo y creación de los ID de los sockets no funciona de una forma como nosotros la imaginamos en un principio. Para facilidad de uso y comodidad, subimos la aplicación a un servidor, para así poder acceder al chat comunal en cualquier momento, desde cualquier dispositivo que soporte navegadores web.