INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

Materia: *Aplicaciones para Comunicaciones en Red*

Grupo: *3CM13*

Nombre del Trabajo:

Reporte de la Práctica 6. Server Con sockets no bloqueantes

Alumnos:

Servín Martinez Victor Saul

Velasco Huerta Angel Eduardo

Profesor: *Moreno Cervantes Axel*

Introducción

Los sockets no bloqueantes son una característica de los sockets de red que permite que un proceso realice otras tareas mientras espera la respuesta de un socket. Con sockets bloqueantes, un proceso se queda bloqueado hasta que se recibe una respuesta o se produce un error. Sin embargo, con sockets no bloqueantes, un proceso puede continuar su ejecución mientras espera una respuesta del socket. Esto permite al proceso realizar otras tareas mientras espera la respuesta, lo que mejora la eficiencia y escalabilidad del sistema.

Los sockets bloqueantes son una forma de comunicación en redes de computadoras en la que un proceso se bloquea hasta que recibe una respuesta de otro proceso. Esto significa que mientras un proceso está esperando una respuesta, no puede hacer nada más.

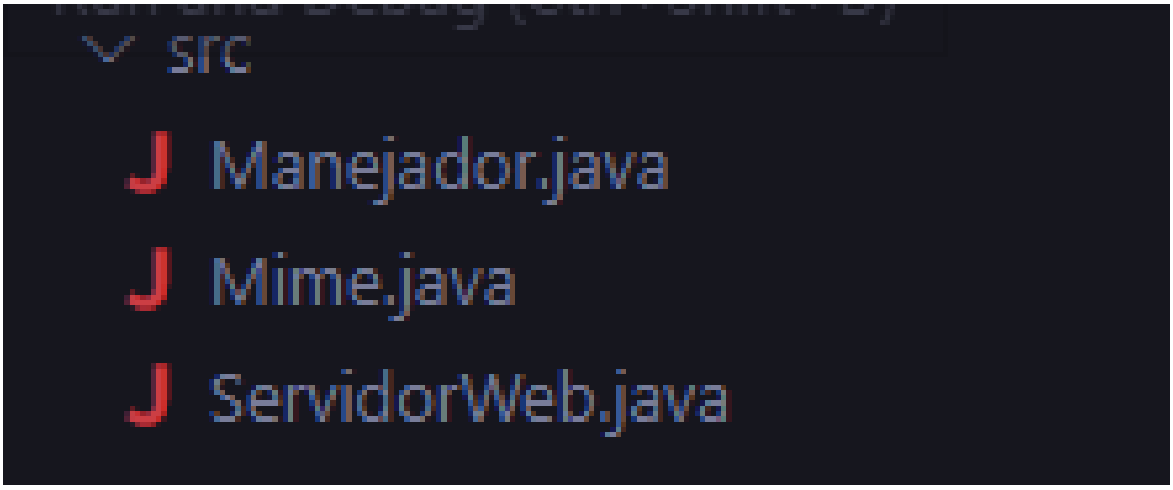
Un beneficio de los sockets bloqueantes es que son fáciles de usar y entender. El flujo de control es simple y predecible, lo que facilita el desarrollo y la depuración de aplicaciones de red. Además, ya que un proceso solo puede realizar una acción a la vez, es menos probable que ocurran errores de sincronización.

Sin embargo, una desventaja importante de los sockets bloqueantes es que pueden causar cuellos de botella en la aplicación. Si un proceso está esperando una respuesta de otro proceso, no puede realizar ninguna otra acción, lo que puede causar un rendimiento deficiente. Además, los sockets bloqueantes son menos escalables que los sockets no bloqueantes, ya que una gran cantidad de procesos bloqueados puede causar sobrecarga en el sistema.

En resumen, los sockets bloqueantes son una forma fácil de comunicación en redes de computadoras, pero también tienen desventajas importantes en términos de rendimiento y escalabilidad. En aplicaciones con requerimientos de rendimiento y escalabilidad altos, se recomienda el uso de sockets no bloqueantes.

Desarrollo

El proyecto cuenta con 3 archivos principales, uno para el manejo de los MIME, otro que es el manejador de las peticiones, y el ejecutable que inicializa el servidor y los métodos para recibir y atender conexiones.



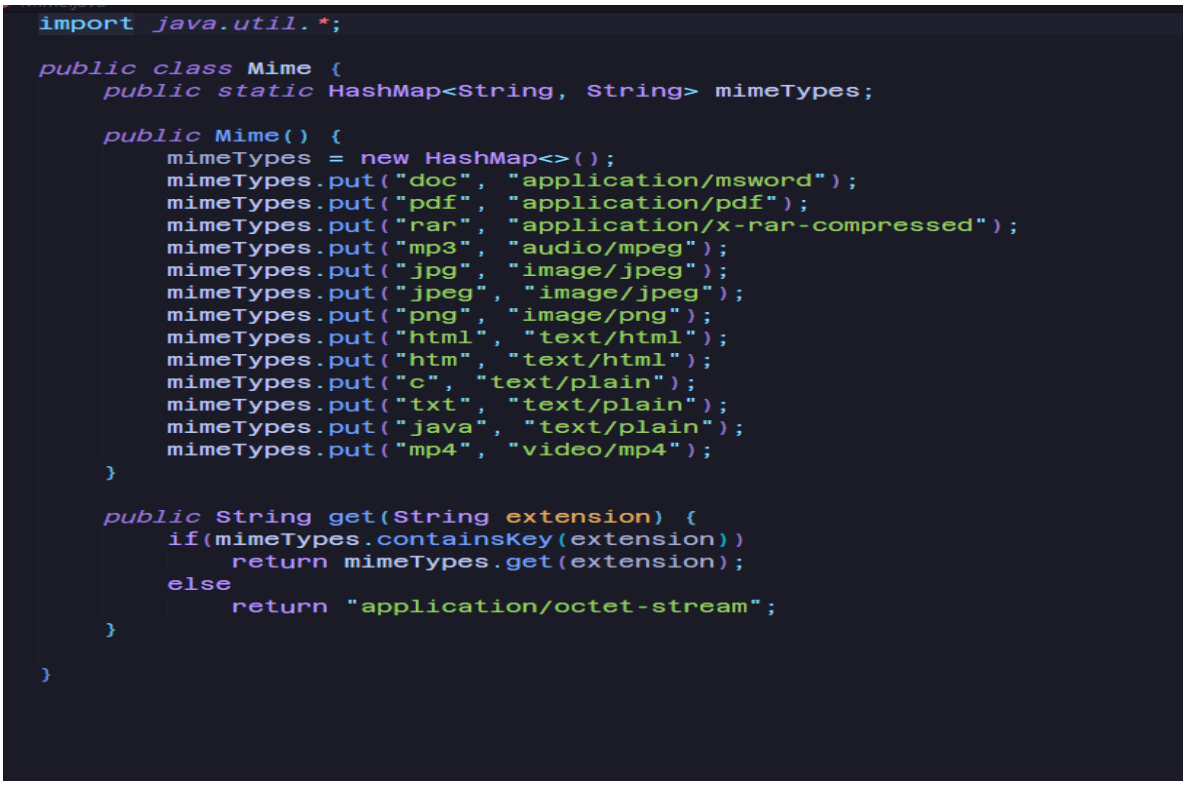
El server:

```java
import java.net.*;
import java.io.*;
import java.util.*;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class ServidorWeb {

    public static void main(String[] args) {
        int pto, tamPool;

        try {
            Scanner sc = new Scanner(System.in);
            System.out.print("Puerto: ");
            pto =  8080;
            tamPool = 5;

            // Pool de Conexiones
            ExecutorService pool = Executors.newFixedThreadPool(tamPool);
            System.out.println("\n\n Iniciando Servidor....");

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

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

El manejador (solo una parte):

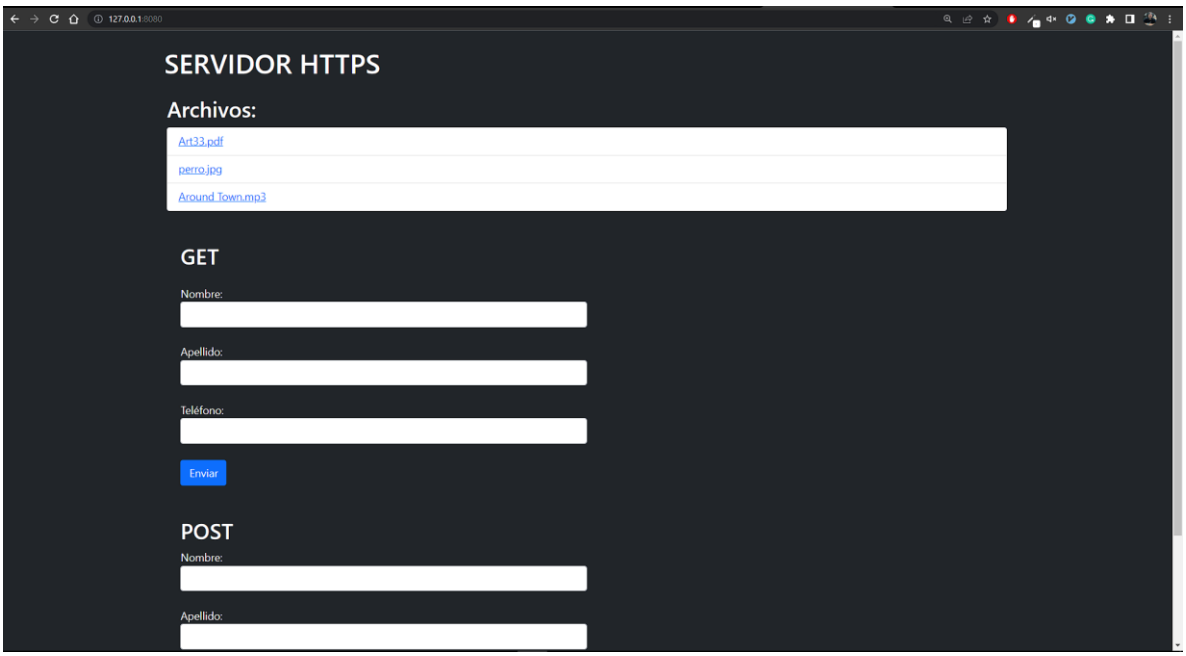
```java
src > J Manejador.java

    public void eliminarRecurso(String arg, String headers) {
        try {
            System.out.println(arg);
            File f = new File(arg);

            if (f.exists()) {
                if (f.delete()) {
                    System.out.println("------> Archivo " + arg + " eliminado exitosamente\n");

                    String deleteOK = headers
                            + "<html><head><meta charset='UTF-8'><title>202 OK Recurso eliminado</title></head
                            + "<body><h1>202 OK Recurso eliminado exitosamente.</h1>"
                            + "<p>El recurso " + arg + " ha sido eliminado permanentemente del servidor."
                            + "Ya no se podra acceder más a él.</p>"
                            + "</body></html>";

                    dos.write(deleteOK.getBytes());
                    dos.flush();
                    System.out.println("Respuesta DELETE: \n" + deleteOK);
                } else {
                    System.out.println("El archivo " + arg + " no pudo ser borrado\n");

                    String error404 = "HTTP/1.1 404 Not Found\n"
                            + "Date: " + new Date() + " \n"
                            + "Server: EnrikeAbi Server/1.0 \n"
                            + "Content-Type: text/html \n\n"
                            + "<html><head><meta charset='UTF-8'><title>404 Not found</title></head>"
                            + "<body><h1>404 Not found</h1>"
                            + "<p>Archivo " + arg + " no encontrado.</p>"
                            + "</body></html>";

                    dos.write(error404.getBytes());
                    dos.flush();
                    System.out.println("Respuesta DELETE - ERROR 404: \n" + error404);
                }
            }
        } catch (Exception e) {
            System.out.println(e.getMessage());
```

Los tipos MIME:

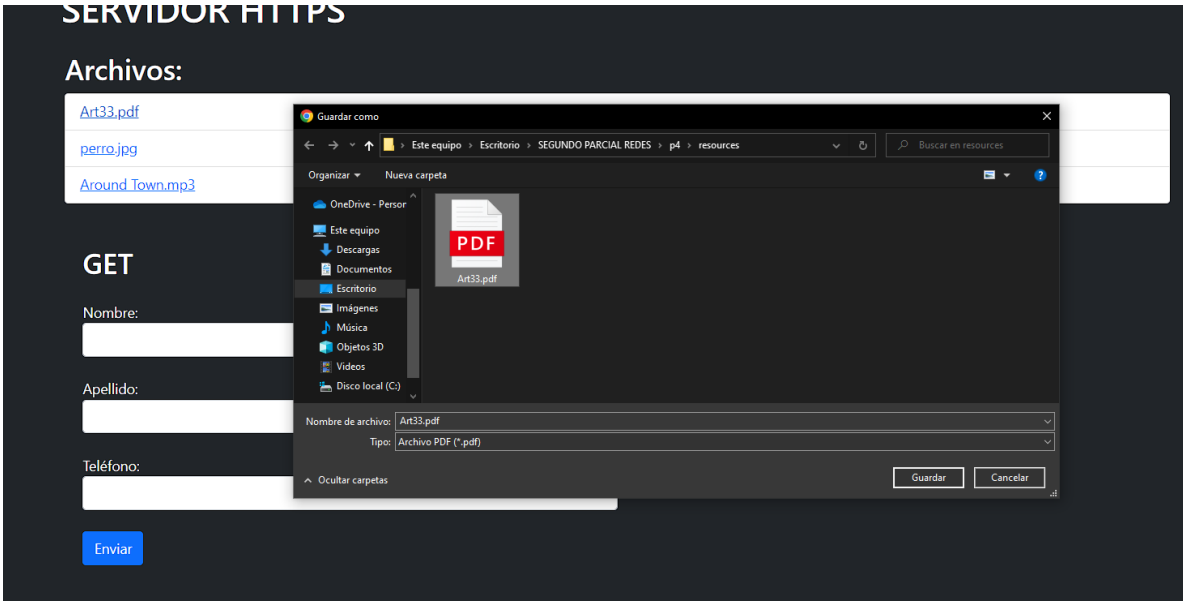


```java
import java.util.*;

public class Mime {
    public static HashMap<String, String> mimeTypes;

    public Mime() {
        mimeTypes = new HashMap<>();
        mimeTypes.put("doc", "application/msword");
        mimeTypes.put("pdf", "application/pdf");
        mimeTypes.put("rar", "application/x-rar-compressed");
        mimeTypes.put("mp3", "audio/mpeg");
        mimeTypes.put("jpg", "image/jpeg");
        mimeTypes.put("jpeg", "image/jpeg");
        mimeTypes.put("png", "image/png");
        mimeTypes.put("html", "text/html");
        mimeTypes.put("htm", "text/html");
        mimeTypes.put("c", "text/plain");
        mimeTypes.put("txt", "text/plain");
        mimeTypes.put("java", "text/plain");
        mimeTypes.put("mp4", "video/mp4");
    }

    public String get(String extension) {
        if(mimeTypes.containsKey(extension))
            return mimeTypes.get(extension);
        else
            return "application/octet-stream";
    }

}
```

Al ejecutar, e ir a la dirección http://127.0.0.1:8080, accederemos a lo siguiente:



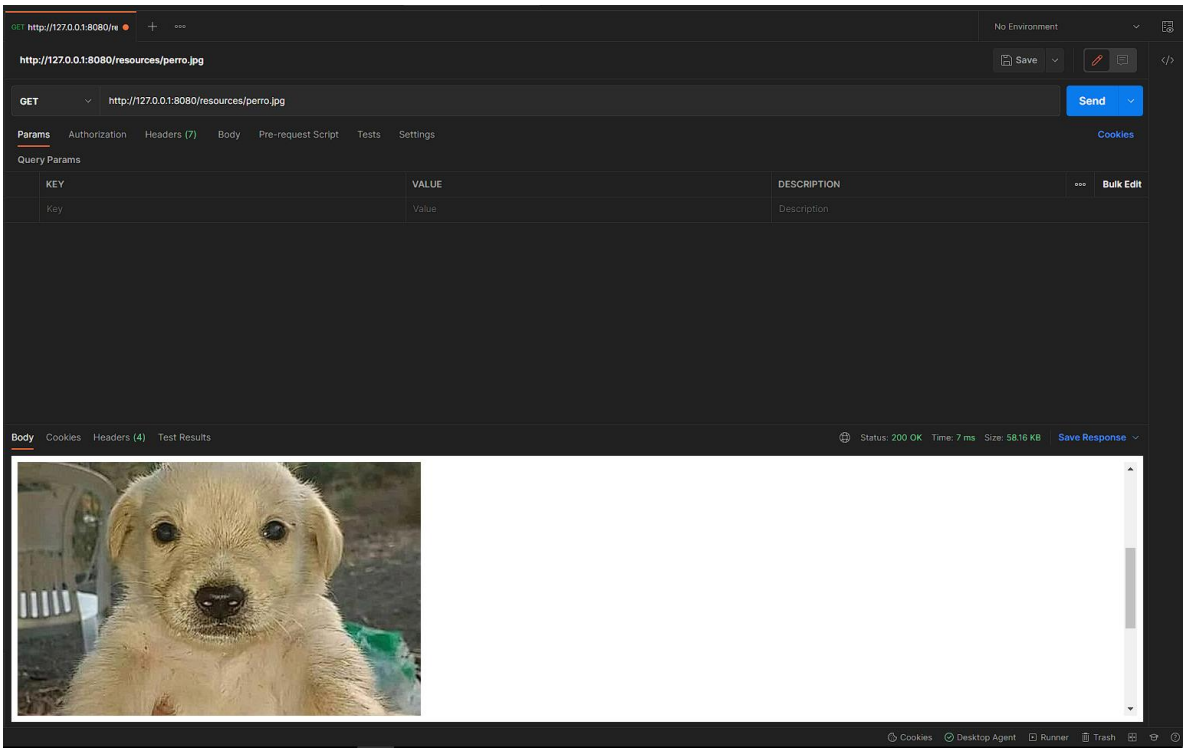
Aquí mismo, podremos descargar los archivos, y enviar peticiones GET y POST



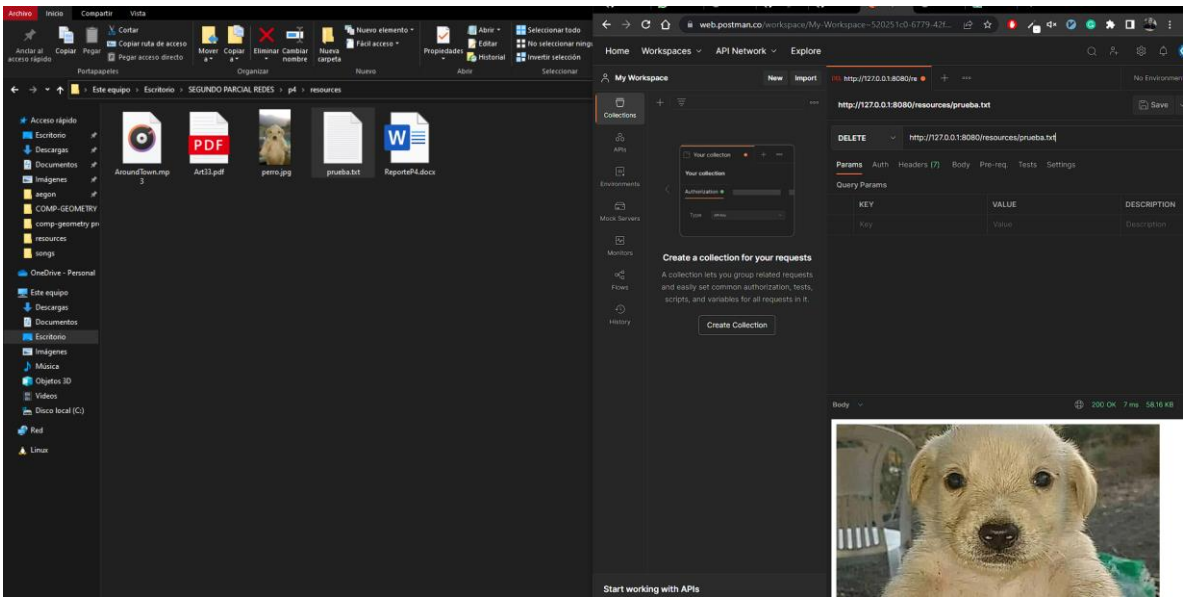
Descarga de Archivo ART33.PDF

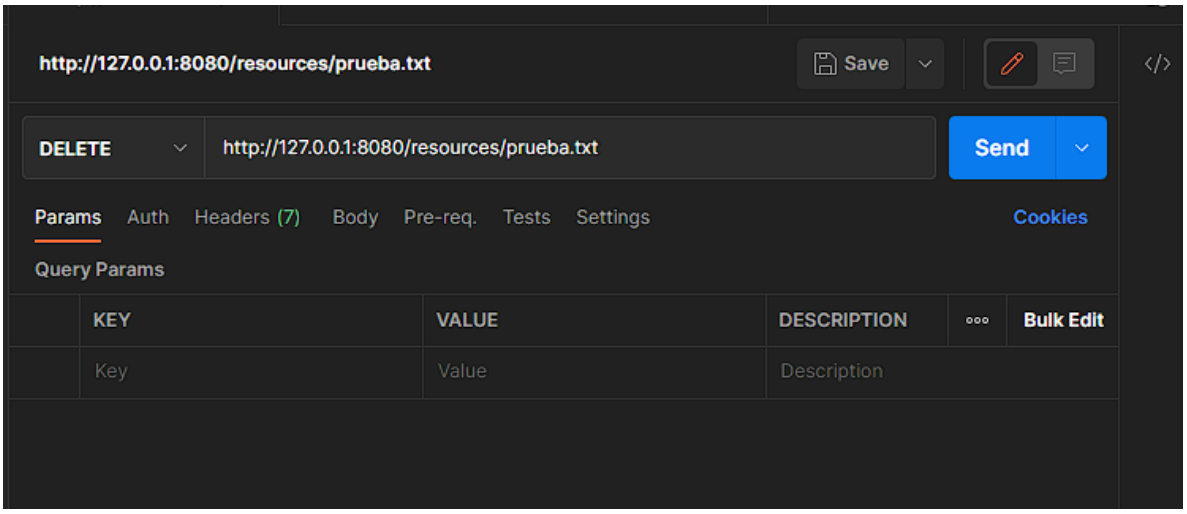
Método GET

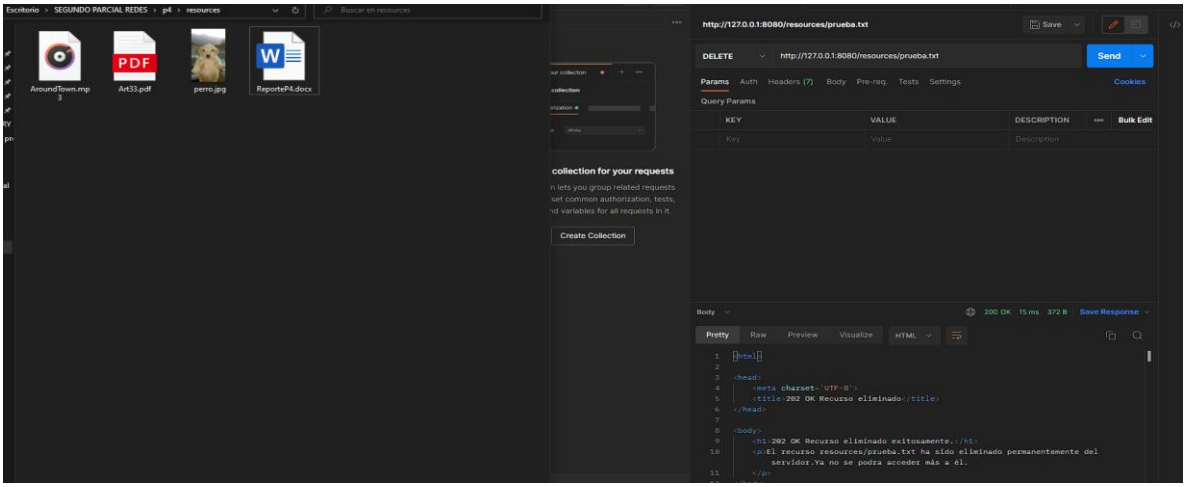
Petición de una Imagen en POSTMAN



Y para probar DELETE, iremos a POSTMAN:







Conclusiones

En conclusión, la realización de una práctica de programación de sockets para hacer servidores HTTP nos permite aprender los siguientes aspectos:

- Cómo utilizar sockets para implementar un servidor HTTP.
- La capacidad de manejar diferentes peticiones HTTP y generar respuestas apropiadas.
- La habilidad para interpretar y procesar las cabeceras HTTP en las peticiones y respuestas.
- La posibilidad de utilizar programación de sockets para crear un servidor HTTP más escalable y eficiente en comparación con los servidores web tradicionales.
- La habilidad para probar y depurar problemas relacionados con la programación de sockets y la implementación del servidor HTTP.

En general, la práctica de programación de sockets para hacer servidores HTTP nos permite comprender mejor cómo funcionan los servidores web y cómo se pueden utilizar los sockets para crear un servidor HTTP personalizado y adaptado a las necesidades específicas del proyecto. También nos permite tener un mejor control sobre el rendimiento y escalabilidad del servidor HTTP.

Referencias

[1] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1," RFC 2616, IETF, June 1999.

[2] A. Luotonen, "Web proxy servers," IEEE Internet Computing, vol. 2, no. 1, pp. 64-71, Jan-Feb 1998.

[3] J. Mogul and P. Leach, "HTTP/1.1, part 1: URIs, Connections, and Message Parsing," RFC 2616, IETF, June 1999.

[4] J. Reschke, "The 'Basic' HTTP Authentication Scheme," RFC 7617, IETF, September 2015.

[5] R. Fielding and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content," RFC 7231, IETF, June 2014.