INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

Materia: *Aplicaciones para Comunicaciones en Red*

Grupo: *3CM13*

Nombre del Trabajo:

Reporte de la Práctica 4. Servidor HTTP

Alumnos:

Servín Martinez Victor Saul

Velasco Huerta Angel Eduardo

Profesor: *Moreno Cervantes Axel*

Introducción

Servidor Web

Un servidor web es un programa o dispositivo que se encarga de alojar y servir páginas web. Cuando un usuario ingresa a una dirección URL en su navegador, este realiza una petición al servidor web para que le envíe el contenido de la página correspondiente. El servidor web, a su vez, busca la página en cuestión en su almacenamiento y la envía al navegador del usuario para que éste la pueda visualizar.

Existen diferentes tipos de servidores web, como Apache, Nginx, IIS, entre otros. Cada uno de ellos tiene sus propias características y ventajas. Por ejemplo, Apache es uno de los servidores web más populares y utilizados en todo el mundo, gracias a su estabilidad y facilidad de uso. Por otro lado, Nginx es conocido por su alta capacidad de manejo de tráfico y su eficiencia en la gestión de peticiones concurrentes.

Además de alojar y servir páginas web, un servidor web también puede ejecutar scripts y aplicaciones, como PHP o Python, para generar dinámicamente el contenido de las páginas. Esto es especialmente útil para sitios web con un gran volumen de tráfico o que requieren una interacción en tiempo real con los usuarios.

En resumen, un servidor web es un programa o dispositivo que se encarga de alojar y servir páginas web a los usuarios de Internet. Es un elemento esencial en la infraestructura de cualquier sitio web y permite a los usuarios acceder al contenido en línea de manera rápida y fiable.

HTTP:

HTTP (Hypertext Transfer Protocol) es un protocolo de red utilizado para transferir información en la World Wide Web. Es el protocolo que permite la comunicación entre un servidor web y un navegador web, y es el fundamento de la navegación en Internet.

HTTP es un protocolo cliente-servidor, lo que significa que un cliente, como un navegador web, envía una solicitud al servidor, el cual responde con la información solicitada. La solicitud se llama "petición HTTP" y la respuesta se llama "respuesta HTTP".

Las peticiones HTTP pueden ser de varios tipos, como GET, POST, PUT, DELETE, entre otros. El tipo más común es GET, que se utiliza para solicitar información de un recurso específico, como una página web o una imagen. El tipo POST se utiliza para enviar información al servidor, como cuando se envía un formulario en un sitio web.

HTTP también utiliza un sistema de "cabeceras" para transmitir información adicional sobre la petición o la respuesta. Estas cabeceras pueden incluir información sobre el tipo de contenido, la codificación de caracteres, las cookies, entre otros.

En resumen, HTTP es un protocolo de red que permite la transferencia de información en la World Wide Web. Es el protocolo que permite la comunicación entre un servidor web y un navegador web, y es el fundamento de la navegación en Internet. Utiliza un sistema de peticiones y respuestas y

cabeceras para transmitir información adicional, y es esencial para la funcionalidad de cualquier sitio web.

Métodos HTTP más comunes

HTTP define un conjunto de métodos de petición para indicar la acción que se desea realizar para un recurso determinado. Aunque también pueden ser sustantivos, estos métodos de solicitud a veces son llamados verbos HTTP. Cada uno de ellos implementan una semántica diferente, pero algunas características son similares compartidas por un grupo de ellos: ej. Un método de solicitud puede ser seguro, idempotente (en-US) o almacenable en caché .

GET
El método GET solicita una representación de un recurso especifico. Las peticiones que usen el método GET sólo deben recuperar datos.

HEAD
El método HEAD pide una respuesta idéntica a la de una petición GET, pero sin el cuerpo de la respuesta.

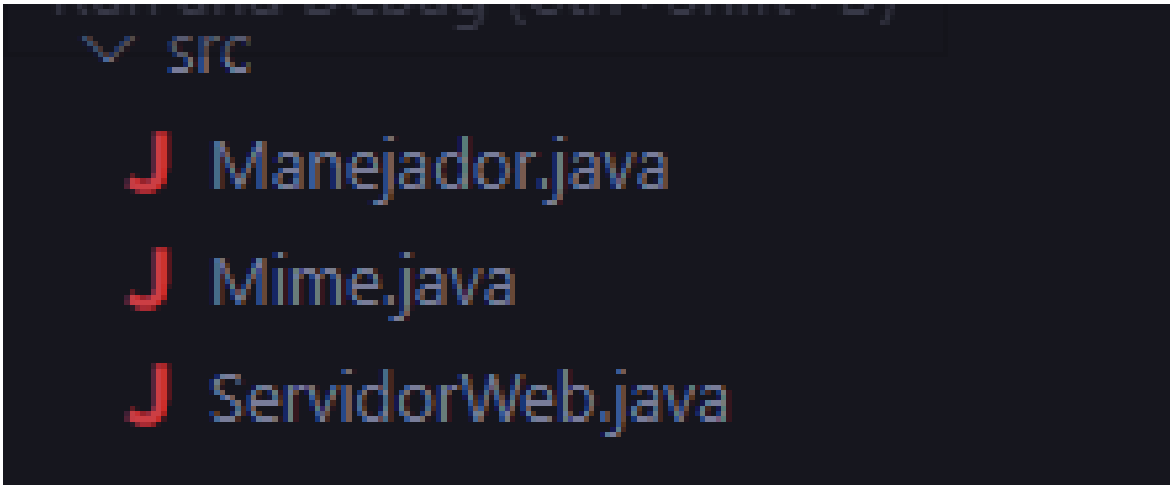
POST
El método POST se utiliza para enviar una entidad a un recurso en específico, causando a menudo un cambio en el estado o efectos secundarios en el servidor.

PUT
El modo PUT reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la petición.

DELETE
El método DELETE borra un recurso en específico.

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.



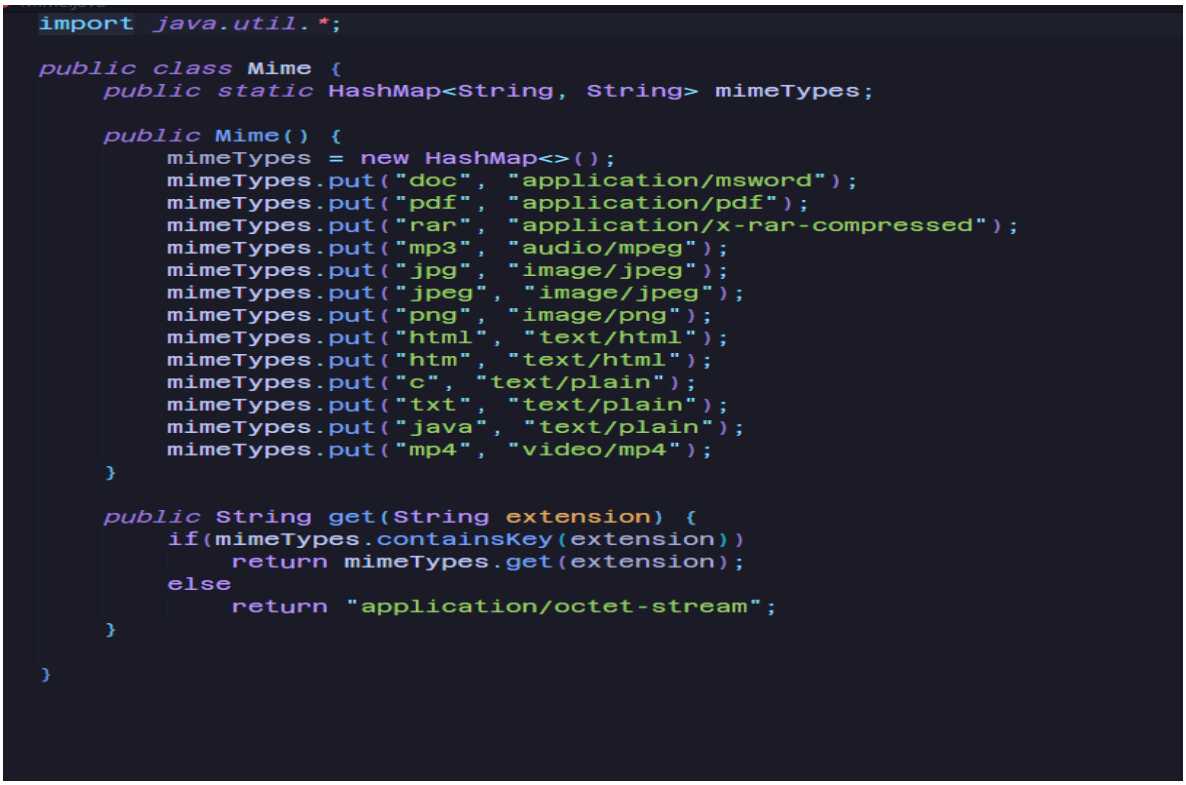
El server:

```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):

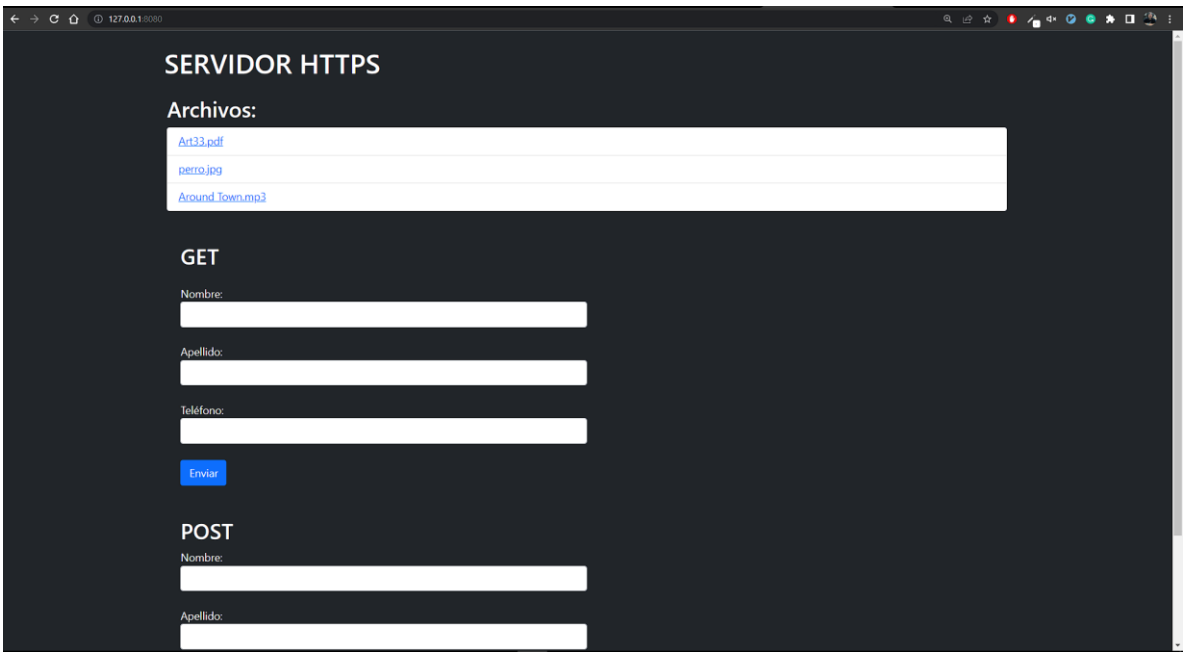
```java
src > J Manejador.java

    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:

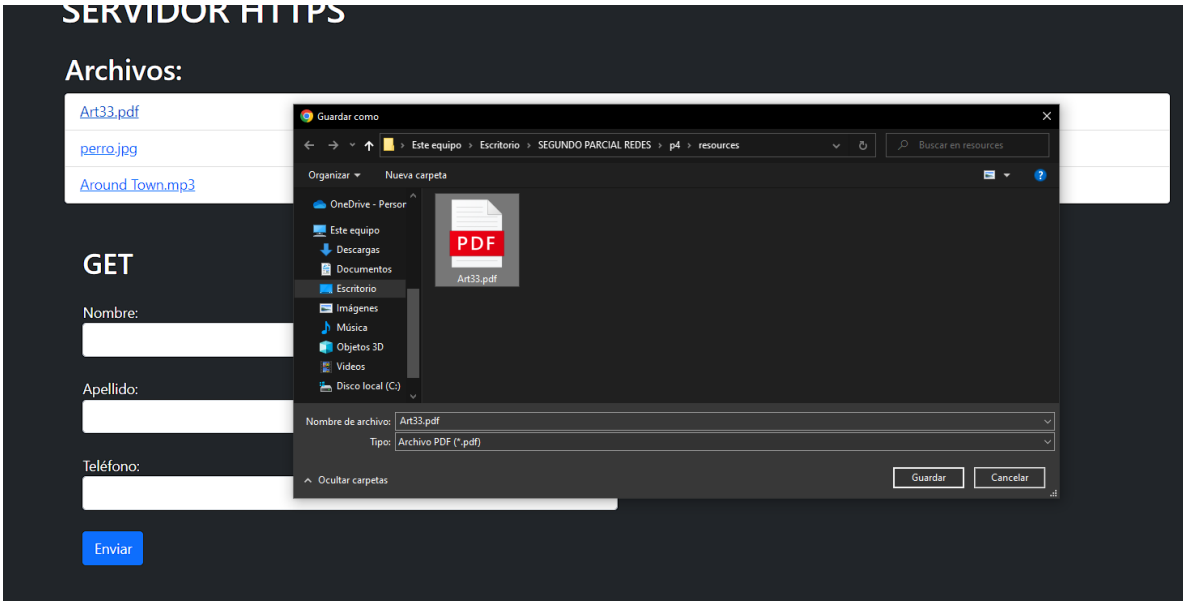


```java
import java.util.*;

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:



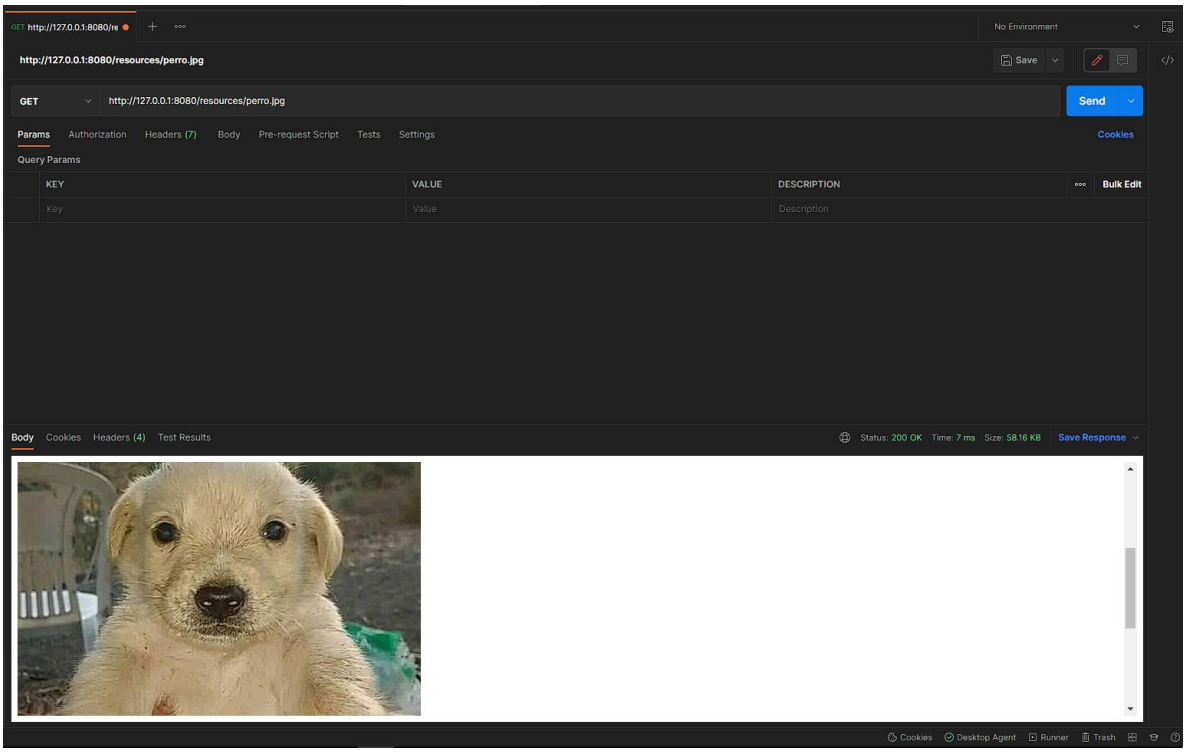
Aquí mismo, podremos descargar los archivos, y enviar peticiones GET y POST



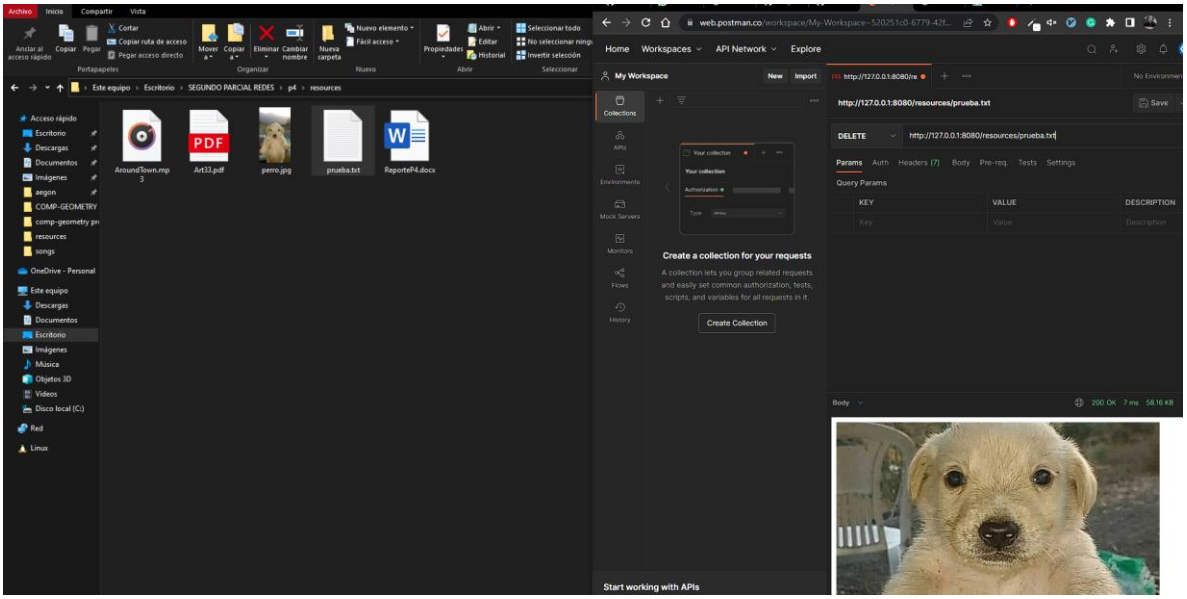
Descarga de Archivo ART33.PDF

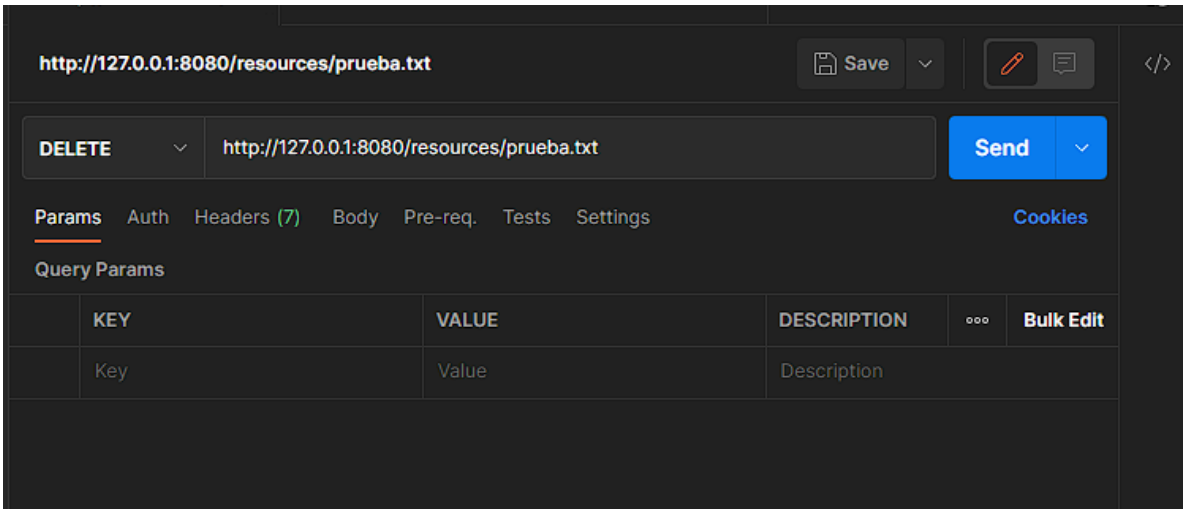
Método GET

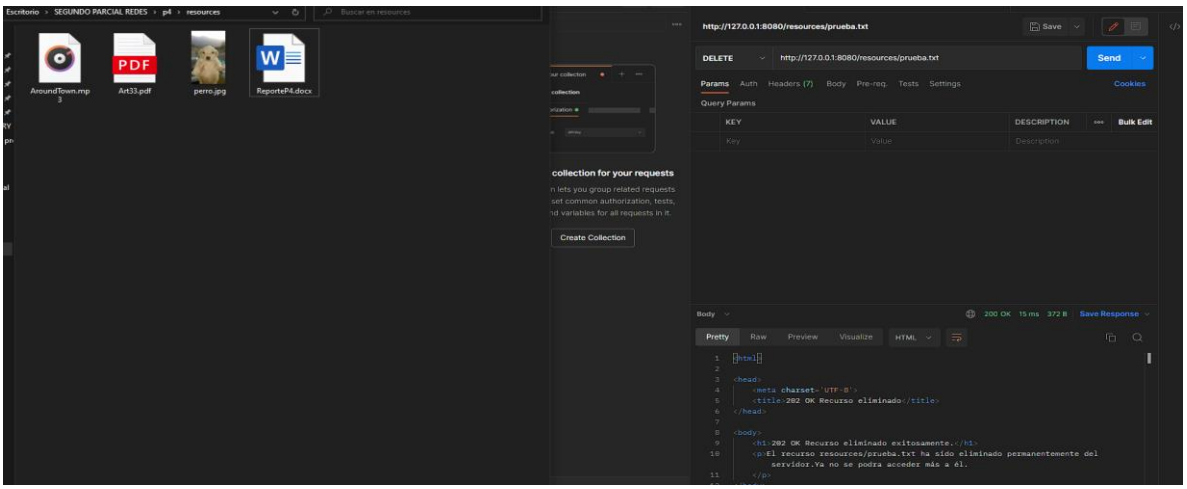
Petición de una Imagen en POSTMAN



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.