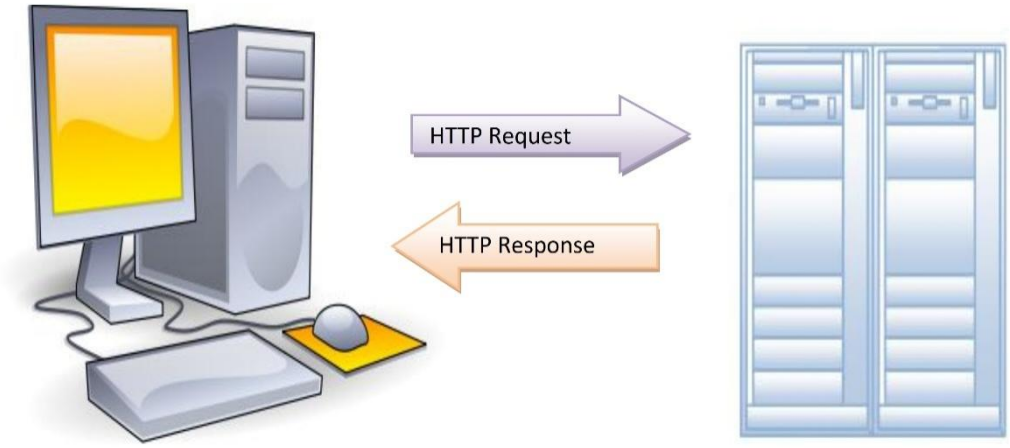
Instituto Politécnico Nacional

Escuela Superior de Computo

Practica 4

SERVIDOR HTTP



MATERIA

Aplicaciones para comunicaciones de red

PROFESOR

Moreno Cervantes Axel Ernesto

GRUPO

6CM2

ALUMNOS

- Guevara Badillo Areli Alejandra
- Ramírez Martínez Alejandro

FECHA

lunes, 2 de junio de 2025

INTRODUCCIÓN

En el contexto de las comunicaciones web, los servidores HTTP juegan un papel fundamental al gestionar solicitudes de los clientes y proporcionar respuestas adecuadas conforme al protocolo. Estas solicitudes pueden involucrar la obtención, creación, actualización o eliminación de recursos a través de métodos como GET, POST, PUT y DELETE. Para mejorar el rendimiento y la eficiencia del servidor, especialmente ante múltiples peticiones concurrentes, es común emplear técnicas como la utilización de albercas de hilos (thread pools). Esta práctica se centra en la creación de un servidor HTTP capaz de manejar diferentes tipos de solicitudes, respondiendo de manera adecuada según el código de estado correspondiente.

La implementación de este servidor permitirá comprender el comportamiento de un servicio web básico, analizar cómo se gestionan los recursos, y estudiar las respuestas ante distintas situaciones, como accesos restringidos o solicitudes a recursos inexistentes.

OBJETIVOS

- Implementar un servidor HTTP capaz de procesar solicitudes mediante los métodos GET, POST, PUT y DELETE.
- Utilizar una alberca de hilos para gestionar múltiples peticiones de forma concurrente.
- Simular el manejo de archivos como respuesta a las solicitudes de los clientes.
- Generar respuestas apropiadas utilizando códigos de estado HTTP según el resultado de la solicitud.
- Evaluar el comportamiento del servidor ante peticiones exitosas, errores por recursos inexistentes, métodos no implementados o accesos no permitidos.

DESARROLLO

Esta práctica implementa un servidor web básico en Java que soporta los métodos HTTP principales (GET, HEAD, POST, PUT, DELETE) y maneja conexiones concurrentes usando un pool de hilos. La práctica está conformada por tres componentes principales:

- **ServidorWeb.java:** Es el núcleo principal del servidor HTTP que inicia el servicio en un puerto específico, configurable por el usuario. Utiliza un pool de hilos fijo (con tamaño definido por entrada) para manejar múltiples conexiones concurrentes de manera eficiente. Cuando un cliente se conecta, el servidor acepta la solicitud mediante un ServerSocket, delega el procesamiento a una instancia de Manejador (que implementa Runnable) y la ejecuta en un hilo del pool, permitiendo así atender varias peticiones simultáneamente sin bloquear la conexión principal. Su diseño sigue el modelo "thread-per-request", optimizando recursos mediante reutilización de hilos.
- **Manejador.java:** Es el componente que procesa cada conexión HTTP individual en un hilo separado. Se encarga de interpretar las solicitudes entrantes (GET, POST, PUT, DELETE, HEAD), gestionar el envío de archivos estáticos (con tipos MIME adecuados), procesar parámetros de formularios, manejar operaciones CRUD (crear, leer, actualizar y eliminar recursos), y generar respuestas HTTP personalizadas incluyendo páginas de error (404, 500, 501) con diseño HTML/CSS embebido. Además, implementa la lógica completa de comunicación cliente-servidor, desde la lectura de cabeceras hasta el streaming de archivos y el cierre controlado de la conexión.
- **Mime.java:** Actúa como un gestor de tipos MIME para el servidor web, proporcionando la correspondencia entre extensiones de archivos (como .jpg, .pdf, .html) y sus tipos MIME estándar (como image/jpeg, application/pdf, text/html). Implementa un mapa de hash (HashMap) que almacena estas asociaciones y ofrece un método get() para consultar el tipo MIME correspondiente a una extensión dada, devolviendo "application/octet-stream" como valor predeterminado cuando la extensión no está registrada. Esta clase permite que el servidor envíe las cabeceras HTTP "Content-Type" correctas en las respuestas, asegurando que los navegadores puedan interpretar adecuadamente los recursos servidos.

SERVIDORWEB.JAVA

La función principal de este script es actuar como el núcleo del servidor HTTP: inicia el servicio en un puerto configurable, gestiona un pool de hilos para conexiones concurrentes, y delega cada solicitud entrante a un Manejador. Su rol clave es aceptar conexiones (mediante ServerSocket) y distribuir su procesamiento de manera eficiente sin bloquear el hilo principal.

Declaración de Clase y Variables

Función:

- Define la clase principal del servidor.
- Declara variables para almacenar:
  - pto: Puerto donde escuchará el servidor.
  - tamPool: Tamaño del pool de hilos (número máximo de conexiones concurrentes).

```
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;
```

Entrada de Configuración (Usuario)

Función:

- Solicita al usuario que ingrese:
  1. El puerto donde se alojará el servidor (ej: 8080).
  2. El tamaño del pool de hilos (ej: 10 para 10 conexiones simultáneas).
- Usa Scanner para leer la entrada por consola.

```
try {
              Scanner sc = new Scanner(System.in);
              System.out.print("Puerto: ");
                     pto = sc.nextInt();
                     System.out.print("Tamanio del pool de conexiones: ");
                     tamPool = sc.nextInt();
```

Inicialización del Pool de Hilos

Función:

- Crea un pool de hilos fijo con Executors.newFixedThreadPool().
  - Cada hilo manejará una conexión cliente.
  - El tamaño del pool (tamPool) limita las conexiones concurrentes.
- Muestra un mensaje de inicio con la configuración.

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

Creación del Socket del Servidor

Función:

- Inicializa un ServerSocket en el puerto especificado (pto).
- El servidor queda en espera (s.accept()) de conexiones entrantes.
- Muestra la URL local donde está accesible el servidor (ej: http://localhost:8080/).

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

Bucle Principal (Atención de Conexiones)

Función:

- Bucle infinito (for(;;)) que acepta conexiones continuamente.
- **Paso a paso:**
    1. s.accept(): Bloquea el hilo hasta que un cliente se conecta. Retorna un Socket para comunicarse con ese cliente.
    2. Manejador manejador = new Manejador(cl): Crea un objeto Manejador (que implementa Runnable) pasándole el Socket del cliente.
    3. pool.execute(manejador): Asigna el Manejador a un hilo del pool para procesar la petición en segundo plano.
- Permite manejar múltiples clientes **en paralelo** gracias al pool de hilos.

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

Manejo de Excepciones

Función:

- Captura errores (ej: puerto en uso, entrada inválida) y muestra el stack trace.
- Asegura que el servidor no falle silenciosamente.

```
catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

MANEJADOR.JAVA

Procesa cada petición HTTP individual en un hilo independiente. Interpreta métodos (GET, POST, PUT, DELETE, HEAD), sirve archivos estáticos, gestiona parámetros de formularios, ejecuta operaciones CRUD (subir/eliminar archivos) y genera respuestas HTTP (incluyendo errores 404/500 con HTML estilizado). Además, maneja el ciclo completo de la comunicación: lectura de cabeceras, envío de datos y cierre seguro de la conexión.

Declaración de Clase y Atributos

Función:

- Extiende Thread para manejar cada conexión en un hilo independiente.
- Atributos clave:
  - cl: Socket para comunicación con el cliente.
  - dos/dis: Flujos de salida/entrada de datos.
  - mime: Instancia de Mime para gestionar tipos de archivo.

```
public class Manejador extends Thread {

    protected Socket cl;

    protected DataOutputStream dos;

    protected Mime mime;

    protected DataInputStream dis;
```

Constructor

Función:

- Inicializa los flujos de E/S (DataInputStream/DataOutputStream) para comunicarse con el cliente.
- Crea una instancia de Mime para resolver tipos de contenido.

```
public Manejador(Socket cl) throws Exception {
        this.cl = cl;
        this.dos = new DataOutputStream(this.cl.getOutputStream());
        this.mime = new Mime();
        this.dis = new DataInputStream(this.cl.getInputStream());
    }
```

Métodos de manejo de errores HTTP

Función:

- Genera respuestas HTML estilizadas para errores:
  - **404:** Recurso no encontrado (incluye la ruta solicitada).
  - **500:** Error interno del servidor.
- Envía las cabeceras HTTP y el cuerpo HTML al cliente.

```
private void enviarError404(String arg) throws IOException {
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"
       + "<!DOCTYPE html>\n"
       + "<html>\n"
       + "<head>\n"
       + "<meta charset='UTF-8'>\n"
       + "<title>404 Not Found</title>\n"
       + "<style>\n"
       + "body { font-family: Arial, sans-serif; background-color: #f5f5f5; margin: 0;
padding: 20px; }\n"
```

```
        + ".container { max-width: 800px; margin: 50px auto; background: white; padding:
30px; border-radius: 8px; box-shadow: 0 2px 10px rgba(0,0,0,0.1); }\n"
        + "h1 { color: #e74c3c; }\n"
        + "p { color: #555; line-height: 1.6; }\n"
        + ".file-path { background-color: #f0f0f0; padding: 8px; border-radius: 4px;
font-family: monospace; }\n"
        + "</style>\n"
        + "</head>\n"
        + "<body>\n"
        + "<div class='container'>\n"
        + "<h1>404 Not Found</h1>\n"
        + "<p>El archivo <span class='file-path'>" + arg + "</span> no existe en el
servidor.</p>\n"
        + "</div>\n"
        + "</body>\n"
        + "</html>";
dos.write(error404.getBytes());
dos.flush();}
private void enviarError500() throws IOException {
String error500 = "HTTP/1.1 500 Internal Server Error\n"
        + "Date: " + new Date() + " \n"
        + "Server: EnrikeAbi Server/1.0 \n"
        + "Content-Type: text/html \n\n"
        + "<!DOCTYPE html>\n"
        + "<html>\n"
        + "<head>\n"
        + "<meta charset='UTF-8'>\n"
        + "<title>500 Server Error</title>\n" + "<style>\n"
        + "body { font-family: Arial, sans-serif; background-color: #f5f5f5; margin: 0;
padding: 20px; }\n"
        + ".container { max-width: 800px; margin: 50px auto; background: white; padding:
30px; border-radius: 8px; box-shadow: 0 2px 10px rgba(0,0,0,0.1); }\n"
        + "h1 { color: #e74c3c; }\n"
        + "p { color: #555; line-height: 1.6; }\n"
        + "</style>\n"
        + "</head>\n"
        + "<body>\n"
        + "<div class='container'>\n"
        + "<h1>500 Internal Server Error</h1>\n"
        + "<p>Ocurrió un error inesperado en el servidor.</p>\n"
        + "</div>\n"
        + "</body>\n"
        + "</html>";
dos.write(error500.getBytes());
dos.flush();}
```

Métodos CRUD

Función:

- Elimina un archivo en resources/ y redirige con código **303** (confirmación de eliminación).
- Maneja casos fallidos (archivo no existe/no se puede borrar) con errores **404**.

```
public void eliminarRecurso(String arg, String headers) throws IOException {
	try {
	// Normalizar ruta (elimina /resources/ si ya está incluido)
	String filePath = arg.startsWith("resources/") ? arg : "resources/" + arg;
	System.out.println("Solicitud DELETE para: " + filePath);
	File f = new File(filePath);
	if (f.exists()) {
		if (f.delete()) {
		System.out.println("------> Archivo " + filePath + " eliminado exitosamente\n");
		// Redirección con parámetros para el modal
		String fileName = filePath.replace("resources/", "");
		String encodedFileName = java.net.URLEncoder.encode(fileName, "UTF-8");
		String deleteOK = "HTTP/1.1 303 See Other\n" +
		"Location: /?status=202&action=delete&file=" + encodedFileName + "\n\n";
		dos.write(deleteOK.getBytes());
		dos.flush();
		} else {
		System.out.println("El archivo " + filePath + " no pudo ser borrado\n");
		enviarError404(filePath);
		}
		} else {
		System.out.println("El archivo " + filePath + " no existe\n");
		enviarError404(filePath);
		}
	} catch (Exception e) {
		System.err.println("Error al eliminar recurso: " + e.getMessage());
		enviarError500();
	}
}
```

Función:

- Sirve archivos estáticos (GET/HEAD) o errores (403/404).
- Usa mime.get(extension) para determinar el Content-Type.
- Bandera: Distingue entre GET (envía contenido) y HEAD (solo cabeceras).

```
    // Método para enviar recursos (GET/HEAD)
    public void enviarRecurso(String arg, int bandera) {
        try {
            File f = new File(arg);
            String sb = "HTTP/1.1 200 OK\n";
            if (!f.exists()) {
                arg = "404.html"; // Recurso no encontrado
                sb = "HTTP/1.1 404 Not Found\n";
            } else if (f.isDirectory()) {
                arg = "403.html"; // Recurso privado
                sb = "HTTP/1.1 403 Forbidden\n";
            }
            DataInputStream dis2 = new DataInputStream(new FileInputStream(arg));
            int tam = dis2.available();
            // Obtenemos extensión para saber el tipo de recurso
            int pos = arg.lastIndexOf(".");
            String extension = pos > 0 ? arg.substring(pos + 1) : "";
            // Enviamos las cabeceras de la respuesta HTTP
            sb = sb + "Date: " + new Date() + " \n"
                    + "Server: EnrikeAbi Server/1.0 \n"
                    + "Content-Type: " + mime.get(extension) + " \n"
                    + "Content-Length: " + tam + " \n\n";
            dos.write(sb.getBytes());
            dos.flush();
            String metodo = bandera == 1 ? "GET" : "HEAD";
            if (bandera == 1) {
                // Respuesta GET, enviamos el archivo solicitado
                byte[] b = new byte[1024];
                long enviados = 0;
                int n = 0;
                while (enviados < tam) {
                    n = dis2.read(b);
                    dos.write(b, 0, n);
                    dos.flush();
                    enviados += n;}}
            System.out.println("Respuesta " + metodo + ": \n" + sb);
            dis2.close();
        } catch (Exception e) {
            System.err.println("Error al enviar recurso: " + e.getMessage());
        }
}
```

Procesamiento de Parámetros

Función:

- Parsea parámetros de formularios (GET: desde la URL, POST: desde el cuerpo).
- Construye una tabla HTML con los resultados (estilizada con CSS embebido).
- Usa URLDecoder.decode() para manejar caracteres especiales.

```
    // Método para obtener parámetros de formularios (GET/POST)
    public String obtenerParametros(String line, String headers, int bandera) {
        String metodo = bandera == 1 ? "POST" : "GET";
        String request2 = line;
        if (bandera == 0) {
            // Para GET: separamos los parámetros de la URL
            System.out.println("Petición GET: " + line);
            StringTokenizer tokens = new StringTokenizer(line, "?");
            tokens.nextToken(); // Saltamos el método y ruta
            request2 = tokens.hasMoreTokens() ? tokens.nextToken() : "";
            // Separamos los parámetros de "HTTP/1.1"
            StringTokenizer tokens2 = new StringTokenizer(request2, " ");
            request2 = tokens2.hasMoreTokens() ? tokens2.nextToken() : "";
        }
        System.out.println("Parámetros crudos: " + request2);
        // Creamos el HTML de respuesta con diseño mejorado
        String html = headers
                + "<!DOCTYPE html>\n"
                + "<html>\n"
                + "<head>\n"
                + "<meta charset='UTF-8'>\n"
                + "<title>Método " + metodo + "</title>\n"
                + "<style>\n"
                + "body { font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
background-color: #f0f8ff; margin: 0; padding: 20px; }\n"
                + ".container { max-width: 800px; margin: 30px auto; background: white;
padding: 30px; border-radius: 10px; box-shadow: 0 4px 8px rgba(0,0,0,0.1); }\n"
                + "h2 { color: #2c3e50; text-align: center; margin-bottom: 30px; }\n"
                + "table { width: 100%; border-collapse: collapse; margin: 20px 0; }\n"
                + "th { background-color: #3498db; color: white; padding: 12px; text-
align: left; }\n"
                + "td { padding: 10px; border-bottom: 1px solid #ddd; }\n"
                + "tr:nth-child(even) { background-color: #f2f2f2; }\n"
                + "tr:hover { background-color: #e6f7ff; }\n"
                + ".method-tag { display: inline-block; background-color: " +
(metodo.equals("GET") ? "#2ecc71" : "#e67e22") + "; color: white; padding: 5px 10px;
border-radius: 4px; font-weight: bold; }\n"
                + "</style>\n"
                + "</head>\n"
                + "<body>\n"
```

```
                + "<div class='container'>\n"
                + "<h2>Parámetros obtenidos por medio de <span class='method-tag'>" +
metodo + "</span></h2>\n"
                + "<table>\n"
                + "<thead>\n"
                + "<tr><th>Parámetro</th><th>Valor</th></tr>\n"
                + "</thead>\n"
                + "<tbody>\n";
        // Procesamos los parámetros si existen
       if (!request2.isEmpty()) {
             StringTokenizer paramsTokens = new StringTokenizer(request2, "&");
             while (paramsTokens.hasMoreTokens()) {
             String parametros = paramsTokens.nextToken();
             StringTokenizer paramValue = new StringTokenizer(parametros, "=");
             String param = paramValue.hasMoreTokens() ? paramValue.nextToken() : "";
             String value = "";
             if (paramValue.hasMoreTokens()) {
             try {
             value = URLDecoder.decode(paramValue.nextToken(), "UTF-8");
             } catch (UnsupportedEncodingException e) {
             // UTF-8 siempre está soportado en Java, pero por si acaso:
             value = paramValue.nextToken(); // Usamos el valor sin decodificar
             System.err.println("Advertencia: UTF-8 no soportado (imposible en Java
estándar), usando valor crudo");}}
            html += "<tr><td><strong>" + param + "</strong></td><td>" + value +
"</td></tr>\n";} else {
           html += "<tr><td colspan='2' style='text-align: center;'>No se recibieron
parámetros</td></tr>\n";}
        html += "</tbody>\n"
                + "</table>\n"
                + "</div>\n"
                + "</body>\n"
                + "</html>";
return html;}
```

Método Principal (run())

Función:

1. Lectura: Obtiene la línea inicial de la petición HTTP.
2. Enrutamiento: Según el método (GET/POST/PUT/DELETE/HEAD), llama al método correspondiente.
3. Respuesta: Envía la respuesta HTTP (archivo, HTML generado, o error).
4. Cierre: Libera recursos (dis.close(), dos.close(), cl.close()).

```
    @Override
    public void run() {
        // Cabeceras base para respuestas HTTP
        String headers = "HTTP/1.1 200 OK\n"
```

```
                + "Date: " + new Date() + " \n"
                + "Server: EnrikeAbi Server/1.0 \n"
                + "Content-Type: text/html \n\n";
        try {
            String line = dis.readLine(); // Lee primera línea de la petición

            if (line == null) {
                String vacia = headers
                        + "<!DOCTYPE html>\n"
                        + "<html>\n"
                        + "<head>\n"
                        + "<title>Línea Vacía</title>\n"
                        + "<style>\n"
                        + "body { font-family: Arial, sans-serif; background-color:
#f5f5f5; text-align: center; padding: 50px; }\n"
                        + "h1 { color: #7f8c8d; }\n"
                        + "</style>\n"
                        + "</head>\n"
                        + "<body>\n"
                        + "<h1>No se recibió ninguna solicitud válida</h1>\n"
                        + "</body>\n"
                        + "</html>";
                dos.write(vacia.getBytes());
                dos.flush();
            } else {
                System.out.println("\n------> Cliente Conectado desde: " +
cl.getInetAddress());
                System.out.println("Por el puerto: " + cl.getPort());
                System.out.println("Petición: " + line + "\n");
                // Método GET
                if (line.toUpperCase().startsWith("GET")) {
                    if (line.indexOf("?") == -1) {
                        // Solicita un archivo
                        String fileName = obtenerNombreRecurso(line);
                        enviarRecurso(fileName, 1); // Bandera GET = 1
                    } else {
                        // Envía parámetros desde un formulario GET
                        String respuesta = obtenerParametros(line, headers, 0); //
Bandera GET = 0
                        dos.write(respuesta.getBytes());
                        dos.flush();
                        System.out.println("Respuesta GET con parámetros enviada");
                    }
                }
```

```
                // Método HEAD
                else if (line.toUpperCase().startsWith("HEAD")) {
                    if (line.indexOf("?") == -1) {
                        // Solicita archivo, solo enviamos cabeceras
                        String fileName = obtenerNombreRecurso(line);
                        enviarRecurso(fileName, 0); // Bandera HEAD = 0
                    } else {
                        // Respuesta HEAD para parámetros
                        dos.write(headers.getBytes());
                        dos.flush();
                        System.out.println("Respuesta HEAD para parámetros enviada");
                    }
                }
                // Método POST
                else if (line.toUpperCase().startsWith("POST")) {
                    // Leemos el cuerpo de la petición POST
                    int tam = dis.available();
                    byte[] b = new byte[tam];
                    dis.read(b);
                    String request = new String(b, 0, tam);
                    // Obtenemos la última línea que contiene los parámetros
                    String[] reqLineas = request.split("\n");
                    int ult = reqLineas.length - 1;
                    String respuesta = obtenerParametros(reqLineas[ult], headers, 1);
// Bandera POST = 1
                    dos.write(respuesta.getBytes());
                    dos.flush();
                    System.out.println("Respuesta POST con parámetros enviada");
                }
                // Método DELETE
                else if (line.toUpperCase().startsWith("DELETE")) {
                    String fileName = obtenerNombreRecurso(line);
                    eliminarRecurso(fileName, headers);
                }
                // Método PUT
                else if (line.toUpperCase().startsWith("PUT")) {
                    String fileName = obtenerNombreRecurso(line);
                    String ruta = "resources/" + fileName;
                    try {
                        // Crear directorio si no existe
                        new File("resources").mkdirs();
                        File file = new File(ruta);
                        boolean fileExisted = file.exists();
                        // Leer todas las cabeceras primero
```

```
                        Map<String, String> hders = new HashMap<>();
                        String headerLine;
                        while (!(headerLine = dis.readLine()).isEmpty()) {
                        int separator = headerLine.indexOf(':');
                        if (separator > 0) {
                        String key = headerLine.substring(0, separator).trim();
                        String value = headerLine.substring(separator + 1).trim();
                                hders.put(key, value);
                        }
                        }
```

Soporte para Métodos HTTP

PUT Función:

- Recibe archivos mediante streaming (buffer de 8KB).
- Verifica el tamaño con Content-Length.
- Redirige con código 303 y parámetros (action=create|update).

DELETE Función:

- Elimina el archivo y notifica el resultado.

```
// Obtener longitud del contenido
                        int                         contentLength                         =
Integer.parseInt(hders.getOrDefault("Content-Length", "0"));

                        // Leer el contenido binario exactamente
                        FileOutputStream fos = new FileOutputStream(file);
                        byte[] buffer = new byte[8192]; // Buffer de 8KB
                        int bytesRead;
                        int totalBytesRead = 0;

                        while (totalBytesRead < contentLength &&
                               (bytesRead = dis.read(buffer, 0, Math.min(buffer.length,
contentLength - totalBytesRead))) != -1) {
                            fos.write(buffer, 0, bytesRead);
                            totalBytesRead += bytesRead;
                        }
                        fos.close();

                        // Verificar integridad del archivo
                        if (file.length() != contentLength) {
                            file.delete(); // Eliminar archivo corrupto
                            throw new IOException("El tamaño del archivo recibido no
coincide con Content-Length");
                        }

                        // Respuesta de éxito
```

```
                        String encodedFileName = java.net.URLEncoder.encode(fileName,
"UTF-8");
                        String putResponse = fileExisted ?
                            "HTTP/1.1 303 See Other\n" +
                            "Location:       /?status=200&action=update&file="       +
encodedFileName + "\n\n" :
                            "HTTP/1.1 303 See Other\n" +
                            "Location:       /?status=201&action=create&file="       +
encodedFileName + "\n\n";


                        dos.write(putResponse.getBytes());
                        dos.flush();
                        System.out.println("Archivo " + fileName + " " + (fileExisted
? "actualizado" : "creado") +
                                                " correctamente. Tamaño: " + file.length() +
" bytes");
```

Utilidades

Función:

- Extrae la ruta del recurso de la solicitud HTTP (ej: de GET /file.html HTTP/1.1 extrae file.html).
- Si es vacío, devuelve index.html por defecto.

```
    // Método para obtener el nombre del recurso de la petición HTTP
    public String obtenerNombreRecurso(String line) {
        int i = line.indexOf("/");
        int f = line.indexOf(" ", i);
        String resourceName = line.substring(i + 1, f);
        // Si es vacío, entonces se trata del index
        if (resourceName.isEmpty()) {
            resourceName = "index.html";
        }
        return resourceName;
    }
```

MIME.JAVA

Funciona como un traductor de extensiones a tipos MIME. Mediante un HashMap, asocia extensiones (ej. .jpg, .pdf) con sus tipos MIME estándar (ej. image/jpeg, application/pdf). Su método get() permite al servidor incluir la cabecera Content-Type correcta en las respuestas, asegurando que los navegadores interpreten adecuadamente los archivos enviados (o usar application/octet-stream como valor predeterminado para extensiones desconocidas).

Declaración de Clase y Variable de Instancia

Función:

- Declara un HashMap estático llamado mimeTypes que almacenará pares de valores:
  - **Clave:** Extensión de archivo (ej: "pdf", "jpg")
  - **Valor:** Tipo MIME correspondiente (ej: "application/pdf", "image/jpeg")

```
import java.util.*;

public class Mime {

	public static HashMap<String, String> mimeTypes;
```

Constructor

Función:

- Inicializa el HashMap y lo carga con mapeos de extensiones a tipos MIME
- Incluye los tipos más comunes para documentos, imágenes, audio, video y texto
- Las asociaciones siguen los estándares IANA para tipos MIME

```
	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");
	}
```

Método get()

Función:

- Recibe una extensión de archivo (sin el punto, ej: "jpg")
- Busca el tipo MIME correspondiente en el HashMap
- Si encuentra la extensión, devuelve el tipo MIME asociado
- Si no la encuentra, devuelve el tipo por defecto "application/octet-stream" (para datos binarios genéricos)

Estructura Completa del Mapeo

Gestiona la correspondencia entre extensiones de archivo (.pdf, .jpg, .html) y sus tipos MIME estándar (application/pdf, image/jpeg, text/html).

Componentes clave:

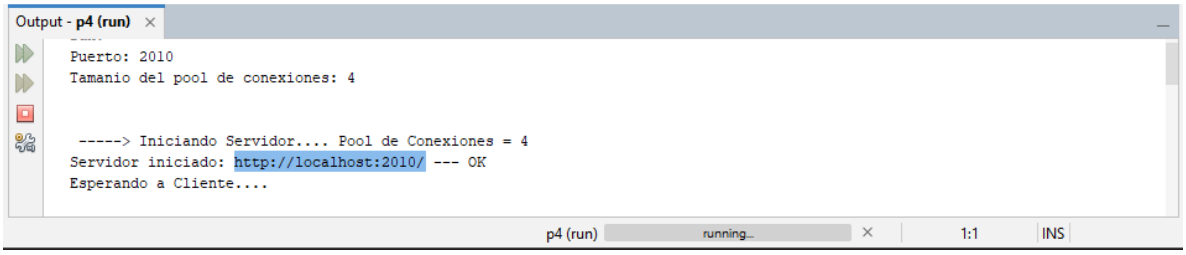
- HashMap estático: Almacena pares (extensión → tipo MIME).
- Constructor: Inicializa el mapa con extensiones comunes
- (documentos, imágenes, audio, texto).
- Método get():
  - Recibe una extensión (ej: "pdf").
  - Retorna el tipo MIME asociado (ej: "application/pdf").
  - Si la extensión no existe, devuelve "application/octet-stream" (tipo genérico para datos binarios).

```
        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");
```

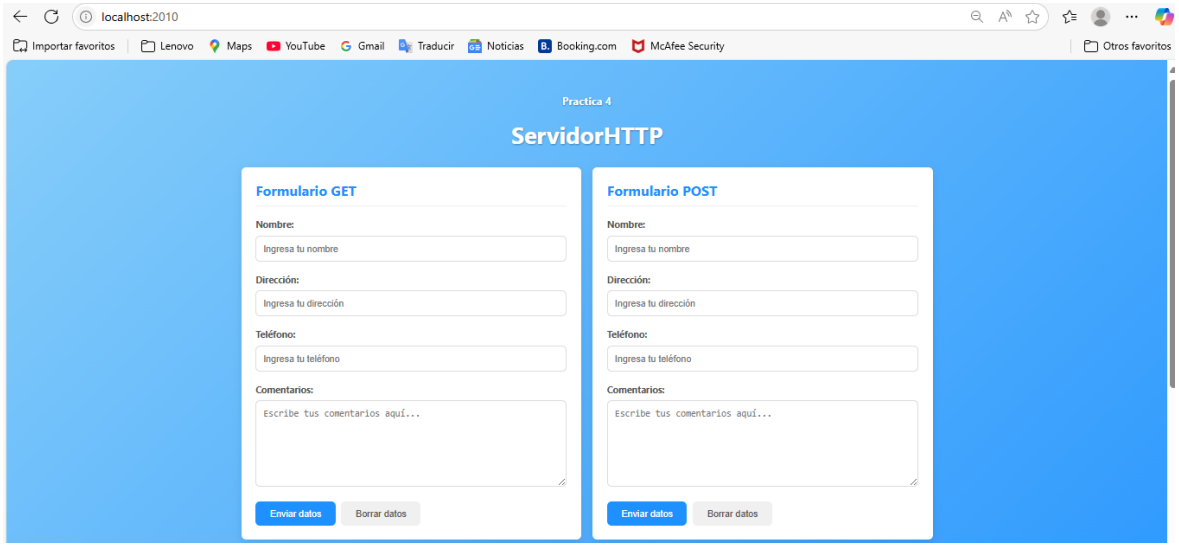
PRUEBAS Y RESULTADOS

Para verificar el correcto funcionamiento de la aplicación, se realizaron pruebas en un entorno simulado, se creó un archivo .html para revisar el correcto funcionamiento de las peticiones que se programaron las cuales son GET, PUT, DELETE y POST.

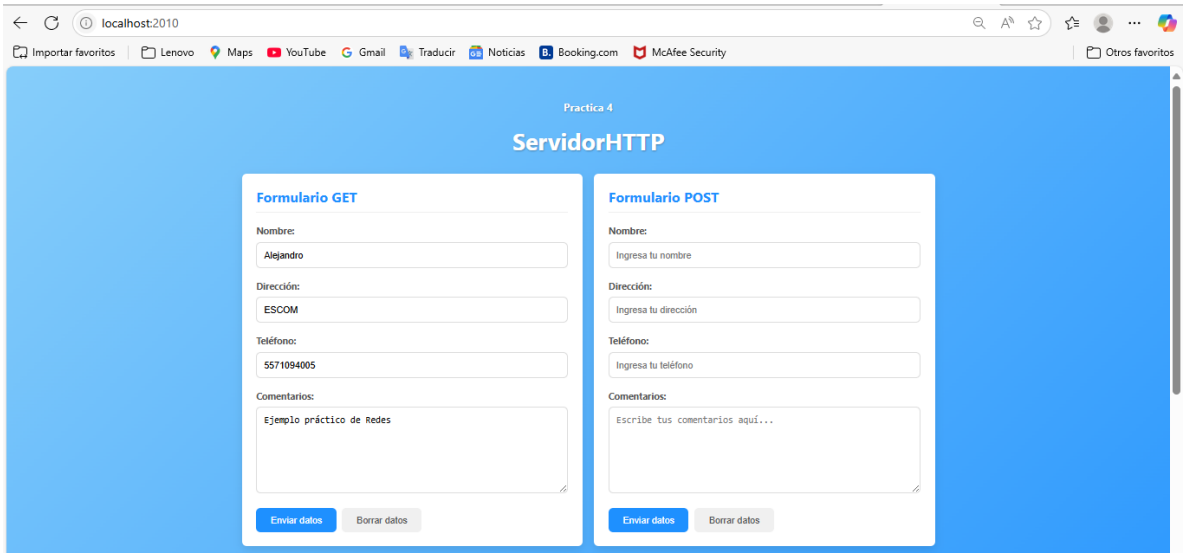
Primero se inicializa el servidor



Ingresamos al URL del servidor que fue iniciado



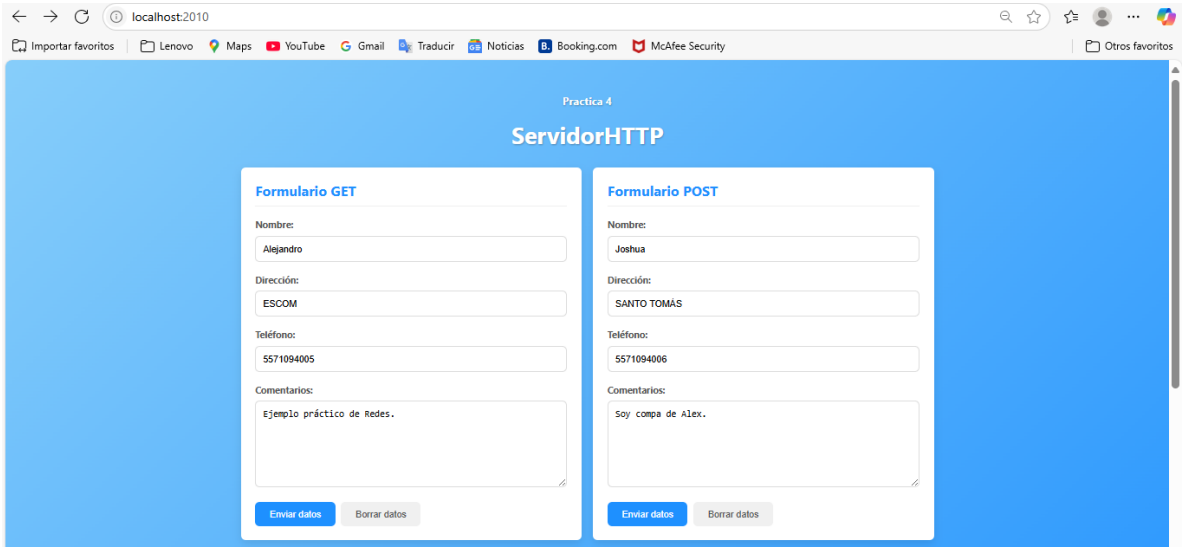
Dentro del url podemos decidir cual petición queremos hacer, para empezar, elegiremos la petición de GET



Al rellenar los campos y darle en el botón enviar, nos aparecerá este resultado en pantalla y en consola


```
------> Cliente Conectado desde: /0:0:0:0:0:0:0:1
Por el puerto: 62088
Petici▯n: GET /get?Nombre=Alejandro&Direccion=ESCOM&Telefono=5571094005&Comentarios=Ejemplo+pr%C3%Alctico+de+Redes. HTTP/1.1

Petici▯n GET: GET /get?Nombre=Alejandro&Direccion=ESCOM&Telefono=5571094005&Comentarios=Ejemplo+pr%C3%Alctico+de+Redes. HTTP/1.1
Par▯metros crudos: Nombre=Alejandro&Direccion=ESCOM&Telefono=5571094005&Comentarios=Ejemplo+pr%C3%Alctico+de+Redes.
Respuesta GET con par▯metros enviada
```

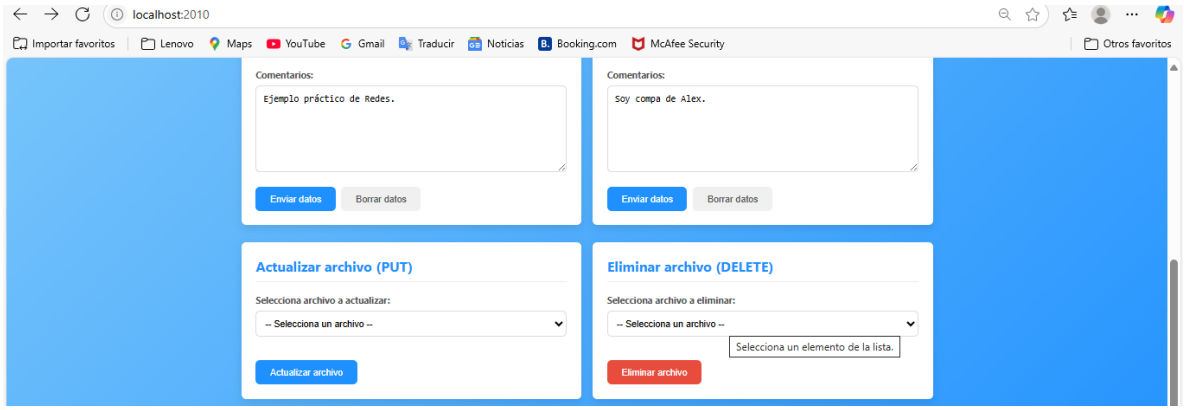
Ahora pasaremos a la petición POST



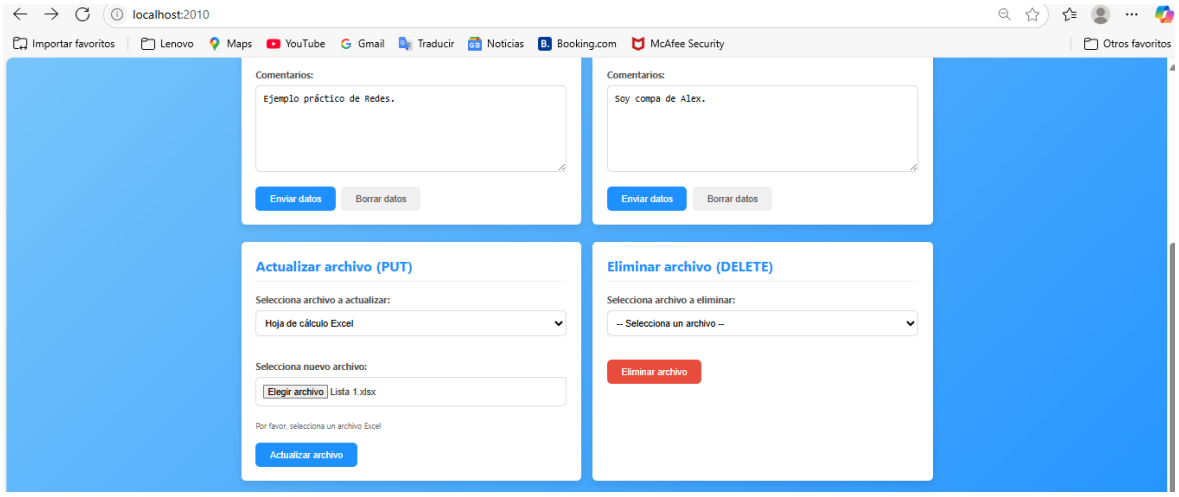
De igual manera, llenaremos los campos del formulario y le daremos enviar lo que nos dará el siguiente resultado


```
------> Cliente Conectado desde: /0:0:0:0:0:0:0:1
Por el puerto: 62134
Petici□n: POST /post HTTP/1.1

Par□metros crudos: Nombre=Joshua&Direccion=SANTO+TOM%C3%81S&Telefono=5571094006&Comentarios=Soy+compa+de+Alex.
Respuesta POST con par□metros enviada
```

La siguiente petición será PUT



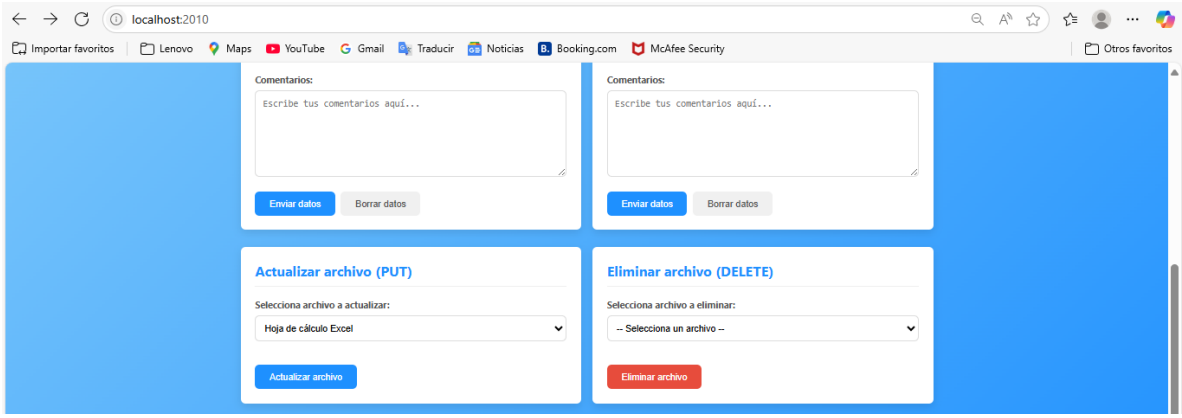
Aquí tendremos que seleccionar el recurso que queremos actualizar, una vez seleccionado tendremos que elegir con recurso lo queremos actualizar



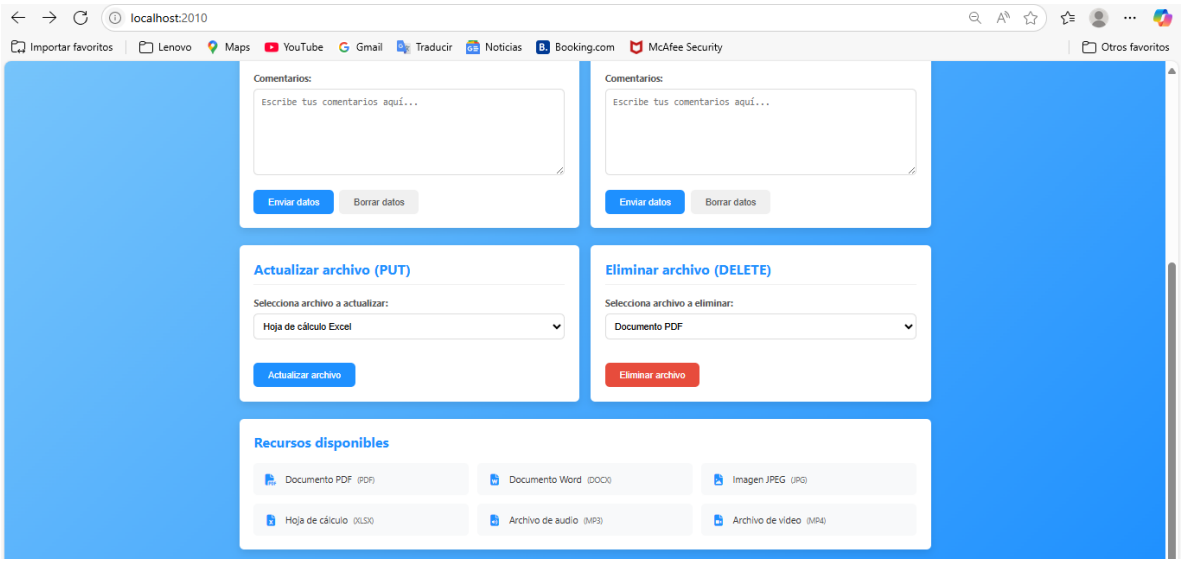
Le daremos en actualizar y nos dará el siguiente resultado


```
------> Cliente Conectado desde: /0:0:0:0:0:0:0:1
Por el puerto: 62260
Petici□n: PUT /hoja.xlsx HTTP/1.1

Archivo hoja.xlsx actualizado correctamente. Tama□o: 159797 bytes

------> Cliente Conectado desde: /0:0:0:0:0:0:0:1
Por el puerto: 62261
Petici□n: GET /?status=200&action=update&file=hoja.xlsx HTTP/1.1

Petici□n GET: GET /?status=200&action=update&file=hoja.xlsx HTTP/1.1
Par□metros crudos: status=200&action=update&file=hoja.xlsx
Respuesta GET con par□metros enviada

------> Cliente Conectado desde: /0:0:0:0:0:0:0:1
Por el puerto: 62262
Petici□n: GET /?status=200&action=update&file=hoja.xlsx HTTP/1.1

Petici□n GET: GET /?status=200&action=update&file=hoja.xlsx HTTP/1.1
Par□metros crudos: status=200&action=update&file=hoja.xlsx
Respuesta GET con par□metros enviada
```

La siguiente y última petición será DELETE



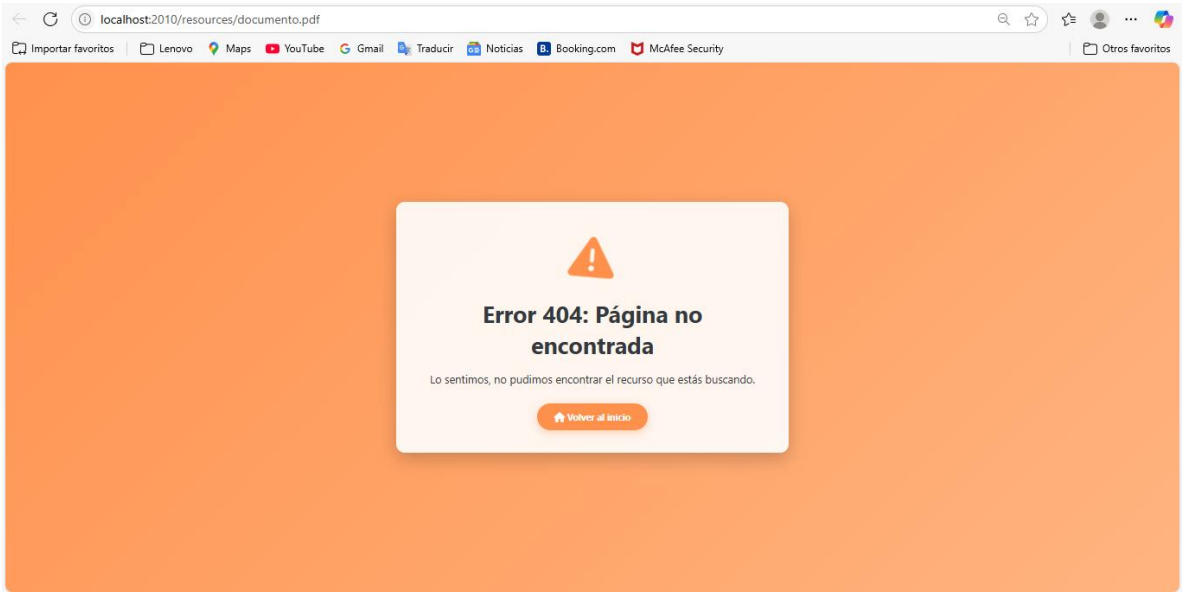
De igual manera tendremos que elegir el recurso a eliminar y posteriormente presionar el botón de “Eliminar”



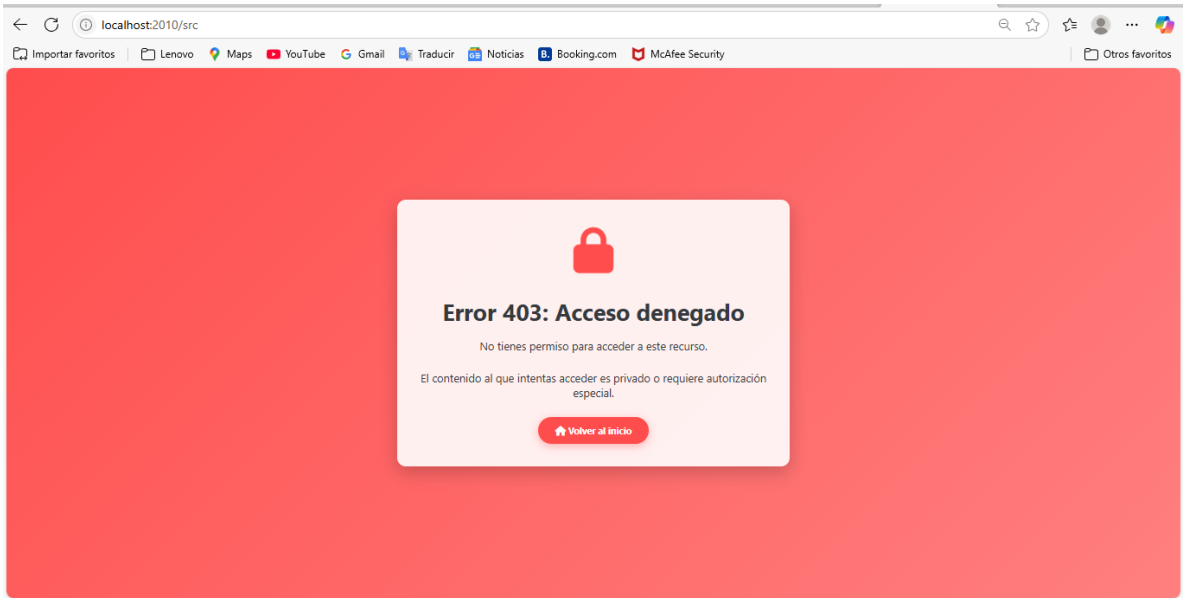
Una vez presionado el botón nos dará el siguiente resultado


```
------> Cliente Conectado desde: /0:0:0:0:0:0:0:1
Por el puerto: 62828
Petici◆n: DELETE /documento.pdf HTTP/1.1

Solicitud DELETE para: resources/documento.pdf
------> Archivo resources/documento.pdf eliminado exitosamente


------> Cliente Conectado desde: /0:0:0:0:0:0:0:1
Por el puerto: 62829
Petici◆n: GET /?status=202&action=delete&file=documento.pdf HTTP/1.1

Petici◆n GET: GET /?status=202&action=delete&file=documento.pdf HTTP/1.1
Par◆metros crudos: status=202&action=delete&file=documento.pdf
Respuesta GET con par◆metros enviada

------> Cliente Conectado desde: /0:0:0:0:0:0:0:1
Por el puerto: 62841
Petici◆n: GET /?status=202&action=delete&file=documento.pdf HTTP/1.1

Petici◆n GET: GET /?status=202&action=delete&file=documento.pdf HTTP/1.1
Par◆metros crudos: status=202&action=delete&file=documento.pdf
Respuesta GET con par◆metros enviada
```

En caso de que no se encuentre el archivo deseado, saldrá el Error 404



En caso de que no se pueda acceder a una carpeta por estar protegida saldrá el Error 403



Los resultados de las pruebas demostraron que:

- La aplicación permite realizar las peticiones de manera correcta mediante el HTML realizado
- Los archivos se actualizan y eliminan de manera correcta
- El uso de los errores 404 y 403 nos permite saber si alguna petición no se puede llevar a cabo de manera satisfactoria.
- Los resultados en la consola nos permiten observar de manera adecuada como trabaja cada una de las peticiones realizadas en esta práctica.

CONCLUSIONES

La implementación de un servidor web multihilo representó una experiencia altamente formativa, que nos permitió afianzar y poner en práctica conocimientos clave en el desarrollo de sistemas en red. Uno de los aprendizajes más significativos fue la comprensión profunda del protocolo HTTP, abarcando desde el análisis detallado de sus cabeceras hasta la correcta construcción de respuestas con los códigos de estado correspondientes.

El uso de un pool de hilos para atender múltiples conexiones simultáneas evidenció el papel fundamental de la programación concurrente en entornos reales, donde la eficiencia y la rapidez de respuesta son esenciales.

A lo largo del desarrollo, nos encontramos con desafíos importantes. Implementar adecuadamente los métodos PUT y DELETE implicó una gestión cuidadosa de los recursos, asegurando que las operaciones sobre archivos se ejecutaran de forma segura y controlada. También fue necesario prestar atención al diseño y funcionalidad de las respuestas de error, como los códigos 404 y 500, integrando HTML y CSS para ofrecer una experiencia más clara y amigable al usuario.

Otro aspecto que destacó fue el manejo de los tipos MIME, lo cual nos enseñó que detalles aparentemente menores, como un encabezado Content-Type incorrecto, pueden tener un gran impacto en la interpretación de los archivos por parte del navegador. La interacción entre las clases principales (ServidorWeb, Manejador y Mime) reforzó conceptos fundamentales como la modularidad y la separación de responsabilidades en el diseño de software.

Este proyecto no solo profundizó nuestra comprensión técnica del funcionamiento interno de un servidor web, sino que también nos brindó herramientas y perspectiva para afrontar retos más avanzados relacionados con la escalabilidad y la seguridad. La integración entre teoría y práctica fue clave para el desarrollo de competencias esenciales en el ámbito del backend y los sistemas distribuidos.