Instituto Politécnico Nacional

Escuela Superior de Cómputo

Departamento de Ciencias e Ingeniería de la Computación

Aplicaciones para Comunicaciones en Red

Practica 04 "Servidor HTTP"

Profesor: Axel Ernesto Moreno Cervantes

Grupo: 6CM1

Integrantes:

Ruvalcaba Flores Martha Catalina

Sandoval Hernández Eduardo

Fecha de entrega: 22/12/2022

Objetivo

Objetivo de la práctica: El estudiante implementará un servidor FTP basándose en sockets de flujo, hilos y principios de funcionamiento del protocolo HTTP.

Planteamiento del problema

El IETF (Internet Engineering Task Force) es un grupo de trabajo encargado de validar y aprobar todos los protocolos que se usan en Internet para garantizar la interoperabilidad entre aplicaciones en red independientemente de arquitecturas, sistemas operativos, lenguajes de programación, etc. Para lograr este objetivo desarrollan especificaciones llamadas RFC (Request For Comments). Una ventaja de contar con la especificación de un protocolo es el poder implementarla o modificar algunas de sus características basándonos en una implementación. En este caso nos gustaría implementar una pequeña versión de un Servidor HTTP la cual se ajuste a la especificación del RFC 2616.

El protocolo de Transferencia de Hipertexto (HTTP, Hyper Text Transfer Protocol) es un protocolo de nivel de aplicación usado para la transferencia de información entre sistemas. Está basado en el modelo ClienteServidor y ha sido utilizado por el World Wide Web desde 1990. Este protocolo permite usar una serie de métodos para indicar la finalidad de la petición. Se basa en estándares como el Identificador de Recurso Uniforme (URI), Localización de Recurso Uniforme (URL) y Nombre de Recurso Uniforme (URN) para indicar el recurso al que hace referencia la petición. El cliente envía una petición en forma de método, una URI y una versión de protocolo, seguida de los modificadores de la petición de forma parecida a un mensaje MIME (Extensiones de Correo Multipropósito de Internet), información sobre el cliente y al final un posible contenido. El servidor contesta con una línea de estado que incluye la versión del protocolo y un código que indica el éxito o error, seguido de la información del servidor en forma de mensaje MIME y un posible contenido.

La sintaxis de la petición es: http://dirección [: puerto][ruta] donde dirección es el nombre calificado o la dirección IP del servidor al que se le solicitará un recurso, el puerto es el identificador de la aplicación servidor que nos brindará el recurso y ruta especifica el nombre o ruta hacia el recurso solicitado Mensaje HTTP: Un mensaje HTTP consiste en una petición de un cliente a un servidor y una respuesta de un servidor al cliente. Las peticiones o respuestas pueden ser simples o completas. La diferencia radica en que una petición completa incluye encabezado y contenido, mientras que una simple solo el encabezado. En el caso de peticiones simples, solo se puede usar el método GET y una respuesta simple solo incluye el contenido. Petición: En el caso de que la petición se haga con el protocolo HTTP/1.0 ó HTTP/1.1 l.

Introducción Teórica

Sockets HTTP

Dentro del mundo de Internet destaca una aplicación que es, con mucho, la más utilizada: la World Wide Web (WWW), a la que nos referiremos coloquialmente como la web. Su gran éxito se debe a la facilidad de uso, dado que simplifica el acceso a todo tipo de información, y a que esta información es presentada de forma atractiva. Básicamente, la web nos ofrece un servicio de acceso a información distribuida en miles de servidores en todo Internet, que nos permite ir navegando por todo tipo de documentos multimedia gracias a un sencillo sistema de hipervínculos.

Para la comunicación entre los clientes y los servidores de esta aplicación, se emplea el protocolo HTTP (Hypertext Transfer Protocol), que será el objeto de estudio de este apartado.

HTTP es un sencillo protocolo cliente-servidor que articula los intercambios de información entre los navegadores web y los servidores web. Fue propuesto por Tim Berners-Lee, atendiendo a las necesidades de un sistema global de distribución de información como la World Wide Web. En la web los servidores han de escuchar en el puerto 80, esperando la conexión de algún cliente web.

Se definió la versión 1.0 del protocolo que añadía nuevos métodos (PUT, POST,..) además de permitir el intercambio de cabeceras entre cliente y servidor.

En esta nueva versión, el navegador se conecta al puerto 80 del servidor y normalmente le envía el comando "GET" seguido de la página que desea obtener. Ahora el navegador añadirá la palabra "HTTP/1.0" para indicar al servidor que quiere utilizar esta nueva versión del protocolo.

A continuación, se incluye una descripción:

- **GET**: Petición de lectura de un recurso.
- **POST**: Envío de información asociada a un recurso del servidor.
- **PUT**: Creación de un nuevo recurso en el servidor.
- **DELETE**: Eliminación de un recurso.
- **HEAD**: El servidor solo transmitirá las cabeceras, no la página

Mime

Un tipo MIME es el tipo Extensiones multipropósito de Correo de Internet (MIME), es una forma estandarizada de indicar la naturaleza y el formato de un documento, archivo o conjunto de datos. Está definido y estandarizado en IETF RFC 6838. La Autoridad de Números Asignados de Internet (IANA) es el organismo oficial responsable de realizar un seguimiento de todos los tipos MIME oficiales, y puede encontrar la lista más actualizada y completa en la página de tipos de medios (Media Types).

Sintaxis

tipo/subtipo

Los navegadores a menudo usan el tipo MIME (y no la extensión de archivo) para determinar cómo procesará un documento; por lo tanto, es importante que los servidores estén configurados correctamente para adjuntar el tipo MIME correcto al encabezado del objeto de respuesta.

La estructura de un tipo MIME es muy simple; consiste en un tipo y un subtipo, dos cadenas, separadas por un '/'. No se permite espacio. El tipo representa la categoría y puede ser de tipo discreto o multiparte. El subtipo es específico para cada tipo.

Un tipo MIME no distingue entre mayúsculas y minúsculas, pero tradicionalmente se escribe todo en minúsculas.

Algunos tipos discretos

- text/plain
- text/html
- image/jpeg
- image/png
- audio/mpeg
- audio/ogg
- audio/*
- video/mp4
- application/octet-stream
- …

Desarrollo de la práctica

En esta práctica se implementó un servidor HTTP que es capaz de reconocer y servir peticiones de recursos mediante los métodos GET, POST, HEAD, PUT Y DELETE (Ver RFC 2616).

A partir del programa ServidorWeb proporcionado por el profesor, se utilizó de ejemplo para desarrollarlo desde el lenguaje Python, de acuerdo con las siguientes modificaciones:

- El programa ServidorWeb implementa un servidor HTTP con funcionalidad limitada en donde solo es capaz de atender peticiones de tipo GET. Basándote en la especificación del protocolo HTTP deberás implementar los métodos de respuesta para peticiones de tipo POST, HEAD, GET y DELETE. El servidor también deberá ser capaz de atender peticiones por lo menos 3 tipos MIME distintos.
- También deberás enviar códigos de respuesta basándote en la especificación de HTTP (ej. 200 para una operación exitosa, 500 para un error interno del servidor).
- Una vez terminado el servidor, deberás probarlo con páginas o aplicaciones web que hagan uso de dichos métodos de envío.
- Implementa una alberca de hilos para que el servidor HTTP pueda atender una cantidad máxima de 100 clientes concurrentemente.

Estructura del programa

El código consta de un archivo "serverSocketHttp.py" el cual hace el levantamiento del servidor HTTP con sockets, y recibe las peticiones del cliente a través de hilos y se identifican dichas peticiones con MIMES, esto permite visualizar cualquier archivo siempre y cuando el formato exista en el servidor HTTP.

Simulación

En las siguientes figuras se muestra la interfaz del servidor junto a las funciones que puede realizar.

Servidor

Inicia el servidor en la figura 1, en espera de alguna petición por parte del cliente.

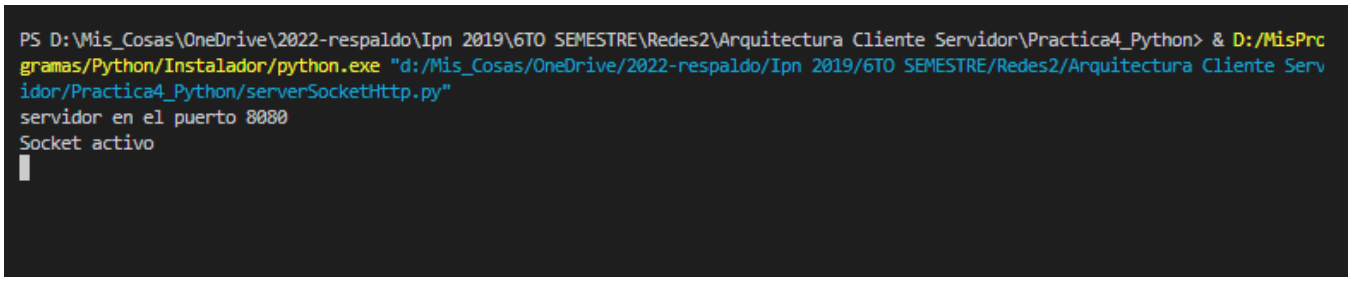

Figura 1. Servidor activo

Si se carga la pagina local con su respectivo puerto se tiene una página html indicando los nombres de los integrantes del equipo, observar la figura 2.

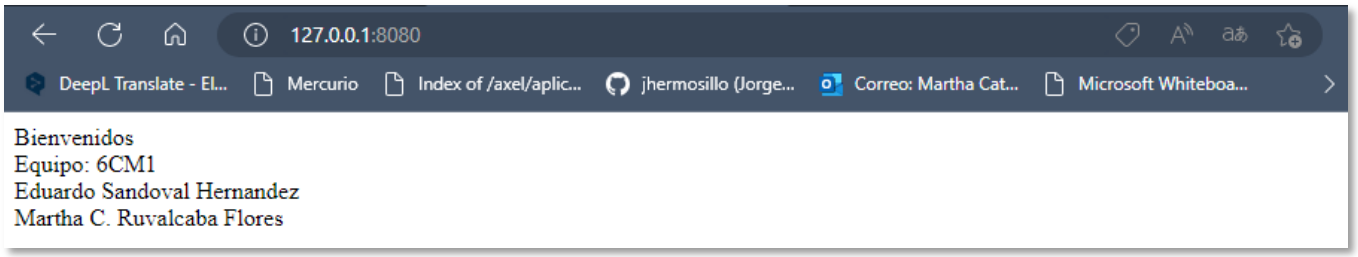

Figura 2. Petición de la página local

Por otro lado, la practica tiene como objetivo utilizar como petición los MIMES a través de las peticiones de respuesta de tipo POST, HEAD, GET y DELETE, para ello se utilizará Postman como la herramienta para el uso de dichos métodos.

Para hacer uso de los MIMES, se crearon archivos de diferentes formatos, tales como: ttf, gif, html, jpg y mp3. A continuación, se mostrará cada uno de estos y algunas peticiones de respuesta.

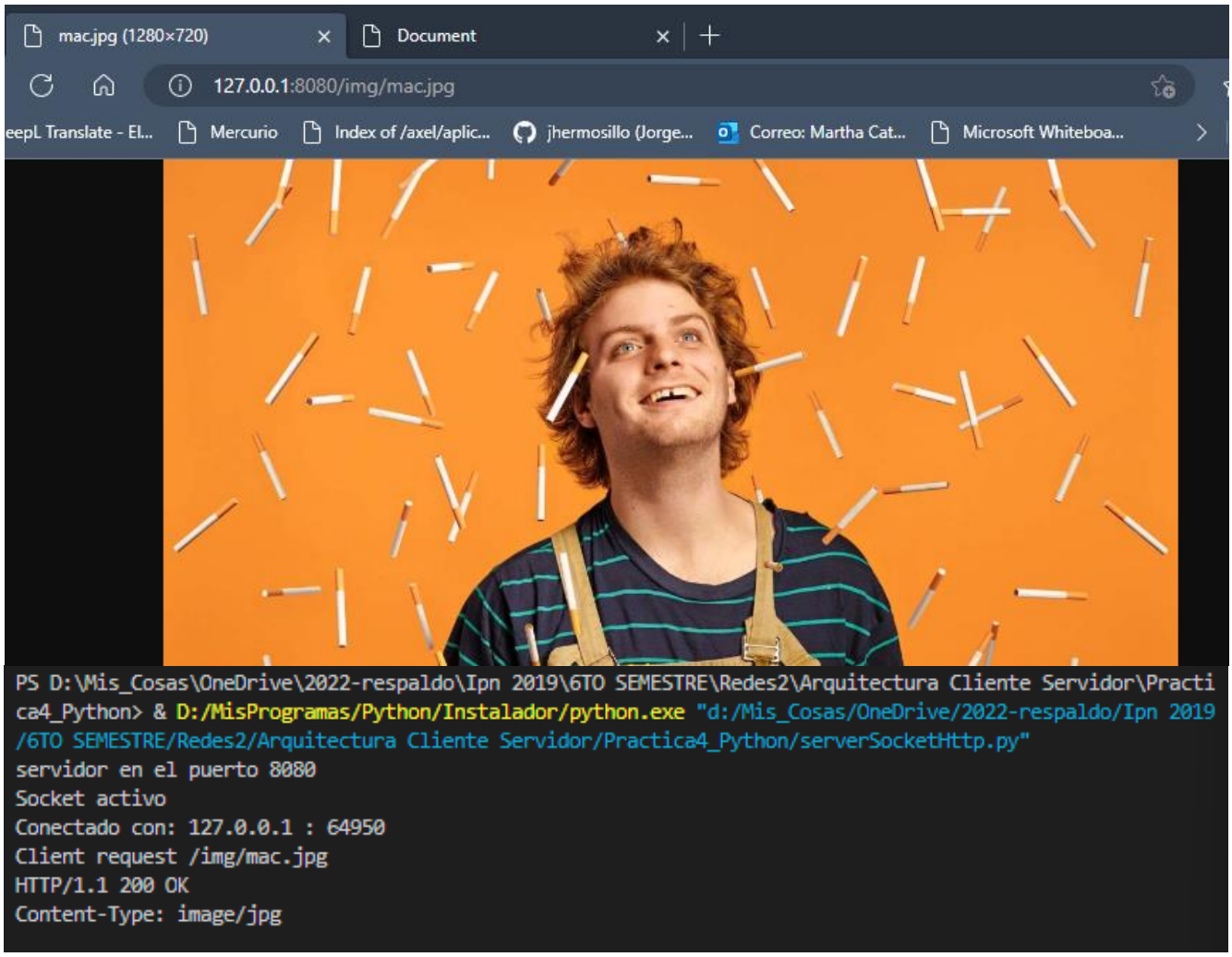

Figura 3. MIME jpg

En la figura 3, se puede observar la petición del MIME jpg, y este se ve reflejado con la terminal del servidor con su respectivo HEADER “Content-Type: image/jpg”. Y el navegador añane la palabra “HTTP/1.1” para indicar al servidor que quiere utilizar esta nueva versión del protocolo. Por otro lado, se puede observar que el hilo se crea al momento de que el cliente realiza una petición, por lo tanto, es posible que existan varias peticiones en el servidor.

Con Postman, aplicamos el método POST y obtenemos una respuesta como lo muestra la figura 4.

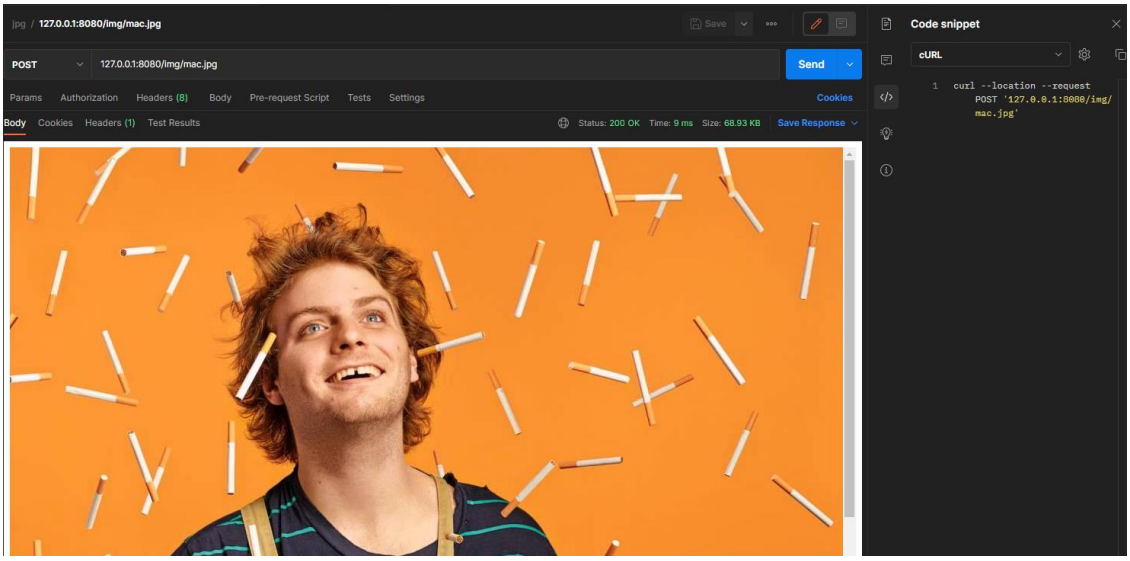

Figura 4. Postman, POST para el MIME jpg

Así como su respectivo HEADER, observar la figura 5.

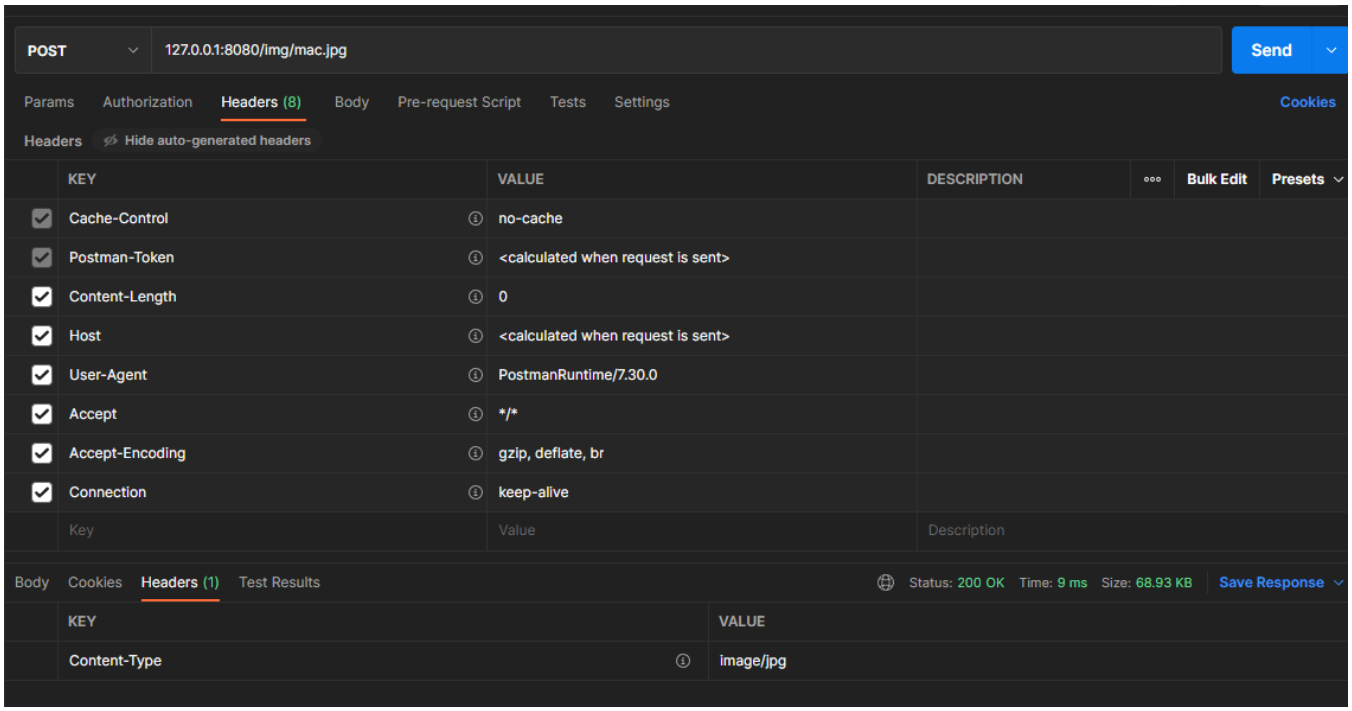

Figura 5. Datos del HEADER acerca del método POST para el MIME jpg

Ahora se realizará una petición del MIME gif, analizar la figura 6 y 7.

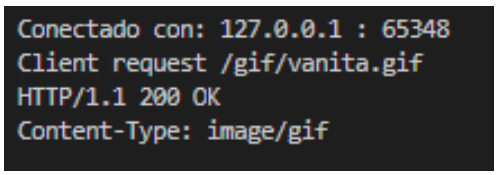

Figura 6. Respuesta del MIME gif

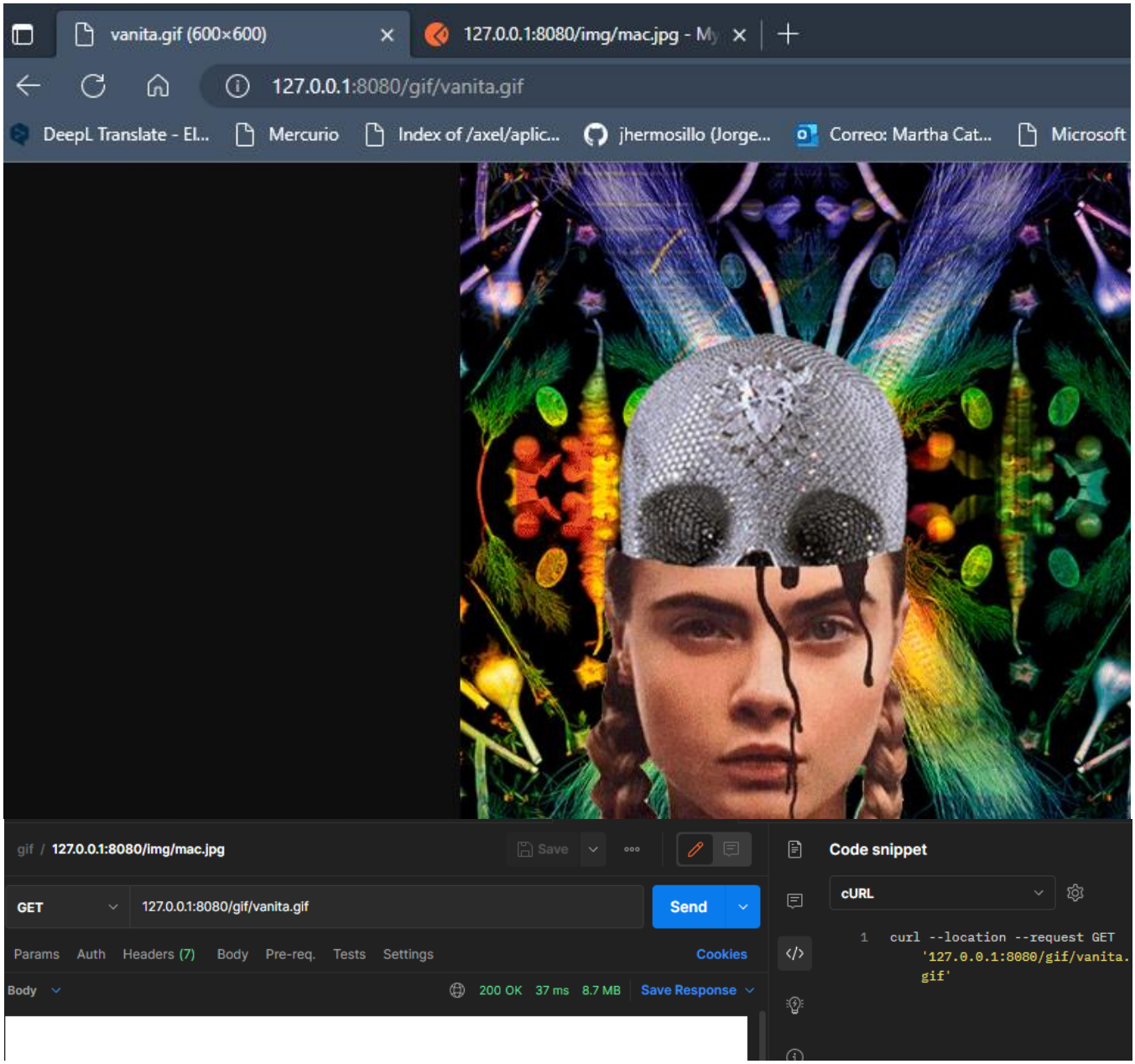

Figura 7. Postman, GET para el MIME gif

Ahora se realizará una petición del MIME ttf, analizar la figura 8 y 9. Cuando se solicita una fuente tipográfica formato ttf, este se descargará automáticamente.

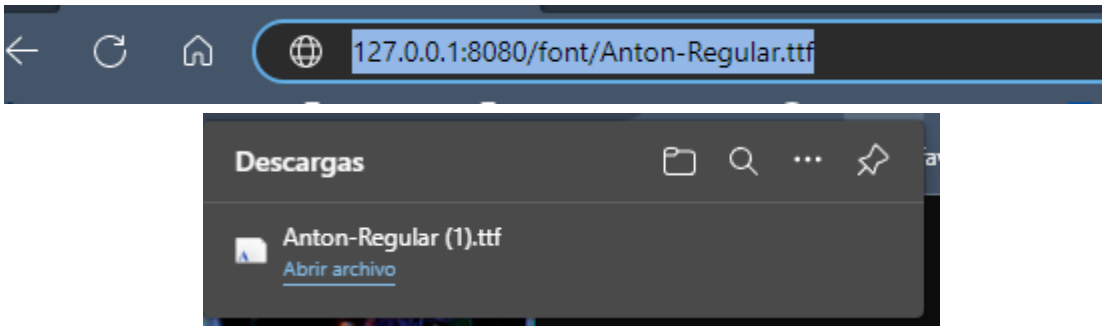

Figura 8. Respuesta del MIME ttf

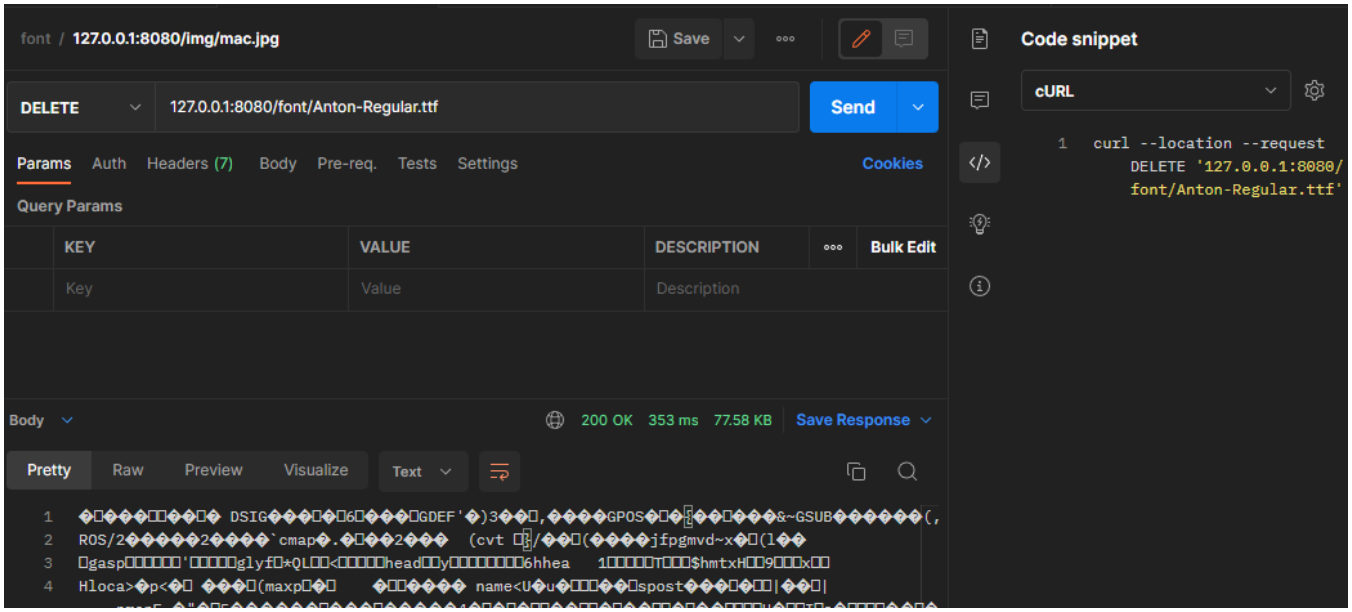

Figura 9. Postman, DELETE para el MIME ttf

Ahora se realizará una petición del MIME mp3, analizar la figura 10 y 11.

```
Conectado con: 127.0.0.1 : 49303
Client request /mp3/JuanGa.mp3
HTTP/1.1 200 OK
Content-Type: audio/mpeg
```

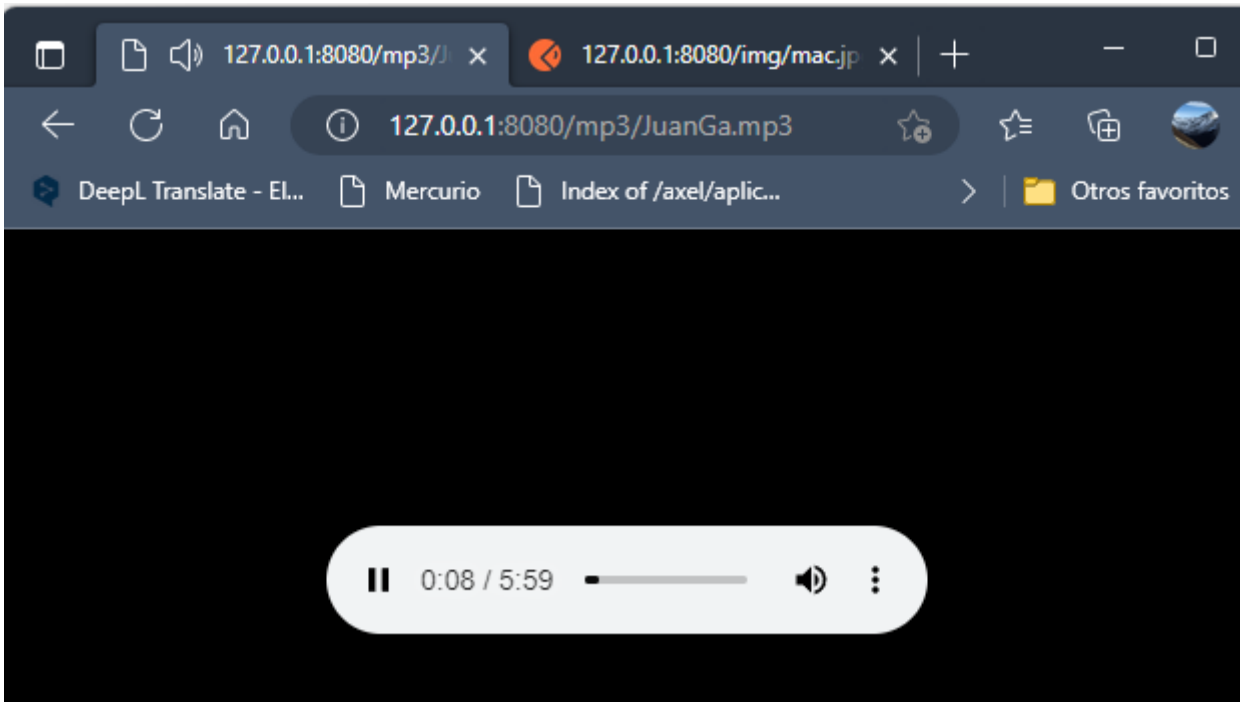

Figura 10. Respuesta del MIME mp3

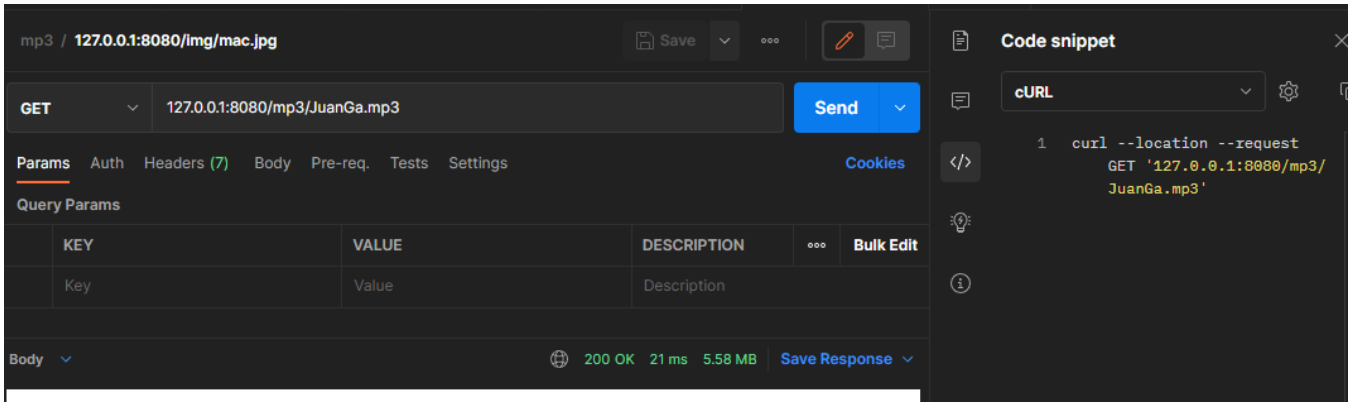

Figura 11 Postman, GET para el MIME mp3

Preguntas sobre el resultado final de la práctica

1. ¿Qué ventajas tiene el uso del método POST vs GET o HEAD?

La diferencia entre los métodos get y post radica en que el método GET envía los datos usando la URL, el método POST los envía en un segundo plano. Tomando en cuenta esto, el método POST puede ofrecer más "discreción" al momento de rellenar formularios puesto que los datos ingresados no se mostrarán en la caché del dispositivo ni en el historial de navegación. Otra de las ventajas del método POST es que no tiene el límite de caracteres que el método GET sí tiene.

2. ¿Qué modificaciones harías al servidor para que fuese capaz de interpretar scriptlets JSP?

Para el caso de esta práctica y debido a su implementación en Python, la propuesta sería utilizar Jython debido a las implementaciones de Python que provee en Java.

Conclusiones Individuales

Ruvalcaba Flores Martha Catalina

Para la realización de la práctica "Servidor HTTP", el equipo se propuso como reto implementarlo en el lenguaje de programación Python. En base a una previa investigación el equipo, se logró entender el MIME, los sockets HTTP y su forma de aplicarlo en conjunto, así mismo las peticiones de respuesta a través de la herramienta Postman y observar su comportamiento en el navegador solicitando varias peticiones haciendo del uso de hilos.

Sandoval Hernández Eduardo

La práctica para el servidor HTTP fue relativamente sencilla de realizar en el lenguaje Python debido a la facilidad que tiene este lenguaje para manejar las funciones necesarias para el protocolo HTTP, además de los hilos y sockets de flujo vistos desde la práctica 1, pudimos aprender más acerca de los distintos tipos de peticiones que se realizan a los servidores HTTP y el cómo responden a cada una de ellas, además de aprender a utilizar la herramienta Postman.

Bibliografía

Máster en Desarrollo de Aplicaciones Android - Unidad 10. Internet: sockets, HTTP y servicios web. (n.d.). Retrieved December 21, 2022, from Androidcurso.com website: http://www.androidcurso.com/index.php/recursos/43-unidad-10-internet-sockets-http-y-servicios-web

Tipos MIME. (n.d.). Retrieved December 21, 2022, from Mozilla.org website: https://developer.mozilla.org/es/docs/Web/HTTP/Basics_of_HTTP/MIME_types

Mozilla (s.f.). HTTP request methods. [En línea]. Disponible en https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

InformIT (2002). Server-Side Web Programming with Jython. Disponible en https://www.informit.com/articles/article.aspx?p=26865&seqNum=10

Anexo 1: Código de serverSocketUDP.py

```
import socket
from _thread import *

host , port = '127.0.0.1' , 8080

serversocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
serversocket.setsockopt(socket.SOL_SOCKET , socket.SO_REUSEADDR , 1)
serversocket.bind((host , port))
serversocket.listen(1)
print('servidor en el puerto',port)

serversocket.listen(10)
print('Socket activo')


def clientThread():
    while True:
        string_list = request.split(' ')
        requesting_file = string_list[1]
        myfile = requesting_file.split('?')[0]
        myfile = myfile.lstrip('/')
        return myfile,requesting_file

while True:
    def do_GET(self):
        print ("[+] New connection: %s:%d" % (self.client_address[0],
self.client_address[1]))
        self.index()

    connection , address = serversocket.accept()
    request = connection.recv(1024).decode('utf-8')
    print("Conectado con: "+address[0]+ " : " +str(address[1]))
    start_new_thread(clientThread,())

    myfile,requesting_file = clientThread()
```

```
    print('Client request',requesting_file)

    if(myfile == ''):
        myfile = 'index.html'

    try:
        file = open(myfile , 'rb')
        response = file.read()
        file.close()

        header = 'HTTP/1.1 200 OK\n'

        if(myfile.endswith('.jpg')):
            mimetype = 'image/jpg'
        elif(myfile.endswith('.ttf')):
            mimetype = 'font/ttf'
        elif(myfile.endswith('.html')):
            mimetype = 'text/html'
        elif(myfile.endswith('.gif')):
            mimetype = 'image/gif'
        elif(myfile.endswith('.mp3')):
            mimetype = 'audio/mpeg'

        header += 'Content-Type: '+str(mimetype)+'\n\n'
        print(header)

    except Exception as e:
        print("-")
        header = 'HTTP/1.1 404 Not Found\n\n'
        print("HTTP/1.1 404 Not Found\n")
        response = '<html><body>Error 404: File not found</body></html>'.encode('utf-8')

    final_response = header.encode('utf-8')
    final_response += response
    connection.send(final_response)
    connection.close()
```