

Instituto Tecnológico de Costa Rica
Sede Campus Tecnológico Central Cartago
Escuela de Ingeniería en Computación
IC6600 - Principios de Sistemas Operativos

Proyecto 1: Streaming Server + CLI and Web Clients

Andrés Felipe Miranda Arias - 2017075170

Josué Daniel Canales Mena - 2017134770

18/07/2020

I semestre 2020

Prof. Ing. Esteban Arias Mendez

Resumen

A HTTP server created entirely in C using sockets, managing threads with the POSIX library. And two different clients consuming the server, one of them is the Command Line Interface (CLI) also developed in C, and a web client using the Angular Framework.

Índice

1. Introducción	3
1.1. Servidor	3
1.2. Clientes	3
1.3. Solución Propuesta	3
2. Marco Teórico	3
2.1. Sockets	3
2.1.1. Atributos	4
2.2. HTTP	4
2.2.1. HTTP/0.9	4
2.2.2. HTTP/1.0	5
2.2.3. HTTP/1.1	5
2.2.4. HTTP responses	6
2.3. MD5sum	6
2.4. POSIX	6
2.5. JSON	6
2.5.1. Estructura	7
2.5.2. json-maker	7
2.5.3. tiny-json	8
2.6. Makefile	8
2.6.1. Reglas	8
2.7. Angular	8
2.8. Reproducción de audio en C	9
2.8.1. Libmad	9
2.8.2. PulseAudio	9
3. Implementación en el proyecto	9
3.1. HTTP	9
3.1.1. /login	9
3.1.2. /register	10
3.1.3. /res	10
3.1.4. /res/[filename.ext]	10
3.1.5. /exit	11
3.2. MD5sum	11
3.3. JSON	11
3.4. json-maker	11
3.5. tiny-json	11
3.6. Makefile	11
4. Estructuras de datos	12
4.1. Multimedia	12
4.1.1. Código	12
4.1.2. Diagrama	12
4.2. User	12
4.2.1. Código	12
4.2.2. Diagrama	13
4.3. Server	13
4.3.1. Código	13
4.3.2. Diagrama	13
4.4. Log	13
4.4.1. Código	13

4.4.2. Diagrama	14
4.5. Logger	14
4.5.1. Código	14
4.5.2. Diagrama	14
4.6. Folder	14
4.6.1. Código	14
4.6.2. Diagrama	15
4.7. User_queue	15
4.7.1. Código	15
4.7.2. Diagrama	15
4.8. Session	15
4.8.1. Código	15
4.8.2. Diagrama	16
4.9. Sessions	16
4.9.1. Código	16
4.9.2. Diagrama	16
5. Diseño de solución	16
5.1. Servidor	16
5.2. Clientes	18
5.2.1. Cliente CLI	18
5.2.2. Cliente Web	19
6. Pruebas de ejecución	21
6.1. Servidor	21
6.2. Cliente CLI	22
6.2.1. Login	23
6.2.2. Register	23
6.2.3. Res	23
6.2.4. File	24
6.2.5. Streamfile	24
6.2.6. Exit	26
6.3. Cliente Web	27
6.3.1. Login	27
6.3.2. Register	28
6.3.3. Streaming de archivos	29
7. Comentarios y observaciones	32
8. Conclusiones	32
9. Referencias	33

1. Introducción

Este proyecto consiste en la programación de un servidor para la distribución en línea, o “streaming”, de archivos multimedia hacia múltiples clientes conectados de forma concurrente al sistema; entiéndase por estos archivos multimedia, archivos de audio, vídeo, imágenes, pdf, texto, entre otros. Los diferentes usuarios, o clientes, deben conectarse de forma remota, en la misma red del servidor, o utilizando un VPN, y podrán reproducir el contenido de estos archivos antes mencionados mediante streaming. [7]

1.1. Servidor

El servidor deberá estar programado completamente sobre un sistema operativo Linux, y en el lenguaje de programación C; deberá usar el compilador `gcc` únicamente. Este deberá usar `sockets` para la comunicación de los clientes remotos hacia un puerto y una dirección IP conocidos, el cual deberá establecer, o bien configurar una resolución de nombres vía DNS o similar.

A su vez, el servidor deberá atender las solicitudes concurrentemente. Al recibir una conexión, deberá crear un proceso liviano tipo *thread* para que atienda a dicho cliente. Para el manejo de solicitudes, el servidor tendrá control de usuarios, y deberá implementar algún mecanismo para el registro, autenticación e inicio de sesión de cada usuario desde múltiples clientes. [7]

1.2. Clientes

Se van a requerir dos tipos de clientes: uno en modo texto, o por línea de comandos en terminal, y otro que permita reproducir vídeos e imágenes principalmente, y adicionalmente otros tipos como audio, pdf, textos, entre otros.

Para iniciar sesión, se le solicitará un nombre de usuario al ingresar. Luego de iniciar sesión, el usuario podrá solicitarle al servidor la lista de archivos multimedia disponibles, y poder seleccionar cualquiera para comenzar la reproducción vía streaming.

El cliente CLI deberá solicitar cualquier tipo de archivo, pero los archivos multimedia serán descargados para reproducción local, los demás archivos sí podrán reproducirse en la terminal. El cliente en modo gráfico podrá ser desarrollado en cualquier tecnología y deberá mostrar todos los archivos. [7]

1.3. Solución Propuesta

La solución propuesta al problema es realizar tanto el cliente CLI como el servidor en el lenguaje establecido, en C, sobre un sistema operativo Linux; en este caso, se trabajará sobre la distribución Ubuntu de Linux. El cliente en modo gráfico se trabajará web, con el framework Angular.

Las estructuras de datos implementadas y la metodología implementada para la conexión y comunicación entre cliente y servidor será descrita más adelante en este documento.

2. Marco Teórico

En esta sección, se definirá cada uno de los protocolos, bibliotecas y frameworks utilizados en proyecto.

2.1. Sockets

Un socket es un mecanismo de comunicación que permite la creación de sistemas cliente/-servidor ya sea de manera local o en diferentes computadoras a través de una red. El mecanismo de socket puede tener a múltiples clientes conectados en un único servidor. [1]

2.1.1. Atributos

Protocolos del socket: Los principales son sockets bajo el protocolo UNIX network, y protocolos bajo el protocolo File System.

Tipos de Socket: Stream sockets y Datagram sockets.

Dominio del socket: El más utilizado es el AF_INET que se refiere la red de internet.

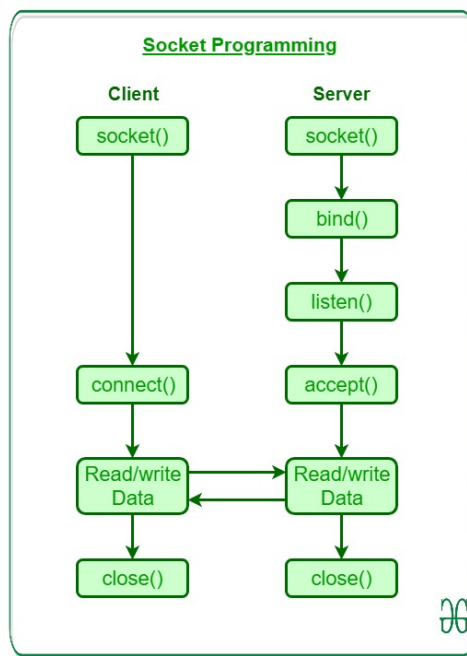


Figura 1: Estructura de un socket

2.2. HTTP

Inventado por Tim Berners-Lee en el CERN entre 1989 y 1991, el protocolo HTTP (Hypertext Transfer Protocol) es el protocolo estándar utilizado en el World Wide Web (www). [2] Este es un protocolo de tipo solicitud-respuesta en el modelo de computación cliente-servidor. Este protocolo cuenta con cuatro versiones:

- HTTP/0.9
- HTTP/1.0
- HTTP/1.1
- HTTP/2.0

2.2.1. HTTP/0.9

Esta es la versión inicial del protocolo. Como característica principal es que consiste en una única línea compuesta por el método y la ruta (path) del documento. Un ejemplo de una petición HTTP/0.9 es el siguiente:

```
GET /index.html
```

métodos válidos: GET

tipos de respuesta: Hypertext

conexión: termina inmediatamente después de la petición

2.2.2. HTTP/1.0

Es una versión de protocolo amigable con los navegadores. Como características agregadas frente a HTTP/0.9 está la integración de campos de cabecera (Header fields). También, soporta más métodos de petición, así como la capacidad de recibir archivos diferentes a HTML. Un ejemplo de una petición HTTP/1.0 es el siguiente:

```
GET /index.html HTTP/1.0
```

```
User-Agent: NCSA_Mosaic/2.0 (Windows 3.1)
```

métodos válidos: GET, HEAD, POST

tipos de respuesta: Definido por el header Content-Type

conexión: termina inmediatamente después de la petición

```
GET /index.html HTTP/1.0
```

```
User-Agent: NCSA_Mosaic/2.0 (Windows 3.1)
```

2.2.3. HTTP/1.1

Esta es la versión estándar en la www actualmente. Agrega mejoras en rendimiento, conexiones persistentes, transferencias en pedazos (chunks, en inglés) y respuestas más rápidas. Un ejemplo de una petición HTTP/1.1 es el siguiente:

```
POST / HTTP/1.1
```

```
User-Agent: PostmanRuntime/7.26.1
```

```
Host: 192.168.0.22:9000
```

```
Accept-Encoding: gzip, deflate, br
```

```
Connection: keep-alive
```

```
Content-Length: 61431
```

```
Content-Type: video/mp4
```

métodos válidos: GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS

tipos de respuesta: Definido por el header Content-Type

conexión: persistente

Para el proyecto se decidió utilizar la versión HTTP/1.1 ya que al día de hoy es el estándar utilizado.

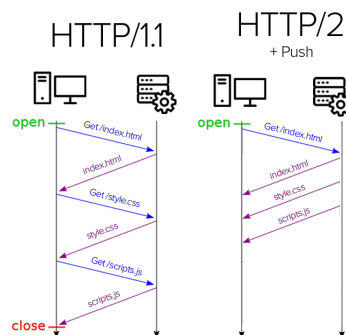


Figura 2: Versiones de HTTP

2.2.4. HTTP responses

Por cada request realizado por el cliente, el servidor debe enviar una respuesta de vuelta (únicamente una respuesta en versiones HTTP/1.1 o anteriores; puede enviar varias depuestas al cliente en la versión HTTP/2.0). El formato es similar al de las peticiones, la primera línea se compone por la versión de HTTP, código de estado del resultado de la operación. Seguido de un conjunto de headers con información relevante para el cliente; y por último del cuerpo del mensaje, esto si lo contiene. A continuación, un ejemplo de una respuesta HTTP/1.1

```
HTTP/1.1 200 OK
Content-Length: 88
Content-Type: audio/mpeg
Chunk: 6546544
Byte-Pos: 0
Next-Index: 5464564
MD5sum: 65465aff4546c464
Total-Bytes: 765465474
Connection: keep-alive
```

2.3. MD5sum

MD5 es un algoritmo de encriptación de 128 bit, el cual genera un código hexadecimal de 32 caracteres sin importar el tamaño del archivo o texto. [3]
md5sum es un programa de Unix para generar estos códigos, es una herramienta de seguridad que sirve para verificar la integridad de los datos.
En terminal, se puede utilizar este programa de la siguiente manera.

```
$ md5sum {archivo}
```

Para poder utilizar esta función en C se debe de instalar el paquete libssl con el comando.

```
sudo apt install libssl-dev
```

Ejemplo de un hash generado de md5.

```
2290babda371e52eeca2a2065a358783
```

2.4. POSIX

POSIX (Portable Operating System Interface for uniX) es una norma escrita por la IEE, es una norma escrita por la IEEE, que define una interfaz estándar del sistema operativo y el entorno. [4]
La biblioteca de POSIX thread son una API de hilos (o subprocesos) basados en estándares para C/C++. Que permite la programación de manera concurrente en de manera más rápida que la utilización de forking de procesos. POSIX también cuenta con una implementación de semáforos para la sincronización de subprocesos. se incluye con la biblioteca semaphore.h

2.5. JSON

JSON (JavaScript Object Notation) es un formato ligero de intercambio de datos. Facil de leer y escribir para los desarrolladores; y facil de analizar sintácticamente para las máquinas. JSON está construido en dos estructuras. [5]

- Una colección de pares nombre/valor.
- Una lista ordenada de valores

2.5.1. Estructura

Un objeto es un conjunto de pares nombre/valor y tiene el siguiente flujo. Un array es una colección ordenada de valores y tiene el siguiente flujo.

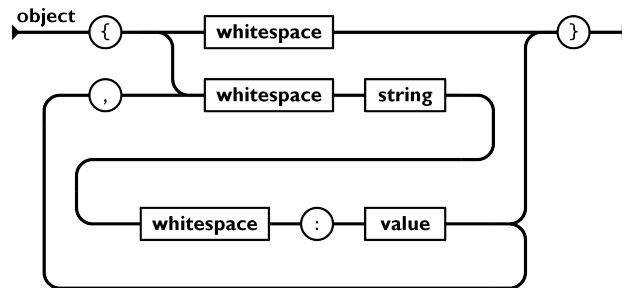


Figura 3: Estructura de un Object

un valor puede ser alguno de los siguientes elementos.

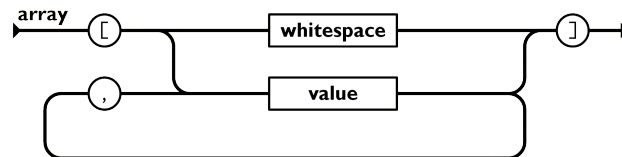


Figura 4: Estructura de un Array

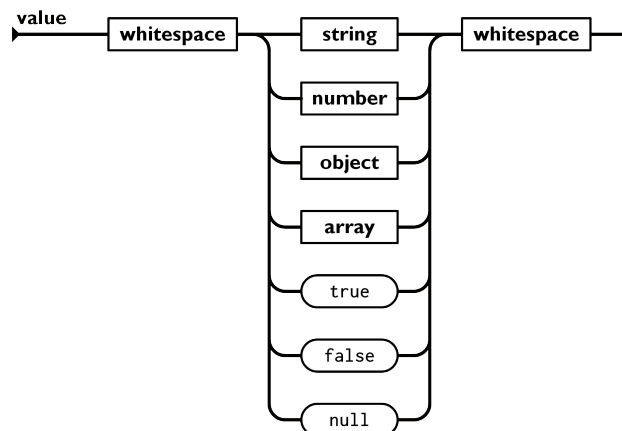


Figura 5: Estructura de un Value

2.5.2. json-maker

Es una biblioteca open-source bajo la MIT-License creada por Rafa García. Es un constructor de json para C, utilizando strings que mediante las funciones de esta librería, concatenan información con el formato JSON. Las principales funciones son:

- `json_objOpen(char *puntero, char *nombre):` agregar "nombre : {"
- `json_int(char *puntero, char *nombre, int valor):` agrega par nombre/valor con entero
- `json_string(char *puntero, char *nombre, char *valor):` agrega nombre/valor con string
- `json_arrOpen(char *puntero, char *nombre):` agrega "nombre: ["
- `json_arrClose(char *puntero):` agrega "]"
- `json_objClose(char *puntero):` agrega "}"

2.5.3. tiny-json

Es una biblioteca open-source bajo la MIT-License creada por Rafa García. es un parser de JSON para C, que ayuda a obtener los diferentes valores de un string con formato JSON. Para la mayoría de operaciones disponibles se utiliza una estructura de datos llamada `json_t`. Las principales funciones son las siguientes:

- `json_create(char *str, json_t pool[], int MAX_FIELDS):` parsea str, recibe un pool del tamaño MAX_FIELDS
- `json_getProperty(json_t parent, "name"):` devuelve un `json_t` (namefield)
- `json_getValue(json_t namefield):` obtiene un `json_t` namefield y devuelve un string con el valor

2.6. Makefile

Makefile es un archivo que contiene un conjunto de directivas como herramienta de construcción automática hacia un target específico. [6]

2.6.1. Reglas

las reglas para crear un makefile con muy simples.

```
target: dependencies
    system command (s)
```

target es una etiqueta, las dependencias son los archivos los cuales se deberán rastrear, en otras palabras, verificar si se dan cambios en estos archivos; y los comandos que debe realizar el sistema

2.7. Angular

Angular es un framework para aplicaciones web escrito principalmente en TypeScript, y fue desarrollado por el Angular Team de Google. En este caso particular, se llamará Angular lo que se conoce como Angular2, el cual no es el mismo framework que Angular.js. Este framework se basa en la modularización de diferentes componentes y servicios que se utilizan a lo largo del desarrollo del proyecto. [8]

El funcionamiento básico de Angular está entre los archivos TypeScript, y los archivos HTML de cada componente. El `.html` tendrá la estructura del archivo, y el `.ts` se encargará de la lógica y funcionamiento de este. Adicionalmente, se pueden adicionar servicios o directivas, que serán módulos adicionales que se pueden utilizar.

Normalmente, los servicios son para hacer llamados asíncronos a bases de datos, APIs u otros, y las directivas se pueden utilizar, por ejemplo, para implementar comandos y condiciones en la parte visual de la aplicación.

2.8. Reproducción de audio en C

2.8.1. Libmad

Libmad es una biblioteca utilizada para la reproducción de sonido en C. Con este API de bajo nivel, se puede programar cada paso de la decodificación en el proceso de manera explícita, y debe manejarse de la misma forma, lo cual, a su vez, permite un gran manejo en el control del audio. [9]

La biblioteca implementa diferentes funciones para el manejo de los archivos, como por ejemplo la función **mad_frame_decode()**, la cual es utilizada para decodificar el siguiente header del frame y genera subbands, o **mad_synth_frame()** que utiliza esos subbands para sintetizarlos en PCM-audio samples. [9]

2.8.2. PulseAudio

La biblioteca PulseAudio permite tomar esos archivos de sonido, pasarlos a un PulseAudio server, y poder reproducir y generar un output de esos archivos, que entran al PulseAudio como PCM-audio samples. PulseAudio permite trabajar con otras bibliotecas sobre esta, como por ejemplo libao, sin embargo, PulseAudio captura los outputs ALSA y los conecta al servidor de audio, el cual puede generar una salida de audio ya sea en audífonos, HDMI, o inclusive Bluetooth. [10]

Con esto, una vez decodificado y reensamblado el audio, es posible tomarlo para generar la salida de audio al cliente directamente en la terminal.

3. Implementación en el proyecto

En esta sección se explicará como se utilizaron las bibliotecas, protocolos y frameworks del proyecto.

3.1. HTTP

Para la implementación de HTTP con los sockets de C, no hubo uso de alguna librería externa para la creación o el parseo de los mensajes. Las solicitudes que se utilizaron son las siguientes

3.1.1. /login

Esta ruta se utiliza para iniciar sesión en el servidor.

Método: POST

Headers:

- Content-Type: application/json
- Content-Length: [Tamaño del JSON]

Body: JSON con el formato "username": usr, "password": psw. Siendo usr el string con el nombre de usuario, y psw el string con la contraseña.

respuestas

- 200 OK, si el cliente pudo iniciar sesión. Si existe una sesión anterior de la cuenta y estaba streameando algún archivo, manda información de ese archivo, la posición en la que iba, el tamaño total.
 - 401 Unauthorized, si el cliente no existe o la contraseña no coincide.
-

3.1.2. /register

Esta ruta registra un cliente en el servidor.

Método: POST

Headers

- Content-Type: application/json
- Content-Length: [Tamaño del JSON]

Body: JSON con el formato {“username”: usr, “password”: psw}. Siendo usr el string con el nombre de usuario, y psw el string con la contraseña.

respuestas

- 200 OK, si el cliente pudo registrar el usuario.
 - 401 Unauthorized, si el cliente ya existe.
-

3.1.3. /res

Solicita información de todos los documentos multimedia del servidor.

Prerrequisito: Haber iniciado sesión

Método: GET

Headers: No requiere headers adicionales.

Body: No requiere body

respuestas

- 200 OK, más un JSON con la lista de archivos.
 - 401 Unauthorized, si no ha iniciado sesión.
-

3.1.4. /res/[filename.ext]

Solicita un archivo multimedia a consumir.

Prerrequisito: Haber iniciado sesión

Método: GET

Headers

- Byte-Pos: posición del byte para empezar a reproducir
- Client: [web — cli]

Body: No requiere body

respuestas

Si NO es un archivo streamable.

- 200 OK, el archivo completo.
- 401 Unauthorized, si no ha iniciado sesión.

Si no es un archivo streamable.

200 OK: Devuelve un chunk del archivo, más los siguientes headers

- Content-Type: Tipo del archivo [audio o video]

- Bytes-Pos: Posición donde empieza a reproducir
- Next-Index: Siguiete posición del byte
- MD5sum: MD5sum del fragmento enviado
- Total-Bytes: Tamaño total del archivo

401 Unauthorized: si no ha iniciado sesión.

3.1.5. /exit

Avisa al servidor para terminar la conexión

Método: GET

Headers: No requiere headers adicionales.

Body: No requiere body

respuestas

- 200 OK, más un JSON con la lista de archivos.
 - 401 Unauthorized, si no ha iniciado sesión.
-

3.2. MD5sum

Este se implementa a la hora del escaneo de los archivos multimedia. También genera un hash cuando el servidor envía el archivo en pedazos.

3.3. JSON

Este es el formato utilizado para el envío de datos estructurados.

3.4. json-maker

Se utiliza para la creación de json de los archivos multimedia que se envía a los clientes cuando utilizan el método HTTP /res.

3.5. tiny-json

En el servidor, se utiliza al momento de obtener el JSON de parte de los cliente que utilizan el método HTTP /register o /login. Por parte del cliente CLI, se utiliza cuando obtiene la lista de los archivos multimedia mediante el método HTTP /res.

3.6. Makefile

El makefile utilizado para la compilación del servidor es el siguiente.

```
server: server.c utils.c server.h
    gcc server.c -o server -lpthread -lcrypto -lrt

test: testfile.c utils.c server.h
    gcc testfile.c -o test -lcrypto
```

4. Estructuras de datos

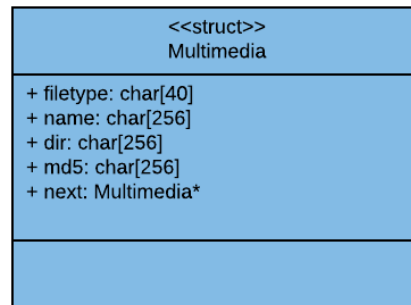
El servidor implementa las siguientes estructuras de datos.

4.1. Multimedia

4.1.1. Código

```
struct Multimedia{
    char filetype[40];
    char name[256];
    char dir[256];
    char md5[35];
    struct Multimedia *next;
};
```

4.1.2. Diagrama

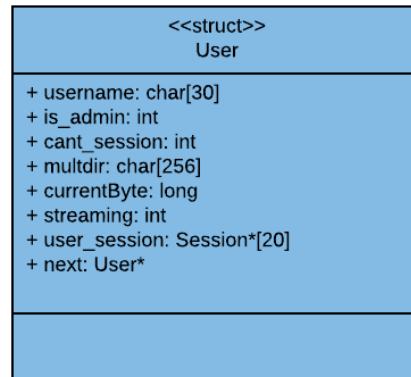


4.2. User

4.2.1. Código

```
struct User{
    char username[30];
    int is_admin;
    int cant_session;
    char multmdir[255];
    long currentByte;
    int streamming;
    struct Session *user_session[20];
    struct User *next;
};
```

4.2.2. Diagrama

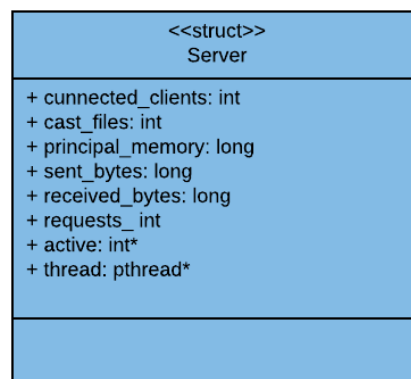


4.3. Server

4.3.1. Código

```
struct Server{
    int connected_clients;
    int cast_files;
    long principal_memory;
    long sent_bytes;
    long received_bytes;
    int requests;
    int *active;
    pthread_t thread;
};
```

4.3.2. Diagrama



4.4. Log

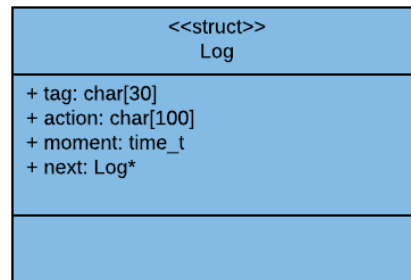
4.4.1. Código

```

struct Log{
    char tag[30];
    char action[100];
    time_t moment;
    struct Log *next;
};

```

4.4.2. Diagrama



4.5. Logger

4.5.1. Código

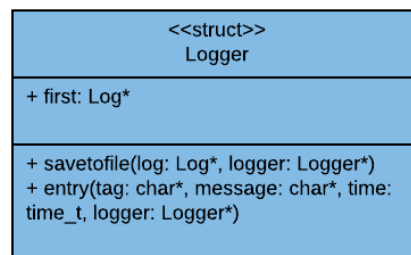
```

typedef int (*SaveOp)(struct Log*, struct Logger *logger);
typedef void (*newEntry)(char *tag, char *message, time_t time, struct Logger *logger)

struct Logger{
    struct Log *first;
    SaveOp savetofile;
    newEntry entry;
};

```

4.5.2. Diagrama



4.6. Folder

4.6.1. Código

```

typedef void (* addmult)(char *, char *, char *, char *, struct Folder *);

struct Folder{

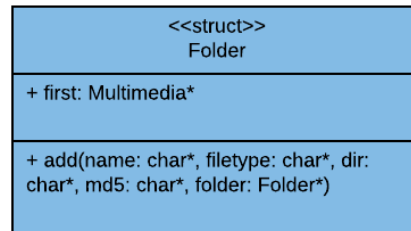
```

```

    struct Multimedia *first;
    addmult add;
};

```

4.6.2. Diagrama



4.7. User_queue

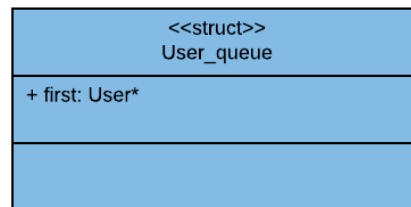
4.7.1. Código

```

struct User_queue{
    struct User *first;
};

```

4.7.2. Diagrama



4.8. Session

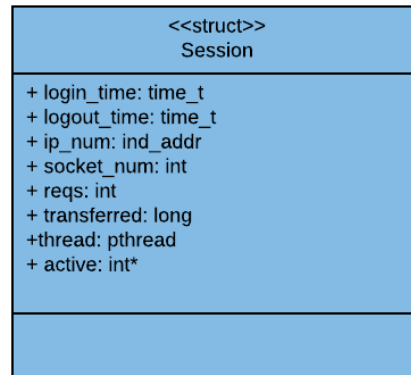
4.8.1. Código

```

struct Session{
    time_t login_time;
    time_t logout_time;
    struct in_addr ip_num;
    int socket_num;
    int reqs;
    long tranferred;
    pthread_t thread;
    int *active;
};

```


4.8.2. Diagrama

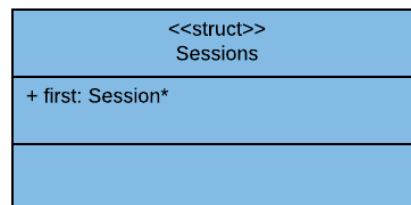


4.9. Sessions

4.9.1. Código

```
struct Sessions{
    struct Session *first;
}
```

4.9.2. Diagrama



5. Diseño de solución

5.1. Servidor

El servidor fue creado en C. Para la conexiones se utilizó un socket de tipo stream conectado a la red de internet. Cuando recibe un nuevo mensaje y este lo acepta la conexión se crea un hilo para manejar dicha conexión con la función **connection_handler(void *)**.

En la función **connection_handler(void *)** es la que se encarga de cada conexión particular, recibe y envía bajo el protocolo HTTP.

Estos mensaje que recibe lo divide en dos, los headers, el body; aquí evalúa que tipo de método HTTP que es y lo redirecciona a las funciones específicas.

LOGIN: Para esta función recibe un JSON con el formato `“username”:user,“password”:pass` con user y pass siendo los datos del usuario. Verifica la existencia de estos en el archivo `accounts.txt` y devuelve lo especificado en la sección `/login`. Si ya existe otra sesión asociada a este usuario. El

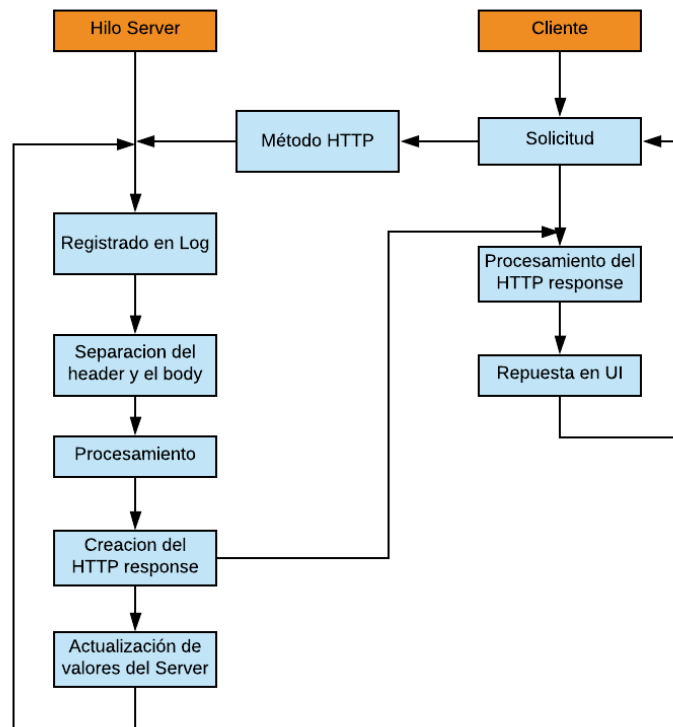


Figura 6: Diagrama general de la comunicación cliente/servidor

servidor cierra la conexión anterior.

REGISTER: Esta función es similar a LOGIN puesto que recibe lo mismo. Revisa si existe la cuenta y si no la escribe en el archivo accounts. Devuelve lo especificado en la sección /register

RES: Recibe la petición. El servidor cuenta con una instancia de los archivos multimedia en la carpeta. con la estructura Folder. Se genera un json de dicho archivo y se envía en el cuerpo de la respuesta.

GET FILE: Recibe la petición, evalúa si existe el archivo. determina si el cliente y el tipo de archivo son streamables, si es así realiza el proceso del streaming, esto se detalla más adelante; Si no es streamable, retorna todo el archivo por medio del socket.

STREAMING: para la realización del streaming, el server lo que posee son funciones para partir el archivo dado cierto rango. Entonces dada una petición de un archivo con un rango incluido, el servidor devolverá un chunk del archivo con los headers especificados en /res/[filename.ext]. Entonces recibe un chunk e información necesaria para la siguiente petición. También, el usuario en el servidor almacena el nombre del archivo y la posición del byte por el que lo pidió, de forma que si inicia sesión en otro dispositivo, esta nueva sesión recibe el la dirección del archivo y su posición actual para encargarse de realizar las peticiones.

SINCRONIZACIÓN: Para evitar la inconsistencia de datos, se utilizan semáforos, el cual realiza dos operaciones, `post` y `wait`.

LOG: Es una lista enlazada con la información del tag, el mensaje del log, el tiempo en que sucedió

5.2. Clientes

Para los clientes, se manejaron dos tipos: el cliente CLI en terminal, y el cliente web.

5.2.1. Cliente CLI

El cliente en terminal, por línea de comandos, se desarrolló en C sobre un sistema operativo Linux, según la especificación del proyecto. El cliente está hecho para compilar y ejecutar con el compilador `gcc`. Este contiene diferentes funciones que le permiten realizar la conexión al servidor, así como su funcionamiento principal con los archivos multimedia.

CONNECTSOCKET: Esta función es la encargada de hacer la conexión con el servidor. Primero se crea el socket y se verifica que no exista error alguno en su creación propiamente, luego se valida también que la conexión al servidor sea la correcta, en términos de dirección IP y puerto, y que no falle la conexión.

PARSEFOLDER: Esta función se encarga de tomar la lista de archivos multimedia que se recibe por respuesta del servidor, en el cuerpo de la misma, y se encarga de capturar cada campo de los archivos, (*name*, *filetype*, *dir*, *md5*), e irlos agregando a una estructura Folder, para poder utilizar dicha estructura de archivos a lo largo de la ejecución del cliente.

PRINTFOLDER: Esta función se encarga de tomar la estructura Folder creada con `parseFolder`, y proceder a imprimir en pantalla cada archivo de forma que sea entendible para el usuario, así como enumerarlos para que este pueda proceder a solicitar *x* archivo con introducir su número, y no su nombre completo, para mayor facilidad.

REQUESTNEXTCHUNK: Esta función es la encargada de solicitar al servidor por HTTP GET el siguiente fragmento del archivo multimedia que se está streaming en ese momento. Una vez obtenida la respuesta, se encarga de abrir el archivo ya existente, y seguir escribiendo los bytes sobre este archivo, **no** lo reescribe, sino que le hace un *append* al mismo.

Aparte de estas funciones principales, se tienen otras funciones auxiliares para el acomodo y visualización de datos.

Por otra parte, la función principal del cliente, el método **main**, es el que lleva la ejecución de acciones y respuestas entre el cliente y el servidor. La lógica de la función es primeramente declarar e inicializar todas las variables que se utilizarán en la ejecución de la misma. Seguidamente, se procede a verificar si la cantidad de parámetros que recibió el programa a la hora de compilar y ejecutar es la correcta, para poder asignar la dirección IP y el puerto asociados al servidor. Seguidamente, asumiendo que la conexión es exitosa, comienza un ciclo en el que el cliente puede empezar a enviar diferentes comandos al servidor para interactuar con este.

LOGIN: En caso de que el usuario ingrese el comando login, se le solicitará el usuario y contraseña. En caso de ser exitoso, el servidor le indicará esto y le pedirá otra opción; caso contrario le indicará que no resultó el inicio de sesión.

REGISTER: En caso de ingresar este comando, se le solicitará el usuario y contraseña para registrarse y el servidor le indicará si fue exitoso o no.

RES: Este comando le permite al usuario mandar a pedir al servidor toda la lista de archivos multimedia disponibles.

EXIT: Con este comando, el cliente desconecta el socket para cerrar la conexión con el servidor.

FILE: Este comando le indica al servidor que se solicitará un archivo. Se le despliega al cliente nuevamente la lista completa de archivos, para que este no tenga que volver a buscarlos, y se le pide que ingrese el número del archivo que desea solicitar al servidor.

STREAMFILE: Este otro comando le indica al servidor que se solicitará un archivo streamable, y la funcionalidad principal es igual a la del comando **file**.

5.2.2. Cliente Web

El cliente web está hecho en Angular, y está conformado estructuralmente por diferentes componentes, módulos, servicios y modelos.

Dentro de los componentes, existen unos que únicamente se encargarán de desplegar datos constantes, como lo son el encabezado y el pie de página, los cuales son consistentes dentro de toda la aplicación. Por otra parte, se tienen otros componentes encargados de la funcionalidad de la misma.

CONTENTVIEWER: Este componente contiene un botón para la navegación entre archivos, el cual se utiliza para retroceder.

HOME: Este componente es el encargado de presentar en la pantalla un espacio inicial, la pantalla de Home, sin ser esta la pantalla principal. Aquí es donde se tiene la opción tanto de registrarse como de iniciar sesión en la aplicación.

IP: El componente IP se encarga de capturar los datos de dirección IP y puerto del componente Login, para poder utilizarlos a la hora de hacer la conexión al servidor.

LOGIN: El componente de login, como su nombre lo indica, es el que se encarga de presentarle al usuario el formulario para iniciar sesión. Este presenta un campo para el nombre de usuario,

un campo para la contraseña, así como también un campo con la dirección IP y el puerto para conectarse al servidor.

MAIN: Este es el componente principal luego de iniciar sesión. Main se encarga de presentar al usuario la información de qué archivos multimedia existen disponibles en el servidor, y es el que se encarga de la principal interacción entre el cliente y el servidor. Aquí se presentan en pantalla los archivos y sus streamings respectivos.

REGISTER: Este componente es el encargado de presentarle al usuario el formulario para registrar un nuevo usuario en el servidor y poder hacer uso de la aplicación, luego de registrarse e iniciar sesión.

STREAMVIEWER: Este componente funciona como un contenedor para el streaming de videos. Aquí es donde se reproducen estos archivos de tipo .mp4, dentro del componente Main.

En lo que respecta a los módulos propios creados para la aplicación, se utilizó solo uno, llamado **app-routing** module, encargado de la navegación de la aplicación. Angular trae por default su propio router, por lo que a la hora de establecer la navegación, es necesario únicamente implementar dicho módulo y proporcionarle el archivo con las rutas de los componentes.

Por otra parte, para los servicios de la aplicación, se cuentan cinco (5) diferentes servicios.

CONTENT: Este servicio no hace llamadas al servidor directamente, pero es el encargado de crear una estructura con los atributos necesarios para utilizar en el componente Main a la hora de desplegar y solicitar los archivos multimedia al servidor.

LOGIN: Este servicio es el encargado de hacer todos los llamados al servidor relacionados al inicio de sesión de usuarios. Contiene dos métodos: **loginClient** el cual se encarga de capturar los datos de usuario y contraseña, y hacer el llamado respectivo al servidor con el método HTTP POST y validar la respuesta del mismo. Y también existe el método **logoutClient** el cual se encarga de mandar la solicitud HTTP GET al servidor para cerrar la conexión al socket.

MEDIA: Este servicio se encarga de todas las solicitudes al servidor relacionadas con los archivos multimedia. Igualmente contiene dos métodos, **requestFile**, el cual se encarga de mandar la solicitud HTTP GET al servidor con el nombre del archivo y la posición en bytes por la que se encuentra, y el otro método **getAllFiles**, el cual manda la solicitud HTTP GET para pedir toda la lista de archivos disponibles en el servidor.

PROCESS-HTTPMSG: Este servicio se encarga de capturar los mensajes de error, en el eventualidad de que alguno llegara a suceder, y se encarga de presentarlos en la pantalla de manera que sean entendibles para el usuario activo. El método **handleError** es el encargado de llevar a cabo dicha acción. Este servicio no realiza ningún tipo de llamadas al servidor.

REGISTER: Este servicio se encarga de las llamadas al servidor con respecto al registro de clientes. Su método principal **registerClient** recibe el usuario y contraseña escogidos por el usuario y se encarga de crear el cuerpo y encabezado del HTTP POST para mandar la solicitud de crear un

nuevo usuario al servidor.

En cuanto a los modelos utilizados, se cuenta con tres (3) diferentes modelos implementados. El primero es una interface **Deserializable**, la cual busca la implementación del método **deserialize**. El segundo modelo implementado es el modelo **Multimedia**, encargado de implementar Deserializable y crear el archivo multimedia a partir de los atributos *name*, *filetype*, *dir* y *md5*. El último modelo, **Res**, permite hacer una lista de objetos multimedia para utilizar dentro de la aplicación.

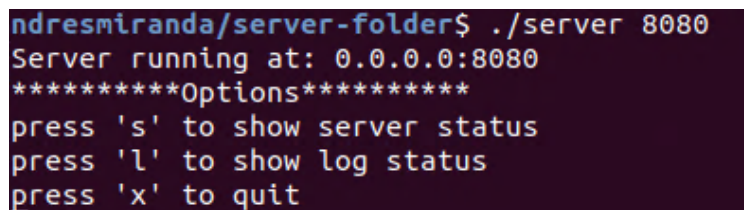
6. Pruebas de ejecución

6.1. Servidor

Para validar el funcionamiento del servidor, es necesario primero compilar y ejecutarlo. para esto, se ingresan los comandos

```
$ make
$ ./server [port_number]
```

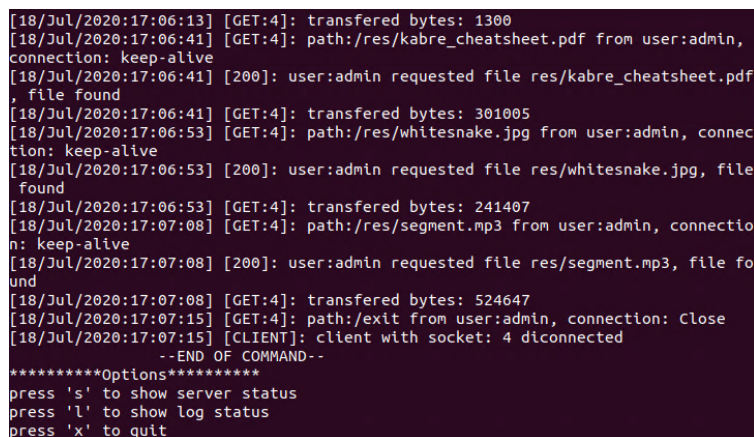
En caso de conectar correctamente, el servidor desplegará en consola que está conectado, y las opciones para consultarlo a él directamente.



```
ndresmiranda/server-folder$ ./server 8080
Server running at: 0.0.0.0:8080
*****Options*****
press 's' to show server status
press 'l' to show log status
press 'x' to quit
```

Figura 7: Conexión exitosa del servidor

Como se observa en la la figura 7, se despliega en consola que el servidor está corriendo en *0.0.0.0:8080*, y también muestra qué opciones se pueden seleccionar para consultar el estado y log del mismo.



```
[18/Jul/2020:17:06:13] [GET:4]: transfered bytes: 1300
[18/Jul/2020:17:06:41] [GET:4]: path:/res/kabre_cheatsheet.pdf from user:admin,
connection: keep-alive
[18/Jul/2020:17:06:41] [200]: user:admin requested file res/kabre_cheatsheet.pdf
, file found
[18/Jul/2020:17:06:41] [GET:4]: transfered bytes: 301005
[18/Jul/2020:17:06:53] [GET:4]: path:/res/whitesnake.jpg from user:admin, connec
tion: keep-alive
[18/Jul/2020:17:06:53] [200]: user:admin requested file res/whitesnake.jpg, file
found
[18/Jul/2020:17:06:53] [GET:4]: transfered bytes: 241407
[18/Jul/2020:17:07:08] [GET:4]: path:/res/segment.mp3 from user:admin, connectio
n: keep-alive
[18/Jul/2020:17:07:08] [200]: user:admin requested file res/segment.mp3, file fo
und
[18/Jul/2020:17:07:08] [GET:4]: transfered bytes: 524647
[18/Jul/2020:17:07:15] [GET:4]: path:/exit from user:admin, connection: Close
[18/Jul/2020:17:07:15] [CLIENT]: client with socket: 4 disconnected
--END OF COMMAND--
*****Options*****
press 's' to show server status
press 'l' to show log status
press 'x' to quit
```

Figura 8: Ejemplo del log del servidor

En caso de escoger la opción 'l', se le mostrará al usuario un log, como se observa en la figura 8 de eventos ocurridos durante esa conexión en el servidor, como inicios de sesión, solicitudes de archivos, registros de nuevos usuarios, entre otros.

Por último, se puede observar en la figura 9, un ejemplo de qué ocurre en caso de escoger la opción 's' del servidor. En este caso particular, se le mostrará al usuario el estatus del servidor, conteniendo este un registro de la cantidad de clientes conectados al momento, la cantidad de solicitudes recibidas, y la cantidad de datos enviados y recibidos.

```
Active clients: 0
Total requests: 6
Data received: 630 Bytes
Data sent: 1.02 MB
Cast files: 0
--END OF COMMAND--
*****Options*****
press 's' to show server status
press 'l' to show log status
press 'x' to quit
```

Figura 9: Ejemplo del status del servidor

6.2. Cliente CLI

La ejecución del cliente por línea de comandos, o terminal, es muy similar al servidor. Para ejecutarlo, se deben ingresar los siguientes comandos para compilar y ejecutar

```
$ make
$ ./client [ip_address] [port_number]
```

Una vez realizada dicha conexión, en caso de conectar correctamente, se le desplegará al cliente en la consola una información como la que se puede observar en la figura 10.

```
ndresmiranda/CLient$ ./client 172.17.92.118 8080
login: Iniciar esion
register: Registrar nuevo usuario
res: Solicitar lista de archivos
file: Solicitar archivo
streamfile: Solicitar archivo multimedia
exit: Cerrar sesión

Escriba una opcion: █
```

Figura 10: Conexión exitosa del cliente

El flujo de acciones de este cliente van a depender de los comandos ingresados, según las opciones desplegadas disponibles.

6.2.1. Login

Si el cliente ingresa la opción *login*, se le solicitará tanto usuario como contraseña antes de poder enviar la solicitud HTTP POST al servidor. En caso de ser un usuario existente, y las credenciales sean las correctas, el servidor le desplegará al usuario el mensaje de éxito en el inicio de sesión, y se seguirá pidiendo más opciones para continuar, como se observa en la figura 11.

En caso de que el usuario no exista, o las credenciales estén incorrectas, se desplegará el mensaje de error en el inicio de sesión, y deberá volver a intentarlo.

```
Escriba una opcion: login
Enter username: admin
Enter password: 1234
Login Successful
login: Iniciar esion
register: Registrar nuevo usuario
res: Solicitar lista de archivos
file: Solicitar archivo
streamfile: Solicitar archivo multimedia
exit: Cerrar sesión

Escriba una opcion: █
```

Figura 11: Inicio de sesión - Cliente CLI

6.2.2. Register

Si se quisieran registrar nuevos clientes, luego de conectar al servidor, se debe ingresar el comando *register*, el cual se solicitará un nombre de usuario y contraseña para validar el registro. En caso de ser exitoso, se desplegará un mensaje de éxito, caso contrario será de error en el registro, como se observa en la figura 12.

```
Escriba una opcion: register
Enter username: admin2
Enter password: test
Registration Successful
login: Iniciar esion
register: Registrar nuevo usuario
res: Solicitar lista de archivos
file: Solicitar archivo
streamfile: Solicitar archivo multimedia
exit: Cerrar sesión
```

Figura 12: Registro de usuario - Cliente CLI

6.2.3. Res

Para que el cliente pueda visualizar la lista de archivos multimedia dentro del servidor al momento, debe ingresar el comando *res*, el cual se encarga de enviar al servidor la solicitud HTTP

GET y le devuelve la lista de archivos, con su respectivo número, como se ilustra en la figura 13.

```

Escriba una opcion: res
1341 < 1341
1. File: asi_fue Filetype: video/mp4
2. File: Tarea1_rotacion Filetype: video/mp4
3. File: asifue Filetype: audio/mpeg
4. File: kabre_cheatsheet Filetype: application/pdf
5. File: whitesnake Filetype: image/jpg
6. File: quarantine Filetype: image/jpg
7. File: gold Filetype: audio/mpeg
8. File: homeless Filetype: video/mp4
9. File: Loyal Filetype: audio/mpeg
10. File: loveit Filetype: audio/mpeg
11. File: 258 Filetype: audio/mpeg
12. File: help Filetype: text/plain
13. File: segment Filetype: audio/mpeg

```

Figura 13: Solicitud de archivos disponibles - Cliente CLI

6.2.4. File

Si el cliente quisiera solicitar archivos al servidor, que no sean archivos de audio, es decir, que no se puedan reproducir directamente en la consola, debe ingresar la opción *file*. La respuesta a esta solicitud es volver a desplegarle la lista de archivos disponibles, para que el cliente no tenga que estar al pendiente de qué archivo lleva cuál número. Seguidamente, deberá ingresar el número de archivo que quiere solicitar, como se puede observar en la figura 14.

En caso de no haber cargado la estructura del folder conteniendo la lista de archivos, si ingresa *file*, este comando procederá a indicarle al cliente que debe hacer la solicitud *res* para cargarlo.

```

Escriba una opcion: file
1. File: asi_fue Filetype: video/mp4
2. File: Tarea1_rotacion Filetype: video/mp4
3. File: asifue Filetype: audio/mpeg
4. File: kabre_cheatsheet Filetype: application/pdf
5. File: whitesnake Filetype: image/jpg
6. File: quarantine Filetype: image/jpg
7. File: gold Filetype: audio/mpeg
8. File: homeless Filetype: video/mp4
9. File: Loyal Filetype: audio/mpeg
10. File: loveit Filetype: audio/mpeg
11. File: 258 Filetype: audio/mpeg
12. File: help Filetype: text/plain
13. File: segment Filetype: audio/mpeg

Enter file number: 4
kabre_cheatsheet.pdf Downloaded Successfully

```

Figura 14: Solicitud para descargar un archivo - Cliente CLI

6.2.5. Streamfile

Por lo contrario, si el cliente desea reproducir un archivo de audio, debe escoger la opción *streamfile*, para poder indicarle al servidor que se solicitará un archivo .mp3, y se deberá repro-

ducir en la consola directamente. De igual manera, se le despliega al cliente la lista de archivos disponibles, como se puede observar en la figura 15, para que pueda ingresar el número del archivo que desea.

Una vez que se comienza a reproducir el archivo, se le despliegan dos opciones al cliente, 'p' para detener o continuar con la reproducción, y la opción 'x' para cancelar la reproducción y volver al menú principal.

Por otra parte, si el cliente está reproduciendo el archivo de audio en una sesión, y se llegara a conectar en otra sesión nueva, la reproducción de audio continuaría en el punto en el que se dejó, esto de manera automática.

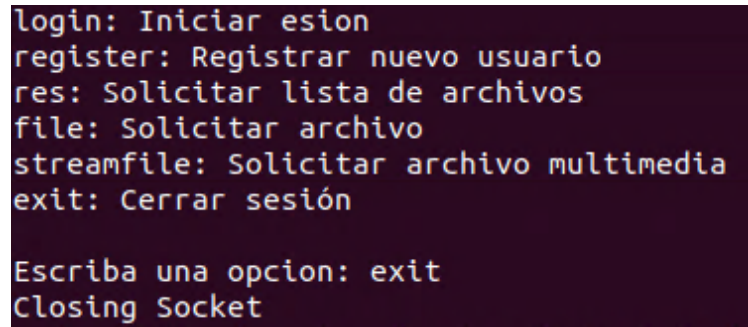
```
Escriba una opcion: streamfile
3. File: asifue Filetype: audio/mpeg
7. File: gold Filetype: audio/mpeg
9. File: Loyal Filetype: audio/mpeg
10. File: loveit Filetype: audio/mpeg
11. File: 258 Filetype: audio/mpeg
13. File: segment Filetype: audio/mpeg

Enter file number: 3
*****res/asifue.mp3*****
press 'p' to play/pause
press 'x' to quit
```

Figura 15: Reproducción de audio - Cliente CLI

6.2.6. Exit

Si el cliente quisiera cerrar la conexión con el socket del servidor, únicamente debe ingresar el comando *exit*, el cual se encargará de enviar la solicitud HTTP GET al servidor, indicando que debe cerrar la conexión, como se observa en la figura 16.

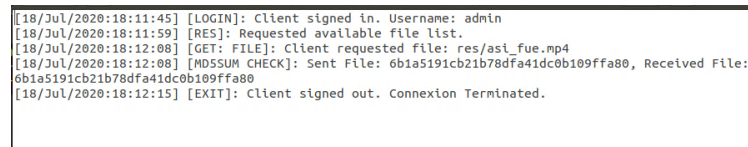


```
login: Iniciar esion
register: Registrar nuevo usuario
res: Solicitar lista de archivos
file: Solicitar archivo
streamfile: Solicitar archivo multimedia
exit: Cerrar sesión

Escriba una opcion: exit
Closing Socket
```

Figura 16: Cerrar sesión - Cliente CLI

Al mismo tiempo, al cerrar la sesión con este comando, se le generará al cliente un log el cual lleva el registro de todas sus transacciones con el servidor para esa sesión en particular; cada sesión genera un nuevo log, como se observa en la figura 17



```
[18/Jul/2020:18:11:45] [LOGIN]: Client signed in. Username: admin
[18/Jul/2020:18:11:59] [RES]: Requested available file list.
[18/Jul/2020:18:12:08] [GET: FILE]: Client requested file: res/asi_fue.mp4
[18/Jul/2020:18:12:08] [MD5SUM CHECK]: Sent File: 6b1a5191cb21b78dfa41dc0b109ffa80, Received File:
6b1a5191cb21b78dfa41dc0b109ffa80
[18/Jul/2020:18:12:15] [EXIT]: Client signed out. Connexion Terminated.
```

Figura 17: Log de transacciones - Cliente CLI

6.3. Cliente Web

El cliente web va a ser, evidentemente, mucho más interactivo con el usuario que el cliente por línea de comando. Al estar hecho en Angular, este framework trae por defecto el puerto 4200, por lo que, en caso de no abrir automáticamente, debe conectarse a **localhost:4200** para poder tener acceso al cliente. Una vez ahí, el cliente podrá ver la siguiente imagen, como se muestra en la figura 18, para poder ya sea registrarse como usuario, o iniciar sesión en caso de ya contar con uno.

Para poder correrlo, se deberá ejecutar el siguiente comando en una terminal, directamente en el root del proyecto

```
npm start
```

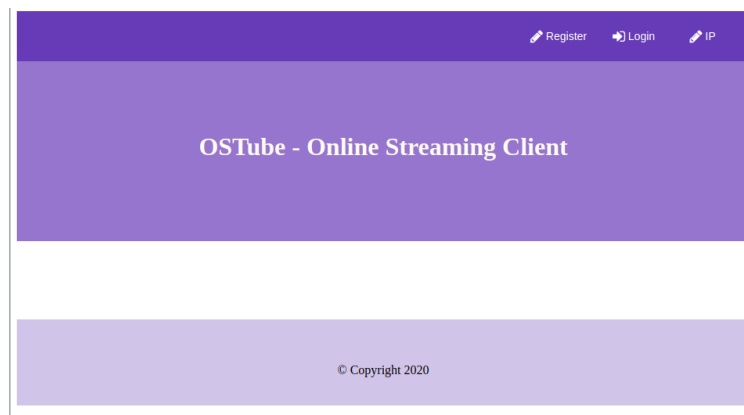
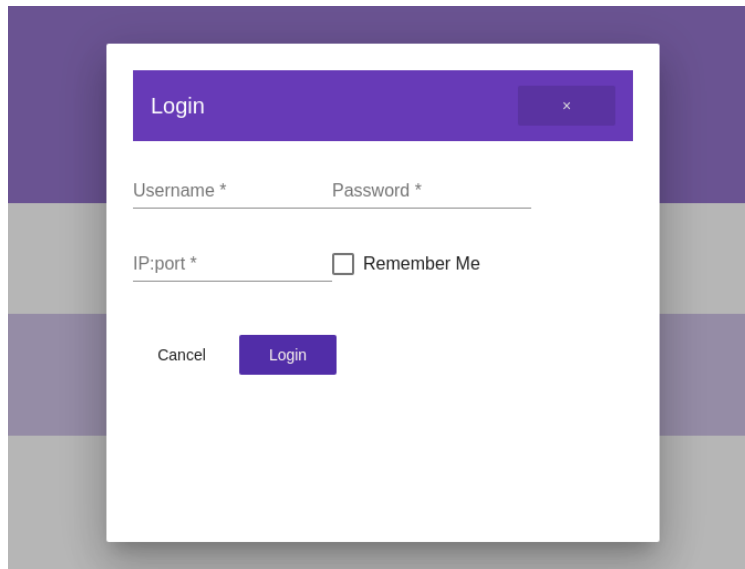


Figura 18: Pantalla principal - Cliente web

6.3.1. Login

En caso de ya contar con un usuario previamente creado, el cliente puede clicar la opción *login*, con la cual se le desplegará un modal con un formulario para ingresar tanto las credenciales, como la dirección IP y el puerto en el que está corriendo el servidor, como se observa en la figura 19.

A login modal form with a purple header bar containing the text "Login" and a close button (X). The form has three input fields: "Username *" and "Password *" on the first line, and "IP:port *" on the second line. To the right of the "IP:port *" field is a checkbox labeled "Remember Me". At the bottom left is a "Cancel" button, and at the bottom right is a purple "Login" button.

Login

Username * Password *

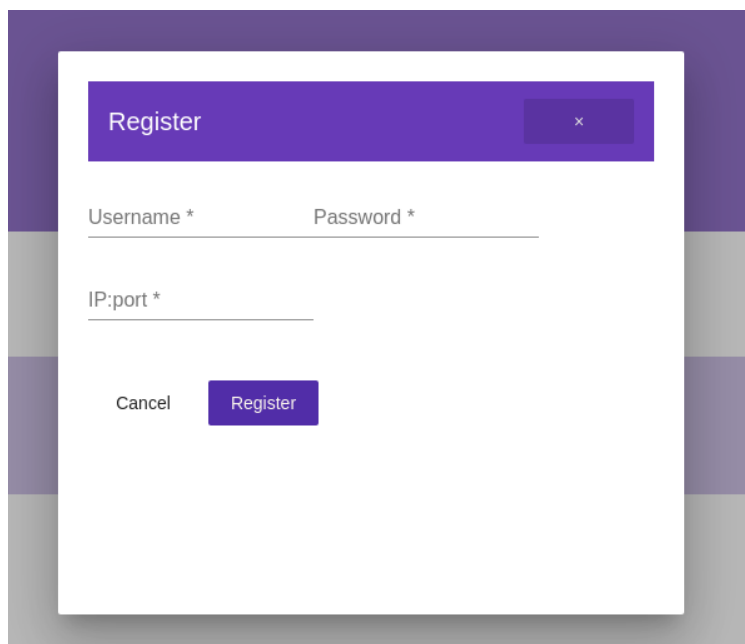
IP:port * ☐ Remember Me

Cancel Login

Figura 19: Inicio de sesión - Cliente web

6.3.2. Register

Por otra parte, si el cliente no cuenta con un usuario, puede clicar la opción *register*, la cual le desplegará otro modal, el cual solicitará la información requerida para crear un usuario y podrá iniciar sesión seguidamente, como se puede ver ilustrado en la figura 20.

A register modal form with a purple header bar containing the text "Register" and a close button (X). The form has three input fields: "Username *" and "Password *" on the first line, and "IP:port *" on the second line. At the bottom left is a "Cancel" button, and at the bottom right is a purple "Register" button.

Register

Username * Password *

IP:port *

Cancel Register

Figura 20: Registro de usuarios - Cliente web

6.3.3. Streaming de archivos

Una vez ingresado en la aplicación, con la debida autenticación, el cliente tiene la opción para acceder a los diferentes archivos multimedia, de una manera más interactiva. El cliente podrá observar en la pantalla principal de la aplicación, una lista con todos los archivos disponibles en el servidor para que este pueda utilizarlos, como se puede observar en la figura 21.

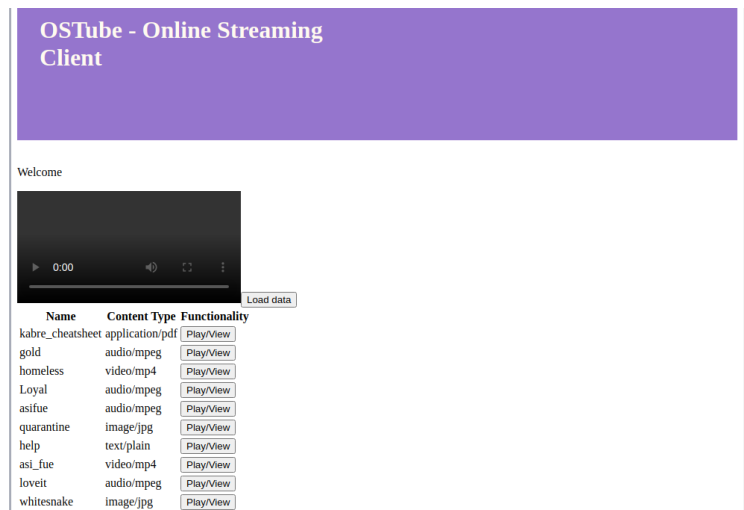


Figura 21: Pantalla principal - Cliente web

En el caso de los archivos de texto, es decir, con extensión *.txt*, estos se le despliegan al cliente en pantalla directamente, con el título y contenido de este, como se puede observar en la figura 22.

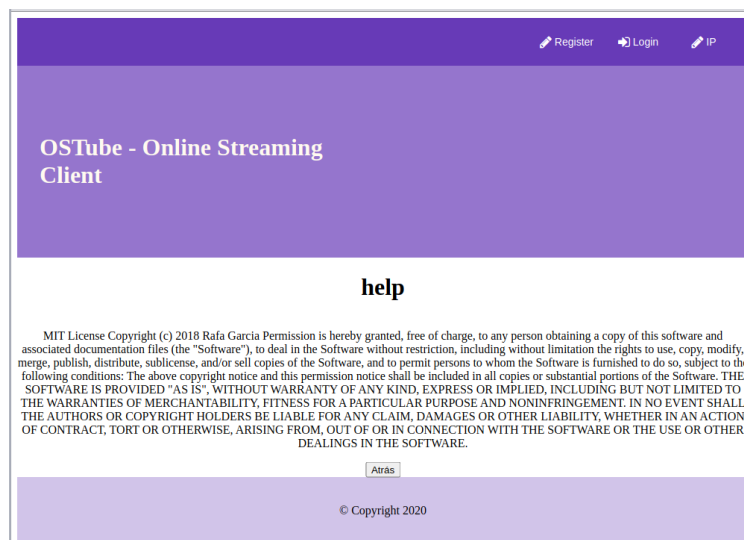


Figura 22: Archivos de texto - Cliente web

Con respecto a los archivos *.pdf*, el cliente igual puede visualizarlos, pero como se acostumbra normalmente, se le desplegará el archivo completo en una nueva pestaña de su buscador, teniendo acceso completo a este tanto para lectura como para descargarlo, como se ilustra en la figura 23.

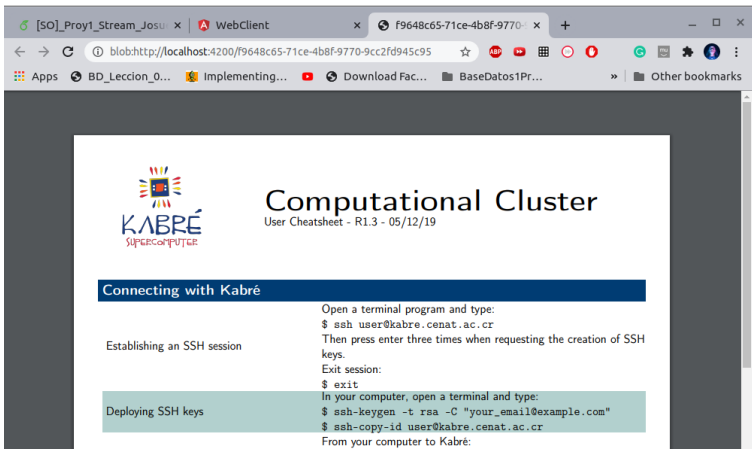


Figura 23: Archivos PDF - Cliente web

En el caso de los archivos streamable, es decir, aquellos archivos de audio *mp3* y vídeo *mp4*, que se pueden reproducir directamente en el cliente, se pueden visualizar en el campo designado para ello.

Primeramente, con los archivos de audio, el cliente podría seleccionar uno, y lo podría visualizar en el cliente directamente, pudiendo reproducirlo con los controladores respectivos para pausar y continuar la misma, como se ilustra en la figura 24.

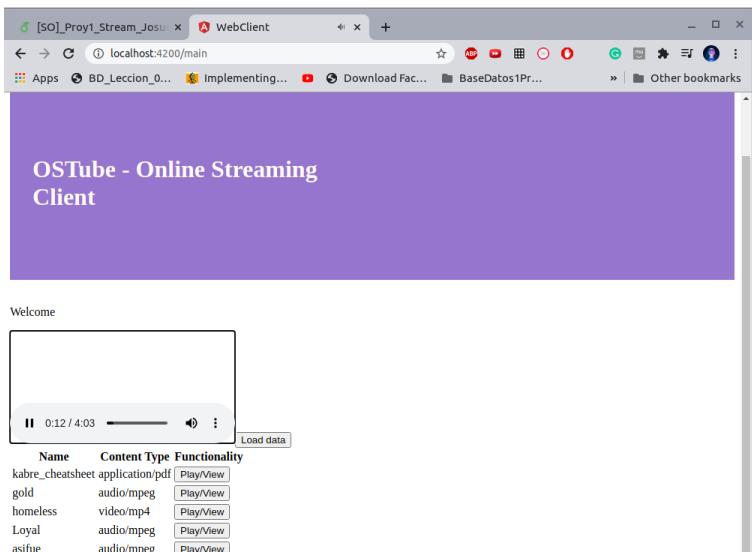


Figura 24: Archivos de audio - Cliente web

Segundo, en lo que respecta al segundo tipo de archivo streamable, es decir, los archivos de vídeo, el cliente puede solicitarlos y de igual manera, reproducirlos directamente en la aplicación, sin necesidad de descargarlo, a diferencia del cliente CLI.

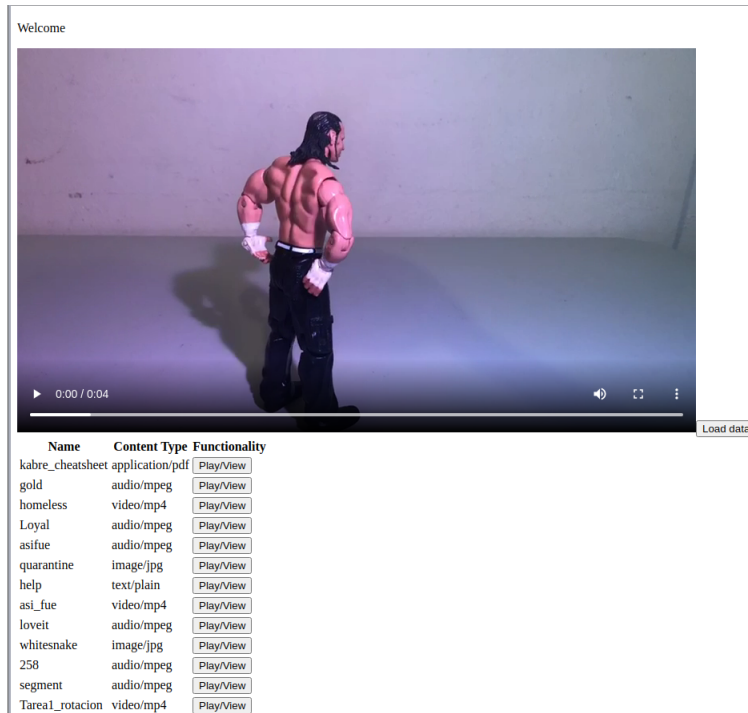


Figura 25: Archivos de vídeo - Cliente web

En cuanto a las imágenes, si el cliente solicitara una imagen, se le desplegaría en pantalla para que este pudiese visualizarla, como se observa en la figura 26¹.

whitesnake



Figura 26: Imágenes - Cliente web

¹Aclarar que se recorta la imagen para el apropiado acomodo en el documento, sin embargo, nótese que el nombre *whitesnake* está referenciado en la figura 25.

7. Comentarios y observaciones

Para la comunicación del cliente web con el servidor, se tuvo que adaptar el servidor ya que al querer hacer una petición, el cliente realizaba una petición de tipo OPTIONS para verificar si el servidor permite el tipo de petición que se quiere mandar. Esto es debido al Cross-Origin Resource Sharing (CORS) que viene integrado en los navegadores.

Para la implementación del streaming en el cliente web, se necesitaba crear un MediaSource para enlazarlo con el elemento de HTML5 video. Este MediaSource posee un SourceBuffer que es al que se le van agregando los bytes del archivo. Las funciones para el manejo de este comportamiento fueron escritas, sin embargo, a la hora de agregar los bytes con la función SourceBuffer.appendByffer(ArrayBuffer) este mandaba un mensaje diciendo que se desvinculó del MediaSource o también decía que el MediaSource estaba indefinido, de manera aleatoria. Por lo que no se pudo implementar de manera eficaz el streaming por segmentos aún el server teniendo la capacidad de soportar este comportamiento.

8. Conclusiones

En conclusión, una vez finalizado el proyecto, se logró cumplir con los objetivos y requerimientos planteados al inicio. Si bien presentó un reto importante, principalmente a la hora del manejo y envío de archivos desde el servidor hacia el cliente, y su debida reproducción en streaming, se logró hallar la forma satisfactoria de completar dicha tarea.

De igual manera, se lograron corregir varios errores en el camino, con el envío de archivos, ya que inicialmente a veces se enviaban completos, a veces se enviaban parcialmente, u otras veces llegaban archivos corruptos, sin conocimiento realmente del por qué de esta situación. No obstante, fue posible completar esta tarea, de manera satisfactoria igualmente, para poder completar la verificación del MD5sum de los archivos, así como su correcta reproducción y/o descarga según la solicitud del cliente.

En lo que respecta particularmente con los objetivos iniciales del proyecto, planteados en la especificación del mismo, se logró realizar el servidor en lenguaje C, permitiendo la conexión por medio de conexiones HTTP, creando diferentes estructuras para almacenar los archivos, manejando sesiones y conexiones con sockets y threads. En resumen, el servidor se logró completar a un 100% de lo estipulado, de acuerdo al conocimiento de los integrantes de este proyecto.

Por otra parte, en respecto al cliente por línea de comandos, se logró cumplir con lo estipulado en este apartado; el cliente se desarrolló en lenguaje C para correr en un sistema operativo Linux, manejando los sockets para la conexión con el servidor. También, se logró manejar correctamente las sesiones, así como las solicitudes HTTP al servidor, y el streaming de los archivos multimedia.

Así, con el cliente web, realizado en Angular, también se logró completar a un 100% de su totalidad tomando como referencia el enunciado, permitiéndole al cliente ingresar, registrar usuarios, manejar los archivos y poder realizar el streaming de los archivos multimedia que lo permitieran, o la descarga en su debido caso.

Para finalizar, se considera que se logró completar todo el proyecto según lo estipulado, de acuerdo al conocimiento de los integrantes del mismo, quedando satisfechos con el servidor, cliente CLI, cliente web, así como también con los resultados de las pruebas de los mismos.

9. Referencias

Referencias

- [1] N. Matthew and R. Stones, Beginning Linux Programming. Hoboken: John Wiley & Sons, Inc., 2011.
- [2] T. Ashen Gamage, "Evolution of HTTP — HTTP/0.9, HTTP/1.0, HTTP/1.1, Keep-Alive, Upgrade, and HTTPS", Medium, 2018. [Online]. Available: <https://medium.com/platform-engineer/evolution-of-http-69cfe6531ba0>
- [3] B. index, M. Support and S. Applications, "C - Where is md5.h - Linux Mint Forums", Forums.linuxmint.com, 2015. [Online]. Available: <https://forums.linuxmint.com/viewtopic.php?t=187683>.
- [4] "POSIX", Es.wikipedia.org, 2020. [Online]. Available: <https://es.wikipedia.org/wiki/POSIX>.
- [5] "JSON", Json.org, 2020. [Online]. Available: <https://www.json.org/json-en.html>
- [6] "Makefile", En.wikipedia.org, 2020. [Online]. Available: <https://en.wikipedia.org/wiki/Makefile>
- [7] E. Arias, Proyecto # 1 Servidor de Streaming. 2020.
- [8] Google Angular Team, "Angular", Angular.io. [Online]. Available: <https://angular.io/docs>. [Accessed: 16- Jul- 2020].
- [9] Doxygen, "libmad: Low-Level API", M.baert.free.fr, 2008. [Online]. Available: <http://m.baert.free.fr/contrib/docs/libmad/doxy/html/low-level.html>. [Accessed: 18- Jul- 2020].
- [10] L. Vösandi, "Lauri's blog — Implementing MP3 player in C", Lauri.xn-vsandi-pxa.com, 2013. [Online]. Available: <https://lauri.xn-vsandi-pxa.com/2013/12/implementing-mp3-player.en.html>. [Accessed: 18- Jul- 2020].
- [11] "MPEG Audio Decoder", En.wikipedia.org, 2020. [Online]. Available: https://en.wikipedia.org/wiki/MPEG_Audio_Decoder. [Accessed: 18- Jul- 2020].