Escuela Politécnica Superior de Linares

Grado en Ingeniería Telemática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Linares

Trabajo Fin de Grado

APLICACIÓN DE STREAMING DE AUDIO INMERSIVO PARA CONCIERTOS DE MÚSICA CLÁSICA

Alumno: **Jaime García Martínez**

Tutores: José Manuel Pérez Lorenzo
José Enrique Muñoz Expósito
Depto.: Ingeniería de Telecomunicación

Junio, 2023

Universidad de Jaén
Escuela Politécnica Superior de Linares

Trabajo Fin de Grado

APLICACIÓN DE STREAMING DE AUDIO INMERSIVO PARA CONCIERTOS DE MÚSICA CLÁSICA

Alumno: **Jaime García Martínez**
Tutores: José Manuel Pérez Lorenzo
José Enrique Muñoz Expósito
Dpto.: Ingeniería de Telecomunicación

Junio, 2023

Firma del autor:

GARCIA MARTINEZ JAIME - ***8705**
21/06/2023

Firma del tutor 1:

PEREZ LORENZO JOSE MANUEL - 23807798S
Firmado digitalmente por PEREZ LORENZO JOSE MANUEL - 23807798S
Fecha: 2023.06.20 21:11:55 +02'00'

Firma del tutor 2:

MUÑOZ EXPOSITO JOSE ENRIQUE - 27386513E
Firmado digitalmente por MUÑOZ EXPOSITO JOSE ENRIQUE - 27386513E
Nombre de reconocimiento (DN): c=ES, serialNumber=IDCES-27386513E, givenName=JOSE ENRIQUE, sn=MUÑOZ EXPOSITO, cn=MUÑOZ EXPOSITO JOSE ENRIQUE - 27386513E
Fecha: 2023.06.21 10:36:21 +02'00'

ÍNDICE DE CONTENIDOS

1. SIGLAS, ACRÓNIMOS Y DEFINICIONES 1

2. RESUMEN 4

3. INTRODUCCIÓN 6

3.1. DESCRIPCIÓN DEL PROBLEMA 6

3.1.1. Disposición de micrófonos para la grabación de la actuación 6

3.1.2. Número de señales de audio disponibles 7

3.1.3. Características de las señales de audio 7

3.2. ALCANCE 8

3.3. ANTECEDENTES 8

4. OBJETIVOS 10

5. REVISIÓN DEL ESTADO DEL ARTE 11

5.1. HLS 12

5.1.1. Visión general 12

5.1.2. Media Playlist 12

5.1.3. Master Playlist 14

5.1.4. Contenedores y códecs compatibles 17

5.2. MPEG-DASH 17

5.2.1. Visión general 17

5.2.2. Modelo de datos 18

5.2.3. Documentos MPD 20

5.2.4. Contenedores y códecs compatibles 22

6. MATERIALES Y MÉTODOS 23

6.1. MATERIALES 23

6.1.1. FFmpeg 23

6.1.2. Node.js 26

6.1.3. Express.js 27

6.1.4. MongoDB 28

6.1.5. Mongoose 29

6.1.6. React 29

6.1.7. Hls.js 32

6.1.8. Dash.js *33*
6.1.9. Web Audio API *34*
6.1.10. Docker *36*
6.1.11. Especificaciones del equipo de pruebas *38*
6.1.12. Base de datos de HRTFs del CIPIC *39*
6.1.13. Git y GitHub *40*
6.1.14. Software adicional *40*
6.2. MÉTODOS 41
6.2.1. Consideraciones para el procesado de las señales de audio *41*
6.2.2. Transmisión de audio procedente de distintas posiciones *42*
6.2.3. Transmisión de audio de distintos instrumentos en una posición *42*
6.2.4. Elección del protocolo de streaming *44*
6.2.5. Espacialización de las señales de audio *45*
6.2.6. Almacenamiento y envío de HRTFs *47*
6.2.7. Adaptación de convenio de coordenadas *48*
6.2.8. Remuestreo de HRTFs *50*
6.2.9. Ajuste de localizaciones tras aplicar rotaciones *51*
6.2.10. Control de versiones *51*
7. DESARROLLO 53
7.1. ARQUITECTURA DEL SERVICIO 53
7.2. COLECCIONES DE BASE DE DATOS 57
7.3. CÓDEC DE AUDIO Y FORMATO CONTENEDOR MULTIMEDIA 59
7.4. TRANSMISIÓN DE CONTENIDO MULTIMEDIA 61
7.4.1. Ingesta de datos *61*
7.4.2. Difusión del contenido multimedia *65*
7.5. ENVÍO DE DATOS ADICIONALES DE UNA PRESENTACIÓN MULTIMEDIA 66
7.5.1. API REST para obtener la información de todas las presentaciones . *67*
7.5.2. API REST para almacenar HRTFs en la base de datos *68*
7.5.3. API REST para obtener un conjunto de HRTFs correspondientes a las localizaciones de los instrumentos de una actuación *68*
7.5.4. API REST para obtener un conjunto de HRTFs correspondientes a las localizaciones de los instrumentos de una actuación tras aplicar una rotación *69*

7.6. CONSUMO DE CONTENIDO MULTIMEDIA Y PROCESADO DE AUDIO EN EL FRONT-END 71

7.6.1. Cadena de audio ... 71

7.6.2. Componentes de la interfaz del front-end ... 75

7.6.3. Funcionamiento de la interfaz de usuario ... 77

8. RESULTADOS Y DISCUSIÓN ... 82

8.1. FUNCIONAMIENTO DE LA APLICACIÓN ... 82

8.2. ANCHO DE BANDA REQUERIDO ... 90

9. CONCLUSIONES ... 93

9.1. VENTAJAS DE LA SOLUCIÓN PROPUESTA ... 93

9.2. LIMITACIONES ... 94

9.3. FUTURAS LÍNEAS DE DESARROLLO ... 94

10. REFERENCIAS ... 96

ANEXO I: MANUAL DE DESPLIEGUE DEL SERVICIO ... 99

ÍNDICE DE FIGURAS

Figura 3.1. Disposición de micrófonos para la grabación de la actuación. Esquema de distribución de instrumentos extraído de [1]. 7

Figura 3.2. Número de suscripciones y reparto de mercado entre los servicios de streaming de música en el segundo trimestre de 2022 (extraída de [3]) 9

Figura 5.1. Ejemplo de una Media Playlist. 13

Figura 5.2. Ejemplo de una Master Playlist con variantes y renditions (extraída de [8]). 16

Figura 5.3. Esquema del modelo de datos de DASH (extraída de [9]). 18

Figura 5.4. Ejemplo de un documento MPD que describe una presentación multimedia en directo. 20

Figura 6.1. Generación de streaming HLS en directo con FFmpeg. 24

Figura 6.2. Generación de streaming DASH en directo con FFmpeg. 25

Figura 6.3. Creación de interfaz de un mezclador de audio con JavaScript estándar. 31

Figura 6.4. Creación de interfaz de un mezclador de audio con React. 32

Figura 6.5. Ejemplo de uso de hls.js. 33

Figura 6.6. Ejemplo de uso de dash.js. 34

Figura 6.7. Ejemplo de uso de Web Audio API. 35

Figura 6.8. Ejemplo de archivo Dockerfile. 36

Figura 6.9. Ejemplo archivo docker-compose. 37

Figura 6.10. Convenio de coordenadas de la base de datos de HRTFs del CIPIC. . 39

Figura 6.11. Ejemplo de señales de audio obtenidas en el micrófono cercano del grupo de flautas. 43

Figura 6.12. Ejemplo de señales de audio obtenidas en el micrófono principal. 44

Figura 6.13. Ejemplo de espacialización de una señal de audio con HRTFs. 46

Figura 6.14. Representación escena sonora resultante de aplicar la operación mostrada en la Figura 6.13. 46

Figura 6.15. Ejemplo del resultado de una rotación del usuario en la escena sonora. 47

Figura 6.16. Cálculo de rotaciones con el convenio de la base de datos de HRTFs del CIPIC. 48

Figura 6.17. Convenio de coordenadas propuesto para facilitar las operaciones de rotación con las HRTFs de la base de datos del CIPIC. 49

Figura 7.1. Esquema de la arquitectura del servicio de streaming. 53
Figura 7.2. Modelos y relación entre colecciones de la base de datos. 57
Figura 7.3. Formato de un contenedor WebM (extraída de [28]). 59
Figura 7.4. Intercambio de información durante la ingesta de datos. 62
Figura 7.5. Estructura de directorios en volumen "media". 63
Figura 7.6. Generación de retransmisión de audio MPEG-DASH con códec OPUS y contenedor WebM y envío de ficheros al servidor. 64
Figura 7.7. Diagrama de flujo del controlador de ingesta de datos multimedia en el back-end. 65
Figura 7.8. Intercambio de información durante la difusión del contenido multimedia. 66
Figura 7.9. Intercambio de información para recuperar los datos de todas las presentaciones. 67
Figura 7.10. Ejemplo de datos recuperados de todas las presentaciones. 67
Figura 7.11. Intercambio de información para almacenar una HRTF al poblar la colección de la base de datos. 68
Figura 7.12. Intercambio de información para recuperar las HRTFs de las localizaciones de los instrumentos de una presentación. 69
Figura 7.13. Ejemplo de HRTFs recuperadas para la actuación de la Figura 7.10.. 69
Figura 7.14. Intercambio de información para recuperar las HRTFs de las localizaciones de los instrumentos de una presentación tras aplicar una rotación. 70
Figura 7.15. Ejemplo de HRTFs recuperadas para una actuación tras aplicar una rotación. 71
Figura 7.16. Representación esquemática de la cadena de procesamiento de audio en el cliente. 72
Figura 7.17. Funcionamiento del combinador de canales con una única entrada. .. 74
Figura 7.18. Estructura de la interfaz del front-end basada en React. 76
Figura 7.19. Diagrama de flujo del funcionamiento del front-end al seleccionar una presentación multimedia. 78
Figura 7.20. Aspecto de la interfaz de usuario durante la reproducción de audio procedente del micrófono principal. 79
Figura 7.21. Aspecto de la interfaz de usuario durante la reproducción de audio procedente del micrófono cercano de flautas. 80

Figura 8.1. Intercambio de información inicial al seleccionar una presentación y cabeceras respuestas HTTP que transportan el archivo de manifiesto MPD. 83
Figura 8.2. Cabeceras de respuestas HTTP que transportan segmentos multimedia. 84
Figura 8.3. Interfaz correspondiente a la posición del micrófono principal. 85
Figura 8.4. Ajuste de mezcla en la posición del micrófono principal. 86
Figura 8.5. Cambio de posición del micrófono principal a micrófono cercano del grupo de flautas. 86
Figura 8.6. Ajuste de mezcla en la posición de los micrófonos cercanos. 87
Figura 8.7. Rotación de 85º hacia la derecha. 88
Figura 8.8. Vuelta a las localizaciones originales con una rotación de 0º. 89
Figura 8.9. Interfaz con una presentación bajo demanda. 89
Figura 8.10. Consumo de ancho de banda durante la ingesta de datos. 91
Figura 8.11. Consumo de ancho de banda durante el consumo de datos multimedia. 92

1. SIGLAS, ACRÓNIMOS Y DEFINICIONES

A continuación se ofrece una lista de siglas, acrónimos y definiciones que aparecen a lo largo del documento.

A

AAC. *Advanced Audio Coding. Es un codec de audio desarrollado por el MPEG en el estándar ISO/IEC 13818-7:2006. Basado en un algoritmo de compresión con pérdidas con rendimiento superior al del MP3 en términos de calidad de audio y eficiencia de compresión.*

AC-3. *Codec de audio propietario desarrollado por Dolby Laboratories, basado en técnicas de codificación con pérdidas.*

API. *Application Programming Interface. Es un conjunto de funciones que proporcionan una forma estructurada y definida para que distintas aplicaciones puedan interactuar entre sí y compartir datos de manera segura y eficiente.*

B

back-end. *En el contexto de una aplicación, es la parte del sistema encargada de implementar la lógica para modelar datos de la aplicación, ofrecer mecanismos para poder realizar operaciones sobre dichos datos e implementar otros mecanismos de gestión y control de acceso.*

binaural. *Referido al audio al que se le aplican técnicas de procesado que buscan simular la experiencia auditiva tridimensional y espacial que perciben los humanos en la vida real.*

C

CDN. *Content Delivery Network. Una red de servidores distribuidos geográficamente que se utiliza para entregar contenido web de manera eficiente y rápida a los usuarios finales.*

E

E-AC-3. *Enhanced AC-3. Desarrollado por Dolby Laboratories como extensión de su otro codec AC-3. Ofrece mejoras en cuanto a calidad de audio y eficiencia de compresión se refiere.*

EBML. *Extensible Binary Meta Language es un formato binario extensible utilizado en el contenedor de medios Matroska. Define una sintaxis y un conjunto de tipos de elementos que permiten la representación de datos en forma de elementos jerárquicos anidados.*

estereofónica. Adjetivo alusivo a mezclas o señales de a*udio de dos canales.*

F

framework. En el contexto del software, un framework es un conjunto de herramientas, bibliotecas y componentes predefinidos que proporcionan una estructura y funcionalidad básica para el desarrollo de aplicaciones. Agiliza el proceso de desarrollo de nuevo software al evitar tener que reinventar toda la infraestructura desde cero.

front-end. *En el contexto de una aplicación, es la parte del sistema que ofrece la interfaz con la que los usuarios interactúan con la aplicación. Se encarga de presentar los datos a los usuarios y se comunica con el back-end para realizar operaciones sobre dichos datos.*

H

HLS. *HTTP Live Streaming.*

HRTF. Head-Related Transfer Function. Es una caracterización de cómo el sonido se modifica al pasar por los oídos de una persona. Se utiliza en aplicaciones de audio espacial para simular de forma realista la forma en al que percibimos el sonido en entornos 3D.

HTML. *HyperText Markup Language. Es un lenguaje de marcas que permite estructurar el contenido de una página web mediante etiquetas de elementos.*

HTTP. *HyperText Transfer Protocol.*

J

JSON. *JavaScript Object Notation. Es un formato de intercambio de datos ligero basado en la notación de objetos en JavaScript que representa la información en forma de objetos y listas. Es independiente del lenguaje de programación y ampliamente usado en aplicaciones web y APIs para el intercambio de datos.*

M

middleware. *Es una capa de software reutilizable situada entre la entrada y salida de datos de una aplicación. Un middleware procesa los datos que la aplicación principal gestiona, añadiendo funcionalidades como autenticación, compresión de datos, ..., sin afectar a la lógica de la aplicación principal.*

MIME. *Multipurpose Internet Mail Extensions. Definen distintos tipos de etiquetas para identificar datos de diferentes formatos, como imágenes, audio y vídeo a través de protocolos como HTTP.*

monofónicas. Adjetivo alusivo a señales de audio de un solo canal

MP3. *Aunque realmente es un formato de archivo de audio, el término MP3 se emplea usualmente para referirse al codec de audio MPEG-1 Audio Capa 3, desarrollado por el MPEG y recogido en el estándar internacional ISO/IEC 11172-3:1993. Está basado en un algoritmo de compresión con pérdidas que elimina información imperceptible por el oído humano.*

MPEG. *Moving Picture Experts Group. Un comité de expertos encargado de desarrollar estándares de compresión y codificación de audio y video.*

MPEG-DASH. *También referido solamente como DASH, significa Dynamic Adaptive Streaming over HTTP.*

MSE. Media Source Extensions. Son un conjunto de APIs de navegador que posibilitan la construcción de streams multimedia para los elementos <audio> y <video> de HTML5. Esto facilita la integración de múltiples aplicaciones entre las que se encuentra el streaming adaptativo.

O

on-demand. *En el contexto de un servicio multimedia, se refiere a la posibilidad de los usuarios de consumir el contenido en cualquier momento de manera flexible, en lugar de estar limitado a un horario de emisión determinado.*

R

REST. *En el contexto de APIs, facilitan el diseño de servicios web con un modelo cliente-servidor que mediante métodos de protocolos estándar, como GET, POST, PUT y DELETE de HTTP, permiten realizar acciones en recursos identificados por URLs sin estado (el servidor procesa las peticiones independientemente de las realizadas anteriormente). Son escalables, flexibles y fáciles de consumir.*

RTCP. *Real-Time Control Protocol. Protocolo complementario a RTP encargado de enviar la información de control asociados a los datos y sesiones multimedia de RTP. Es transportado típicamente por UDP.*

RTMP. *Real-Time Messaging Protocol.*

RTP. *Real-Time Transport Protocol. Es un protocolo destinado a la transmisión en streaming confiable de datos multimedia. Es transportado por UDP.*

S

SDP. *Session Description Protocol. Describe sesiones multimedia proporcionando información relevante y necesaria para iniciar y mantener dichas sesiones.*

SQL. *Structured Query Language. Es un lenguaje de programación utilizado para gestionar y manipular bases de datos relacionales.*

stream. *Secuencia continua de datos multimedia que puede incluir audio, video u otros tipos de datos relacionados, se transmite en forma de paquetes o fragmentos.*

T

TCP. *Transmission Control Protocol. Es un protocolo en la capa de transporte del modelo TCP/IP que ofrece comunicaciones confiables y orientadas a conexión entre equipos.*

U

UDP. *User Datagram Protocol. Protocolo de capa de transporte en el modelo TCP/IP que proporciona comunicaciones no orientadas a conexión. Empleado normalmente cuando los requisitos de velocidad y baja latencia priman sobre la integridad de los datos.*

unidades de acceso. *Unidad mínima de datos multimedia perteneciente a un stream que se puede decodificar y reproducir de forma independiente. Representa una porción discreta de audio, vídeo o datos similares. En el caso de audio esto se corresponde con una muestra de audio o un grupo de muestras de audio.*

2. RESUMEN

En este trabajo de tipo teórico/experimental se propone una solución de servicio de streaming de audio para retransmitir actuaciones de música clásica.

Los usuarios podrán en todo momento ajustar la mezcla de audio en tiempo real a sus preferencias personales, pudiendo realzar, aislar o suprimir grupos de instrumentos que forman parte de la actuación.

También es posible seleccionar entre un conjunto de posiciones predefinidas del escenario, las cuales se corresponden con localizaciones cercanas a los grupos de instrumentos o donde se sitúa el micrófono principal.

La mezcla de audio con las señales de audio captadas por el micrófono principal es espacializada en tiempo real, generando una mezcla binaural, dotando de un carácter más inmersivo el contenido de audio que los usuarios consumen.

Además, los usuarios pueden realizar rotaciones, esto es, simular que miran hacia otra dirección en la escena sonora. Esto hace que se los instrumentos se perciban desde otras direcciones distintas a las originales.

El código fuente desarrollado está disponible en el repositorio del proyecto. Informar de problemas de funcionamiento, sugerir nuevas ideas o incluso involucrarse directamente con el desarrollo del servicio es siempre bien recibido.

En el capítulo de INTRODUCCIÓN se ofrece una descripción del problema y de las características de las señales de audio que se pretenden distribuir. También se fija el alcance del trabajo. Además, se nombran algunos servicios de streaming de audio conocidos, los cuales pueden constituir un punto de partida para el desarrollo del servicio propuesto.

En el capítulo de OBJETIVOS se establecen las metas que deben alcanzarse en el desarrollo del servicio propuesto.

En el capítulo de REVISIÓN DEL ESTADO DEL ARTE se estudiarán distintas aproximaciones para servir contenido multimedia en streaming, además de analizar dos protocolos de soporte de multimedia ampliamente utilizados actualmente.

En el capítulo MATERIALES Y MÉTODOS se proporciona una lista con todos los materiales utilizados para la realización de este trabajo y se describen las pautas y procedimientos llevados a cabo durante el transcurso del proyecto.

Los detalles de implementación son comentados en el capítulo DESARROLLO, donde se ofrece una explicación de la arquitectura del servicio y de todos sus componentes en detalle.

RESUMEN

En el capítulo RESULTADOS Y DISCUSIÓN se comprueba el funcionamiento del servicio con una actuación de ejemplo.

En el capítulo de CONCLUSIONES se exponen los principales aspectos favorables y limitaciones del servicio, y se citan algunas líneas de desarrollo que pueden ser abordadas en el futuro.

La lista con todas las referencias consultadas se encuentra en el capítulo de REFERENCIAS.

3. INTRODUCCIÓN

En este apartado se describe el problema que se pretende solucionar, cuál es el alcance del trabajo realizado y se citan qué servicios de naturaleza similar existen a día de hoy que pueden constituir una buena referencia para el desarrollo del servicio propuesto.

3.1. Descripción del problema

Se pretende desarrollar un servicio de streaming de audio de carácter inmersivo para retransmitir conciertos de música clásica.

Los usuarios podrán interactuar con la aplicación de tal modo que se permitirá que seleccionen una posición de un conjunto de localizaciones en el escenario predefinidas. Acorde a esta posición, se permitirá el aislamiento, realzado o supresión de los distintos grupos de instrumentos que integran la actuación.

Adicionalmente, para mejorar la experiencia de inmersión, se implementará un sistema de espacialización de audio que permita recrear, en la mezcla de audio estereofónica reproducida en el cliente, la sensación de que los instrumentos están en distintas localizaciones del espacio, así como definir la orientación del usuario. Una vez situados los instrumentos en una escena sonora, y definidos la posición y orientación del usuario, este sistema deberá adaptar la mezcla de audio a dicha configuración. Los siguientes subapartados describen cuáles son las posiciones predefinidas y algunas características de las señales de audio a tener en cuenta para su mejor tratamiento.

3.1.1. Disposición de micrófonos para la grabación de la actuación

Para la grabación de la actuación se emplearán distintos micrófonos dependiendo de la posición en el escenario e instrumentos que se quieren captar.

Un ejemplo de la disposición de micrófonos para grabar una actuación de una orquesta con una distribución de instrumentos típica se puede observar en la Figura 3.1.

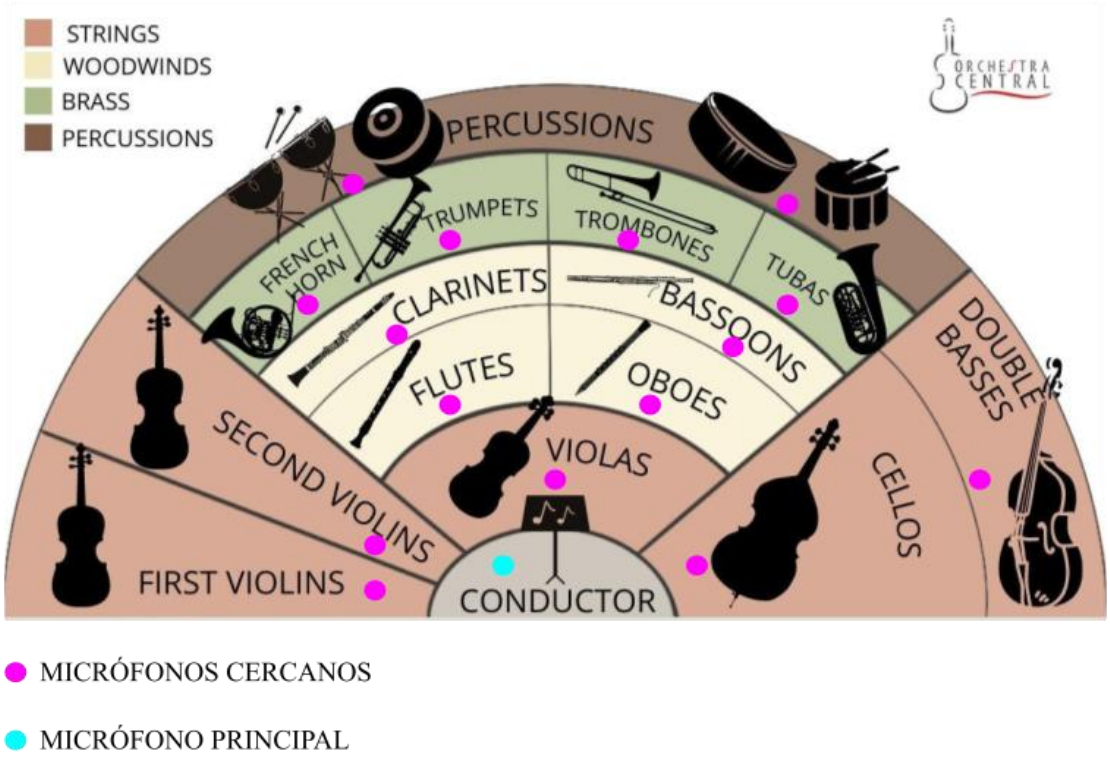

Figura 3.1. Disposición de micrófonos para la grabación de la actuación. Esquema de distribución de instrumentos extraído de [1].

Cada grupo de instrumentos tendrá un micrófono cercano directivo, por ejemplo, con patrón cardioide o supercardioide, destinado a captar las señales procedentes de los instrumentos del grupo en cuestión.

Adicionalmente, se empleará un micrófono principal (*main*) omnidireccional que captará las señales procedentes de toda la orquesta, situado en una posición cercana a la del director.

3.1.2. Número de señales de audio disponibles

Las señales de audio que se pretenden distribuir son las resultantes de aplicar un algoritmo de separación de fuentes a cada uno de los micrófonos disponibles en la actuación. En esta ocasión se ha empleado el método propuesto en [2], el cual procesa cada una de las señales captadas por los micrófonos y obtiene una estimación de la señal procedente de cada grupo de instrumentos. Esto significa que, si hay N micrófonos y J grupos de instrumentos, el número total de señales de salida es de J·N.

3.1.3. Características de las señales de audio

Las señales de audio procedentes del mismo grupo de instrumentos obtenidas al aplicar el algoritmo de separación de fuentes en cada micrófono son distintas entre sí debido

a que, por lo general, los micrófonos cercanos son directivos al estar destinados a captar lo mejor posible el sonido de un grupo de instrumentos, lo que provoca una atenuación en las señales procedentes de otros instrumentos situados en las distintas posiciones. Esto hace que, a excepción de la señal de audio del grupo de instrumentos cercano al micrófono, las señales resultantes tengan muy poca energía, siendo en algunos casos apenas audibles.

Esto no ocurre en el micrófono principal, ya que, al tener un patrón directivo onminidireccional, capta el sonido procedente de todas las direcciones sin atenuar ninguna de las señales sonoras procedentes de cada uno de los grupos de instrumentos. Esto no quiere decir que todas las señales se capten con la misma intensidad sonora, ya que, evidentemente, el sonido se atenúa en función de la distancia y si un grupo está más alejado en el escenario, se percibirá algo más atenuado que uno que esté situado cerca del micrófono.

Además, en las señales obtenidas de los micrófonos cercanos no se percibe la contribución de la acústica de la sala de actuación al sonido resultante, lo que resulta en una sensación de falta de ambiente, al contrario que en las señales del micrófono "main", donde sí que existe esa "sensación de sala".

3.2. Alcance

El alcance de este trabajo consiste en desarrollar un servicio de streaming que permita consumir el contenido de audio anteriormente descrito en tiempo real.

Esto incluye la implementación de un back-end encargado de gestionar y retransmitir el contenido multimedia y un front-end que permita consumir este contenido.

La obtención de la separación de las señales de audio de cada grupo de instrumentos no está dentro del ámbito de este proyecto.

3.3. Antecedentes

A día de hoy existen distintos servicios de streaming de audio entre los que podemos destacar YouTube Music, Amazon Music, Apple Music y, el que más suscripciones acumulaba en el segundo trimestre del año 2022, Spotify, con 187,8 millones de suscripciones [3].

La Figura 3.2 muestra el número de suscripciones y reparto de mercado entre los que se incluyen los servicios comentados anteriormente.

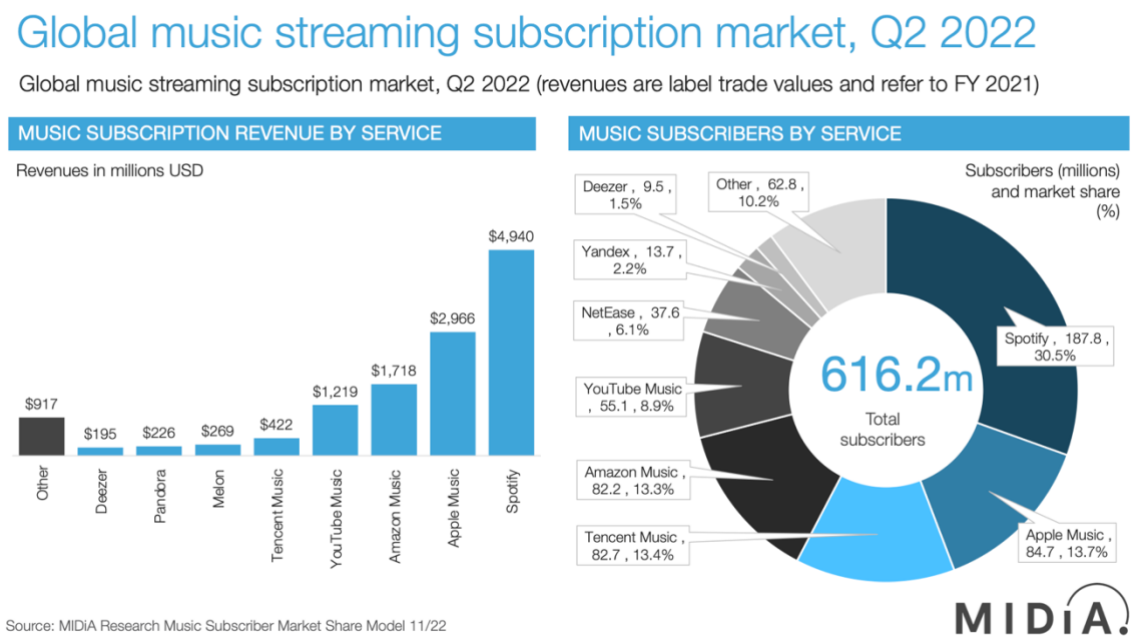

Figura 3.2. Número de suscripciones y reparto de mercado entre los servicios de streaming de música en el segundo trimestre de 2022 (extraída de [3])

Estos servicios se caracterizan por ofrecer música y podcasts ajustándose a un paradigma de streaming "on-demand", en el que los usuarios pueden consumir los contenidos ofrecidos por los servicios en cualquier momento.

Otros servicios de streaming los cuales tienen la capacidad de retransmitir el contenido multimedia en tiempo real son Twitch o YouTube, más enfocados a la emisión de vídeo.

Todas estas aplicaciones difieren de la que se pretende desarrollar en este trabajo en que los usuarios no tienen gran capacidad para interactuar con el contenido multimedia que están consumiendo, por ejemplo, en Spotify o Amazon Music no es posible seleccionar qué instrumento escuchar mientras se reproduce una canción, tampoco se puede ajustar la mezcla de todos los instrumentos y voz según las preferencias del usuario, etc.

4. OBJETIVOS

A fin de facilitar la planificación de las tareas a realizar para el desarrollo del servicio con las especificaciones detalladas en el apartado 3.1 se han establecido los objetivos específicos que se citan a continuación:

1. Retransmitir en tiempo real las señales de audio procedentes de distintos instrumentos de una orquesta.
2. Permitir ajustar la mezcla de estas señales para realzar o suprimir los distintos instrumentos de la actuación.
3. Espacializar la mezcla de audio para obtener una mezcla de audio inmersiva.
4. Ajustar la espacialización de los instrumentos en la escena sonora coherentemente con posibles cambios de orientación del usuario.

5. REVISIÓN DEL ESTADO DEL ARTE

Los servicios mencionados en el apartado 3.3 más atrás emplean diversos protocolos y estrategias para su funcionamiento, los cuales serán discutidos en este apartado y serán considerados como punto de partida para el desarrollo del servicio de streaming mencionado en el apartado 3.2.

El principio de retransmisión de audio en Spotify es sencillo. Las pistas de audio están almacenadas como ficheros en servidores HTTP. A la hora de reproducirlas, la aplicación de Spotify descarga el fichero correspondiente por segmentos de 512 kilobytes [4].

Amazon Music se basa en el protocolo MPEG-DASH para servir el contenido. De forma similar a la metodología de Spotify, los ficheros de audio son divididos en segmentos más pequeños, los cuales pueden ser codificados con distintos niveles de calidad. Antes de que la aplicación comience a descargar estos segmentos mediante peticiones HTTP, previamente descarga un archivo de manifiesto que contiene información necesaria para poder recuperar todo el contenido [5].

Twitch emplea distintos protocolos para la ingesta de datos, esto es, el envío de datos de quien está transmitiendo a los servidores de Twitch, y la difusión del contenido, esto es, el envío de datos de los servidores de Twitch a los usuarios finales. Para la ingesta de datos emplea RTMP, este protocolo proporciona un método de enviar contenido generado en tiempo real de forma rápida al receptor. Sin embargo, para hacer posible la difusión de contenido a miles de espectadores, a la hora de distribuir el contenido se emplea HLS, cuyo funcionamiento es muy similar a MPEG-DASH [6].

YouTube tiene un funcionamiento similar a Twitch, pero para la ingesta de datos soporta RTMP, HLS y MPEG-DASH [7].

Las distintas soluciones planteadas tienen en común el uso de HTTP para el envío de contenido multimedia y, por lo tanto, toda la información viaja sobre TCP. Además, HLS y DASH crean, actualizan y difunden los descriptores de la sesión multimedia junto con los contenidos multimedia, todo en el mismo protocolo, en contraposición a los protocolos multimedia clásicos como RTP/RTCP, SDP o RTSP, los cuales dependían generalmente de UDP.

En base a la información que se acaba de presentar se puede constatar que HLS y MPEG-DASH son los protocolos de soporte de multimedia que estas aplicaciones emplean (salvo Spotify). A continuación, se repasarán los aspectos fundamentales y el funcionamiento de los mismos.

5.1. HLS

HLS son las siglas de HTTP Live Streaming, protocolo desarrollado por Apple en 2009 cuya especificación se recoge en la RFC 8216.

5.1.1. Visión general

HLS ofrece un método confiable y eficiente para la entrega continua de vídeo y audio empleando el protocolo HTTP para el envío de datos. Este protocolo permite al receptor adaptar la calidad del contenido al estado de la red para asegurar una reproducción sin interrupciones a la mejor calidad posible [8].

HLS también facilita la encriptación de los datos, permite servir contenido en streaming tanto en tiempo real como "on-demand" y soporta múltiples versiones del mismo contenido, por ejemplo un vídeo con distintos idiomas.

Una presentación multimedia viene definida mediante un Identificador de Recurso Uniforme (URI) que hace referencia a una lista de reproducción (***Playlist***). Estas Playlists pueden ser de dos tipos: ***Media Playlist*** o ***Master Playlist***, son básicamente ficheros de manifiesto con extensión .m3u8 que contienen todos los datos necesarios para poder reproducir el contenido multimedia que describen.

Para poder reproducir el contenido descrito en la "*Playlist*", el cliente descarga el archivo .m3u8 y seguidamente descarga los segmentos descritos en este. En el caso de "*Playlist*" de contenido "on-demand", el cliente descarga este archivo una única vez. Sin embargo, en emisiones en tiempo real, el cliente debe descargarlo periódicamente, ya que el archivo es actualizado continuamente para añadir nuevos segmentos.

5.1.2. Media Playlist

Una "*Media Playlist*" contiene la lista de segmentos del contenido multimedia correspondiente a un stream, junto con la duración y el URI de cada uno de ellos. Por ejemplo, la Figura 5.1 se corresponde con una "*Media Playlist*" con 5 segmentos multimedia.

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-TARGETDURATION:10
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-PLAYLIST-TYPE:VOD
#EXTINF:10.007800,
http://media.example.com/pirates0.ts
#EXTINF:10.007800,
http://media.example.com/pirates1.ts
#EXTINF:9.984589,
http://media.example.com/pirates2.ts
#EXTINF:10.007800,
http://media.example.com/pirates3.ts
#EXTINF:9.832633,
http://media.example.com/pirates4.ts
#EXT-X-ENDLIST
```

Figura 5.1. Ejemplo de una Media Playlist.

A continuación se describe brevemente el significado de cada etiqueta mostrada en la Figura 5.1:

- La primera línea se corresponde con la etiqueta de identificación de formato #EXTM3U.
- La etiqueta #EXT-X-VERSION:3 indica la versión del protocolo empleada.
- #EXT-X-TARGETDURATION:10 indica que la duración de los segmentos multimedia es de alrededor de 10 segundos, admitiendo un cierto nivel de desviación.
- #EXT-X-MEDIA-SEQUENCE:0 especifica que el número de secuencia del primer segmento es el 0.
- #EXT-X-PLAYLIST-TYPE:VOD indica que esta "*Playlist*" es para servir contenido "on-demand". En caso de que el contenido se sirviera en directo, el valor de esta etiqueta sería "*event*".
- #EXTINF identifica los segmentos multimedia junto con la duración y el URI de los mismos.

- #EXT-X-ENDLIST es una etiqueta propia de Playlist de tipo “on-demand”. Identifica el final de la transmisión.

5.1.3. Master Playlist

Una presentación más compleja puede ser descrita mediante una Master Playlist.

De acuerdo a lo establecido en [8], una Master Playlist proporciona un conjunto de streams variantes (***Variants***), esto es, el mismo contenido multimedia codificado con un formato, resolución y bitrate particulares para los recursos multimedia.

Además, estos streams variantes pueden especificar un conjunto de interpretaciones (***Renditions***), esto es, versiones alternativas del contenido, como audio en un distinto idioma o vídeo grabado desde otro ángulo de cámara.

La Figura 5.2 se corresponde con una Master Playlist con tres variantes de calidad (baja, media y alta) y tres interpretaciones (Main, Centerfield y Dogout).

Existen etiquetas con atributos que no aparecen en una Media Playlist, cuyos significados se describen brevemente a continuación:

- EXT-X-MEDIA identifica un tipo particular de contenido. Esta etiqueta tiene distintos atributos:
  - TYPE: tipo de contenido, en este ejemplo todos son vídeo.
  - GROUP-ID: identifica la variante a la que pertenece el contenido.
  - NAME: identifica la interpretación (rendition).
  - DEFAULT: especifica qué interpretación el cliente deberá seleccionar por defecto cuando el reproductor comience a reproducir el contenido.
  - URI: el URI de la Media Playlist correspondiente al contenido.
- #EXT-X-STREAM-INF identifica a una variante de stream. Tiene los siguientes atributos:
  - BANDWIDTH: indica el bitrate en bits por segundo de la variante.
  - CODECS: atributos del códec empleado por la variante.
  - VIDEO: identifica el nombre de la variante. Esta etiqueta depende del tipo de contenido, pudiendo ser VIDEO, AUDIO y SUBTITLES

El URI correspondiente a cada variante es la misma que la especificada como por defecto y hacen referencia a una Media Playlist.

Nótese que los atributos de los códecs han sido omitidos para condensar la información. El símbolo “\” indica continuación en la siguiente línea eliminando los espacios en blanco.

```
#EXTM3U
#EXT-X-MEDIA:TYPE=VIDEO,GROUP-ID="low",NAME="Main", \
        DEFAULT=YES,URI="low/main/audio-video.m3u8"
#EXT-X-MEDIA:TYPE=VIDEO,GROUP-ID="low",NAME="Centerfield", \
        DEFAULT=NO,URI="low/centerfield/audio-video.m3u8"
#EXT-X-MEDIA:TYPE=VIDEO,GROUP-ID="low",NAME="Dugout", \
        DEFAULT=NO,URI="low/dugout/audio-video.m3u8"

#EXT-X-STREAM-INF:BANDWIDTH=1280000,CODECS="...",VIDEO="low"
low/main/audio-video.m3u8

#EXT-X-MEDIA:TYPE=VIDEO,GROUP-ID="mid",NAME="Main", \
        DEFAULT=YES,URI="mid/main/audio-video.m3u8"
#EXT-X-MEDIA:TYPE=VIDEO,GROUP-ID="mid",NAME="Centerfield", \
        DEFAULT=NO,URI="mid/centerfield/audio-video.m3u8"
#EXT-X-MEDIA:TYPE=VIDEO,GROUP-ID="mid",NAME="Dugout", \
        DEFAULT=NO,URI="mid/dugout/audio-video.m3u8"

#EXT-X-STREAM-INF:BANDWIDTH=2560000,CODECS="...",VIDEO="mid"
mid/main/audio-video.m3u8

#EXT-X-MEDIA:TYPE=VIDEO,GROUP-ID="hi",NAME="Main", \
        DEFAULT=YES,URI="hi/main/audio-video.m3u8"
#EXT-X-MEDIA:TYPE=VIDEO,GROUP-ID="hi",NAME="Centerfield", \
        DEFAULT=NO,URI="hi/centerfield/audio-video.m3u8"
#EXT-X-MEDIA:TYPE=VIDEO,GROUP-ID="hi",NAME="Dugout", \
        DEFAULT=NO,URI="hi/dugout/audio-video.m3u8"

#EXT-X-STREAM-INF:BANDWIDTH=7680000,CODECS="...",VIDEO="hi"
hi/main/audio-video.m3u8
```

Figura 5.2. Ejemplo de una Master Playlist con variantes y renditions (extraída de [8]).

5.1.4. Contenedores y códecs compatibles

Según las especificaciones en [8], HLS es compatible con 2 tipos de formato contenedor para fragmentar los contenidos multimedia descritos en los archivos de Playlist:

- MPEG-2 Transport Streams, definidos en la norma ISO/IEC 13818-1:2007
- MPEG-4 Fragments, definidos en la norma ISO/IEC 14496-12:2015

Adicionalmente, para segmentos de audio, el protocolo soporta únicamente los códecs de audio mencionados a continuación:

- AAC
- MP3
- AC-3
- E-AC-3

5.2. MPEG-DASH

DASH son las siglas de Dynamic Adaptive Streaming over HTTP. Es un protocolo desarrollado por el MPEG y estandarizado internacionalmente en 2012 en la norma ISO/IEC 23009-1, revisada en varias revisiones, siendo la última en 2022. Los servicios que ofrece el protocolo y el esquema de funcionamiento general son muy similares al anteriormente revisado HLS.

5.2.1. Visión general

DASH proporciona un mecanismo que permite la entrega continua de contenido multimedia desde servidores HTTP estándar a clientes HTTP, adaptando la calidad y el bitrate de los datos entregados al cliente dependiendo de las condiciones de la red con el fin de proporcionar una reproducción continua y sin interrupciones [9].

Adicionalmente, DASH permite almacenar en cachés HTTP estándar el contenido multimedia, permitiendo a CDNs servir más rápidamente estos contenidos, facilita la división de la presentación completa en secciones, permitiendo la inclusión de anuncios u organizar por capítulos el contenido completo de una forma muy sencilla y posibilita la encriptación del contenido de la presentación multimedia.

Una presentación multimedia se describe mediante un modelo de datos jerárquico en un documento conocido como ***Media Presentation Description*** (MPD), que es un documento XML. Esto es, básicamente, un archivo de manifiesto en el que se encuentran los datos necesarios para reproducir el contenido multimedia.

El contenido puede ser servido tanto en tiempo real como "on-demand". Para presentaciones multimedia "on-demand", este documento MPD suele ser un fichero estático y todos los segmentos del contenido multimedia se encuentran disponibles en el servidor desde el primer momento. Sin embargo, para transmisiones en directo, este fichero es descargado periódicamente, pues se va actualizando a medida que avanza el tiempo al ir añadiendo nuevos segmentos.

5.2.2. Modelo de datos

Los datos que describen una presentación multimedia en DASH para la correcta reproducción del contenido mantienen una organización por niveles, desde lo más general a lo más particular, la cual se puede observar en la Figura 5.3.

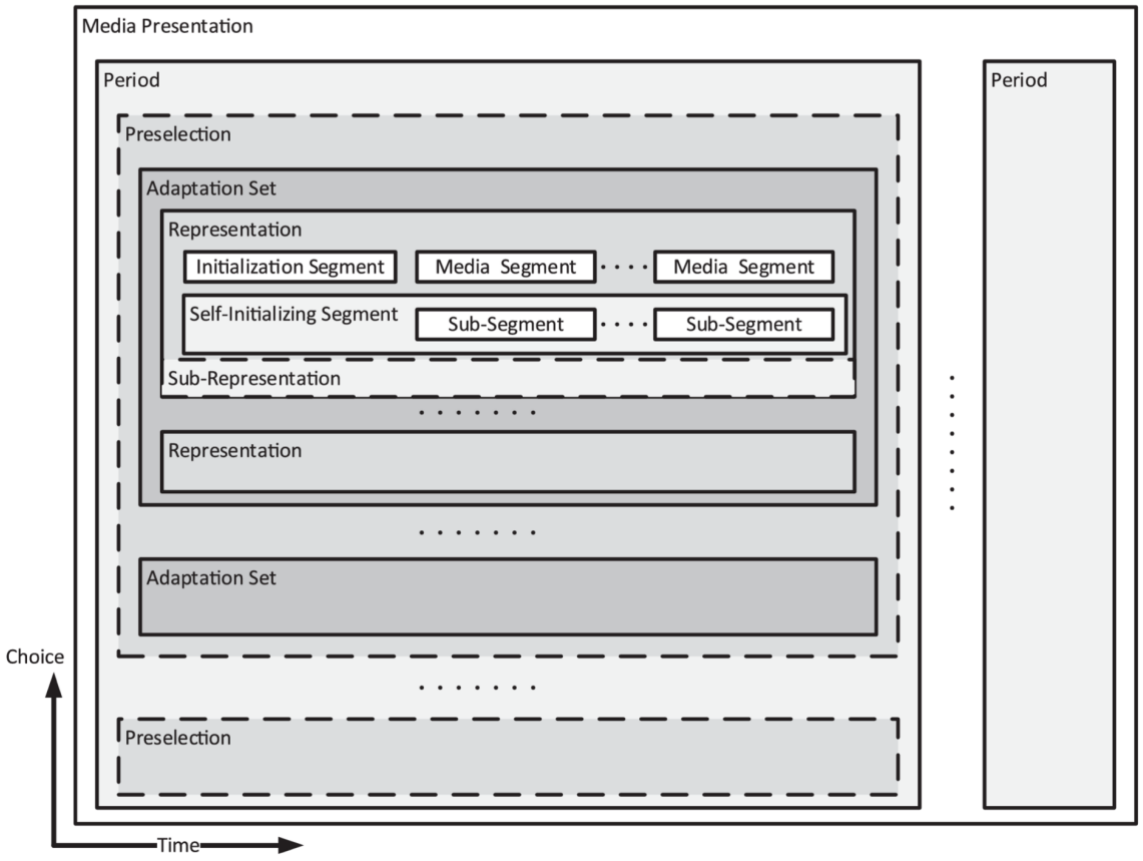

Figura 5.3. Esquema del modelo de datos de DASH (extraída de [9]).

De acuerdo con lo establecido en [9], una presentación multimedia se compone de una sucesión de Periodos (***Periods***) a lo largo del tiempo. Estos elementos representan contenido multimedia que mantiene cierta relación en un intervalo temporal, como podrían ser capítulos, anuncios o cualquier tipo de división lógica del contenido.

Dentro de un mismo *Period* pueden existir Conjuntos de Adaptación (***Adaptation Sets*****),** que representan un conjunto de versiones del mismo contenido multimedia codificadas de manera similar y que son intercambiables entre sí. Por ejemplo, puede haber un solo "*Adaptation Set*" para el componente principal de vídeo y tres "*Adaptation Sets*" para el audio, que está disponible en inglés, francés y español, de tal forma que el usuario puede seleccionar el idioma a la hora de reproducir el contenido.

El concepto de Preselecciones (***Preselections***) fue introducido en la tercera revisión del estándar con la motivación de señalar combinaciones apropiadas de elementos de audio que son ofrecidos en distintos "*Adaptation Sets*".

Los "*Adaptation Sets*" contienen un conjunto de Representaciones (***Representations***), que son distintos formatos de la misma versión del contenido multimedia del "*Adaptation Set*" al que pertenece. Esto es, por ejemplo, distintas resoluciones para un componente de vídeo o distintas tasas de bits para el audio.

El contenido de cada "*Representation*" se divide en Segmentos (***Segments)***, a los que a cada uno le corresponde un URL, y es, por lo tanto, la unidad más grande de información que se puede recuperar en una sola petición HTTP. Existen distintos tipos de "*Segments*":

- Segmentos de Inicialización (***Initialization Segments***), que contienen metadatos estáticos de la Representation.
- Segmentos Multimedia (***Media Segments***), contienen muestras del contenido multimedia.
- Segmentos de Auto-inicialización (***Self-initializing Segment***), que contienen tanto metadatos como datos multimedia. Pueden ser subdivididos en Subsegmentos (***Subsegments***), que contienen un número completo de unidades de acceso.

Puede darse la ocasión de que una "*Representation*" incluya varios tipos de contenido multimedia, por ejemplo, de vídeo y audio. Las Subrepresentaciones (***Subrepresentations***) permiten incluir información que solo es aplicable a un solo stream dentro de este tipo de "*Representation*".

5.2.3. Documentos MPD

El modelo de datos presentado en el apartado 5.2.2 se utiliza en los documentos MDP, que son archivos XML con extensión .mpd generalmente. Estos documentos cumplen la función de archivos de manifiesto para obtener la información necesaria para la correcta reproducción del contenido de la presentación multimedia que describen.

La Figura 5.4 es un ejemplo de este tipo de archivos, el cual describe jerárquicamente una presentación multimedia de un solo "*Period*", 2 "*Adaptation Sets*" y 2 "*Representations*" para el segundo "*Adapation Set*".

```
<?xml version="1.0" encoding="utf-8"?>
<MPD xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="urn:mpeg:dash:schema:mpd:2011"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xsi:schemaLocation="urn:mpeg:DASH:schema:MPD:2011 http://standards.iso.org/ittf/PubliclyAvailableStandards/MPEG-DASH_schema_files/DASH-MPD.xsd"
    profiles="urn:mpeg:dash:profile:isoff-live:2011"
    type="dynamic"
    minimumUpdatePeriod="PT10S"
    suggestedPresentationDelay="PT10S"
    availabilityStartTime="2023-06-01T15:19:06.272Z"
    publishTime="2023-06-01T15:20:06.298Z"
    maxSegmentDuration="PT10.0S"
    minBufferTime="PT20.0S">
    <ProgramInformation>
    </ProgramInformation>
    <ServiceDescription id="0">
    </ServiceDescription>
    <Period id="0" start="PT0.0S">
        <AdaptationSet id="0" contentType="audio" startWithSAP="1" segmentAlignment="true" bitstreamSwitching="true" lang="violinUNO">
            <Representation id="0" mimeType="audio/webm" codecs="opus" bandwidth="128000" audioSamplingRate="48000">
                <AudioChannelConfiguration schemeIdUri="urn:mpeg:dash:23003:3:audio_channel_configuration:2011" value="2" />
                <SegmentTemplate timescale="1000" initialization="init-stream$RepresentationID$.webm" media="chunk-stream$RepresentationID$-$Number%05d$.webm" startNumber="1">
                    <SegmentTimeline>
                        <S t="0" d="10001" />
                        <S d="10000" r="4" />
                    </SegmentTimeline>
                </SegmentTemplate>
            </Representation>
        </AdaptationSet>
        <AdaptationSet id="1" contentType="audio" startWithSAP="1" segmentAlignment="true" bitstreamSwitching="true" lang="main">
            <SupplementalProperty schemeIdUri="urn:mpeg:dash:srd:2014" value="main"/>
            <Representation id="1" mimeType="audio/webm" codecs="opus" bandwidth="128000" audioSamplingRate="48000">
                <AudioChannelConfiguration schemeIdUri="urn:mpeg:dash:23003:3:audio_channel_configuration:2011" value="15" />
                <SegmentTemplate timescale="1000" initialization="init-stream$RepresentationID$.webm" media="chunk-stream$RepresentationID$-$Number%05d$.webm" startNumber="1">
                    <SegmentTimeline>
                        <S t="0" d="10001" />
                        <S d="10000" r="4" />
                    </SegmentTimeline>
                </SegmentTemplate>
            </Representation>
            <Representation id="2" mimeType="audio/webm" codecs="opus" bandwidth="256000" audioSamplingRate="48000">
                <AudioChannelConfiguration schemeIdUri="urn:mpeg:dash:23003:3:audio_channel_configuration:2011" value="15" />
                <SegmentTemplate timescale="1000" initialization="init-stream$RepresentationID$.webm" media="chunk-stream$RepresentationID$-$Number%05d$.webm" startNumber="1">
                    <SegmentTimeline>
                        <S t="0" d="10001" />
                        <S d="10000" r="4" />
                    </SegmentTimeline>
                </SegmentTemplate>
            </Representation>
        </AdaptationSet>
    </Period>
</MPD>
```

Figura 5.4. Ejemplo de un documento MPD que describe una presentación multimedia en directo.

Se observa que los elementos descritos en el apartado 5.2.2 se incluyen en el documento como etiquetas, siendo la etiqueta raíz la etiqueta <MPD>, que tiene una serie de atributos entre los que se distinguen:

- *"**Type**"*, en este caso toma el valor de "dynamic" al tratarse de una presentación en directo. En caso de presentaciones "on-demand", este atributo toma el valor de "static".
- *"**minimumUpdatePeriod**"* indica el intervalo de tiempo que debe transcurrir entre peticiones sucesivas del documento MPD, de tal forma que el cliente

pueda tener la información de la presentación multimedia actualizada. Este atributo es propio de presentaciones en directo.

- *"**availabilityStartTime**"* es un sello temporal que representa el instante en el que el documento MPD fue generado por primera vez. Este atributo es propio de presentaciones en directo.
- *"**publishTime**"* es un sello temporal que indica cuándo fue la última vez que el documento fue actualizado. Este atributo es propio de presentaciones en directo.
- *"**maxSegmentDuration**"* indica la duración máxima que tendrán los segmentos de contenido multimedia.

En documentos MPD que describen presentaciones "on-demand" (etiqueta "*type*" con valor "*static*") aparece el atributo "***mediaPresentationDuration***", que indica la duración total de la presentación.

Nótese cómo en la presentación descrita en la Figura 5.4, el documento MPD incluyen información adicional a parte de los datos presentados anteriormente en el apartado 5.2.2, integrada como atributos para las etiquetas correspondientes. A continuación se describirán los atributos más relevantes de cara a la implementación de la aplicación deseada:

- Se indica el instante de comienzo de cada *Period* con la etiqueta "***start***".
- Cada "*Adaptation Set*" tiene un atributo "***contentType***" que identifica el contenido multimedia que representa. En este caso, todo el contenido multimedia se trata de audio.
- Cada "*Representation*" tiene asociados atributos para identificar el formato contenedor y codec empleado, el ancho de banda del contenido multimedia, la frecuencia de muestreo en caso de ser audio.
- La etiqueta "*SegmentTemplate*" contiene atributos que proporcionan el patrón para obtener los URL de cada "*Initialization Segment*" y "*Media Segment*".
- Cada segmento contenido dentro de la etiqueta "*SegmentTimeline*" tiene asociados atributos que indican el tiempo de presentación (atributo t), la duración del mismo (atributo d) o el número de segmentos contiguos con idéntica duración menos 1 (atributo r). Esto es, en el ejemplo de la Figura 5.4,

el atributo r toma el valor de 4, por lo que existen en total 5 segmentos contiguos con igual duración que la indicada en el segundo segmento.

Además, en caso de que el contenido sea audio, existe la etiqueta "***AudioChannelConfiguration***", que describe el número de canales de audio que componen la representación en cuestión.

5.2.4. Contenedores y códecs compatibles

Aunque en el estándar el término "***codec-agnostic***" no se menciona explícitamente, el diseño y principios de funcionamiento de MPEG-DASH lo hacen en la práctica compatible con una amplia gama de códecs.

Esto significa que una de las ventajas más importantes de MPEG-DASH es que es capaz de transportar contenido multimedia independientemente del códec utilizado.

Algunos formatos contenedores soportados por DASH son:

- MPEG-2 Transport Streams
- MPEG-4 parte 14 (.mp4)
- WebM

A su vez, cabe destacar algunos de los códecs de audio compatibles con DASH, como los siguientes:

- AAC
- MP3
- Vorbis
- Opus

6. MATERIALES Y MÉTODOS

En este capítulo se ofrece una descripción de los materiales utilizados y de los criterios y procedimientos metodológicos en los que se basa la solución de streaming propuesta.

Debido al gran número de tecnologías que se describirán en este capítulo, se recomienda observar la Figura 7.1, en la que se representa la arquitectura final del servicio propuesto. Esta representación visual proporciona una perspectiva más clara y comprensible de lo que se va a discutir a continuación.

6.1. Materiales

A continuación se describirá todos los materiales, tanto software como hardware, que forman parte del servicio de streaming o que de alguna manera se han empleado a lo largo del transcurso del proyecto.

6.1.1. FFmpeg

Para este proyecto se ha empleado FFmpeg en su versión 5.1.2.

FFmpeg es el "*framework*" multimedia líder, capaz de decodificar, codificar, transcodificar, multiplexar, demultiplexar, retransmitir, filtrar y reproducir básicamente cualquier contenido. Soporta desde los formatos multimedia más antiguos hasta los más novedosos. No importa si fueron diseñados por algún comité de estándares, la comunidad o una corporación. Es además altamente portable, pudiendo ser ejecutado en Linux, Mac OS y Windows [10].

FFmpeg se puede ejecutar como un programa de línea de comandos y puede generar contenido multimedia en directo con los protocolos HLS y MPEG-DASH, como se detalla en [11].

La Figura 6.1 es un ejemplo de comando para generar un streaming HLS en directo.

```
ffmpeg -re \
-i ../audio/Unaligned/VC.wav \
-i ../audio/Unaligned/FL.wav \
-c:a aac -b:a 128k -ac 2 \
-map 0:a -map 1:a \
-f hls -hls_playlist_type event -hls_time 10 \
-hls_segment_filename 'stream_%v/segment_%d.ts' \
-var_stream_map 'a:0,agroup:audio,name:vn a:1,agroup:audio,name:vc' \
-master_pl_name master.m3u8 \
stream_%v.m3u8
```

Figura 6.1. Generación de streaming HLS en directo con FFmpeg.

En este ejemplo, la opción '*-re*' hace que la lectura de los ficheros de audio de entrada se haga a la frecuencia de muestreo del fichero, en caso contrario FFmpeg intentaría leer los ficheros lo más rápido posible. De esta forma se puede simular la generación de contenido en tiempo real.

La opción '*-i*' indica un fichero de entrada y se puede repetir múltiples veces. En esta ocasión hay 2 ficheros de entrada, por lo que se repite 2 veces.

La opción '*-c:a*' establece el códec de audio con el que procesar los ficheros de entrada, en este caso es el AAC. De forma similar, '*-b:a*' especifica el bitrate del stream de audio resultante, en esta ocasión, 128 kilobits por segundo. Para establecer el número de canales de audio se emplea la opción '*-ac*' seguida del número de canales, en este caso, 2.

La opción '*-map*' se utiliza para seleccionar los datos de un flujo multimedia específico. En este ejemplo 0:a es el audio del primer fichero y 1:a, el audio del segundo.

Para especificar el formato de salida se usa la opción '*-f* '. Este ejemplo genera un streaming en directo, por lo que las "*Playlists*" HLS deben ser de tipo "*event*", como se indica con la opción '*-hls_playlist_type*', además de establecer el tiempo de actualización de las "*Playlists*" a 10 segundos mediante la opción '*-hls_time*'.

El nombre de los segmentos se establece con '*-hls_segment_filename*', que admite establecer marcadores como '*%v*' y '*%d*', los cuales serán sustituidos por el índice los "*Variant Streams*" y el número de secuencia de cada segmento, respectivamente.

Para indicar las variantes de la presentación multimedia se utiliza la opción '*var_stram_map*'. En esta ocasión existen 2 variantes llamadas "vn" y "vc".

Por último, se indica el nombre de la "*Master Playlist*" con '*-master_pl_name*' y el patrón para nombrar a las "*Media Playlists*" de salida.

Se puede generar una presentación multimedia equivalente utilizando el protocolo MPEG-DASH con el comando que se muestra en la Figura 6.2.

```
ffmpeg -re \
-i ../audio/Unaligned/VC.wav \
-i ../audio/Unaligned/FL.wav \
-map 0:a -c:a libopus -b:a:0 128k -mapping_family 255 -vn \
-map 1:a -c:a libopus -b:a:1 128k -mapping_family 255 -vn \
-f dash -dash_segment_type webm -seg_duration 10 -update_period 10 \
-adaptation_sets "id=0,streams=0 id=1, streams=1 " \
manifest.mpd
```

Figura 6.2. Generación de streaming DASH en directo con FFmpeg.

Algunas opciones de este comando son iguales que las anteriormente explicadas. En esta ocasión, los streams de audio se codifican utilizando el códec Opus, con un bitrate de 128 kilobits por segundo e indicando mediante la opción '*-mapping_family 255*'que los canales de audio se deben tratar como canales discretos sin ningún procesado adicional típico de disposiciones de audio como 5.1 o similares.

El formato contenedor para los segmentos del contenido de esta presentación es WebM, como se indica con '*-dash_segment_type*'. Además, '*-seg_duration*' establece que la duración de dichos segmentos ha de ser igual o inferior a 10 segundos y que el fichero de manifiesto deberá actualizarse cada 10 segundos de acuerdo a la opción '*-update_period*'.

Los "*Adaptation Sets*" se indican con la opción '*adaptation_sets*'. En este caso existen 2 de ellos, cada uno incluyendo el contenido de uno de los ficheros de audio de entrada.

Por último, se establece el nombre del fichero de manifiesto MPD de salida. Nótese que no se ha especificado ningún patrón para nombrar los segmentos multimedia, por lo tanto, FFmepg utilizará el patrón por defecto, esto es:

- Init_stream{Adaptation_Set_ID} para los segmentos de inicialización, donde '*Adaptation_Set_ID*' es el ID especificado en el comando anterior.

- Chunk-stream{Adaptation_Set_ID}-{Seq_Number} para los segmentos multimedia, donde '*Seq_Number*' es el número de secuencia del segmento correspondiente.

Los valores entre llaves en la lista anterior son los marcadores en los nombres de los ficheros de salida que FFmpeg sustituirá a medida que procese el contenido de entrada.

Es importante destacar que la implementación de FFmpeg para el protocolo MPEG-DASH presenta una característica relevante. Al finalizar una transmisión en directo, bien porque se ha interrumpido manualmente o bien porque se ha procesado por completo el contenido de los ficheros de entrada, FFmpeg realiza modificaciones automáticas en los archivos MPD. Uno de los cambios realizados consiste en la modificación del valor del atributo "*Type*" de "*dynamic*" a "*static*", reflejando que la presentación multimedia ya no es en directo, sino que está disponible "on-demand", como se comentó en el apartado 5.2.3.

6.1.2. Node.js

El back-end del servicio de streaming se fundamenta en Node.js en su versión 18.16.0.

Node.js es un entorno de ejecución de JavaScript de código abierto y multiplataforma. Ejecuta el motor de JavaScript V8, núcleo de Google Chrome, fuera del navegador. Permite crear programas de línea de comandos y es una de las opciones preferidas para desarrollar aplicaciones de back-end.

Una aplicación de Node.js corre en un solo proceso, sin crear nuevos hilos para atender cada petición. Por esta razón, proporciona un conjunto de funciones de entrada/salida asíncronas. Las bibliotecas estándar de Node.js previenen que el código JavaScript bloquee la ejecución del programa.

Por ejemplo, cuando una acción como acceder a un sistema de base de datos se realiza, Node.js retomará las operaciones cuando recibe una respuesta. Esto permite a Node.js gestionar un gran número de conexiones concurrentes sin la necesidad de trabajar con procesos multi-hilo [12].

A continuación se listan algunas ventajas de Node.js frente a otras alternativas para el desarrollo de aplicaciones de servidor, de acuerdo con [13]:

- Node.js es una buena solución para aplicaciones web en tiempo real por su alta eficiencia.

- Todo el código es JavaScript, esto agiliza el desarrollo de aplicaciones web al permitir el uso del mismo lenguaje de programación para desarrollar el código del lado del cliente o del servidor.
- Es compatible con otros lenguajes que son compilados o convertidos a JavaScript, como puede ser TypeScript, Scala, LiveScript, etc.
- El gestor de paquetes de Node.js (npm, del inglés "*Node Package Manager*") facilita el acceso a miles de paquetes reutilizables para crear nuevas aplicaciones.
- Es portable, pudiendo ser ejecutado en una gran gama de sistemas operativos entre los que se incluyen Linux, Mac OS y Windows.
- Cuenta con una comunidad muy activa, lo que significa que hay abundante información y documentación disponible.

6.1.3. Express.js

Además de Node.js, para el desarrollo del back-end se ha utilizado Express.js en su versión 4.18.2.

Express.js es un "*framework*" minimalista y flexible de Node.js para desarrollo de aplicaciones web. Dispone de una gran cantidad de utilidades, métodos y "*middleware*" HTTP que aceleran y simplifican la creación de robustas APIs [14].

La característica más destacable de Express.js es su potente sistema de enrutamiento de peticiones HTTP. Con Express.js, el código del servidor se puede estructurar en controladores que procesan las peticiones realizadas a determinadas rutas en función del método HTTP (GET, POST, PUT, DELETE, etc.). Estas rutas se corresponden con URLs que pueden contener parámetros que pueden ser empleados por los controladores para el procesado de las peticiones. Todo esto favorece la reutilización y mantenimiento del código de la aplicación.

Adicionalmente, Express.js ajusta el tipo de contenido de las respuestas HTTP en función de los recursos que transporten, si procede, y ajusta el código de las respuestas automáticamente. Por ejemplo, si al procesar una petición en la que se solicita un recurso y no se ha podido localizar en el servidor, Express.js ajusta el código de la respuesta a 404. No obstante, Express.js ofrece la flexibilidad de ajustar los códigos de las respuestas según la lógica que se programe manualmente.

6.1.4. MongoDB

Para este proyecto se ha usado MongoDB en su versión 6.0.6.

MongoDB es una base de datos NoSQL diseñada para facilitar el desarrollo de nuevas aplicaciones y la escalabilidad.

MongoDB almacena registros de datos como Documentos (***Documents***) JSON binarios (BSON, del inglés "*Binary JSON*"), que son agrupados en Colecciones (***Collections***).

Estas Colecciones son análogas a las tablas en bases de datos relacionales. Sin embargo, a diferencia de las bases de datos relacionales, los Documentos de una Colección no tienen por qué compartir el mismo Esquema (***Schema***), es decir, los Documentos no tienen por qué tener los mismos campos o el tipo de dato de un campo en particular puede variar entre todos los Documentos de la Colección [15].

Este diseño aporta a MongoDB una serie de ventajas sobre las bases de datos relacionales tradicionales, de acuerdo a [16]:

- Gran flexibilidad. Modelar la información como Documentos que no tienen por qué compartir el mismo Esquema permite almacenar y manipular cualquier tipo de dato. Al mismo tiempo, MongoDB soporta la creación de Esquemas para imponer que todos los Documentos compartan los mismos campos y tipos de datos, de esta forma se pueden construir bases de datos dependiendo de las necesidades de cada aplicación.
- Ampliamente compatible. Se puede acceder a la información con cualquier lenguaje de programación utilizando estructuras de datos nativas de estos lenguajes, como podrían ser los diccionarios en Python, objetos en JavaScript, mapas en Java, etc.
- Diseño adaptable a cambios. Es posible modificar los Esquemas de los Documentos de una Colección en cualquier momento y agregar nuevos registros de datos sin tener que eliminar los antiguos.
- Sistema de consulta potente. MongoDB está diseñado para acceder a la información de forma muy sencilla, haciendo que realizar uniones o transacciones sea poco común. No obstante, MongoDB permite realizar consultas y realizar complejos procesos de análisis con tan solo unas pocas líneas de código.

- Alto rendimiento. Al ser posible almacenar toda la información en un mismo Documento en lugar de dividirla en múltiples tablas, se puede recuperar en una sola consulta. Además, MongoDB proporciona mecanismos para hacer inserciones o actualizaciones de múltiples documentos en una misma operación, lo que mejora el rendimiento en comparación con las inserciones por lotes en bases de datos tradicionales.
- Soporte técnico y documentación disponible. MongoDB dispone de una extensa cantidad de documentación oficial y tutoriales, así como una comunidad activa en los foros.

6.1.5. Mongoose

Para este proyecto se ha usado Mongoose en su versión 7.0.3.

Mongoose es una biblioteca de modelado de datos de MongoDB para Node.sj. Permite modelar documentos de la base de datos como objetos JavaScript actuando como una herramienta de mapeo de documentos a objetos (ODM, del inglés "*Object-Document Mapper*"). Para ello, los objetos se asocian a modelos definidos en un esquema (*schema*). Además, simplifica la interacción con la base de datos al abstraer la sintaxis de MongoDB proporcionando una API asíncrona fácil de usar.

Mongoose también valida la creación de nuevas instancias de objetos para asegurar que cumplen las reglas definidas en el esquema de los modelos, realiza conversiones de tipos de datos automáticamente y permite definir métodos asociados a los modelos de objetos para añadir funcionalidades personalizadas.

6.1.6. React

Para este proyecto se ha usado React en su versión 18.2.0.

React es una biblioteca de JavaScript de código abierto para creación de interfaces de usuario compuestas por bloques más pequeños llamados componentes, que se combinan para formar pantallas, páginas y aplicaciones.

Estos componentes son funciones que utilizan una extensión de la sintaxis de JavaScript llamada JSX, que combina código JavaScript con lenguaje de marcas similar a HTML.

Los componentes reciben datos de entrada y devuelven lo que se debería ver en pantalla, React se encarga de actualizar la interfaz cada vez que un componente recibe nuevos datos [17].

Otro aspecto importante de React es la existencia de dos tipos de variables, variables con estado o referencias. Las variables con estado disparan una renderización de los componentes que están asociados a estas variables cuando se produce un cambio de valor de estas variables, mientras que las referencias no disparan renderizaciones al ser actualizadas, pero sus valores persisten entre renderizaciones consecutivas. En React, las variables típicas de JavaScript son eliminadas después de una renderización.

React se fundamenta en un paradigma de programación declarativo, a diferencia del enfoque imperativo utilizado en el desarrollo tradicional de interfaces de usuario. Esto se ejemplifica en la Figura 6.3 y la Figura 6.4, que muestran cómo se construiría la misma interfaz.

En la Figura 6.3 el código JavaScript describe una serie de elementos HTML que tienen que ser añadidos al documento, junto con una serie de parámetros y funciones para procesar los datos que puedan ser introducidos por el usuario. Esto ejemplifica un enfoque de programación imperativo.

```
// Crea faders de volumen para cada canal
for (let i = 0; i < maxChannelNumber; i++) {
        let fader = document.createElement('input')
        fader.setAttribute('type', 'range')
        fader.setAttribute('min', '0')
        fader.setAttribute('max', '1')
        fader.setAttribute('step', '0.1')
        fader.setAttribute('value', '0.5')
        fader.setAttribute('id', i)
        // Modifica la ganancia del nodo correspondiente
        fader.addEventListener('input', (event) => {
        let id = event.target.id
        let gain = event.target.value
        gainNodes[id].gain.value = gain
        })
        // Añade los faders al documento
        document.body.appendChild(fader)
}
```

Figura 6.3. Creación de interfaz de un mezclador de audio con JavaScript estándar.

En la Figura 6.4 el código se centra en describir el resultado que se desea obtener, que es la lista de "*faders*" que devuelve el componente en cuestión y el comportamiento que se desea lograr al interactuar con esta interfaz. Esto ejemplifica un enfoque de programación declarativo.

```
import Fader from './Fader'
export default function Mixer({ gains, setGains, numFaders }) {
        const faders = [];
        const setGain = (index) => (event) => {
                const nextGains = gains.map((gain, i) => {
                        if (i === index) {
                                return event.target.value;
                        } else {
                                return gain;
                        }
                });
            setGains(nextGains);
        }
        for (let i = 0; i < numFaders; i++) {
                faders.push(
                <Fader
                        key={i}
                        gain={gains[i]}
                        setGain={setGain(i)}
                />
            );
        }
        return (
                <div>
                        {faders}
                </div>
        )}
```

Figura 6.4. Creación de interfaz de un mezclador de audio con React.

Nótese también cómo el componente de la Figura 6.4 hace uso de otro componente más pequeño llamado "*Fader*" para construir la interfaz deseada.

6.1.7. Hls.js

Para este proyecto se ha usado hls.js en su versión 1.3.3.

Hls.js es una biblioteca de JavaScript que implementa un cliente HLS. Se basa en el elemento <video> de HTML5 y las **MSE** para la reproducción del contenido, esto significa que hls.js no es compatible con navegadores que no soporten las MSE [18].

La Figura 6.5 muestra un ejemplo de utilización de hls.js, en donde se comprueba en primer lugar que el navegador es compatible. Si es así, vincula una presentación HLS a un elemento <audio> de HTML5.

```
import Hls from 'hls.js'
const audioTag = new Audio()
if (Hls.isSupported()){
        const hls = new Hls()
        hls.loadSource('/media/Pirates_4channels/master.m3u8')
        hls.attachMedia(audioTag)
}
```

Figura 6.5. Ejemplo de uso de hls.js.

Es importante destacar que el control de pausa, reproducción, volumen, adelantar y retroceder en el tiempo se puede realizar con el elemento multimedia HTML al que el cliente HLS está vinculado.

6.1.8. Dash.js

Para este proyecto se ha usado dash.js en su versión 4.6.0.

Dash.js es una implementación de código abierto en JavaScript de un cliente MPEG-DASH. Es una iniciativa de la DASH Industry Forum para establecer un marco de trabajo de calidad para desarrollar reproductores de vídeo y audio que puedan reproducir contenido MPEG-DASH usando bibliotecas de JavaScript en el lado del cliente y aprovechando el conjunto de APIs de las MSE [19].

En la Figura 6.6 se muestra un ejemplo de cómo se puede utilizar dash.js.

```
import dashjs from 'dashjs'
// URL manifiesto dash
const src = '/media/Unaligned/manifest.mpd'
// Instancia reproductor dash
const player = dashjs.MediaPlayer().create()
// Inicia el reproductor dash y comienza la reproducción automáticamente (true)
player.initialize(audio, src, true)
```

Figura 6.6. Ejemplo de uso de dash.js.

Dash.js requiere instanciar en primer lugar un reproductor, el cual se inicializa indicando el URL de un manifiesto MPD DASH y un elemento multimedia HTML5, como puede ser <audio> o <video> que queda vinculado con el reproductor DASH.

De forma análoga a hls.js, el control de pausa, reproducción, volumen y demás se puede realizar con el elemento multimedia HTML al que el reproductor DASH queda vinculado.

6.1.9. Web Audio API

Web Audio API proporciona un potente y versátil sistema para procesar audio en la web. Este procesado se hace dentro de un Contexto de Audio (***Audio Context***), en el que se definen nodos que pueden realizar diversos tipos de operaciones, desde ajustar volumen hasta realizar una convolución. Estos nodos están conectados entre sí formando un grafo de enrutamiento de audio [20], como el de la Figura 7.16.

El flujo de trabajo típico en Web Audio API incluye:

1. Crear un Contexto de Audio.
2. Crear una o varias fuentes de audio.
3. Crear nodos intermedios encargados de procesar las señales de audio de las fuentes, como pueden ser nodos de ganancia, filtros, compresores, etc.
4. Definir el destino final de la señal de audio, por ejemplo, los altavoces del Pc.
5. Conectar los nodos entre sí.

Web Audio API admite señales de audio con distintas distribuciones de canal, incluso dentro del mismo Contexto de Audio.

A la hora de crear un nuevo Contexto de Audio, es necesario especificar la frecuencia de muestreo. Por defecto, la frecuencia de muestreo se ajusta con la del dispositivo de salida

del usuario, por ejemplo, la tarjeta de sonido del Pc. Web Audio API se encarga de realizar el remuestreo de las señales de audio cuya frecuencia de muestreo difiere de la frecuencia de muestreo del Contexto de Audio para garantizar una reproducción de audio correcta.

Esto se puede comprobar en la Figura 6.7, que muestra un ejemplo de uso de Web Audio API.

```
// crea un contexto de audio con la frecuencia de muestreo del dispositivo de salida
audioCtx = new AudioContext();
// buffer con un tono de 1 KHz de duracion 2 segundos
// frecuencia de muestreo de 192 KHz
f0 = 1e3;
d = 2;
sr = 192e3;

buffer = audioCtx.createBuffer(1,d*sr,sr);
// rellena el buffer
channelData = buffer.getChannelData(0);
for (let n = 0; n < channelData.length; n++){
        channelData[n] = Math.cos(2*Math.PI*f0*n/sr);
}
// Crea un nodo fuente con el buffer
const source = audioCtx.createBufferSource();
source.buffer = buffer;
// Conecta a la salida
source.connect(audioCtx.destination);
// reproduce
source.start();
```

Figura 6.7. Ejemplo de uso de Web Audio API.

El tono que se genera tiene una frecuencia de muestreo de 192 KHz, mientras que el Contexto de audio se ha inicializado a una frecuencia de muestreo de 48 KHz. No obstante, la reproducción es correcta gracias a las operaciones de remuestreo, en este caso hacia abajo, que Web Audio API realiza automáticamente.

6.1.10. Docker

Para este proyecto se han usado Docker en su versión 20.10.

Docker es una plataforma abierta para desarrollar, distribuir y ejecutar aplicaciones. Permite separar las aplicaciones de la infraestructura subyacente al proporcionar la capacidad de empaquetarlas y ejecutarlas en entornos aislados llamados **contenedores**.

Estos contenedores contienen todo lo necesario para ejecutar la aplicación, por lo que no hay que preocuparse de si las dependencias de la aplicación están instaladas en el host. Además, al ser muy ligeros, se pueden ejecutar simultáneamente varios contenedores en el mismo host sin problema [21].

Gracias a esto, el despliegue de aplicaciones es mucho más consistente que de la forma tradicional, ya que pueden ser ejecutadas en cualquier sistema operativo, incluso si no son compatible de forma nativa.

Los contenedores no son más que instancias de **imágenes**. Las imágenes son representaciones estáticas de una aplicación o servicio, y son creadas a partir de archivos comúnmente llamados ***Dockerfile*** los cuales tienen un aspecto similar al ejemplo de la Figura 6.8, y son donde se especifican las instrucciones y comandos para construir la imagen deseada.

```
FROM node:18
WORKDIR /usr/src/app
# Copia el package.json y el package-loc.json e instala dependencias
COPY package*.json ./
RUN npm install
# Copia todos al directorio de trabajo (salvo los que aparecen en el .dockerignore)
COPY . .
# El servidor escucha en el puerto 8080
EXPOSE 8080
CMD ["node","main.js"]
```

Figura 6.8. Ejemplo de archivo Dockerfile.

Cada contenedor tiene su propio sistema de archivos. Cuando un contenedor se destruye, también lo hace toda la información que este contenía. No obstante, es posible crear y montar volúmenes de datos o asociar directorios del host al sistema de archivos del contenedor para la persistencia de información.

Además de permitir ejecutar aplicaciones de forma aislada, Docker permite configurar redes virtuales en las que los distintos contenedores pueden intercambiar información entre sí con un servicio de resolución de nombres de dominio que asocia las direcciones de cada contenedor con su nombre, facilitando el direccionamiento.

Una forma de describir servicios completos es mediante un archivo llamado comúnmente como **docker-compose**, que permite definir los nombres de los contenedores que se desplegarán, las imágenes a partir de las cuales se crearán estos contenedores, la asociación de puertos de los contenedores con el host, los volúmenes de datos o directorios que se desean montar para la persistencia de información y las redes virtuales para la comunicación entre contenedores. Por defecto, todos los contenedores del archivo se desplegarán dentro de la misma red virtual, si no existe, se crea automáticamente.

La Figura 6.9 es un ejemplo de un archivo docker-compose en el que se describen dos contenedores, uno con nombre "*front*end" y otro con nombre "*backend*". Para cada contenedor, se especifica el directorio donde se encuentra el fichero Dockerfile del que construir la imagen de cada contenedor y la asociación de puertos de los contenedores con los del host, entre otras instrucciones.

```
version: '3.8'

services:
  frontend:
    build: ./client
    ports:
      - '3000:3000'
  backend:
    build: ./server
    restart: always
    ports:
      - '8080:8080'
    volumes:
      - ./media:/usr/src/media
```

Figura 6.9. Ejemplo archivo docker-compose.

Los contenedores ejecutados en este proyecto están basados en las siguientes imágenes:

- node:18, la imagen oficial de Docker de Node.js.
- mongo:6.0.6, la imagen oficial de Docker de MongoDB
- mongo-express: 1.0.0-alpha.4, interfaz de administración de MongoDB basada en Express.js.

6.1.11. Especificaciones del equipo de pruebas

Para realizar pruebas de funcionamiento se ha utilizado un equipo Ubuntu 22.04.2 LTS "Jammy" con las siguientes especificaciones hardware:

- Arquitectura: x86_64
- CPU: High Performance Datacenter vCPU
  - Familia de CPU: 6
  - Modelo: 61
  - Núcleos: 2
- Virtualización: VT-x
  - Proveedor de hipervisor: KVM
  - Tipo de virtualización: completa
- Cachés:
  - L1d: 64 KiB (2 instancias)
  - L1i: 64 KiB (2 instancias)
  - L2: 4 MiB (1 instancia)
  - L3: 16 MiB (1 instancia)
- NUMA:
  - Nodo(s) NUMA: 1
  - CPU(s) del nodo NUMA0: 0, 1
- Memoria del sistema
  - tamaño: 4GiB
  - capacidades: ecc
  - configuración: errordetection=multi-bit-ecc
  - Banco:
    - memoria RAM DIMM
    - fabricante: QEMU
    - ranura: DIMM 0
    - tamaño: 4GiB

6.1.12. Base de datos de HRTFs del CIPIC

Para la espacialización de la mezcla de audio de las actuaciones se usarán HRTFs.

La base de datos de HRTFs del CIPIC (*Center for Imaging Processing and Integrated Computing*) es una base de datos de dominio público que contiene HRTFs obtenidas en el U. C. Davis CIPIC Interface Laboratory. En su versión 1.0 se incluyen las respuestas impulsivas de 45 sujetos para un total de 1250 direcciones (25 distintos azimuts y 50 distintas elevaciones) [22].

Los datos están disponibles en ficheros .mat, adecuados para su lectura con MATLAB. Para cada HRTF se tiene la respuesta correspondiente al oído izquierdo y oído derecho. Esto permite espacializar una fuente de sonido monofónica, resultando en una señal de audio estereofónica.

Las coordenadas de las HRTFs se definen utilizando una combinación de azimut y elevación. El azimut representa la dirección horizontal, mientras que la elevación representa la dirección vertical. El convenio de coordenadas es el que aparece representado en la Figura 6.10.

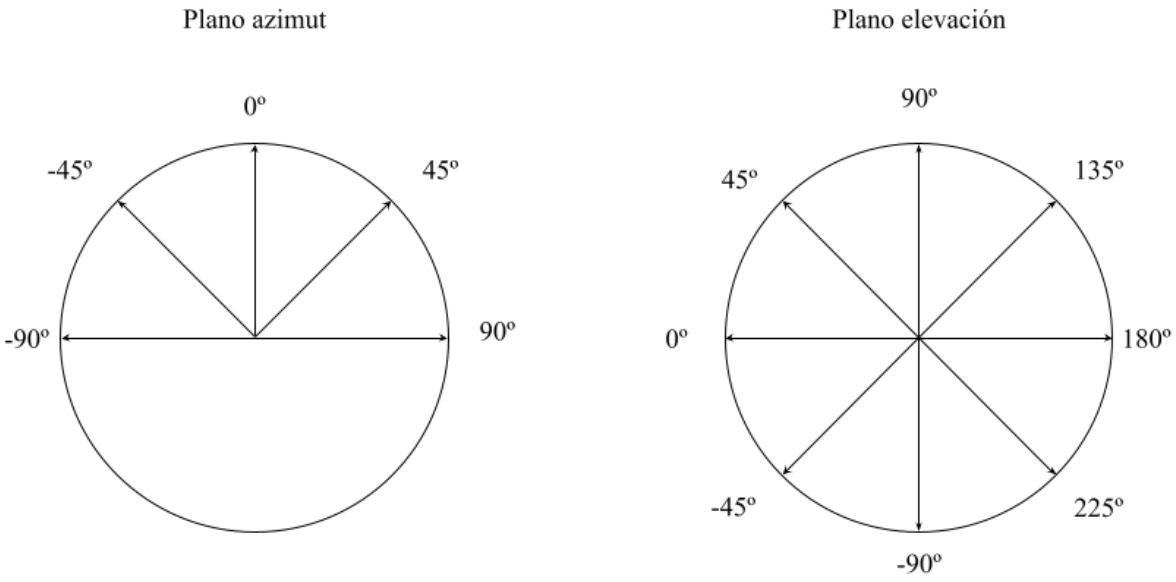

Figura 6.10. Convenio de coordenadas de la base de datos de HRTFs del CIPIC.

Nótese como el azimut está definido dentro del intervalo [-90º, 90º], mientras que la elevación pertenece al intervalo [-90º,270º). A continuación, se exponen algunos ejemplos de cómo se denotarían ciertas direcciones siguiendo este convenio.

- Azimut = 0º, elevación = 0º es un punto situado justamente en frente.

- Azimut = 80º, elevación = 0º es un punto situado a la derecha, delante del oyente.
- Azimut = -80º, elevación = 0º es un punto situado a la izquierda, delante del oyente.
- Azimut = 0º, elevación = 180º es un punto situado justamente a la espalda.
- Azimut = 0º, elevación = 90º es un punto situado justamente encima.
- Azimut = 80º, elevación = 180º es un punto situado a la derecha, detrás del oyente.
- Azimut = -80º, elevación = 180º es un punto situado a la izquierda, detrás del oyente.

6.1.13. Git y GitHub

Para este proyecto se ha usado Git en su versión 2.39.2.

Git es un software muy popular de control de versiones gratis y de código abierto. Git soporta tanto repositorios locales en la máquina en la que está instalado, como repositorios remotos alojados en plataformas como GitHub, GitLab o Bitbucket.

GitHub es una plataforma ampliamente conocida que ofrece un servicio de alojamiento de repositorios Git en la nube. Además, GitHub facilita la colaboración entre desarrolladores y la distribución de código.

6.1.14. Software adicional

Para realizar las operaciones de remuestreo y reasignación de localizaciones y poblar la colección de HRTFs de la base de datos se hizo uso de las bibliotecas Scipy en su versión 1.10.0, concretamente se utilizó la función "*resamplePoly*" del módulo "*Signal*", Numpy en su versión 1.23.5 y Requests en su versión 2.28.1, con un intérprete de Python en su versión 3.10.9.

Para poder leer los conjuntos de HRTFs en Python, se usó la función "*loadmat*" del módulo "IO" de la biblioteca Scipy, la misma versión especificada anteriormente.

Para realizar las capturas de tráfico se usó TShark en su versión 3.6.2, herramienta con la que también se obtuvieron los datos necesarios para la representación de los consumos de ancho de banda en la ingesta de datos multimedia y en el consumo.

Para representar estos datos se hizo uso del módulo "*pyplot*" de la biblioteca de Python "*Matplotlib*" en su versión 3.7.0.

6.2. Métodos

A lo largo de este apartado se exploran distintos enfoques y procedimientos y se expondrán las razones fundamentales que llevaron a la elección de los métodos específicos adoptados para desarrollar el servicio de streaming.

6.2.1. Consideraciones para el procesado de las señales de audio

Atendiendo a las características de las señales de audio resultantes descritas en el apartado 3.1.3 más atrás, se han tomado las siguientes decisiones para el procesado de las mismas:

1. Las posiciones con las que el usuario podrá interactuar se corresponden con las de los distintos micrófonos dispuestos en el escenario, esto es, los micrófonos cercanos de cada grupo de instrumentos y el micrófono principal.
2. Debido a que las señales de audio de los instrumentos en los micrófonos cercanos son atenuadas salvo la de su grupo respectivo, no es eficaz aplicar técnicas de espacialización en estas condiciones, puesto que algunos instrumentos no se perciben correctamente.
3. No existen razones para ajustar la mezcla individual de instrumentos pertenecientes a otros grupos en el micrófono cercano del grupo asociado. Por ejemplo, tiene más sentido que, en lugar de escuchar los contrabajos desde la posición de los timbales, se seleccione directamente la posición del grupo de instrumentos que se desea escuchar y ajustar su nivel correspondiente.
4. Las señales obtenidas en el micrófono principal sí que permiten realizar la espacialización de la escena sonora correctamente, manteniendo además la sensación de ambiente que produce la sala de actuación, por lo que la mezcla binaural se realizará con estas señales cuando el usuario seleccione la posición del micrófono principal.
5. El procesado de las señales de audio se realizará en el cliente, el cual incluye el ajuste del nivel de los instrumentos y la espacialización en la mezcla. El servidor se encargará de retransmitir el contenido multimedia junto con otros datos necesarios para el correcto tratamiento del audio demandado en todo momento.

De esta forma se consigue reducir la cantidad de recursos necesarios en el servidor y simplificar la complejidad de las señales de audio que se van a transmitir, mejorando la eficiencia de transmisión al enviar solamente información útil para la reproducción del contenido.

6.2.2. Transmisión de audio procedente de distintas posiciones

El audio de los instrumentos procedentes de distintas posiciones del escenario se puede entender como distintas versiones de la misma presentación multimedia. Esto es, todos los instrumentos están interpretando la misma obra musical, pero evidentemente el sonido procedente de cada grupo de instrumentos es único y distinto de los demás grupos.

Por esta razón, el audio procedente de cada grupo de instrumentos captado por su micrófono cercano correspondiente o por el micrófono principal se puede enviar al cliente bien como un "*Variant Stream*" de HLS o un "*Adaptation Set*" de MPEG-DASH.

Esto hace que el número de variantes o conjuntos de adaptación sea igual al número de micrófonos cercanos dispuestos en el escenario además del micrófono principal.

6.2.3. Transmisión de audio de distintos instrumentos en una posición

Ajustándose a las soluciones descritas en el apartado 6.2.1, el procedimiento para la transmisión de audio procedente de distintos instrumentos al seleccionar una posición seguirá las siguientes pautas:

1. Las señales de los micrófonos cercanos se enviarán como señales de audio estereofónicas, con la siguiente distribución de canal:
    a. El canal izquierdo contendrá la señal de audio del grupo de instrumentos correspondiente al micrófono cercano.
    b. El canal derecho contendrá la mezcla de señales de audio de todos los instrumentos captados excepto la del grupo cercano.

Con esta distribución de canales, el cliente puede extraer las señales de cada uno de los canales y procesarlas por separado, siendo posible de esta manera ajustar el nivel de señal de los instrumentos asociados al micrófono cercano para realzarlos o suprimirlos de la mezcla, e igual ocurre con la señal del resto de la orquesta. Una vez procesadas, las señales se combinarán para formar la mezcla estereofónica deseada.

Esto significa que, aunque se envíen señales estereofónicas en las que en el canal izquierdo transporta únicamente la señal del grupo de instrumentos correspondiente al micrófono cercano y en el canal derecho la señal del resto de la orquesta, el resultado es una señal estéreo en la que las ambas señales de audio se escuchan por ambos altavoces, tanto el izquierdo como el derecho, proporcionando una experiencia estéreo completa.

En la Figura 6.11 se ofrece un ejemplo de este tratamiento para las señales obtenidas en el micrófono cercano del grupo de flautas.

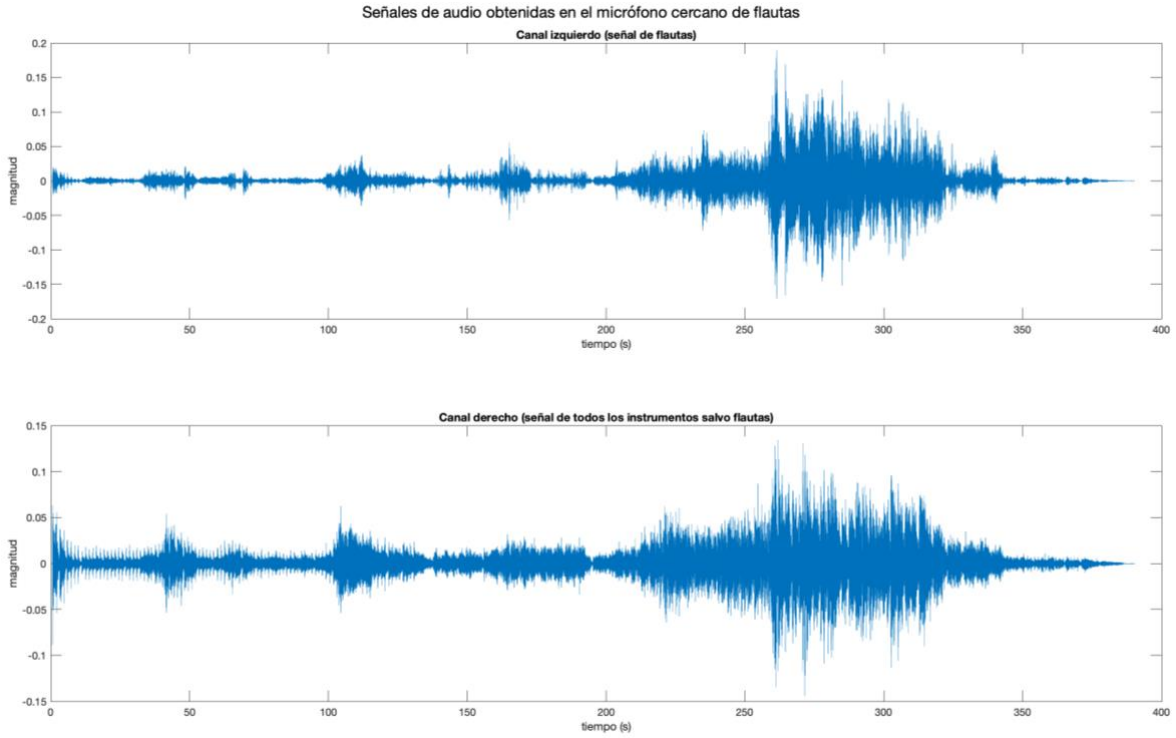

Figura 6.11. Ejemplo de señales de audio obtenidas en el micrófono cercano del grupo de flautas.

2. Las señales de audio en la posición del micrófono principal serán señales en formato multicanal, con un número de canales igual al número de grupos de instrumentos del algoritmo de separación de fuentes. En cada uno de los canales de audio se enviará la señal de cada grupo de instrumentos de la orquesta, como se muestra en la Figura 6.12. Esto permite ajustar los niveles de cada una de las señales de la presentación y aplicar técnicas de espacialización a cada una de las fuentes sonoras.

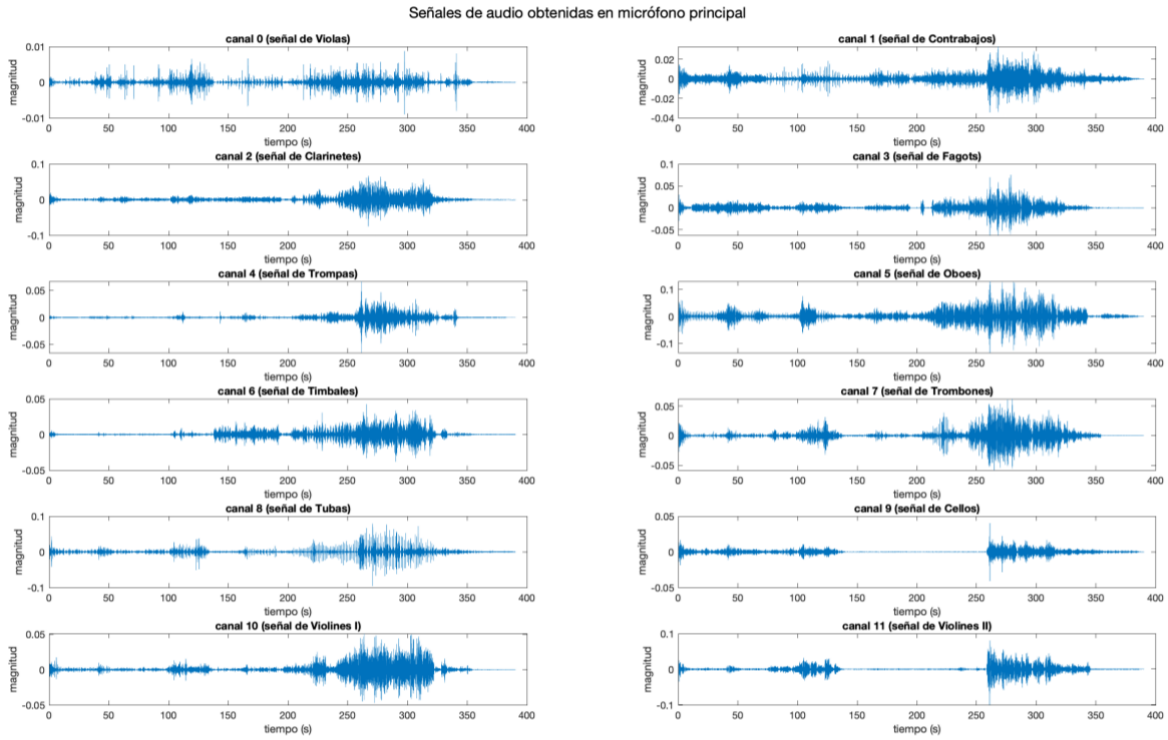

Figura 6.12. Ejemplo de señales de audio obtenidas en el micrófono principal.

Las ventajas de esta solución en comparación con la de crear más "*variant streams*" o "*adaptations sets*" y combinarlos para crear la mezcla de todas las señales radican en que las implementaciones de reproductores HLS o DASH descritas en los apartados 6.1.7 y 6.1.8 no se permite realizar esta operación.

Esto hace que sea necesario crear múltiples instancias de estos reproductores que lancen peticiones al servidor para recuperar el contenido multimedia correspondiente, lo cual implica manejar varios "*streams*" y varias conexiones HTTP en paralelo.

Transmitiendo audio multicanal, todo el contenido multimedia se envía en el mismo "*stream*", simplificando la complejidad de los archivos de manifiesto y la gestión de las conexiones HTTP.

6.2.4. Elección del protocolo de streaming

Las alternativas consideradas para servir el contenido multimedia se basaron en las soluciones comentadas en el apartado 5.

El enfoque de segmentación de archivos de audio de Spotify resultó ser demasiado ineficiente puesto que requeriría implementar múltiples mecanismos de control y gestión para el funcionamiento en directo, lo cual no es necesario teniendo en cuenta que los protocolos HLS y MPEG-DASH ofrecen soluciones más adecuadas para la transmisión de contenido multimedia en streaming.

Inicialmente se consideró utilizar HLS. Sin embargo, se encontraron ciertas limitaciones prácticas que llevaron a la elección de MPEG-DASH como protocolo de soporte multimedia para el servicio desarrollado. Dichas limitaciones se describen a continuación.

La limitación más decisiva de HLS es que es compatible con una restringida gama de códecs, como se comentó en el apartado 5.1.4. MP3, AC-3 y E-AC3 no permiten codificar audio con un gran número de canales, teniendo que emplearse el códec AAC.

En teoría, AAC soporta un número máximo de canales de 48 [23]. Sin embargo, con FFmpeg no es posible codificar audio con AAC con una distribución de canales discreta, sino que tiene que ajustarse a algún diseño como 5.1 o 7.1, lo que significa que ninguno de los códecs de audio compatibles con HLS se ajusta a los requisitos de diseño de la aplicación.

Por otro lado, MPEG-DASH es compatible con una gama de códecs de audio mucho más amplia que HLS, lo que permite usar códecs como Opus, que soporta señales de audio de hasta 255 canales con distribuciones discretas.

Adicionalmente, aunque HLS está ampliamente adoptado en servicios de streaming no es un estándar internacional, al contrario que MPEG-DASH.

Por último, la implementación de HLS de FFmpeg no modifica los ficheros de manifiesto de presentaciones en directo para que sean consideradas como "*on-demand*" una vez la transmisión ha finalizado. Sin embargo, esto sí que ocurre con MPEG-DASH.

Por estos motivos, el protocolo de streaming que se eligió como base del servicio de streaming desarrollado fue finalmente MPEG-DASH.

6.2.5. Espacialización de las señales de audio

Para la espacialización de las señales de audio se utilizarán HRTFs debido a la simplicidad del procesado de audio necesario para conseguir de forma muy eficaz una experiencia de sonido tridimensional para señales estereofónicas.

Este procesado de audio implica realizar operaciones de convolución de las muestras de audio de las señales de cada grupo de instrumentos obtenidos en el micrófono principal con las respuestas impulsivas del canal izquierdo y derecho de cada una de las HRTFs asociadas a las direcciones en las que se quiere situar cada grupo de instrumentos. Dichas respuestas impulsivas reciben el nombre de HRIRs (*Head-Related Impulse Response*). No obstante, aunque técnicamente incorrecto, se utilizará el término HRTF por simplificación.

La Figura 6.13 muestra un ejemplo de la operación descrita para espacializar la señal del grupo de flautas, nótese que la señal de entrada es monofónica, mientras que la de salida es estereofónica.

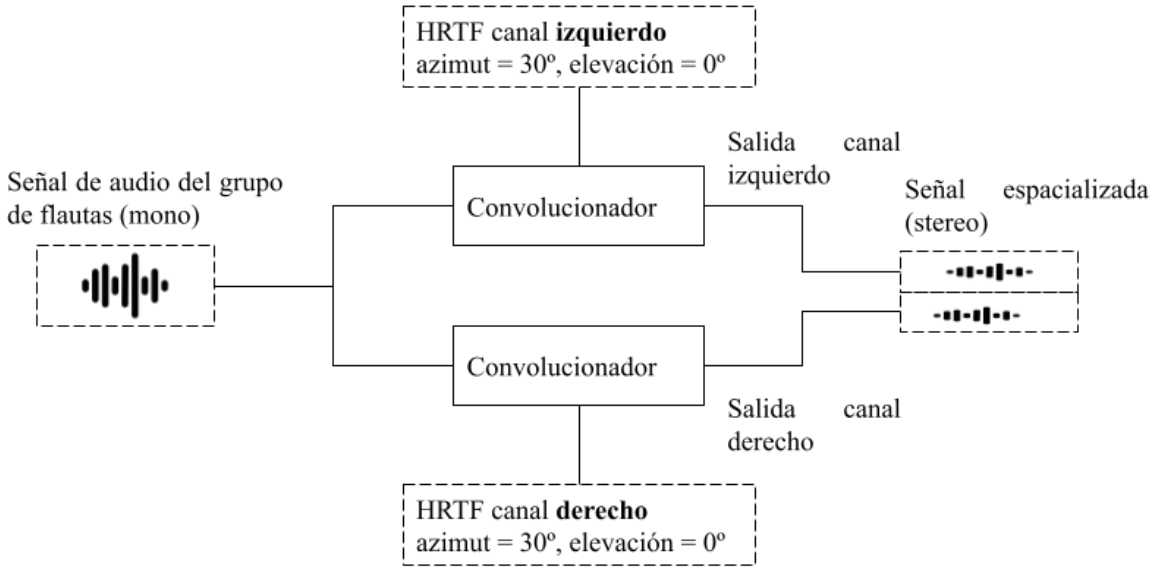

Figura 6.13. Ejemplo de espacialización de una señal de audio con HRTFs.

La operación representada en la Figura 6.13 daría como resultado una escena sonora igual a la mostrada en la Figura 6.14, en donde la flecha indica la dirección en la que el usuario se encontraría mirando. La señal de audio se percibiría como si estuviera situada unos 30º a la derecha, es decir, se percibiría con más presencia en el oído derecho.

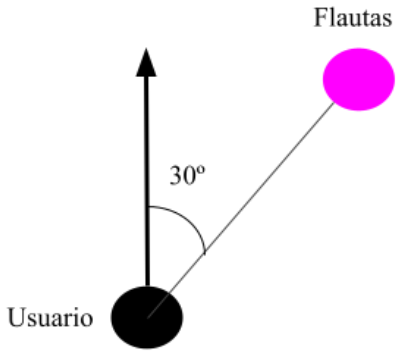

Figura 6.14. Representación escena sonora resultante de aplicar la operación mostrada en la Figura 6.13.

Web Audio API permite definir nodos que realizan esta operación de convolución, a los que hay que especificar como respuesta impulsiva la HRTF de la dirección deseada (ver apartado 6.1.12) para realizar la espacialización.

Las HRTFs deberán ser reemplazadas cada vez que se produce una rotación del usuario. En la Figura 6.15 representa el resultado de aplicar una rotación en la escena sonora inicial, representada en la Figura 6.14.

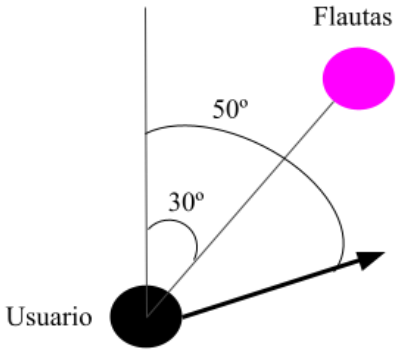

Figura 6.15. Ejemplo del resultado de una rotación del usuario en la escena sonora.

En la Figura 6.15 se observa que el usuario se encuentra mirando en dirección 50º hacia la derecha. En esta situación, debería percibir el sonido de las flautas llegando de una dirección de 20º hacia la izquierda, esto es, con más presencia en el oído izquierdo.

Esto implica cambiar las HRTFs utilizadas en un principio por otras cuya dirección sea de 20º hacia la izquierda en azimut.

6.2.6. Almacenamiento y envío de HRTFs

En los ejemplos de uso de los nodos convolucionadores en [20], las respuestas al impulso se cargan a partir de un fichero .wav. No obstante, la solución adoptada para enviar las repuestas impulsivas al cliente se basa en utilizar el formato JSON debido a las siguientes ventajas:

- Flexibilidad en la estructura de datos: Además de las muestras de las respuestas impulsivas de cada canal, se pueden incluir otro tipo de datos como, por ejemplo, la orientación en azimut y elevación y la frecuencia de muestreo.
- Fácil identificación de cada HRTF: Las respuestas impulsivas se identifican por sus orientaciones espaciales azimut y elevación. Solo existe 1 respuesta para 1 orientación en el espacio. Incluir esta información en el documento JSON junto con las muestras de audio hace que sea muy sencillo recuperar las HRTFs deseadas de la base de datos, además de facilitar otro tipo de operaciones, como calcular rotaciones.
- Formato adecuado para intercambio de datos: El uso de JSON permite que el cliente puede recuperar fácilmente las HRTFs y su información asociada mediante una API REST.

6.2.7. Adaptación de convenio de coordenadas

El convenio de coordenadas usado por el CIPIC y descrito en el apartado 6.1.12 dificulta realizar operaciones de rotación.

Por ejemplo, sea un punto localizado en azimut igual a -80º y elevación 0º, el usuario rota hacia la derecha 60º, la dirección del punto tras la rotación es -140º en azimut y 0º en elevación.

Sin embargo, el valor de -140º de azimut no está definido en el convenio de coordenadas de la base de datos de HRTFs del CIPIC, por lo que habría que expresar el punto con dirección -140º en azimut y 0º de elevación como -40º en azimut y 180º en elevación, como se muestra en la Figura 6.16.

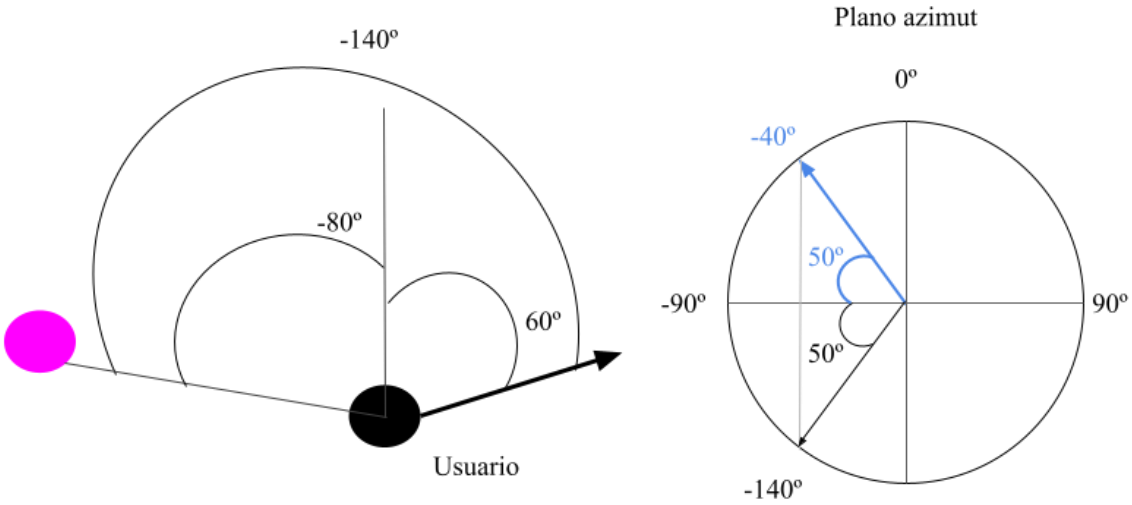

Figura 6.16. Cálculo de rotaciones con el convenio de la base de datos de HRTFs del CIPIC.

Las operaciones de rotación se podrían simplificar utilizando otro convenio de coordenadas en el que los valores de azimut estuvieran definidos en el intervalo [-180º, 180º) y los de elevación en el intervalo [-90º, 90º], como se muestra en la Figura 6.17.

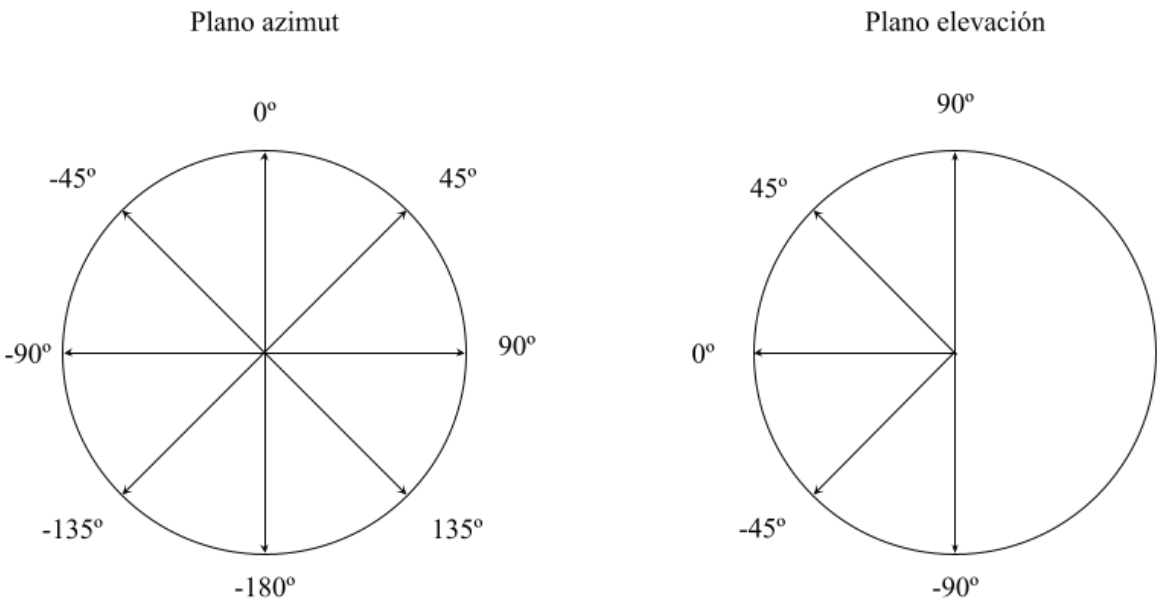

Figura 6.17. Convenio de coordenadas propuesto para facilitar las operaciones de rotación con las HRTFs de la base de datos del CIPIC.

Denotando el valor de azimut como ϕ, el de elevación como θ, los valores de azimut y elevación tras aplicar el cambio de convenio, ϕ' y θ', se calculan de la siguiente manera:

$$\forall\theta \in [-90º, 90º] \rightarrow \begin{cases} \phi' = \phi \\ \theta' = \theta \end{cases} \quad (1)$$

$$\forall\theta \in (90º, 270º) \rightarrow \begin{cases} \phi' = sign(\phi + \epsilon) \cdot 180 - \phi \\ \theta' = 180 - \theta \end{cases} \quad (2)$$

En la ecuación (2), sign es la función matemática signo y ε es el "épsilon de la máquina", esto es, el menor valor numérico que puede representar el computador en el que se está realizando la operación que sumado a 1,0 se obtiene un valor distinto de 1,0.

De esta forma, la dirección del ejemplo anterior de -40º en azimut 180º de elevación se transforma en -140º en azimut y 0º de elevación.

Igual ocurre con las direcciones de ejemplo en el apartado 6.1.12, las cuales ahora se denotarían de la siguiente forma:

- Azimut = 0º, elevación = 0º es un punto situado justamente en frente.

- Azimut = 80º, elevación = 0º es un punto situado a la derecha, delante del oyente.
- Azimut = -80º, elevación = 0º es un punto situado a la izquierda, delante del oyente.
- Azimut = -180º, elevación = 0º es un punto situado justamente a la espalda.
- Azimut = 0º, elevación = 90º es un punto situado justamente encima.
- Azimut = 100º, elevación = 0º es un punto situado a la derecha, detrás del oyente.
- Azimut = -100º, elevación = 0º es un punto situado a la izquierda, detrás del oyente.

Redefinir las localizaciones de todas las HRTFs de la base de datos del CIPIC empleando este convenio permite aplicar rotaciones de una forma más sencilla, rápida y directa.

6.2.8. Remuestreo de HRTFs

No se entrarán en detalles de la operación de remuestreo, si es de especial interés para el lector, se recomienda leer el capítulo 4.6 de [24].

La operación de remuestreo es una técnica de procesado de señales digitales que permite cambiar la frecuencia de muestreo de una secuencia, aumentándola o disminuyéndola.

Las HRTFs de la base de datos del CIPIC fueron obtenidas con una frecuencia de muestreo de 44100 Hz, sin embargo, las señales que se pretenden distribuir tendrán una frecuencia de muestreo de 48000 Hz. Como las muestras de las HRTFs se envían en JSON en lugar de en ficheros con extensión "*wav*", Web Audio API no realizará ninguna operación de remuestreo.

Para que las HRTFs tengan una frecuencia de muestreo de 48000 Hz, a la hora de poblar la colección de HRTFs de la base de datos y después de haber adaptado las coordenadas al convenio descrito en el apartado 6.2.7, a cada una de estas se le aplica la función '*resample_poly*' del módulo "*signal*" de la biblioteca Scipy de Python.

Esta función recibe como parámetros las muestras de la secuencia que se quiere procesar y un valor de "*up*" y "*down*". Estos valores son factores que indican el número de veces que la secuencia de entrada es, en primer lugar, interpolada, lo que aumenta el número

de muestras de la misma en "*up*" veces, después es filtrada por un filtro paso bajo y, finalmente, diezmada, que disminuye el número de muestras "*down*" veces.

Estos valores de "*up*" y "*down*" se obtienen como la relación entre la frecuencia de muestreo deseada y la frecuencia de muestreo original de la secuencia de entrada. En este caso, el valor fraccionario 48000/44100 puede ser simplificado a 160/147. El valor del parámetro "*up*" es precisamente el valor del numerador, mientras que "*down*" el del denominador. Pueden introducirse los valores de 48000 y 44100, la función internamente reducirá todo lo posible el valor fraccionario que describen hasta que sea irreducible, en este caso es 160 y 147, para realizar el menor número de operaciones.

6.2.9. Ajuste de localizaciones tras aplicar rotaciones

Las HRTFs empleadas para la espacialización están definidas para localizaciones discretas del espacio. Puede ocurrir que, tras aplicar una rotación, un grupo de instrumentos quede localizado en una dirección en la que no existe una HRTF definida.

En estas situaciones, al grupo de instrumentos se le asigna la dirección más cercana para la que se dispone de una HRTF en la base de datos.

Por ejemplo, existen HRTFs definidas para direcciones de azimut de -80º y -65º. Si tras aplicar una rotación, un grupo de instrumentos queda localizado en dirección -75º, se le asignará una dirección de -80º de azimut.

6.2.10. Control de versiones

Además de los procedimientos para el desarrollo e implementación, el control de versiones no solo es una buena práctica, sino que constituye una parte fundamental de este proyecto por las siguientes razones.

- Permite registrar de forma sistemática y ordenada los cambios realizados en el código fuente y otros archivos, proporcionando un historial completo del progreso del proyecto.
- En caso de introducir cambios no deseados o errores en el código, el control de versiones permite regresar a una versión anterior del proyecto, restaurando rápidamente una versión funcional.
- Facilita trabajar en nuevas características y experimentos a través de la creación de ramas aislada. Lo que permite trabajar de forma independiente sin afectar a la versión principal del proyecto.

- Facilita monitorizar el progreso del proyecto al poder comprobar los cambios realizados en el código a lo largo del tiempo.

7. DESARROLLO

En este apartado se describe en detalle la materialización de las ideas siguiendo los procedimientos y métodos comentados más atrás, haciendo uso de las tecnologías descritas en el apartado 6.1.

Se abordarán los aspectos clave de implementación que permitieron alcanzar los objetivos del proyecto y se describirán los intercambios de información entre los distintos componentes del servicio.

7.1. Arquitectura del servicio

La Figura 7.1 proporciona una descripción visual de la arquitectura del servicio de streaming desarrollado, con la representación de los contenedores desplegados, las tecnologías más relevantes y otros aspectos relacionados con la comunicación y persistencia de datos.

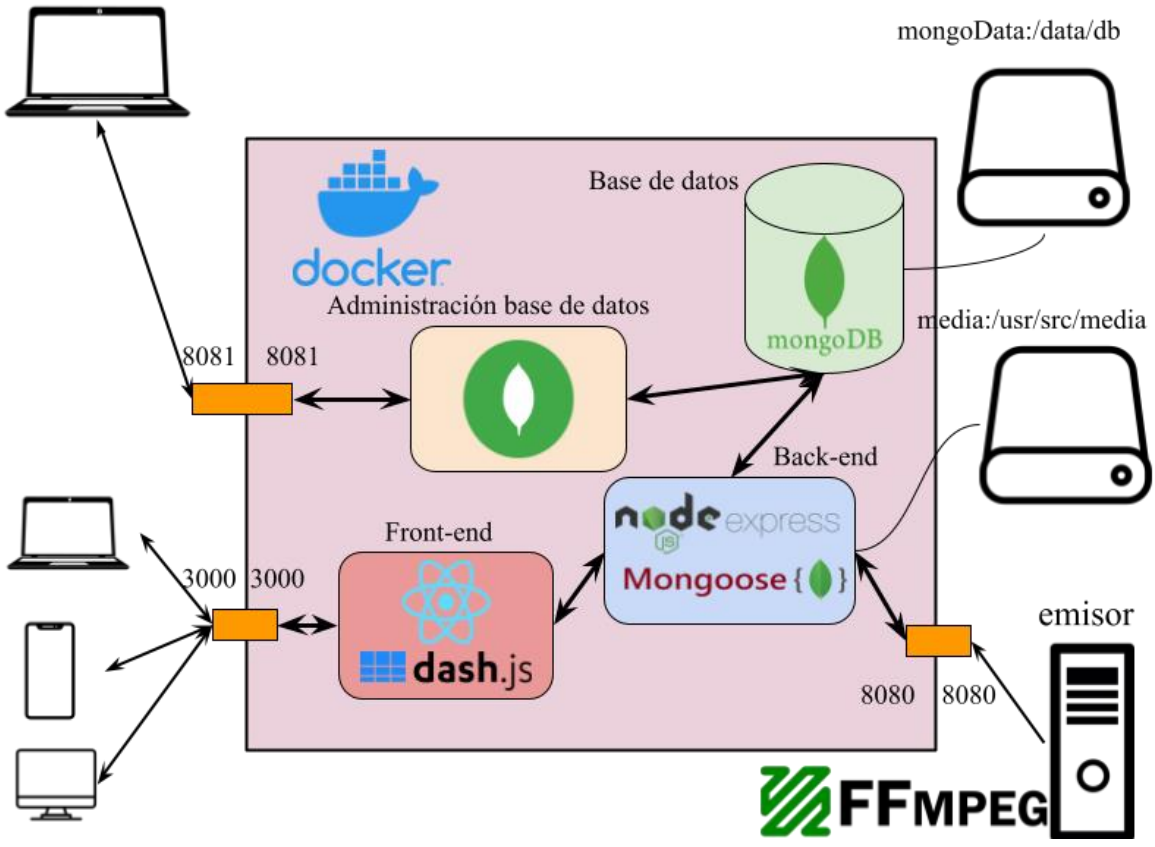

Figura 7.1. Esquema de la arquitectura del servicio de streaming.

El despliegue de la infraestructura definida con Docker se hace mediante un fichero *docker-compose.yml*. Se puede observar que existen cuatro contenedores diferenciados que

ejecutan distinto software y tecnologías ya comentadas en el capítulo 6. Estos contenedores son:

- Front-end: Construido sobre React. Para reproducir el contenido multimedia utiliza dash.js. La imagen de Docker a partir de la que se construye el contenedor es la oficial de Node. El front-end se encarga no solo de presentar el contenido de la actuación de música, sino también de ofrecer una interfaz dinámica con la que el usuario interactúa y de realizar todo el procesado de audio, como ajuste de la mezcla y espacialización, en tiempo real.
- Back-end: Basado en Node-Express. Igual que el front-end, la imagen sobre la que se instancia el contenedor es la oficial de Node en Docker. Procesa las distintas peticiones recibidas de los clientes, entre las que se incluyen no solo las del front-end, sino la de los equipos que están generando contenido multimedia con ayuda de software como FFmpeg para ser distribuido en streaming. Interactúa con la base de datos mediante mongoose para almacenar y recuperar información requerida por el front-end.
- Base de datos: Se trata de una base de datos MongoDB que se encarga de almacenar la información de múltiples instancias de los modelos de datos definidos en el apartado 7.2. El contenedor se crea a partir de la imagen oficial de MongoDB para Docker.
- Interfaz de administración de base de datos: Se crea con la imagen mongo-express. Es una interfaz web de administración de la base de datos basada en Express. Permite comprobar cuántas colecciones hay en la base de datos y realizar operaciones, por ejemplo, agregar más colecciones o documentos, eliminarlos o modificar campos de los documentos.

Todos los contenedores de la figura están definidos dentro de una misma **red virtual** de tipo "*bridge*". Este tipo de redes conectan los distintos contenedores en la **capa de enlace** del modelo TCP/IP, es decir, es el equivalente a una red local física.

Los contenedores pertenecientes a este tipo de redes están aislados del resto de redes, en el sentido de que mantienen un espacio de direcciones independientes al del host y al de otras redes de tipo "*bridge*", por lo que, por defecto, pueden comunicarse entre ellos, pero no con hosts del exterior.

Además, las redes de tipo "*bridge*" proveen un servicio de resolución de nombres de dominio, o también conocido como **servicio DNS** (*Domain Name System*), para los contenedores conectados a la red, lo que significa que, por ejemplo, para hacer peticiones desde el front-end al back-end no es necesario conocer la dirección IP del contenedor, que además puede cambiar entre distintos despliegues a lo largo del tiempo, sino que basta con conocer su nombre.

Es posible configurar los contenedores de una red de este tipo para que puedan comunicarse con máquinas del exterior. En este caso, esto se ha configurado utilizando las funcionalidades de **mapeo de puertos** de Docker. El mapeo de puertos permite exponer puertos de los contenedores Docker y vincularlos con puertos del host en el que se ejecutan, pudiendo así comunicarse con el exterior.

Poniendo como ejemplo la arquitectura del servicio, el back-end está configurado en el código fuente para escuchar en el puerto 8080. Nótese como en el esquema de la Figura 7.1 se ha realizado un mapeo de puertos del 8080 del contenedor con el 8080 del host. Supóngase que la dirección IP del host es la 185.253.152.19. Con la configuración descrita, los equipos del exterior pueden comunicarse con el back-end a través de la dirección 185.253.152.19:8080. El mapeo no tiene por qué ser entre pares de puertos idénticos del contenedor y host, en el ejemplo anterior, el puerto 8080 del contenedor podría haberse vinculado al 5432 del host, y para poder comunicarse con el back-end se utilizaría la dirección 185.253.152.19:5432.

La situación descrita implica definir una serie de reglas de **NAT** (*Network Address Translation*), las cuales se detallan más en profundidad en [25]. Básicamente, Docker especifica estas reglas de NAT para redes de tipo "*bridge*" haciendo uso de iptables, utilidad de cortafuegos en sistemas Linux, de la siguiente forma:

- NAT de origen, conocido como SNAT (*Source NAT*), permite que los contenedores puedan comunicarse con los equipos externos. Se configura añadiendo a la cadena de *POSTROUTING* la operación *MASQUERADE*, con lo que la dirección IP de origen de los contenedores se traduce por la dirección IP de la tarjeta de red del host.
- NAT de destino, conocido como DNAT (*Destination NAT*), permite que los equipos del exterior puedan comunicarse con los contenedores. Se configura añadiendo a la cadena de *DOCKER* (esta cadena no está definida por defecto en iptables, permite definir reglas para paquetes que entran/salen de la

interfaz "*bridge*" de Docker) una regla para traducir la dirección IP y puerto de destino vinculado con el host por la dirección IP y puerto expuesto del contenedor.

Los contenedores que componen el servicio se destruyen al detener su ejecución junto con todos los datos que contienen. Sin embargo, es deseable mantener la información almacenada en la base de datos y los archivos multimedia de las distintas emisiones entre sucesivas ejecuciones. Para ello, se puede hacer uso de **volúmenes** y **directorios vinculados** en Docker.

Los directorios vinculados permiten que ficheros o directorios del host sean montados dentro de un contenedor. Si estos ficheros o directorios no existen en el host, se crearán automáticamente. Los directorios vinculados dependen de la estructura del sistema de ficheros del sistema operativo del host y pueden localizarse en cualquier lugar en el sistema de ficheros del host. Si dentro del contenedor existe un directorio con distintos archivos y se vincula un directorio en la misma ruta, los archivos originales del contenedor son sustituidos por los del directorio vinculado. Un efecto de este comportamiento es que los contenedores incrementan de tamaño según el de los directorios vinculados.

Mientras que los directorios vinculados dependen de la estructura de archivos del host, los volúmenes son creados en un directorio específico en el host que es únicamente gestionado por Docker, brindando un mayor aislamiento del resto de archivos del host. Un mismo volumen puede ser utilizado por múltiples contenedores simultáneamente. Además, el tamaño de los contenedores no se incrementa de forma proporcional con el del volumen que monten.

Tal y como se muestra en la Figura 7.1, se hace uso de dos volúmenes para la persistencia de datos:

- El volumen "media" se encarga de almacenar todos los ficheros multimedia de las distintas presentaciones, esto es, los manifiestos MPD y los segmentos.
- El volumen "mongoData" contiene toda la información almacenada en la base de datos.

Durante el desarrollo, los archivos multimedia procesados por el back-end se almacenaban en un directorio vinculado. Esto permitía acceder fácilmente a los distintos ficheros MPD y segmentos. No obstante, al término del desarrollo, y como se puede observar en la Figura 7.1, este directorio vinculado fue reemplazado por un volumen.

7.2. Colecciones de base de datos

En la base de datos existen dos Colecciones, "*Stream*" y "*HRTF*", cuyos documentos se modelan y se relacionan como se muestra en la Figura 7.2.

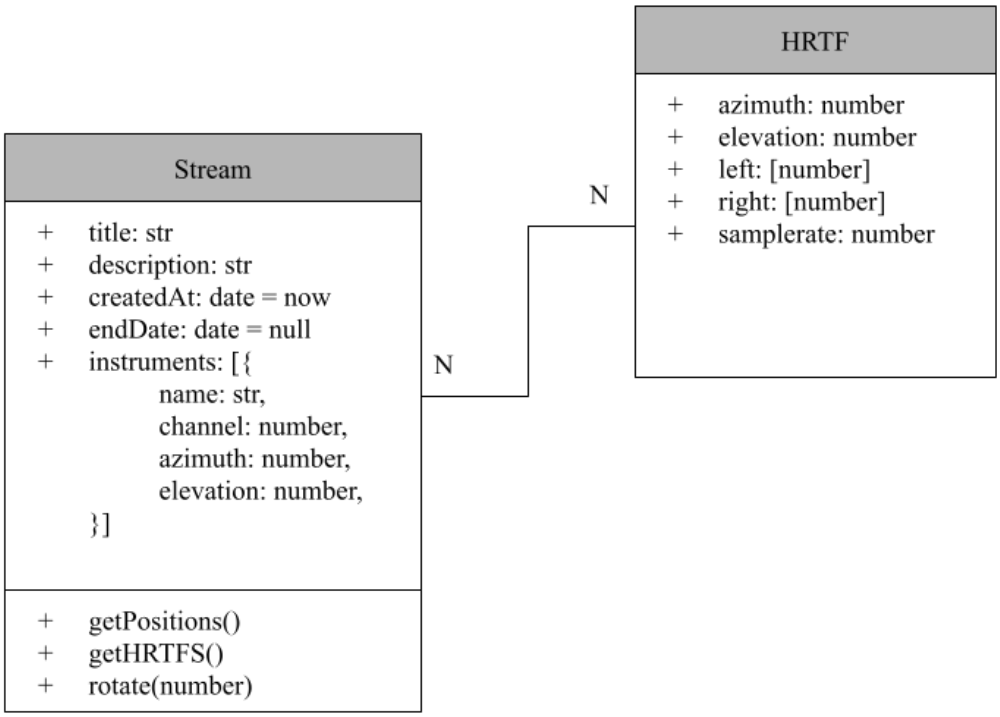

Figura 7.2. Modelos y relación entre colecciones de la base de datos.

En el esquema de la Figura 7.2 se describen los campos de cada modelo indicando los tipos de datos de cada uno de ellos siguiendo la siguiente sintaxis: *nombre: tipo de dato*.

A continuación se ofrece una explicación breve de cada uno de estos tipos de datos:

- "*str*" es la abreviación de "*string*", se trata de una cadena de caracteres alfanuméricos.
- "*date*" representa una fecha en UTC, en formato ISO-8601.
- "*number*" representa un número, tanto entero como decimal.
- Los corchetes "[]" sirven para indicar que el campo se trata de un "*array*" cuyos ítems son del tipo de datos entre corchetes.
- Las llaves "{}" representan objetos cuyos campos son los indicados dentro de las mismas.

Aunque no están representados en el esquema, ambos modelos contienen un campo "*_id*" que MongoDB añade a todos los documentos agregados a la base de datos y que los identifica de forma unívoca a modo de clave primaria.

El modelo de "*HRTF*" contiene las muestras de las respuestas impulsivas del canal izquierdo y derecho junto con la frecuencia de muestreo. Los campos de "*azimuth*" y "*elevation*" forman un **índice compuesto único**, lo que significa que no puede haber dos documentos dentro de la colección que tengan valores idénticos de azimut y elevación simultáneamente. Los índices en MongoDB mejoran el rendimiento a la hora de realizar consultas a la base de datos, los índices compuestos son el mismo concepto, pero sobre múltiples campos de un documento. Conceptualmente, los índices son el equivalente a las claves en bases de datos relacionales.

El modelo de "*Stream*" contiene información relativa al título, una descripción de la emisión, la fecha de creación, cuyo valor por defecto es la fecha en la que se crea una instancia de este modelo, y finalización, cuyo valor por defecto es nulo, y, además, una lista con los nombres de los instrumentos de la actuación, además de su canal correspondiente en la señal de audio multicanal que se va a transmitir y los valores de azimut y elevación que permiten localizar la dirección de cada uno de los instrumentos.

Adicionalmente, para este modelo se han definido tres métodos que se describen a continuación:

- "getPositions()" no recibe ningún parámetro de entrada y devuelve los valores de azimut y elevación de cada uno de los instrumentos que forman parte de la actuación transmitida. Realmente es usado como método auxiliar en los otros métodos del modelo.
- "getHRTFS()" tampoco recibe ningún parámetro de entrada. Devuelve el conjunto de HRTFs correspondientes a cada uno de los instrumentos de acuerdo a sus valores de azimut y elevación.
- "rotate(number)" recibe un valor numérico entero como parámetro. Devuelve otro conjunto de HRTFs resultado de aplicar la rotación en grados especificada con el valor del parámetro de entrada.

7.3. Códec de audio y formato contenedor multimedia

El códec de audio utilizado debe soportar audio multicanal con distribución discreta y un número máximo de canales considerable, como ya se comentó en el apartado 6.2.4, y el que más se ajusta a estos requisitos es **Opus**.

Opus puede trabajar con frecuencias de muestreo de 8, 12, 16, 24 y 48 KHz, esta última es la que preferiblemente debería utilizar el audio de las emisiones que se pretenden distribuir con el servicio propuesto. La frecuencia de muestreo de 48 KHz implica que el audio codificado tendrá un ancho de banda efectivo de 24 KHz, sin embargo, Opus no codifica el audio por encima de los 20 KHz, ya que esto es considerado el límite superior del ancho de banda de escucha del oído humano [26].

El formato contenedor será WebM, de código abierto y desarrollado por el WebM Project. WebM está pensado para poder usarse en la web, proporcionando una excelente integración con HTML5. WebM se trata de un subconjunto del formato contenedor multimedia Matroska (.mkv) [27].

Los contenedores que transportan únicamente audio deben indicarlo ajustando el tipo MIME a "*audio/webm*". El formato WebM contiene los siguientes elementos mostrados en la Figura 7.3.

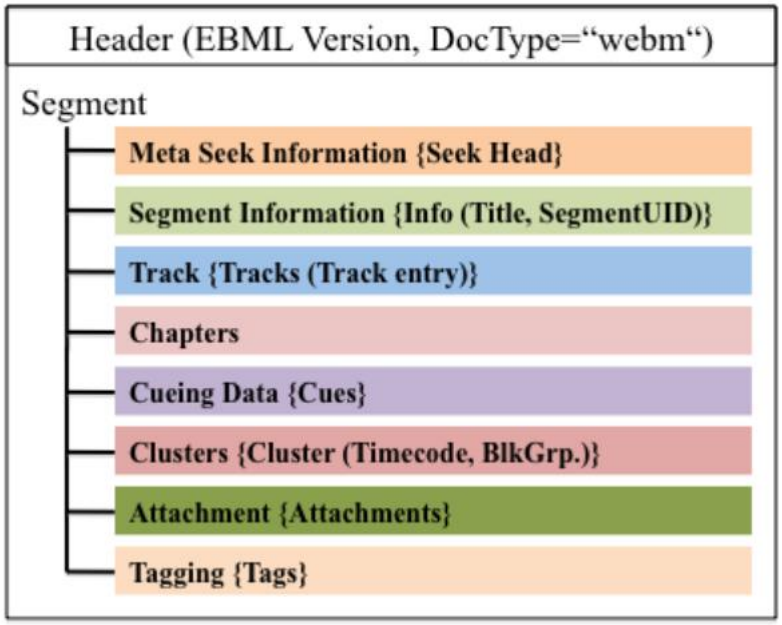

Figura 7.3. Formato de un contenedor WebM (extraída de [28]).

Los elementos mostrados en la Figura 7.3 también aparecen en el formato Matroska. A continuación se ofrece una breve descripción de los mismos según [29].

- "*Header*" contiene información esencial del archivo, como la versión del formato EBML y el tipo de documento Matroska.
- El elemento "*Seek Head*" contiene un índice con las ubicaciones de los demás elementos dentro del segmento y evita tener que recorrer todo el archivo para llegar a ellos.
- "*Info*" incluye datos que identifican el segmento, como el título y un identificador único para dicho segmento.
- Un mismo contenedor puede transportar múltiples pistas de audio y vídeo. El elemento "*Tracks*" contiene datos como el nombre, número y tipo (audio, vídeo, subtítulos, …) de las pistas que el contenedor transporta. Además, contiene toda la información necesaria para que el códec pueda decodificar los datos multimedia de cada una de las pistas.
- "*Chapters*" identifica todos los capítulos, que son instantes temporales predefinidos a los que se puede saltar en el momento de la reproducción.
- "*Cues*" es similar al elemento "*SeekHead*". Contiene un índice con sellos temporales de presentación de los bloques de datos multimedia del segmento.
- "*Clusters*" contienen los datos multimedia de cada pista, divididos en bloques.
- "*Attachments*" permite adjuntar otro tipo de archivos como fuentes o páginas web.
- "*Tags*" contienen otro tipo de metadatos, como puede ser el género de una película, el reparto de actores, fechas, comentarios, etc.

El cambio más importante de WebM respecto a Matroska es que el atributo "*DocType*" debe ser obligatoriamente ajustado al valor "webm". Existen otra serie de cambios menores que afectan a la definición de las referencias dentro del elemento "*Cues*" y cómo se agrupan los "*Clusters*" [28].

Opus y WebM son totalmente compatibles con la implementación de FFmpeg de MPEG-DASH, se sugiere volver a observar la Figura 6.2 en la que se muestra un ejemplo de generación de una presentación MPEG-DASH con este códec y formato contenedor multimedia.

7.4. Transmisión de contenido multimedia

La transmisión del contenido multimedia comprende dos fases:

1. **Ingesta de datos**, en la que el equipo que está generando el contenido multimedia envía estos datos al back-end, junto con otros metadatos.
2. **Difusión de contenido**, en la que estos datos multimedia generados son enviados a los clientes que consumen el contenido.

7.4.1. Ingesta de datos

FFmpeg implementa un cliente HTTP que puede lanzar peticiones en cuyo cuerpo pueden transportar los datos multimedia generados. Estos datos multimedia son, básicamente, los segmentos y el manifiesto MPD de la presentación MPEG-DASH, el cual se va actualizando periódicamente en caso de que las transmisiones sean en vivo.

Junto con los datos multimedia, es necesario incluir metadatos que permitan identificar los siguientes aspectos de una retransmisión:

- Qué "*Adaptation Set*" corresponde a cada posición.
- En la posición del director (micrófono principal), qué canal está asociado a cada grupo de instrumentos.
- En la posición del director, qué dirección tienen cada uno de los grupos de instrumentos para realizar la espacialización.
- Nombre de la emisión, descripción de la misma, fecha de creación y, cuando proceda, fecha de finalización.

Para realizar la ingesta de datos se definen **dos** puntos finales (***endpoints***) en el back-end:

- **/stream/**, para crear una nueva retransmisión mediante el método HTTP **POST**. En el cuerpo de la petición se envía el título de la retransmisión, una descripción y una lista con los instrumentos que forman parte de la actuación, junto con sus localizaciones y el canal al que sus señales están asociadas en la señal de audio multicanal del micrófono principal (ver Figura 7.2)
- **/ingest/*:stream_id*/*:filename***, para enviar el contenido multimedia, incluyendo el manifiesto MPD, también mediante el método HTTP **POST**.

Este "*endpoint*" es el que realmente se encarga de procesar la ingesta de datos multimedia. Nótese que se envían como parámetros en el URL el ID de la presentación multimedia y el nombre del fichero que se envía en el cuerpo de la petición.

Si hay éxito al procesar los datos en el back-end y almacenarlos en la base de datos, el "*endpoint*" para crear una nueva retransmisión devuelve el valor del ID que se debe usar en el "*endpoint*" para enviar los datos multimedia. Este intercambio de información se representa en la Figura 7.4.

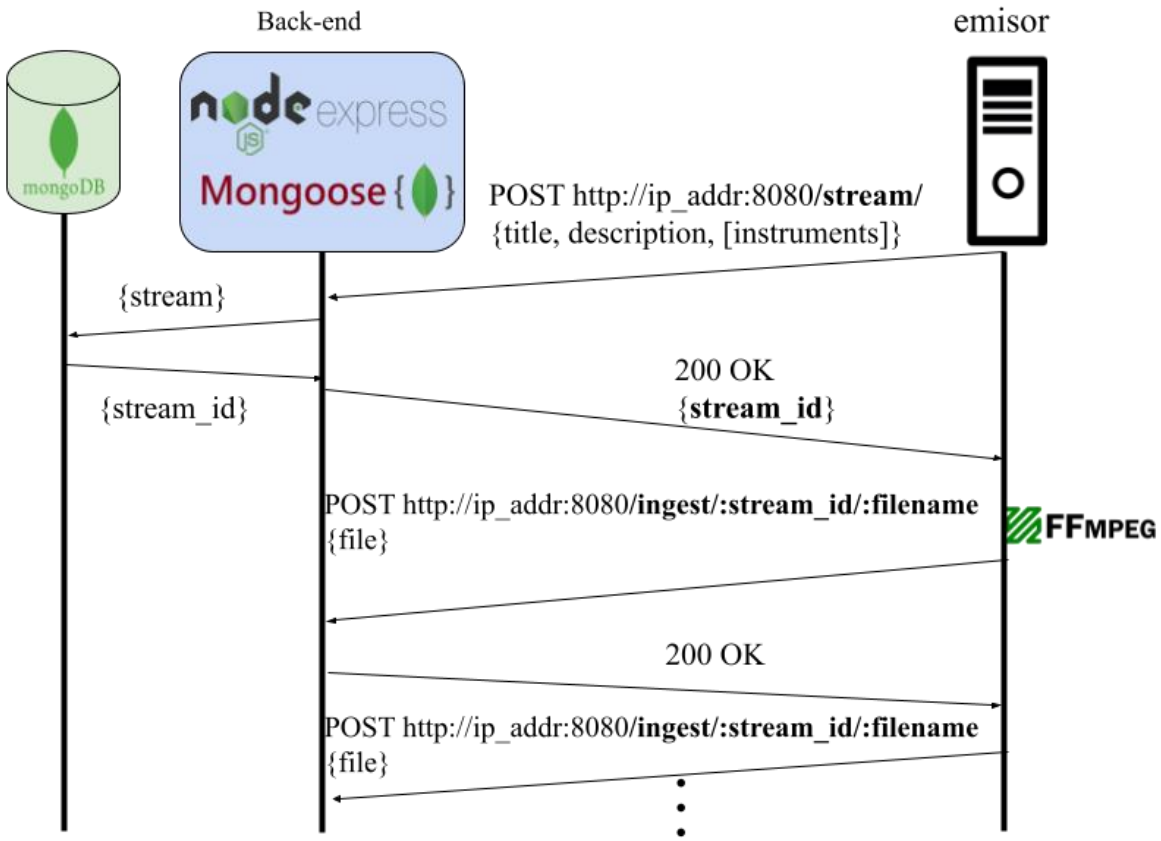

Figura 7.4. Intercambio de información durante la ingesta de datos.

Los ficheros MDP y segmentos multimedia serán almacenados en el volumen "media" (ver Figura 7.1) montado en el contenedor del back-end. Estos ficheros se ordenarán en directorios cuyo nombre es igual al del ID de la retransmisión a la que pertenecen.

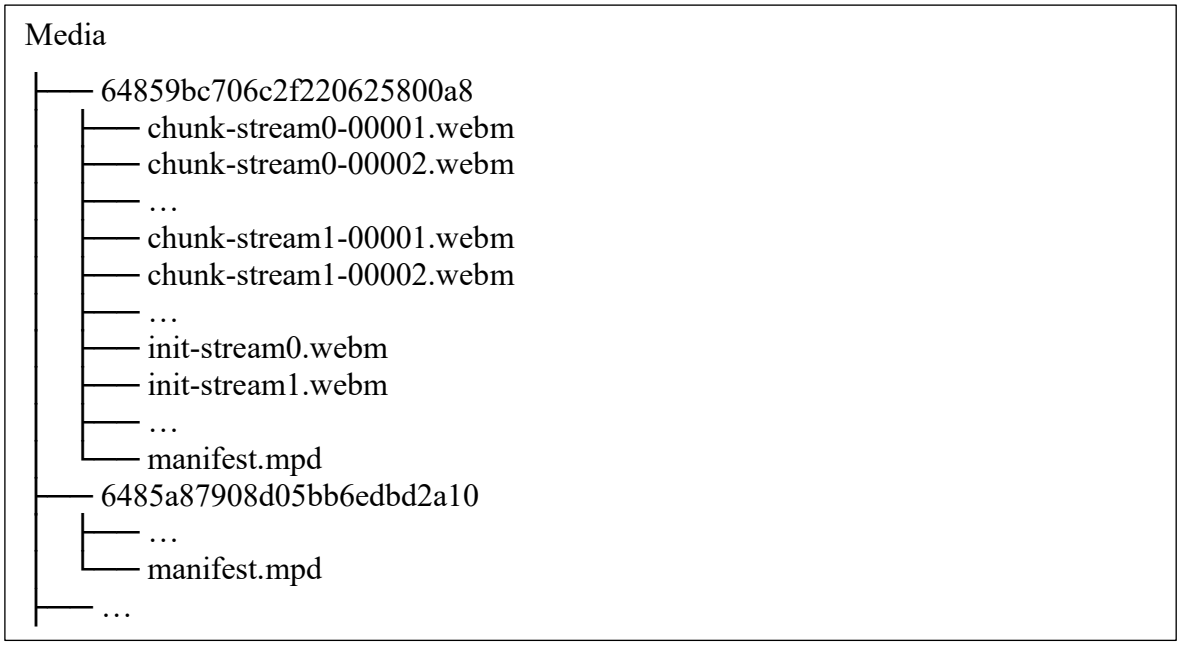

Figura 7.5. Estructura de directorios en volumen "media".

En la Figura 7.5 se comprueba que cada presentación multimedia tiene un archivo de manifiesto MPD y múltiples segmentos multimedia WebM. Estos segmentos se nombran en función del ID del "stream" de audio al que pertenecen y del índice del segmento, el cual es un contador que se incrementa para los segmentos de un mismo stream. Se observa además los distintos tipos de segmentos de inicialización y segmentos multimedia que ya se comentaron en el apartado 5.2.2.

En los intercambios de información anteriormente descritos se envían todos los metadatos necesarios a excepción de la identificación de cada "*Adaptation Set*" con cada posición en el escenario.

Estos metadatos pueden ser incrustados directamente en el manifiesto MPD en la etiqueta de idioma, el reproductor dash.js es capaz de recuperar sin problema estos metadatos para cada uno de los "*Adaptation Sets*".

En la Figura 7.6 se muestra un ejemplo de cómo se puede usar FFmpeg para generar una retransmisión DASH de audio, utilizando el códec Opus y contenedores multimedia WebM, en la que se utiliza la etiqueta de idioma para identificar cada uno de los "*Adaptation Sets*".

```
ffmpeg -re \
-i VC.wav \
-i FL.wav \
-i VN_I.wav \
-i  MAIN.wav \
-map 0:a -c:a libopus -mapping_family 255 -vn -metadata:s:a:0 language=cello \
-map 1:a -c:a libopus -mapping_family 255 -vn -metadata:s:a:1 language=flute \
-map 2:a -c:a libopus -mapping_family 255 -vn -metadata:s:a:2 language=violin \
-map 3:a -c:a libopus -mapping_family 255 -vn -metadata:s:a:3 language=main \
-f dash -dash_segment_type webm -seg_duration 4 -update_period 8 -target_latency 30 \
-adaptation_sets "id=0,streams=0 id=1,streams=1 id=2,streams=2 id=3,streams=3 " \
http:// 185.253.152.19:8080/ingest/64859bc706c2f220625800a8/manifest.mpd
```

Figura 7.6. Generación de retransmisión de audio MPEG-DASH con códec OPUS y contenedor WebM y envío de ficheros al servidor.

Nótese como en la Figura 7.6 se han añadido metadatos para el idioma de cada una de las señales de audio, indicando la posición en el escenario. De este modo, en el archivo MPD, el "*Adaptation Set*" con ID 0 describe el contenido de audio perteneciente al micrófono cercano de grupo de violoncellos, el "*Adaption Set*" con ID 1, el de flautas, el de ID 2, el de violines y, finalmente, el "*Adaptation Set*" con ID 3 describe el contenido en el micrófono principal.

De este modo, el cliente dispone de todos los metadatos necesarios a la hora de consumir el contenido multimedia para su correcta reproducción. Es también importante recordar que, cuando esta transmisión en directo finalice, FFmpeg modificará el manifiesto para que la transmisión sea identificada como "*on-demand*".

El **controlador** asociado al "*endpoint*" '***/ingest/:stream_id/:filename***', mediante el que se envían los datos multimedia al back-end mantiene la lógica de procesamiento descrita en el diagrama de flujo de la Figura 7.7.

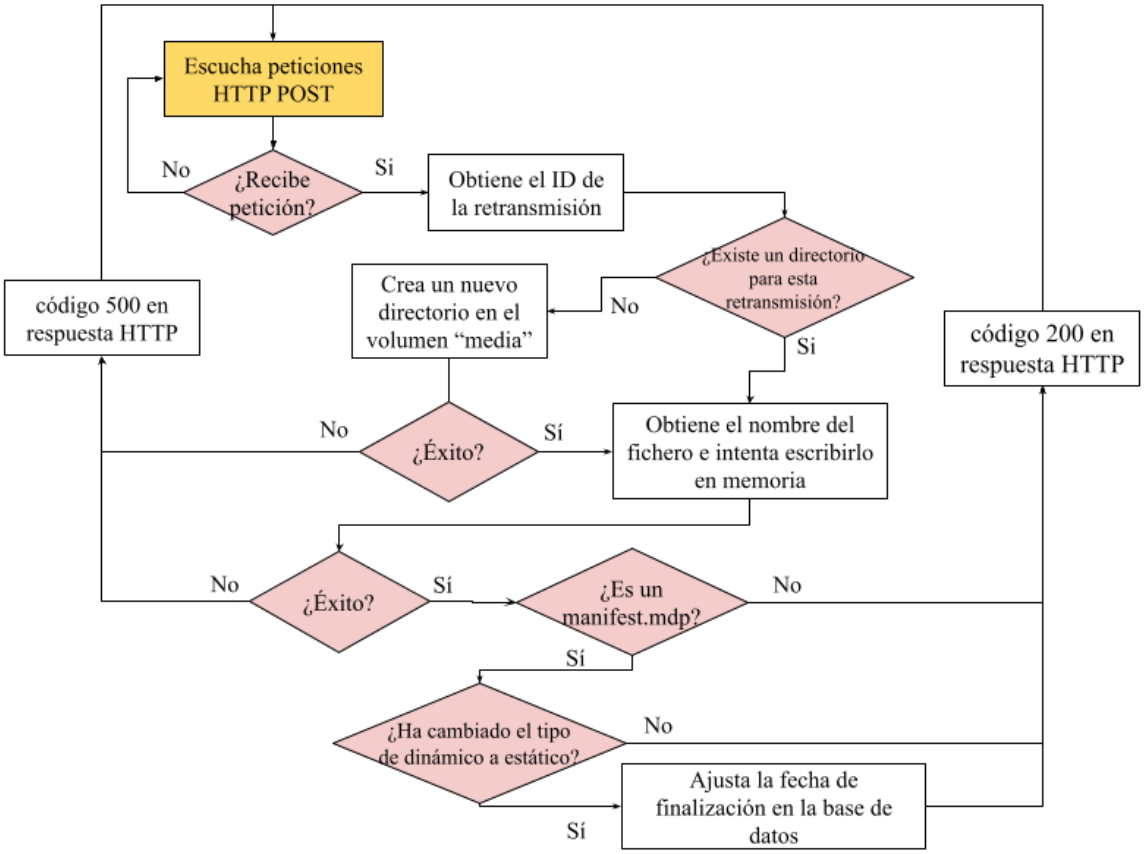

Figura 7.7. Diagrama de flujo del controlador de ingesta de datos multimedia en el back-end.

7.4.2. Difusión del contenido multimedia

La difusión del contenido multimedia es, en esencia, similar a servir cualquier tipo de recurso en un servidor HTTP. El reproductor dash.js enviará peticiones GET al back-end para recuperar inicialmente el fichero MPD. Una vez hecho esto, el cliente realiza peticiones HTTP sucesivas en las que recupera los segmentos multimedia. En caso de transmisiones en vivo, el cliente también realiza peticiones periódicas del fichero MPD para actualizarlo de acuerdo con los parámetros de la presentación multimedia.

Para servir los ficheros multimedia de una presentación, en el back-end se ha habilitado el "*endpoint*" ***/media/:stream_id/:filename***, que procesa peticiones HTTP **GET**. Es importante recalcar que en el URL se envían los parámetros del ID de la presentación multimedia y el nombre del fichero.

La Figura 7.8 representa el intercambio de información descrito entre el front-end y el back-end.

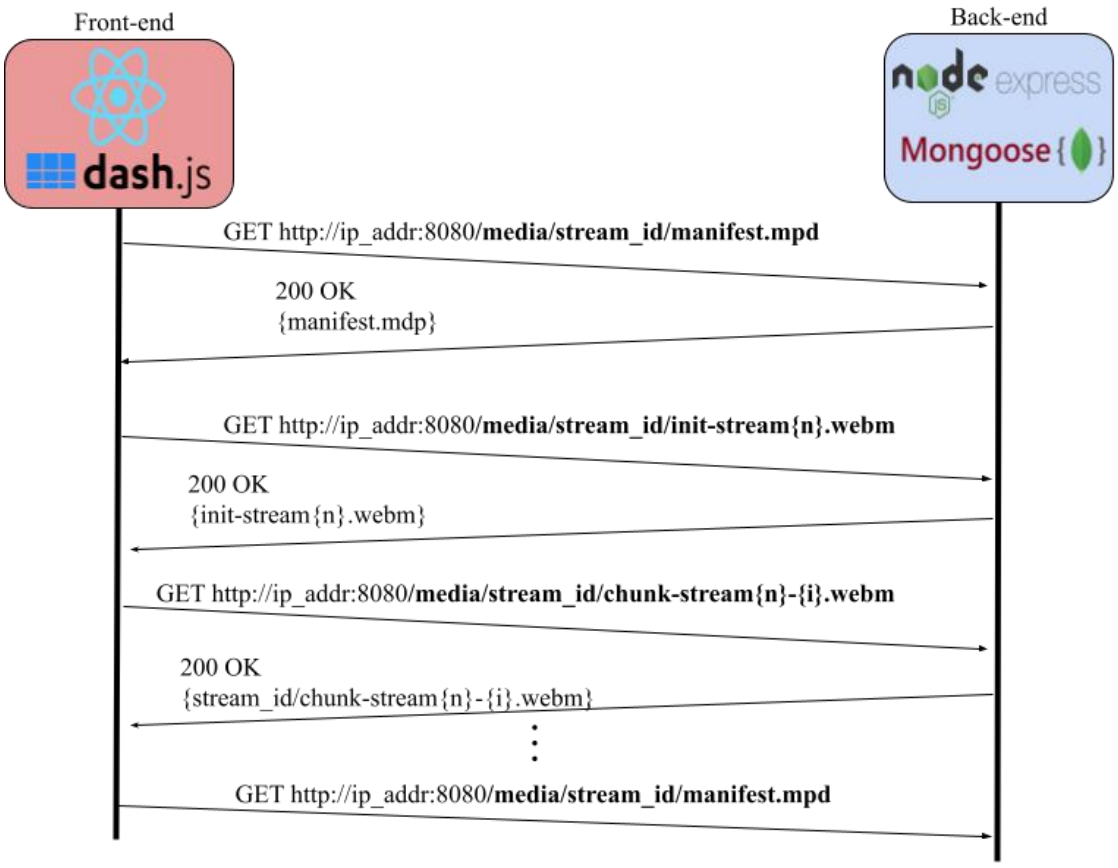

Figura 7.8. Intercambio de información durante la difusión del contenido multimedia.

En la Figura 7.8, los distintos streams que componen la presentación DASH son identificados con el índice 'n', mientras que los segmentos de inicialización y multimedia son indexados con el contador ascendente 'i'.

7.5. Envío de datos adicionales de una presentación multimedia

A parte de difundir contenido multimedia, el back-end debe enviar la información relacionada con cada retransmisión, comentada en el apartado 7.4.1 y un conjunto de HRTFs.

Los datos de cada retransmisión permiten mostrar una lista con las presentaciones disponibles para su reproducción. Además, para la espacialización de la mezcla de audio en el cliente será necesario recuperar las HRTFs correspondientes a las direcciones de los distintos instrumentos especificadas a la hora de crear una nueva emisión.

Adicionalmente, ante la posibilidad de que el usuario pueda realizar rotaciones, es necesario disponer de un mecanismo para recuperar un nuevo conjunto de HRTFs que ajuste la escena sonora a la rotación realizada.

Todos estos datos pueden enviarse en formato JSON mediante APIs REST implementadas en el back-end y que se describirán a continuación.

7.5.1. API REST para obtener la información de todas las presentaciones

Para obtener la información de todas las presentaciones, el back-end tiene habilitado el "*endpoint*" ***/stream/***, que mediante el método HTTP **GET** devuelve una lista con todas las instancias del modelo "*stream*" (ver Figura 7.2) almacenadas en la base de datos.

La Figura 7.9 ejemplifica el intercambio de información que se produce para obtener los datos comentados, donde el modelo de Mongoose "*Stream*" se denota con la primera letra en mayúscula, mientras que las instancias de ese modelo se denotan en minúscula.

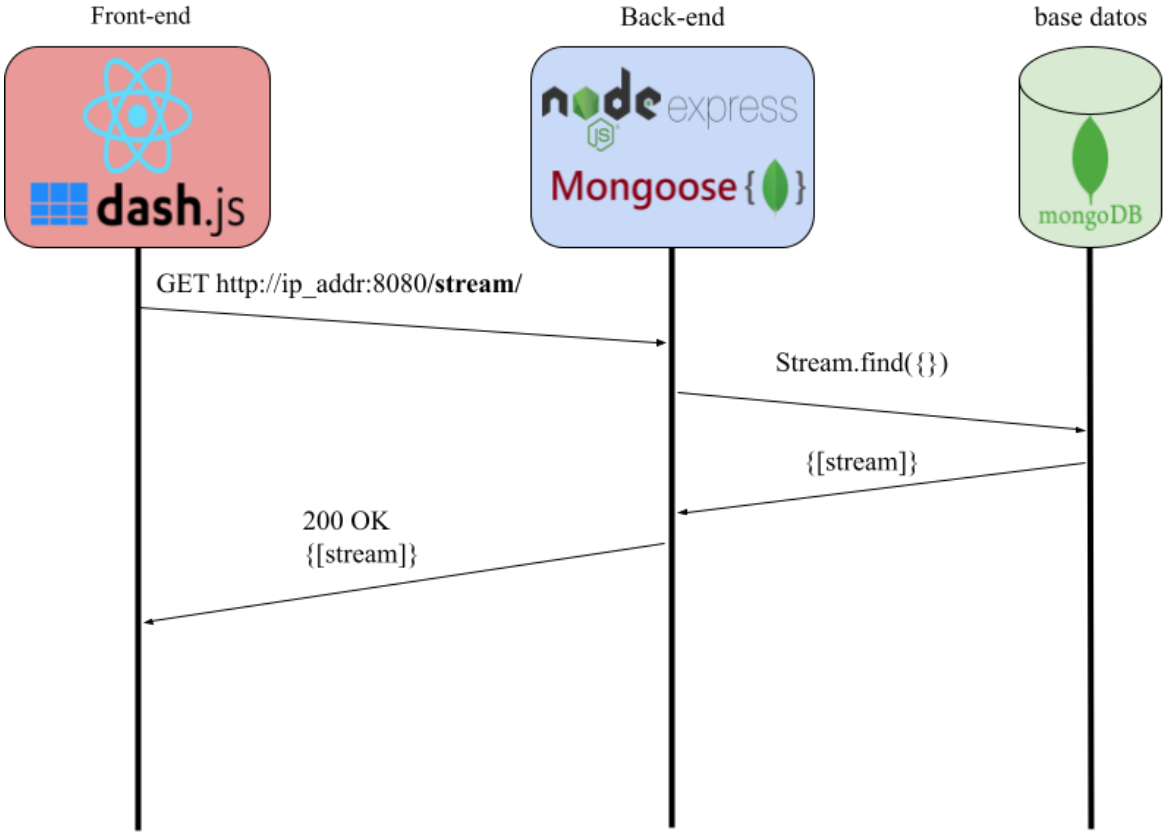

Figura 7.9. Intercambio de información para recuperar los datos de todas las presentaciones.

El resultado es una lista como la que se ofrece de ejemplo en la Figura 7.10. En este caso, la lista solo contiene un ítem.

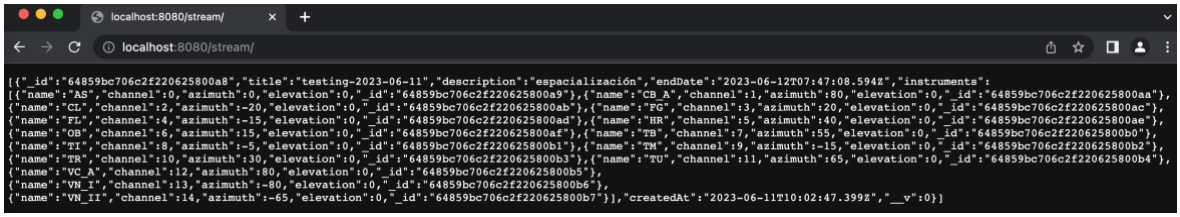

Figura 7.10. Ejemplo de datos recuperados de todas las presentaciones.

7.5.2. API REST para almacenar HRTFs en la base de datos

Para poblar la colección de HRTFs de la base de datos de Mongo, se ha habilitado el "*endpoint*" ***/hrtf/***, que mediante el método HTTP **POST**, permite enviar en el cuerpo de las peticiones los datos correspondientes a una instancia del modelo HRTF descrito en el apartado 7.2.

Este modelo permite almacenar documentos que representan cada una de las HRTFs disponibles en la base de datos que se envíen al back-end después de realizar las operaciones comentadas en los apartados 6.2.7 y 6.2.8.

El intercambio de información se representa en la Figura 7.11.

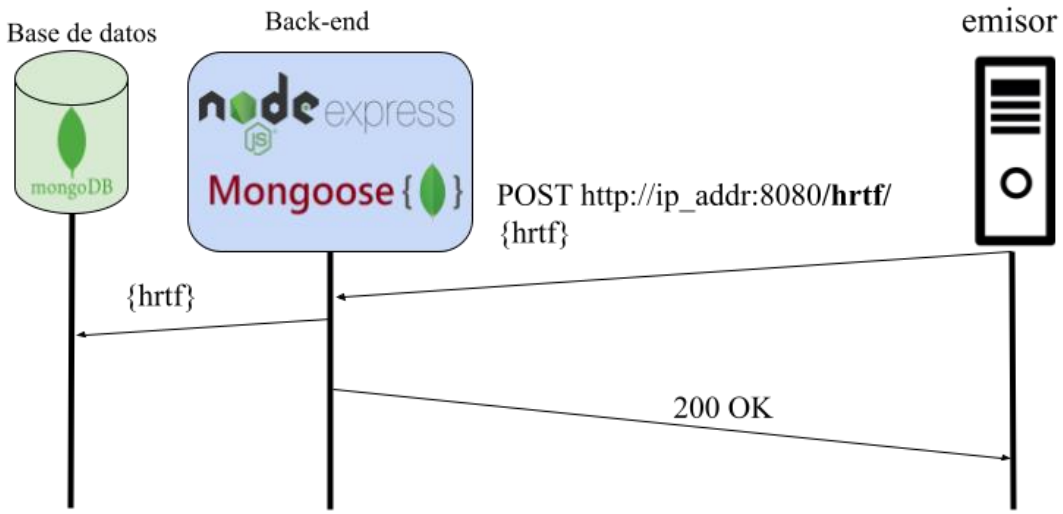

Figura 7.11. Intercambio de información para almacenar una HRTF al poblar la colección de la base de datos.

7.5.3. API REST para obtener un conjunto de HRTFs correspondientes a las localizaciones de los instrumentos de una actuación

De forma similar a lo descrito en el apartado 7.5.1, para obtener el conjunto de HRTFs cuyas posiciones corresponden a la localización especificada de cada uno de los instrumentos de la actuación se dispone del "*endpoint*" ***/stream/:id/hrtfs***, el cual, mediante el método HTTP **GET** devuelve una lista con las HRTFs necesarias para la espacialización de la mezcla siguiendo el intercambio de información mostrado en la Figura 7.12. Es importante resaltar que el ID de la retransmisión es enviado como parámetro en el URL.

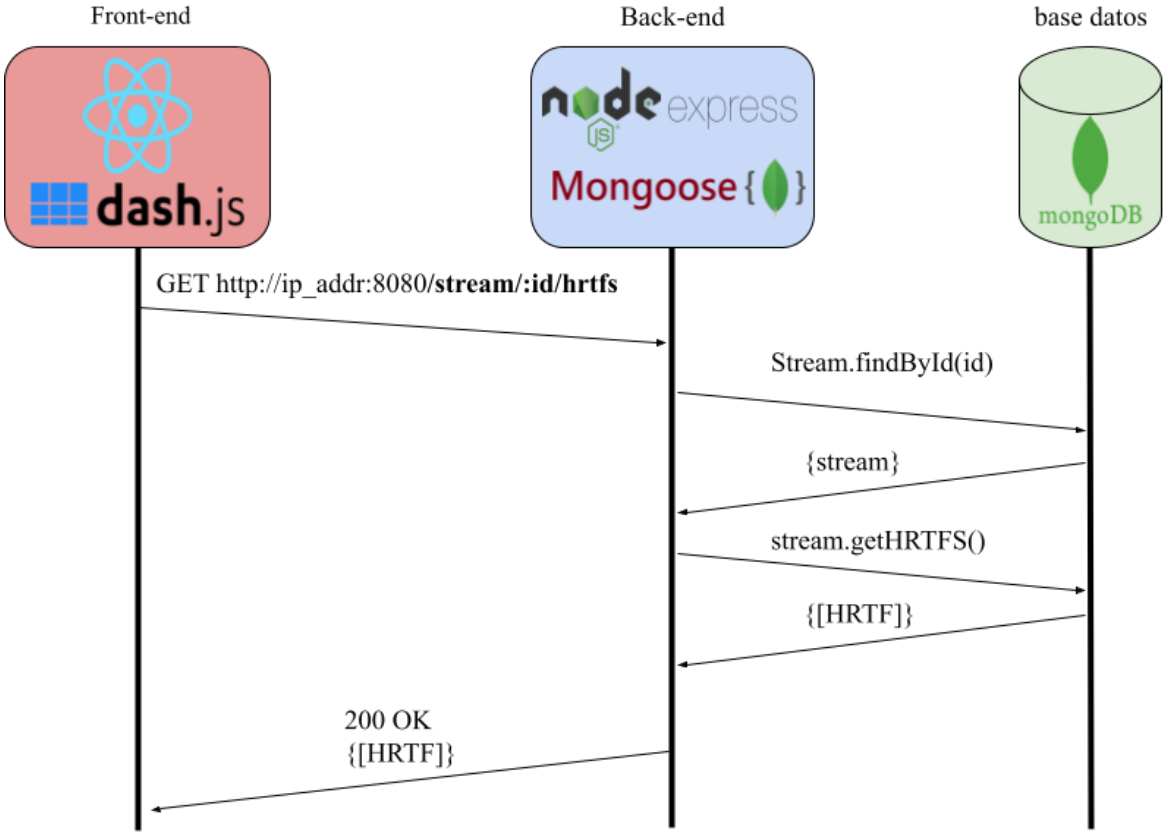

Figura 7.12. Intercambio de información para recuperar las HRTFs de las localizaciones de los instrumentos de una presentación.

En la Figura 7.13 se ofrece un ejemplo de las HRTFs recuperadas para la presentación mostrada en la Figura 7.10.

Figura 7.13. Ejemplo de HRTFs recuperadas para la actuación de la Figura 7.10.

7.5.4. API REST para obtener un conjunto de HRTFs correspondientes a las localizaciones de los instrumentos de una actuación tras aplicar una rotación

Empleando el método HTTP **GET** en el "*endpoint*" ***/stream/:id/hrtfs/:rotation***, que recibe como parámetros el ID de la presentación y el valor de la rotación (tanto negativo

como positivo) que se desea aplicar, se puede recuperar un conjunto de HRTFs para espacializar los instrumentos de la actuación tras aplicar la rotación especificada como parámetro.

La rotación es en referencia al movimiento del usuario y se expresa en grados. Esto quiere decir que un valor de 20º significa que el usuario ha girado hacia la derecha y, por lo tanto, los instrumentos deberían percibirse con más presencia en el oído izquierdo. Por el contrario, un valor de -20º implica haber girado hacia la izquierda y los instrumentos deberían percibirse con más presencia en el oído derecho.

Igual que en los apartados anteriores, el intercambio de información que se produce para recuperar las HRTFs tras aplicar una rotación viene representado en la Figura 7.14.

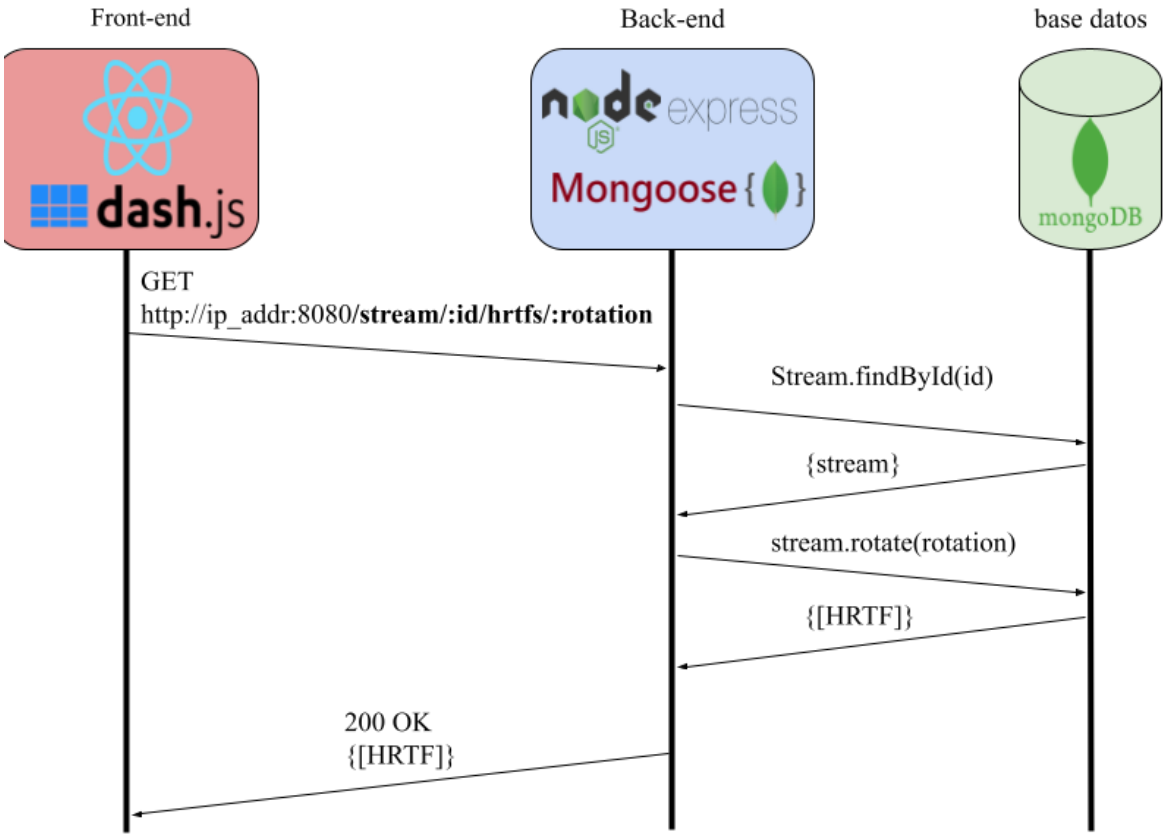

Figura 7.14. Intercambio de información para recuperar las HRTFs de las localizaciones de los instrumentos de una presentación tras aplicar una rotación.

Siguiendo con el ejemplo de la presentación anterior, la Figura 7.15 muestra los datos recuperados con esta API después de que el usuario haya realizado una rotación de 40º hacia la derecha.

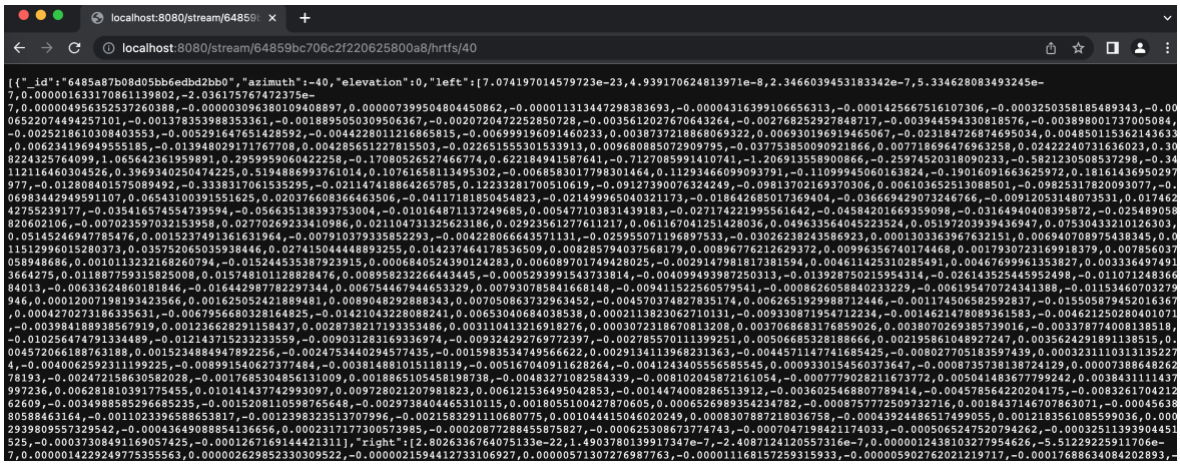

Figura 7.15. Ejemplo de HRTFs recuperadas para una actuación tras aplicar una rotación.

Nótese cómo la primera HRTF recuperada tiene un valor de azimut de -40º, mientras que en la Figura 7.13 tenía un valor de 0º. Esto quiere decir que la fuente de sonido que antes se localizaba en el centro de la escena sonora ahora se localiza en el lado izquierdo.

7.6. Consumo de contenido multimedia y procesado de audio en el front-end

Los apartados anteriores han estado centrados en aspectos del back-end. Se ha detallado cómo se envían datos multimedia y otro tipo de información asociada a las presentaciones que se pretenden retransmitir y se han descrito los mecanismos mediante los cuales estos datos se pueden recuperar.

En este apartado se detallan los procesos que ocurren en el front-end para consumir el contenido multimedia y realizar las distintas operaciones de procesado de audio en tiempo real.

7.6.1. Cadena de audio

La cadena de audio es el nombre que recibe el grafo que conforman los distintos nodos y conexiones entre ellos de Web Audio API, diseñado para el procesado de audio en el cliente en tiempo real. Junto al reproductor DASH, la cadena de audio es el elemento más importante del front-end. Su estructura se muestra en la Figura 7.16.

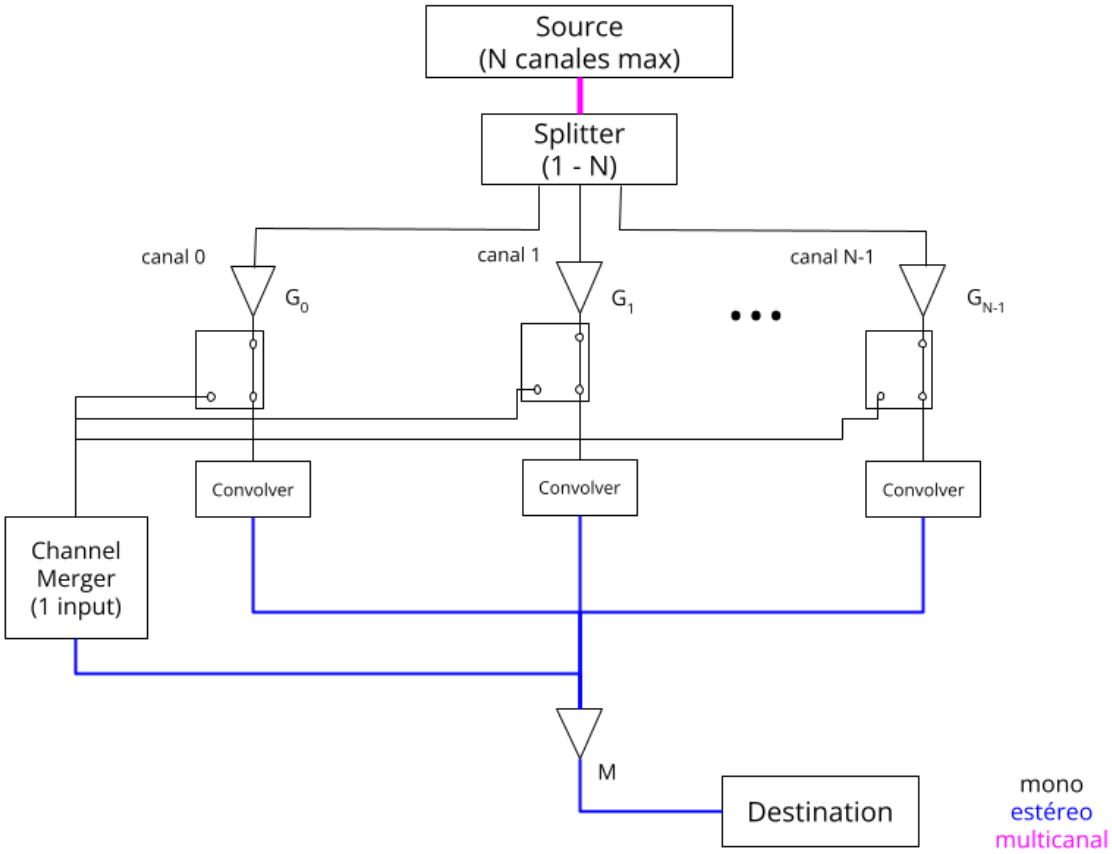

Figura 7.16. Representación esquemática de la cadena de procesamiento de audio en el cliente.

A continuación, se presentan los nombres de los nodos que conforman la cadena de audio representada en la Figura 7.16 junto con una descripción de las operaciones que realizan.

- **Source**: Concretamente se trata de un "*MediaElementSource*", ya que está vinculado a la misma etiqueta <audio> de HTML5 que a su vez está vinculada con el reproductor DASH. Constituye el punto de inicio del recorrido que las señales de audio de la presentación DASH realizarán atravesando el resto de los nodos.
- **Channel Splitter**: Es el primer nodo de procesado de audio que las señales procedentes del nodo "*source*" encuentran en su camino hacia el destino. Este nodo toma la señal de audio de entrada y divide sus canales en señales de audio independientes en cada una de sus salidas. El número de salidas es igual al número máximo de canales de las señales de audio de la presentación, es decir, la del micrófono principal. Si el número de canales de la señal de entrada es menor que el número de salidas configuradas, el funcionamiento

es exactamente el mismo, solo que algunas salidas no tendrán señal de audio disponible.

- **Gain**: Cada salida del "*splitter*" está conectada a un nodo de ganancia, representados con un triángulo. Estos nodos permiten ajustar el nivel de la señal de entrada. De esta forma, es posible ajustar el nivel de cada uno de los grupos de instrumentos en la mezcla de audio final.
- **Convolver**: Las señales de salida de los nodos de ganancia, cuando la señal del nodo fuente es la del micrófono principal, se dirigen hacia la entrada de los convolucionadores. Estos nodos permiten realizar la operación de filtrado de las señales de entrada con respuestas impulsivas, que en este caso son HRTFs. Es importante recalcar que, en el caso de que la señal de entrada sea monofónica, como es el caso, y la respuesta impulsiva tenga varios canales, como es el caso de las HRTFs, que tienen 2 canales, la salida será el resultado de aplicar la operación de convolución de la señal de entrada por la señal de cada uno de los canales de la respuesta impulsiva. Es decir, como las HRTFs tienen 2 canales, la señal de salida será estereofónica.
- **Channel Merger**: Cuando la señal del nodo fuente no es la del micrófono principal, las señales de salida de los nodos de ganancia son dirigidas hacia la entrada de este nodo. Este nodo se encarga de combinar múltiples señales a la entrada en una única señal multicanal a la salida. En este caso particular, el funcionamiento del nodo se representa en la Figura 7.17.

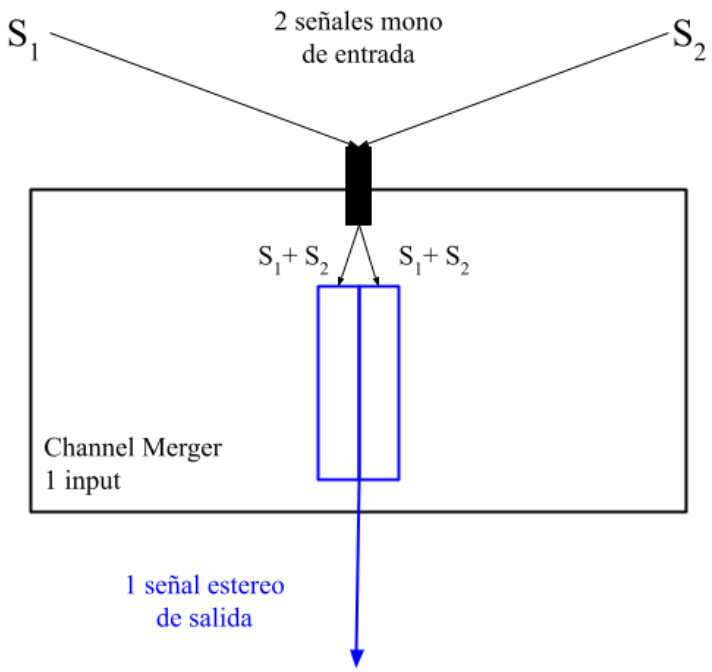

Figura 7.17. Funcionamiento del combinador de canales con una única entrada.

De esta forma se consigue una señal estereofónica a la salida del combinador de canales a partir de las señales de entrada.

- **Master Gain**: Realmente es un nodo de ganancia como los descritos anteriormente. Está situado justo antes del nodo de destino y sirve para ajustar el volumen de la mezcla completa.
- **Destination**: Representa el hardware de salida de audio del equipo en el que se está reproduciendo el contenido, por ejemplo, los altavoces del ordenador.

Además de estos nodos definidos en Web Audio API, en la representación de la cadena de audio de la Figura 7.16 aparecen unos interruptores situados entre los nodos de ganancia de cada canal y los convolucionadores. Estos interruptores no existen realmente, pero ayudan a representar que los flujos de las señales de audio de los canales pueden dirigirse, o bien hacia los convolucionadores, o bien hacia el combinador de canales.

La **cadena de audio** se representa como un **objeto** de implementación personal, para el cual se han definido los siguientes **métodos asociados**:

- constructor(htmlAudioElement, maxNumChannels, defaultGain, hrtfs): Recibe como parámetros el elemento <audio> que será la fuente de la cadena de audio, el número máximo de canales, el valor de la ganancia que se aplicará por defecto a cada uno de los grupos de instrumentos (que no el master) y el conjunto de HRTFs para inicializar los convolucionadores. Este método instancia una nueva cadena de audio al crear un contexto de audio con una frecuencia de muestreo de 48000 Hz y generar los nodos y conexiones representados en la Figura 7.16.
- setFadersGain(gains): Recibe como parámetro una lista con un número de valores igual al número de nodos de ganancia, excluyendo el nodo maestro. Permite ajustar el valor de amplificación de cada uno de estos nodos.
- getFadersGain(): No recibe ningún parámetro. Devuelve una lista con los valores de amplificación de los nodos de ganancia de la cadena de audio, excluyendo el nodo maestro.
- setMasterGain(gain): Recibe como parámetro un valor de amplificación y ajustar el nodo de ganancia maestro a este nivel.
- getMasterGain(): No recibe ningún parámetro. Devuelve el valor de amplificación del nodo de ganancia maestro.
- loadHRTFS(hrtfs): Recibe como parámetro la lista con un número de HRTFs igual al número de nodos convolucionadores. Carga en cada uno de estos nodos las respuestas impulsivas de la lista.
- bypassConvolvers(bypass): Recibe como parámetro un valor booleano, es decir, o verdadero o falso. Desconecta la salida de los nodos de ganancia (salvo el maestro) de la entrada de los nodos convolucionadores y conecta con la entrada del combinador de canal.

7.6.2. Componentes de la interfaz del front-end

La cadena de audio descrita en el apartado 7.6.1 permite realizar el procesado de audio en el cliente en tiempo real. Sin embargo, es necesario desarrollar una interfaz a través de la cual el usuario pueda interactuar para poder realizar todas las operaciones que la cadena de audio brinda.

Esta interfaz no solo debe incluir elementos que permitan con una interacción intuitiva con la cadena de audio, sino que debe ser lo suficientemente dinámica como para mostrar la información del estado de la cadena de audio ante cualquier acción del usuario.

Esta interfaz está fundamentada en React y está formada por los componentes que se muestran en la Figura 7.18.

Figura 7.18. Estructura de la interfaz del front-end basada en React.

El fichero **App.js** es el componente principal de la interfaz. Hace las funciones de punto de entrada y es donde se renderiza la interfaz de forma dinámica.

En el directorio **components** se almacenan los distintos componentes reutilizables de la interfaz.

El componente "*AudioChain*" no es realmente un componente con sintaxis de React, sino de JavaScript normal. Este componente modeliza la cadena de audio descrita en el apartado 7.6.1.

"*Fader*" modeliza un componente que contiene un control deslizante y una etiqueta de identificación. Este componente permite ajustar un valor de amplificación deseado para un canal de audio, identificado con la etiqueta asociada, en función de la posición del control deslizante.

A lo largo de los siguientes apartados del documento, se empleará el término "ganancia" para referirse a lo que verdaderamente es "amplificación" por simplificación y abuso del lenguaje. Técnicamente "ganancia" se utiliza para referirse al valor en decibelios de la unidad lineal que describe el término "amplificación". Por lo tanto, entiéndase "amplificación" cuando aparezca el término "ganancia".

"*Mixer*" modeliza un mezclador de audio. Este componente contiene una serie de "*Faders*" en su interior. El número de "*faders*" depende de los canales de audio de la señal de audio que se está procesando.

"*RotationSelector*" define un selector mediante el cual el usuario puede realizar una rotación en grados hacia la izquierda o hacia la derecha.

"*TrackSelector*" permite seleccionar entre todos los "*Adaptation Sets*" de la presentación multimedia. Recibe este nombre porque en el contexto del reproductor de dash.js, los "*Adaptation Sets*" que describen audio son llamados "*Tracks*".

El directorio **pages** contiene componentes que representan páginas específicas de la aplicación.

"*StreamList*" muestra una lista con las presentaciones multimedia disponibles para reproducir. Al seleccionar una de estas presentaciones se renderiza el componente "*stream*".

"*Stream*" contiene un reproductor dash.js y es donde los demás componentes reutilizables del directorio **components** son usados para construir la interfaz que permita interactuar con la cadena de audio en tiempo real mientras se reproduce audio.

7.6.3. Funcionamiento de la interfaz de usuario

En el apartado 7.6.1 se detalló la cadena de audio mediante la cual el audio se procesa en el cliente mientras se reproduce una presentación multimedia. En el apartado 7.6.2 se describieron los distintos componentes que conforman la interfaz a través de la que el usuario puede interactuar con la cadena de audio.

En este apartado se discute la lógica de funcionamiento del front-end cuando se selecciona una presentación multimedia, es decir, cuando se muestra la página "*stream*". El funcionamiento aparece representado en forma de diagrama de flujo en la Figura 7.19.

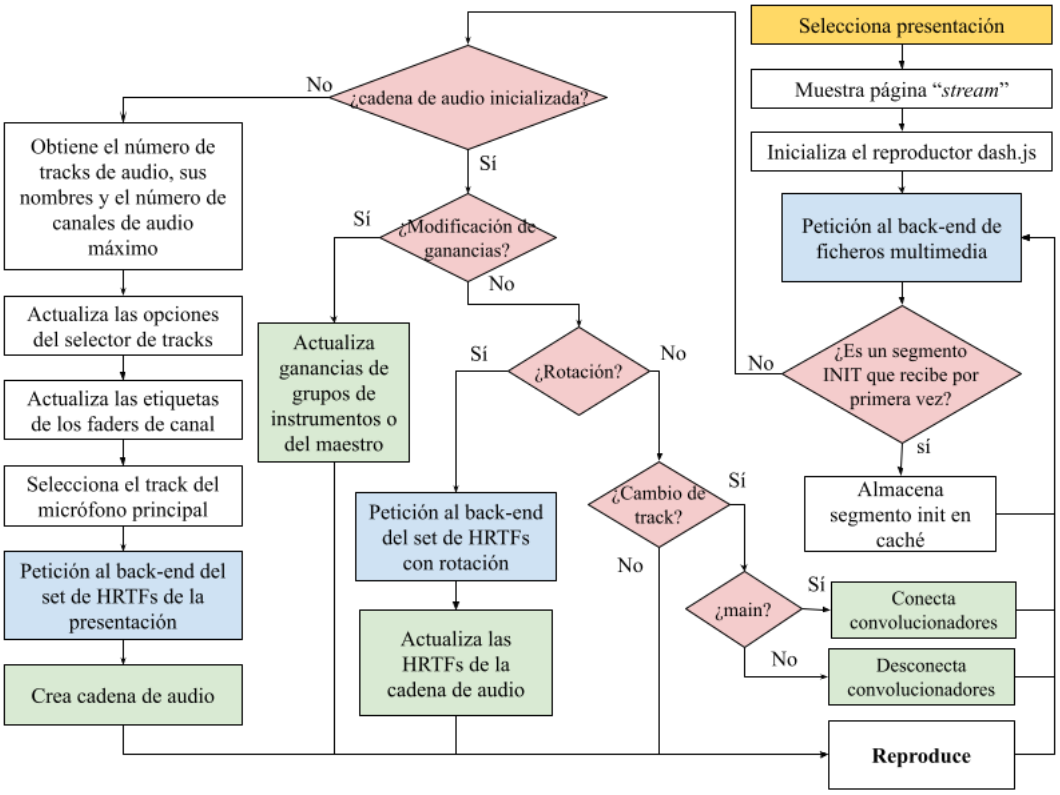

Figura 7.19. Diagrama de flujo del funcionamiento del front-end al seleccionar una presentación multimedia.

En el diagrama de flujo de la Figura 7.19, las acciones que implican utilizar métodos de la cadena de audio, descritos en el apartado 7.6.1, se representan en color verde. De forma similar, las acciones que implican realizar peticiones al back-end son representadas en color azul.

La relación con los métodos de la cadena de audio y las acciones mostradas en el diagrama es la siguiente:

- Para crear la cadena de audio será necesario invocar al '*constructor*'.
- Actualizar las ganancias de los grupos de instrumentos o del maestro se hace mediante los métodos '*setFadersGain(gains)*' o '*setMasterGain(gain)*', dependiendo de qué valor es el que ha sido modificado.
- Para actualizar las HRTFs se usa el método '*loadHRTFS(hrtfs)*'.
- Conectar los convolucionadores implica invocar al método '*bypassConvolvers(false)*', igual que para para desconectarlos se invocará el método , '*bypassConvolvers(true)*'.

Las peticiones de recursos al back-end se hacen a través de los siguientes "*endpoints*".

- Los archivos multimedia se obtienen mediante una petición GET a */media/:stream_id/:filename*. Ver apartado 7.4.2.
- Los conjuntos de HRTFs se obtienen realizando peticiones GET a los dos "*endpoints*" siguientes:
  - */stream/:id/hrtfs si la rotación es de 0 grados. Ver apartado 7.5.2.*
  - */stream/:id/hrtfs/:rotation* si la rotación es de un valor distinto de 0 grados. Ver apartado 7.5.4.

Cuando se selecciona la presentación multimedia, el cliente dispone de la información asociada a esta, pues para mostrar la lista de presentaciones disponibles, previamente se ha tenido que hacer una petición al back-end para recuperar la información de todas ellas.

Esta información incluye el número de grupos de instrumentos, sus nombres, el canal de la señal del micrófono principal que corresponde a cada uno de ellos y el ID con el que construir los URLs para realizar las peticiones HTTP que procedan al back-end.

Los elementos visuales que forman la página "*stream*" se listan a continuación:

- Un elemento <audio> HTML5.
- Un componente "M*ixer*" con un número de "*Faders*".
- Un "*Fader*" maestro.
- Un "*TrackSelector*".
- Un "*RotationSelector*".

La Figura 7.20 muestra el aspecto de la interfaz construida con los componentes que se acaban de describir.

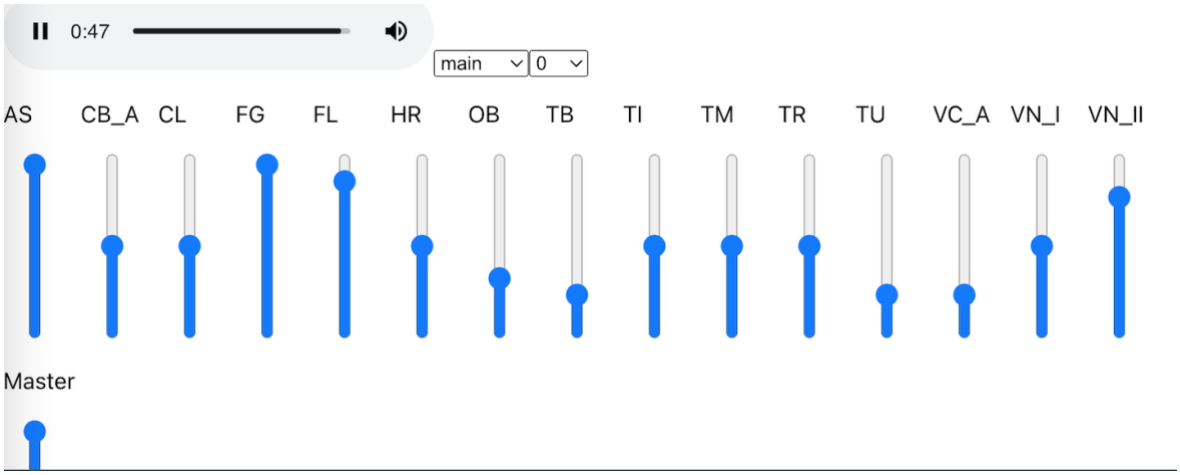

Figura 7.20. Aspecto de la interfaz de usuario durante la reproducción de audio procedente del micrófono principal.

Los valores de las ganancias para cada uno de los grupos de instrumentos, el "*Fader*" maestro, la selección de grupo de instrumentos que escuchar y el valor de la rotación se definirán como **variables con estado** de React. Esto implica que cada vez que se cambien los valores de estas variables, se disparará un renderizado de la parte de la interfaz que refleja la modificación. Por ejemplo, al cambiar el volumen de un grupo de instrumentos, se volverá a renderizar el "*Fader*" correspondiente, además de que se ajustará el valor de las ganancias de la cadena de audio.

Adicionalmente, la interfaz solo muestra los "*Faders*" necesarios en todo momento. Esto significa que, si se selecciona el "*track*" de un micrófono cercano, solo se mostrarán 2 "*Faders*", el primero para ajustar el volumen del grupo de instrumentos correspondiente y el segundo para ajustar el volumen del resto de instrumentos, como ocurre en la Figura 7.21.

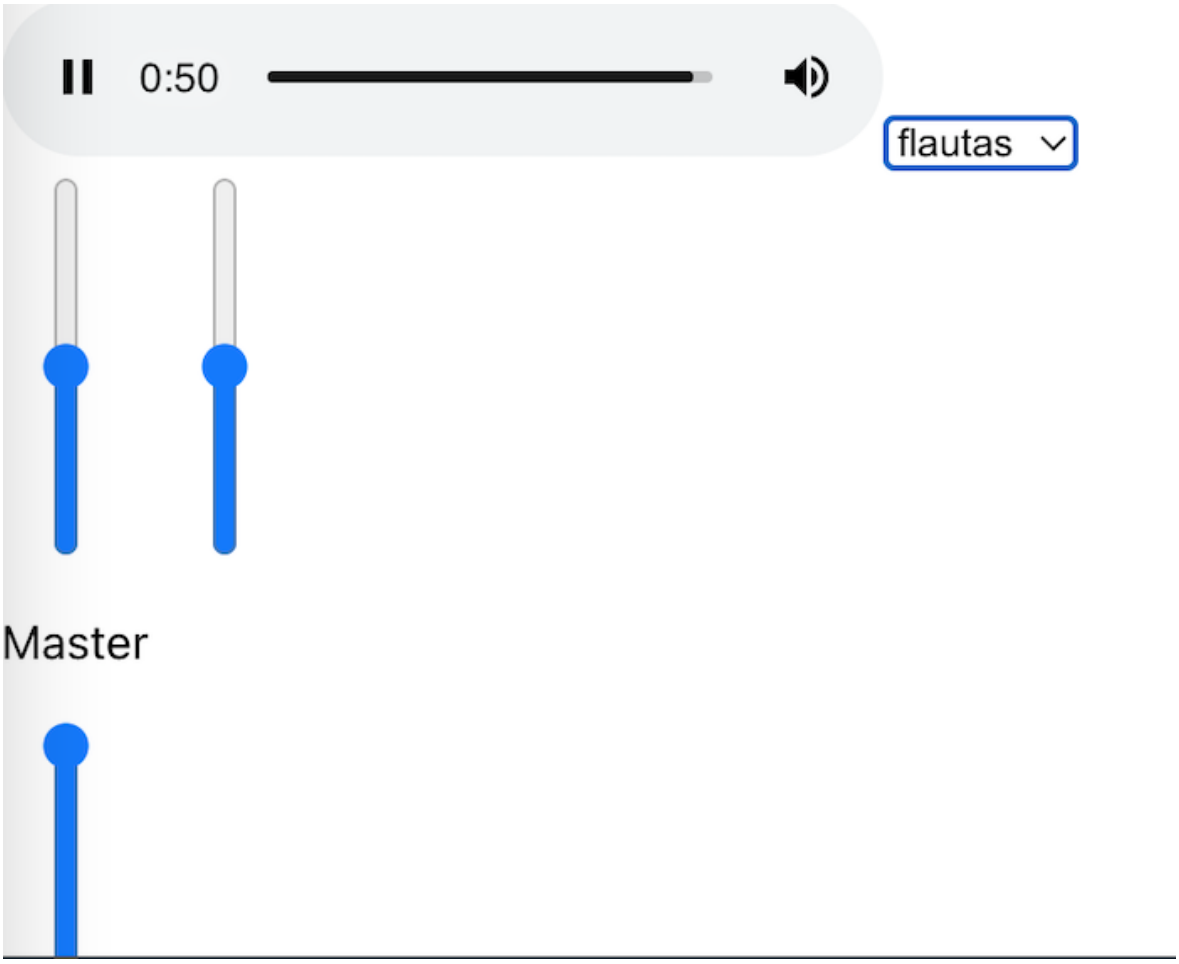

Figura 7.21. Aspecto de la interfaz de usuario durante la reproducción de audio procedente del micrófono cercano de flautas.

En cambio, si la señal de audio que se está reproduciendo es la del micrófono principal, se mostrará un número de "*Faders*" igual al número de grupos de instrumentos que transporta la señal de audio junto con unas etiquetas para identificar qué "*Fader*" está asociado a cada grupo de instrumentos, como ocurría en la Figura 7.20. Básicamente, **el**

número de "*Faders*" que se renderizan es igual al número de canales de la señal de audio que se está reproduciendo.

Existen otras variables, como la **cadena de audio** o el **reproductor DASH**, que no necesitan estado, pero sí que deben persistir entre sucesivas renderizaciones. Estas variables se almacenan como **referencias** de React.

Un aspecto importante que aparece reflejado en el diagrama de flujo de la Figura 7.19 es el almacenamiento de los segmentos "*init*" en la memoria caché del navegador. Los segmentos "*init*" contienen información del códec necesaria por el decodificador para poder reproducir el audio correctamente.

El comportamiento por defecto en dash.js es no almacenar estos segmentos en caché, de tal modo que el cliente siempre lanza una petición al servidor para recuperarlos cuando la reproducción comienza o se cambia de "*track*", aunque este ya haya sido anteriormente seleccionado.

Almacenar estos segmentos en la memoria caché del navegador hace que el cliente no necesite lanzar esta petición si el "*track*" al que cambia ya ha sido seleccionado anteriormente, lo que permite eliminar peticiones redundantes al servidor.

8. RESULTADOS Y DISCUSIÓN

En este capítulo se utilizará un ejemplo de retransmisión en directo de una actuación que servirá para apoyar la explicación del funcionamiento del servicio para consumir el contenido generado de una presentación.

Adicionalmente, se ofrece un pequeño estudio del consumo de ancho de banda a la hora de realizar la ingesta de contenido multimedia de una presentación y en la reproducción de este mismo contenido.

8.1. Funcionamiento de la aplicación

La actuación de ejemplo está compuesta por un total de 15 grupos de instrumentos. Solo se han procesado las señales procedentes de 3 micrófonos cercanos y el micrófono principal, es decir, existen 4 posiciones entre las que el usuario puede elegir.

A continuación se listan los nombres de los grupos de instrumentos de la actuación y las abreviaturas que reciben las etiquetas de los "*faders*" de cada uno de ellos.

- Violas, AS
- Contrabajos, CB_A
- Clarinetes, CL
- Fagots, FG
- Flautas, FL
- Trompas, HR
- Oboes, OB
- Trombones, TB
- Timbales, TI
- Tambores, TM
- Trompeta, TR
- Tubas, TU
- Violoncellos, VC_A
- Primer grupo de violines, VN_I
- Segundo grupo de violines, VN_II

Se utilizó el nombre de estas etiquetas por ser precisamente las abreviaturas de instrumentos de música clásica típicas. Evidentemente, estos nombres pueden ser especificados a la hora de realizar la ingesta de datos.

Las 4 posiciones disponibles para esta demostración se corresponden con los siguientes micrófonos:

- Micrófono cercano al grupo de violoncellos.
- Micrófono cercano al grupo de flautas.
- Micrófono cercano al primer grupo de violines.
- Micrófono principal

Al seleccionar una presentación multimedia de la lista ocurrirá el siguiente intercambio de información representado en la Figura 8.1.

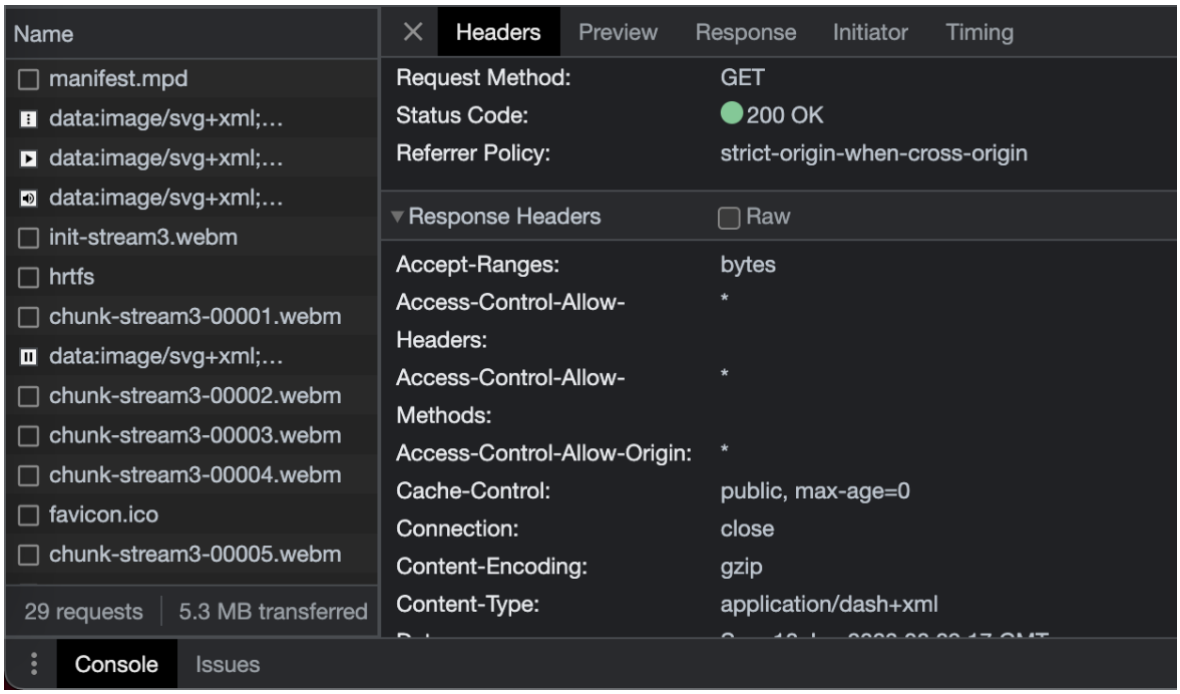

Figura 8.1. Intercambio de información inicial al seleccionar una presentación y cabeceras respuestas HTTP que transportan el archivo de manifiesto MPD.

En la Figura 8.1 se observa cómo el cliente solicita en primer lugar el fichero de manifiesto MPD. Tras esto, solicita el segmento de inicialización correspondiente a la posición del micrófono principal, el conjunto de HRTFs para inicializar la cadena de audio y a partir de entonces comienza a recuperar segmentos multimedia para su reproducción. También se comprueba que el valor de la cabecera "*Content-Type*" de la respuesta que contiene el archivo MPD tiene el valor de "*application/dash+xml*".

Las cabeceras de las respuestas HTTP que transportan segmentos multimedia pueden observarse en la Figura 8.2, donde la cabecera "*Content-Type*" tiene el valor de "*audio/webm*", ya que el único contenido multimedia que estos segmentos contienen es audio.

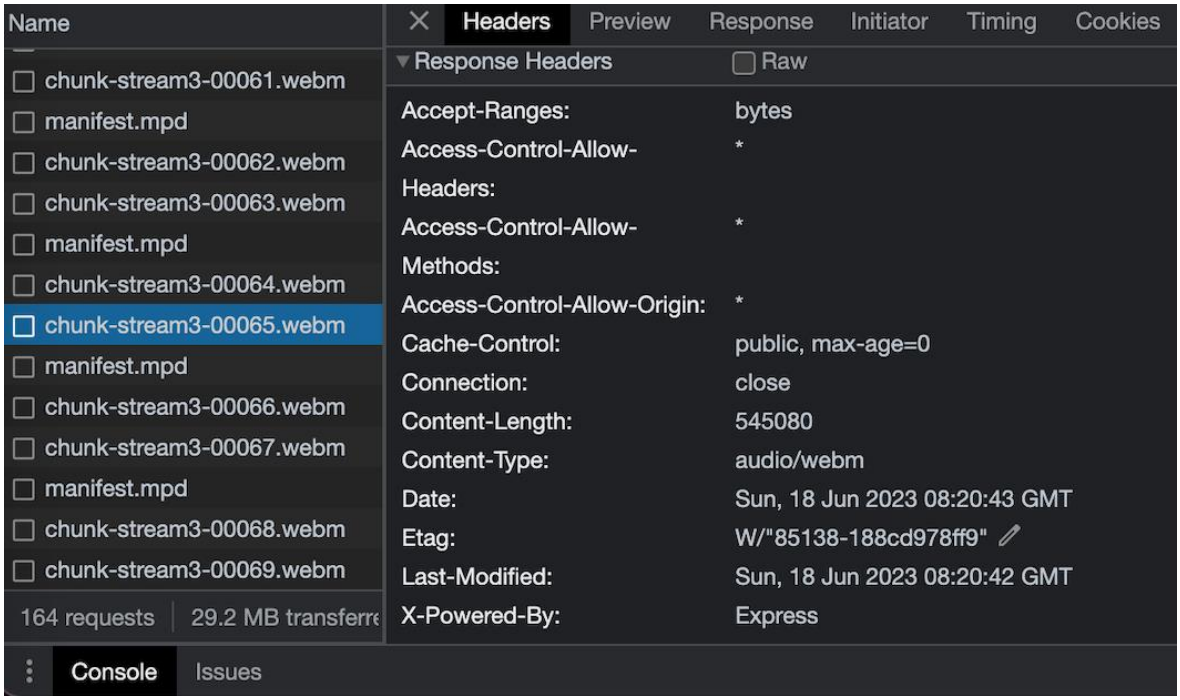

Figura 8.2. Cabeceras de respuestas HTTP que transportan segmentos multimedia.

Al mismo tiempo que ocurre el intercambio inicial de información de la Figura 8.1, se mostrará una ventana en el navegador como la que se puede observar en la Figura 8.3.

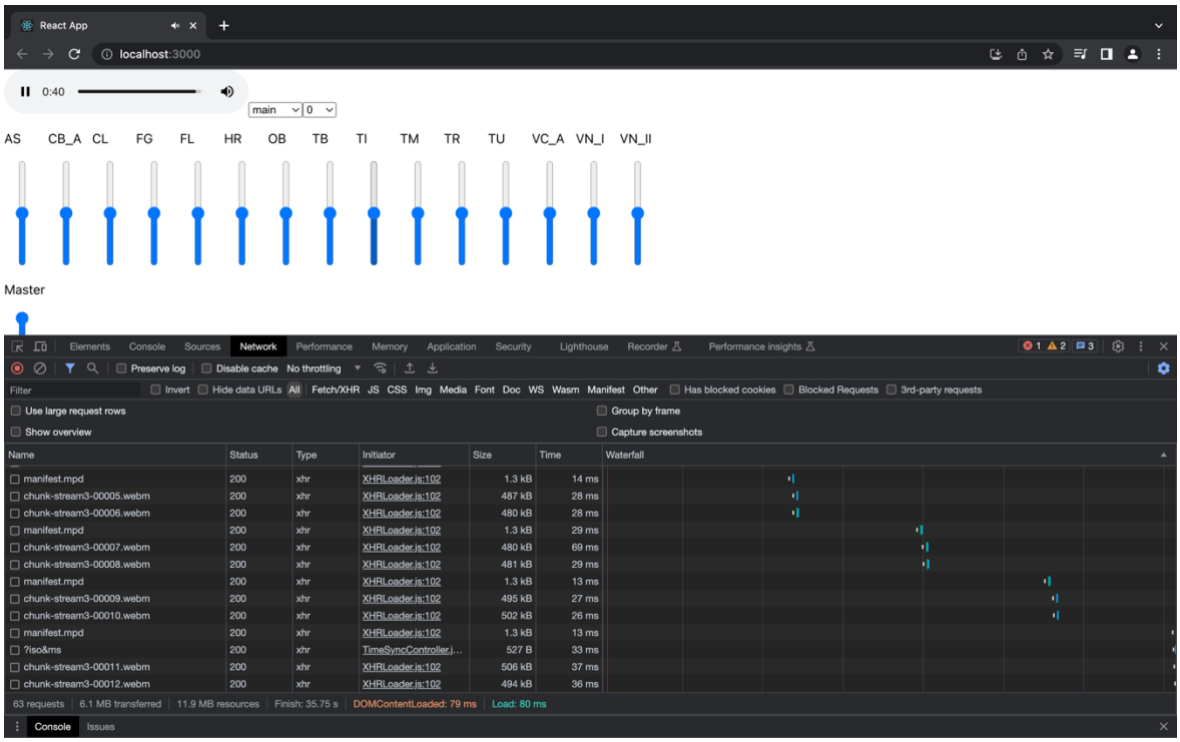

Figura 8.3. Interfaz correspondiente a la posición del micrófono principal.

En la Figura 8.3 se comprueba que la interfaz está formada por un reproductor HTML5 de audio junto con un selector de "*tracks*" con la posición del micrófono principal seleccionada, un selector de rotación, con valor de 0º, un mezclador, en el que se muestran todos los "*faders*" para modificar el nivel de señal de cada uno de los grupos de instrumentos y un "*fader*" maestro para controlar el volumen de la mezcla final.

Nótese cómo el **reproductor** de audio **no indica la duración del contenido** de audio que está reproduciendo. Esto es debido porque se trata de una **presentación en directo** y, por lo tanto, no se dispone de la información relativa a la duración total de la misma.

Es también de interés fijarse en la ventana de la herramienta de inspección de actividad de red en la parte inferior de la Figura 8.3. En ella se muestra cómo el fichero de manifiesto MPD es solicitado periódicamente, en este caso, cada dos segmentos multimedia.

La mezcla de audio en la posición del micrófono principal está espacializada con las HRTFs que se han recuperado en el intercambio de información inicial de la Figura 8.1. Haciendo uso de los "*faders*" de cada uno de los grupos de instrumentos se puede ajustar la mezcla, como se muestra en la Figura 8.4.

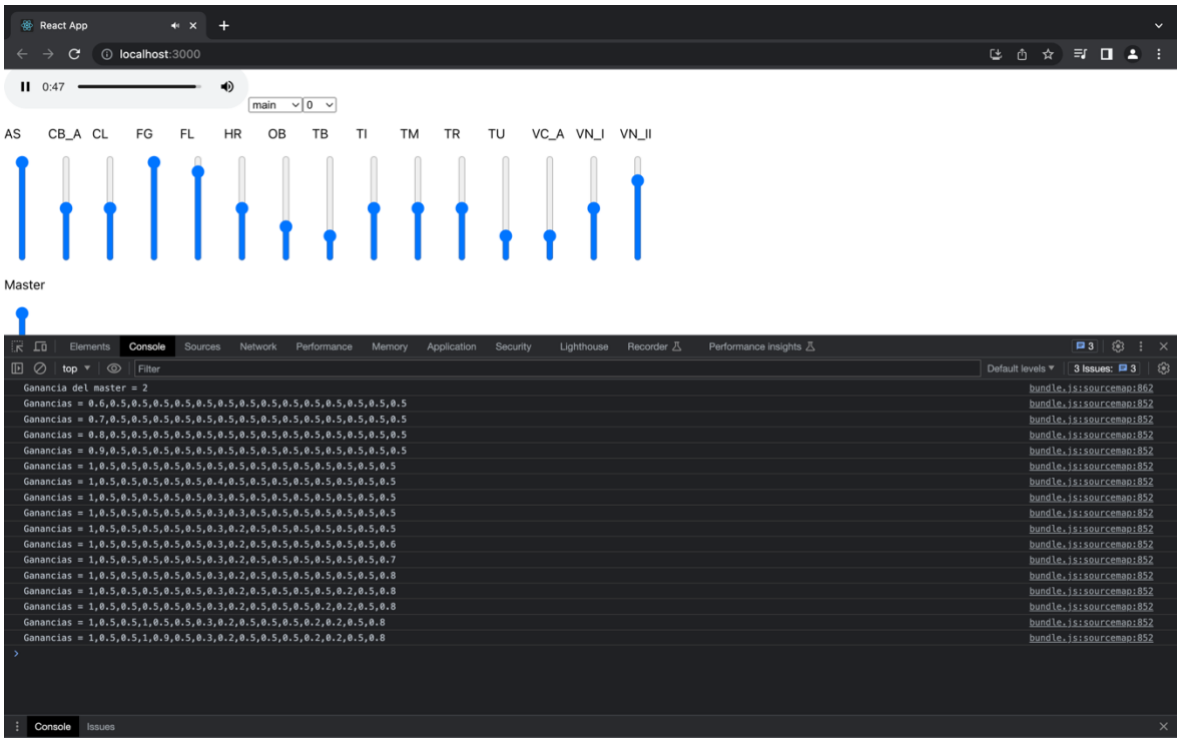

Figura 8.4. Ajuste de mezcla en la posición del micrófono principal.

Los valores de las ganancias, salvo el maestro, son almacenados en forma de lista, como se muestra en los mensajes de depuración de la consola en la herramienta de inspección del navegador.

Haciendo uso del selector de "*tracks*" se puede cambiar de posición o grupo de instrumentos, como se representa en la Figura 8.5.

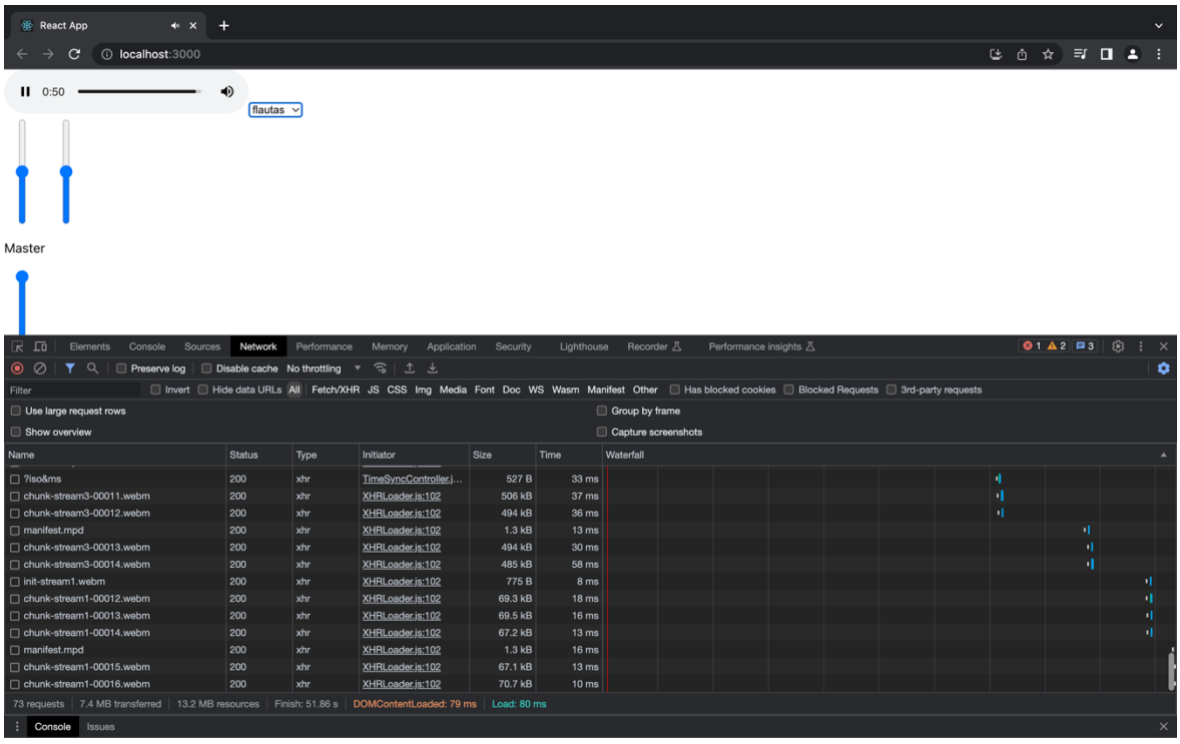

Figura 8.5. Cambio de posición del micrófono principal a micrófono cercano del grupo de flautas.

Es importante observar la **actividad de red** mostrada en la Figura 8.5 al producirse el cambio de posición. Se observa cómo el cliente pasa de solicitar segmentos multimedia para el "*stream3*", que es el "*Adaptation Set*" número 3, a realizar una petición para recuperar, en primer lugar, un **segmento de inicialización** para el "*stream1*" y, a continuación, recuperar segmentos multimedia correspondientes a ese "*stream*".

Al seleccionar una posición correspondiente a micrófonos cercanos, la interfaz muestra únicamente 2 "*faders*", a excepción del maestro, para poder modificar los niveles de señal del grupo de instrumentos asociado y del resto de la orquesta, tal y como se observa en la Figura 8.6.

Es importante recordar que, en este momento, la señal que se está distribuyendo es una señal estereofónica, en la que en el canal izquierdo transporta la señal del grupo de instrumentos del micrófono cercano, en este caso, el grupo de flautas, y el canal derecho transporta la señal del resto de la orquesta.

Además, en las posiciones de los micrófonos cercanos no se realiza espacialización, por lo que al cambiar de posición se ha debido invocar al método "*bypassConvolvers(true)*" de la cadena de audio.

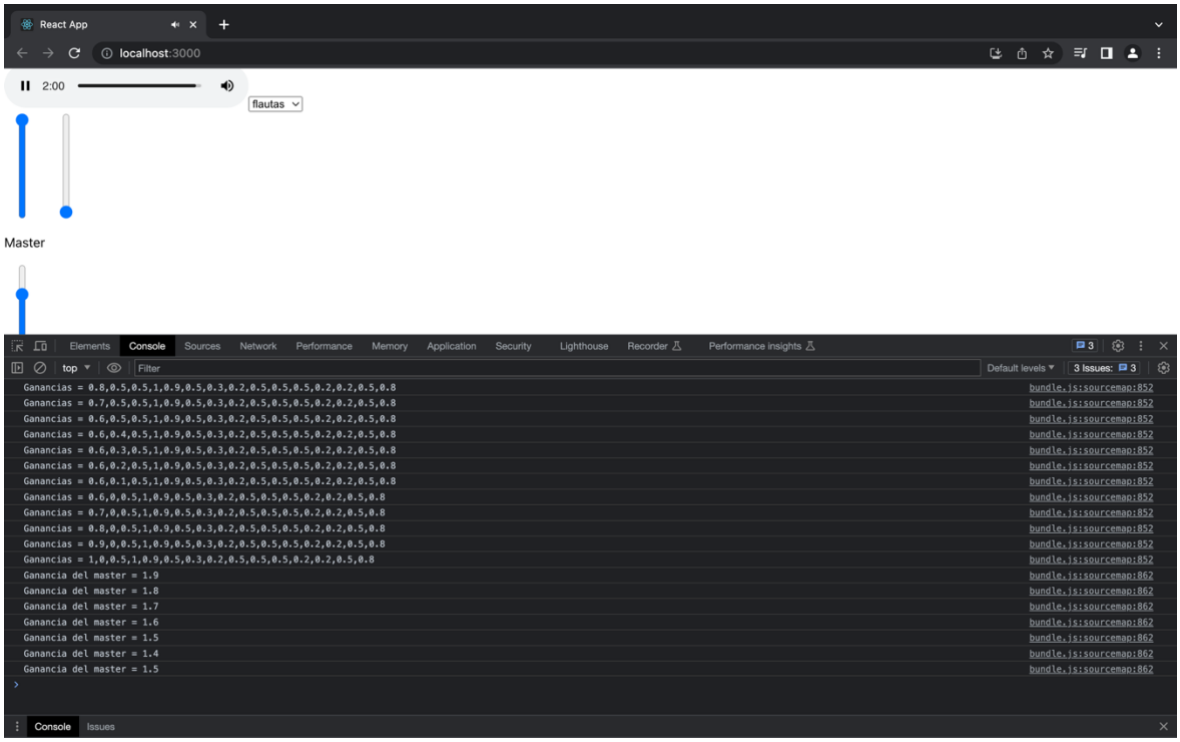

Figura 8.6. Ajuste de mezcla en la posición de los micrófonos cercanos.

En los mensajes de depuración de la Figura 8.6 se comprueba que el valor de las ganancias que se ajustan con los "*faders*" de las posiciones de micrófonos cercanos se

corresponden con el primer y segundo valor de la lista, el resto permanecen inmutables. De esta forma, se mantiene la configuración de los niveles de señal del resto de grupos de instrumentos cuando se vuelva a la posición del micrófono principal.

Volviendo a la posición del micrófono principal, es posible realizar una rotación haciendo uso del selector de rotación como aparece representado en la Figura 8.7.

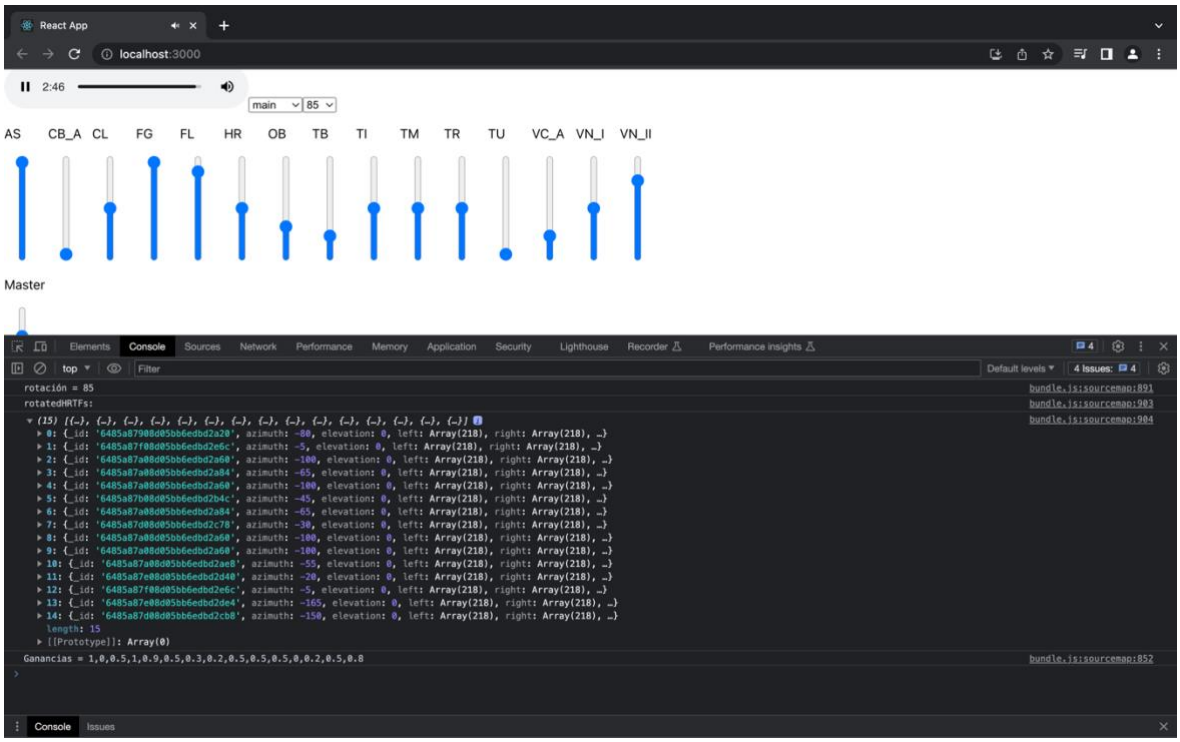

Figura 8.7. Rotación de 85º hacia la derecha.

En la Figura 8.7 se puede comprobar la lista de HRTFs recuperadas tras aplicar la rotación, las cuales son cargadas en la cadena de audio para ajustar la escena sonora ajustando las direcciones de los grupos de instrumentos según las direcciones que se observan en el mensaje de depuración.

Aplicar una rotación de 0º es realmente volver a la configuración de localizaciones de los grupos de instrumentos original, como se muestra en la Figura 8.8.

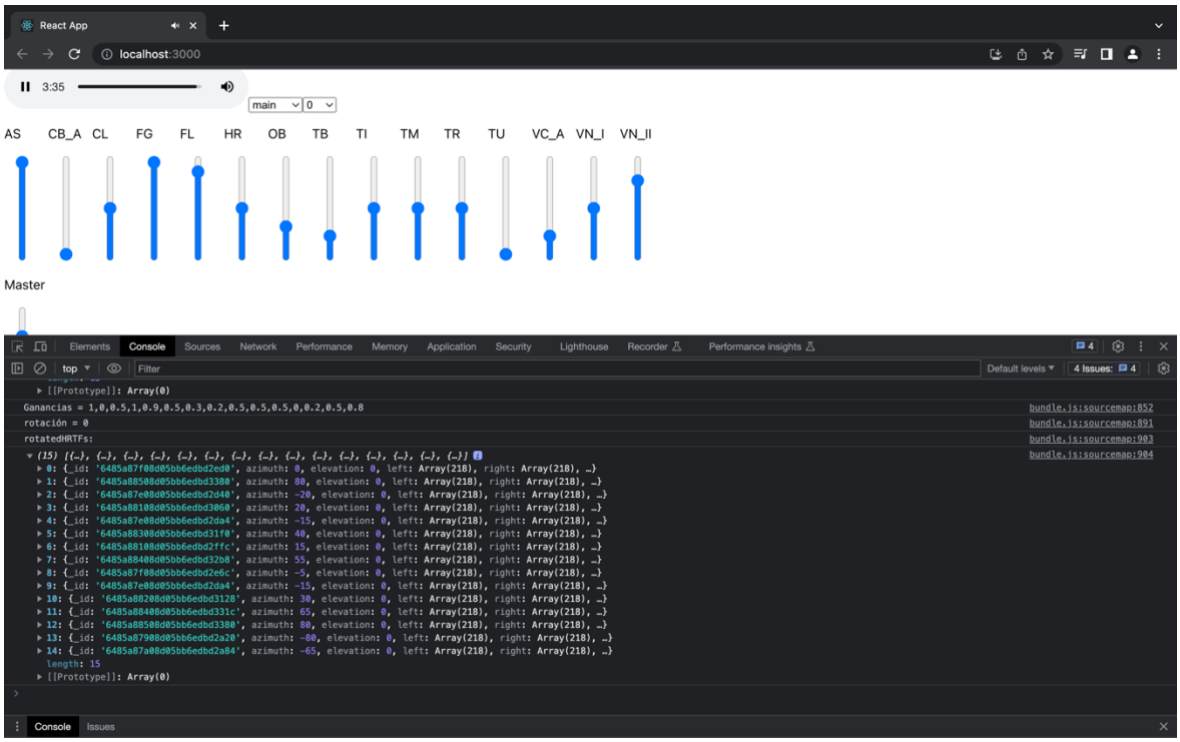

Figura 8.8. Vuelta a las localizaciones originales con una rotación de 0º.

Por último, cuando la presentación de directo termina, el archivo de manifiesto MPD es modificado para describir una presentación multimedia bajo demanda. Para presentaciones bajo demanda, el reproductor de audio muestra la duración del contenido multimedia de la misma, como en la Figura 8.9.

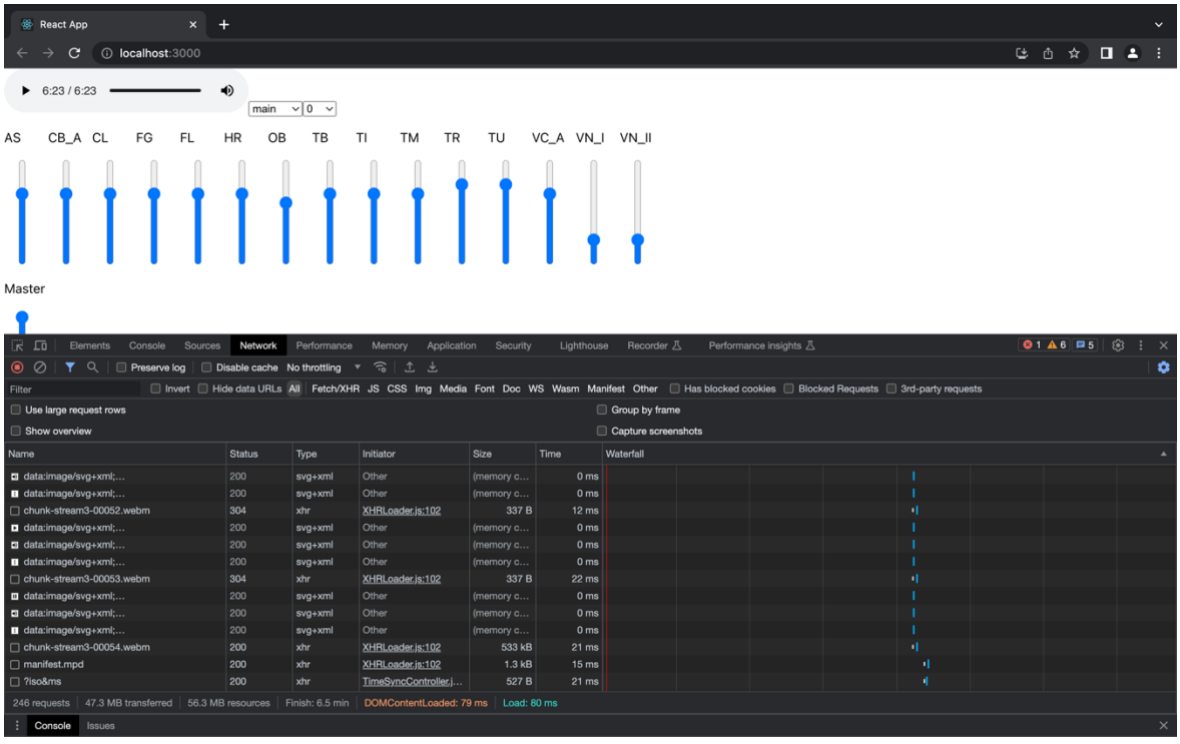

Figura 8.9. Interfaz con una presentación bajo demanda.

8.2. Ancho de banda requerido

En este apartado se estudiará el ancho de banda que el servicio necesita para su correcto funcionamiento.

Los valores que se van a obtener dependen de los parámetros específicos de la transmisión de la presentación multimedia MPEG-DASH y son, evidentemente, susceptibles a cambiar al usar otros parámetros.

A continuación se enumera una serie de datos necesarios para realizar los distintos cálculos.

- Señales distribuidas: 3 señales estereofónicas y una señal multicanal de 15 canales, cada una de ellas se corresponde con un "*Adaptation Set*".
- Códec Opus, bitrate de 64 Kbps (kilobits por segundo) por canal.
- Segmentos multimedia de duración 4 segundos.
- Periodo de actualización del archivo MPD de 4 segundos.
- No se generan "*Representations*" adicionales para cada "*Adaptation Set*".

Esto significa que, teóricamente, el ancho de banda que será requerido para poder enviar los datos multimedia hasta el back-end se calcula multiplicando el total de canales de audio por el bitrate de salida del codificador.

Al tener 15 canales de la señal multicanal y 6 canales de las tres señales estereofónicas, en total suman 21 canales de audio, que con un bitrate de 64 Kbps por canal, el **ancho de banda** a la salida del **codificador** es de unos **1,33 Mbps** (Megabits por segundo).

La Figura 8.10 representa el consumo de ancho de banda real durante el proceso de ingesta de datos multimedia de la presentación bajo estudio.

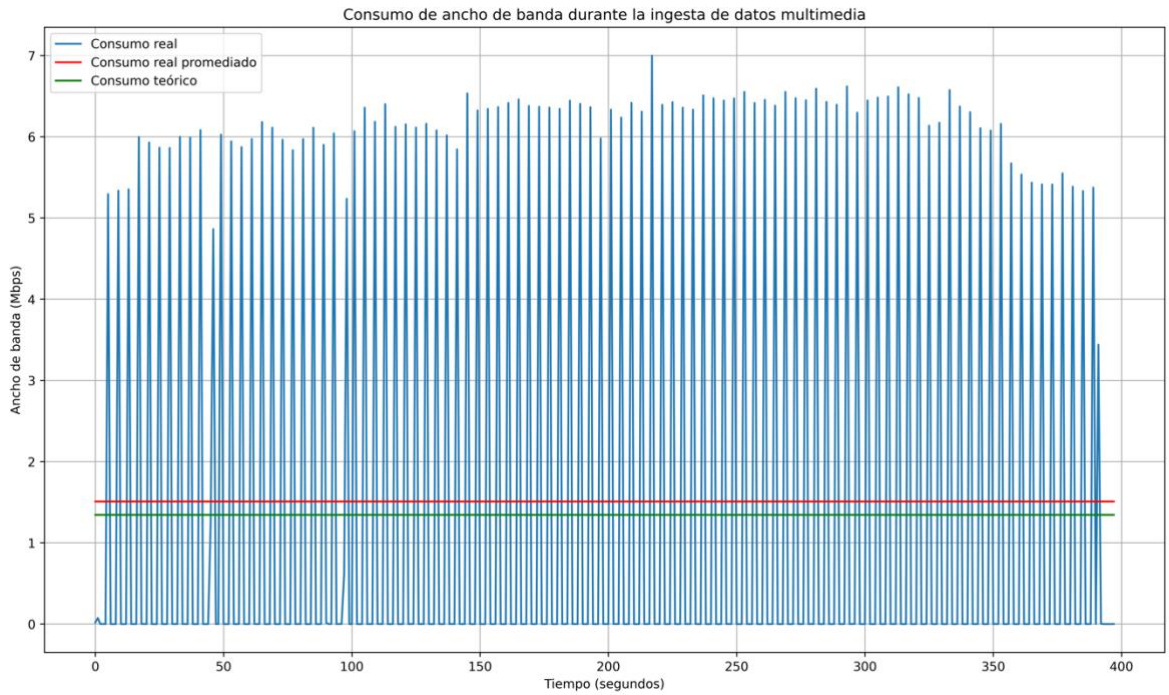

Figura 8.10. Consumo de ancho de banda durante la ingesta de datos.

En la Figura 8.10 se observa que el consumo real no es para nada continuo, sino que la **transmisión de datos** es a **ráfagas**.

FFmpeg codifica las señales de audio y forma segmentos de 4 segundos, cuando estos segmentos han terminado de procesarse, son enviados al back-end. Los picos que se observan en la Figura 8.10 son las distintas peticiones HTTP que FFmpeg realiza para subir estos segmentos multimedia, además del archivo de manifiesto. La separación entre estos picos es de, precisamente, alrededor de 4 segundos.

Promediando los valores del ancho de banda del consumo real a lo largo de todo el intervalo de observación se obtiene el consumo real promediado. Este valor se puede comparar con el consumo teórico y, como cabría esperar, es mayor debido al proceso de encapsulación que ocurre en la transmisión de información por la red.

El **consumo real promediado es de unos 1,51 Mbps**. Con los valores del consumo real promediado y el consumo teórico, es posible calcular la **eficiencia** de transmisión, obteniéndose un valor del **88,99%**.

En la Figura 8.11 puede observarse el consumo de ancho de banda durante la reproducción del contenido multimedia en el cliente.

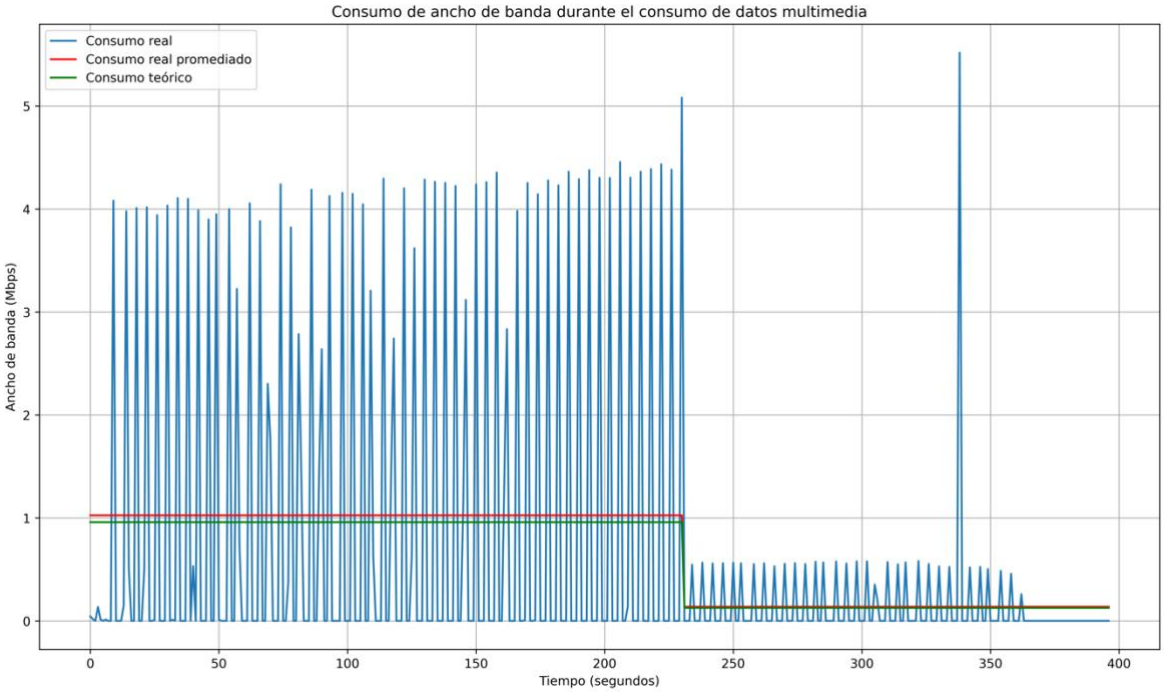

Figura 8.11. Consumo de ancho de banda durante el consumo de datos multimedia.

Se comprueba que, igual que ocurre en la ingesta de datos, el consumo del ancho de banda durante la reproducción del contenido es a ráfagas.

En la Figura 8.11 se visualiza un cambio de la posición del micrófono principal a una posición de algún micrófono cercano alrededor del segundo 230. Esto se deduce debido a la caída de ancho de banda consumido que se produce en dicho instante temporal.

Existen, por lo tanto, dos intervalos distintos que se deben estudiar, uno cuando se reproduce el audio captado en el micrófono principal y otro cuando se reproduce el audio de un micrófono cercano.

En el intervalo en el que se reproduce el contenido multimedia de la posición del **micrófono principal**, el **ancho de banda teórico**, a la salida del codificador, es de **0,96 Mbps**, mientras que el **promedio** del **consumo real** del ancho de banda es algo menos de **1,03 Mbps**, resultando en una eficiencia de transmisión del **93,54%**.

Por otro lado, en el intervalo de reproducción del contenido del **micrófono cercano**, el **ancho de banda teórico** es de **128 Kbps**, mientras que el **promedio** del **consumo real** es de algo menos de **139,05 Kbps**, lo que significa una **eficiencia** de transmisión del **92,05%**.

Estos cálculos permiten afirmar que la **eficiencia de transmisión es mayor al enviar señales de audio con más canales**.

9. CONCLUSIONES

A continuación, se expondrán distintas conclusiones obtenidas del funcionamiento de la solución de difusión de contenido de audio inmersivo de una actuación en directo.

A modo de recapitulación, en este trabajo se ha propuesto una solución de "*streaming*" para difundir contenido multimedia de audio generado en conciertos de música clásica y, a diferencia de otros servicios similares, el usuario puede interactuar en tiempo real con la aplicación para ajustar la mezcla de audio a sus preferencias, suprimiendo o realzando grupos de instrumentos e incluso cambiar de posición. Adicionalmente, la mezcla de la orquesta completa está espacializada, dotando de un carácter mucho más inmersivo la actuación que el usuario escucha.

9.1. Ventajas de la solución propuesta

Entre las ventajas de la solución propuesta, cabe destacar los enfoques del envío de contenido multimedia con señales de audio multicanal y la realización del procesado de audio en el front-end.

La transmisión del contenido de audio en formato multicanal ha demostrado ser muy versátil y conveniente. Permite transportar toda la información requerida por el cliente para la correcta reproducción de la presentación y es más eficiente que enviar múltiples "*streams*" de datos.

Realizar el procesado de audio en el front-end permite ahorrar recursos en el back-end. La carga de computación que supone este procesado de audio no recae sobre este, teniendo únicamente que encargarse de procesar las peticiones que transportan el contenido multimedia y, posteriormente, difundirlo junto con los datos necesarios, como las HRTFs y la información acerca de las distintas presentaciones.

El soporte multimedia del servicio está basado en el protocolo MPEG-DASH, lo que implica que todos intercambios de información, tanto multimedia como de otro tipo, se llevan a cabo mediante HTTP como protocolo único. Esto permite unificar la ingesta y difusión de contenido multimedia con el envío de recursos de otro tipo en servidores HTTP estándar.

Adicionalmente, por los principios de desarrollo seguidos, el servicio puede ser empleado para difusión de contenidos en vivo o bajo demanda sin realizar ninguna modificación, y se facilita el despliegue y escalabilidad del mismo al estar fundamentado en tecnologías de contenerización.

9.2. Limitaciones

El servicio solo es compatible con aquellos navegadores con soporte de las MSE y del formato contenedor WebM.

Aunque los navegadores web más usados mantienen un historial de versiones desde que incluyeron soporte a las MSE y WebM, cabe destacar que las versiones de Safari para dispositivos móviles con sistema operativo IOS no tienen compatibilidad completa, como se indica en [30, 31].

Opera Mini no tiene soporte para estas tecnologías.

Safari en su versión de escritorio soporta las MSE desde su versión 8 y tiene compatibilidad completa con WebM desde la versión 16.0.

Firefox soporta las MSE desde la versión 42, siendo compatible con WebM desde su versión 28.

Las pruebas de funcionamiento se han realizado con el navegador web Google Chrome en su versión 114. En general, los navegadores web basados en Chromium, como también lo son Edge u Opera en sus versiones recientes, son perfectamente compatibles con estas tecnologías.

Durante las pruebas de funcionamiento se ha detectado un fallo ocasional en la decodificación de audio cuando se cambia desde una posición de micrófono cercano al micrófono principal. Este error provoca que la interrupción de la reproducción del contenido, siendo necesario recargar la página para solucionarlo.

Solucionar este problema constituye una prioridad a corto plazo en el posterior desarrollo del servicio, ya que, aunque ocurre de forma muy puntual, resulta bastante molesto.

9.3. Futuras líneas de desarrollo

En este apartado se citan brevemente algunas líneas de desarrollo que podrían ser incorporadas al servicio propuesto en un futuro.

- Extender el sistema de rotaciones para permitir a los usuarios cambiar la dirección en la que se encuentran mirando no solo en azimut, sino también en elevación.
- Configurar distribuciones de los instrumentos que conforman una presentación libremente. Ya que se dispone de una base de datos con HRTFs

en prácticamente todas las direcciones espaciales, se podría dar la opción a los usuarios de poder espacializar los instrumentos a su gusto si no prefieren ajustarse a la distribución original de la actuación.

- Incluir más opciones de procesado de audio para personalizar aún más la mezcla de audio. El procesado de audio en el front-end ha funcionado mejor de lo esperado. Se podrían incluir más nodos de procesado, como ecualizadores, tanto en la mezcla general como para los canales individuales. También se podrían incorporar algunos efectos, como reverberación.
- Incluir sistemas de usuarios. Este sistema incluso podría distinguir entre usuarios individuales y de otros tipos, como orquestas. Esto abriría la puerta a incluir aspectos algo más sociales, como sistemas de seguimiento de artistas, notificaciones de emisiones en directo, etc.
- Desarrollar un sistema de entradas virtuales. MPEG-DASH soporta sistemas de administración de derechos digitales (*DRM*, del inglés *Digital Rights Management*) haciendo posible cifrar los contenidos de una presentación. Este sistema se puede desarrollar junto con el sistema de usuarios, y permitiría restringir la distribución del contenido a un cierto número de usuarios, según las preferencias de los creadores.

10. REFERENCIAS

1. HANNAH. Orchestra Arrangement: How Is It Arranged? Online. 13 September 2021. [Accessed 9 June 2023]. Available from: https://orchestracentral.com/orchestra-arrangement-how-is-it-arranged/

2. CABAÑAS-MOLERO, Pablo, MUÑOZ-MONTORO, Antonio J., VERA-CANDEAS, Pedro and RANILLA, José. The music demixing machine: toward real-time remixing of classical music. *The Journal of Supercomputing*. Online. 6 April 2023. DOI 10.1007/s11227-023-05192-5.

3. MULLIGAN, Mark. Music subscriber market shares 2022. *Midia Research*. Online. 7 December 2022. Available from: https://midiaresearch.com/blog/music-subscriber-market-shares-2022

4. CARLSSON, Erik and KAKOGIANNI, Eirini. Smoother Streaming with BBR. *Spotify Engineering*. Online. 31 August 2018. [Accessed 14 February 2023]. Available from: https://engineering.atspotify.com/2018/08/smoother-streaming-with-bbr/

5. Amazon. Amazon Music Playback. *Amazon Developer*. Online. 2 March 2023. [Accessed 14 February 2023]. Available from: https://developer.amazon.com/docs/music/playback_overview.html

6. GONG, Jeff, DHANJU, Sahil, LU, Chih-Chiang and SHEN, Yueshi. Live Video Transmuxing/Transcoding: FFmpeg vs TwitchTranscoder, Part I. *Twtich Blog*. Online. 10 October 2017. [Accessed 15 February 2023]. Available from: https://blog.twitch.tv/en/2017/10/10/live-video-transmuxing-transcoding-f-fmpeg-vs-twitch-transcoder-part-i-489c1c125f28/

7. Google. YouTube Live Streaming Ingestion Protocol Comparison. *Google for developers*. Online. 4 November 2022. [Accessed 9 March 2023]. Available from: https://developers.google.com/youtube/v3/live/guides/ingestion-protocol-comparison

8. PANTOS, Roger and MAY, William. *HTTP Live Streaming*. Online. August 2017. RFC Editor. [Accessed 20 March 2023]. Request for Comments. Available from: https://www.rfc-editor.org/info/rfc8216Published: RFC 8216DOI: 10.17487/RFC8216

9. ISO/IEC. *23009-1:2022. Information technology — Dynamic adaptive streaming over HTTP (DASH) — Part 1: Media presentation description and segment formats*. Online. 2022. Standard. August 2022. Switzerland. 23009–1. [Accessed 27 March 2023]. Available from: https://www.iso.org/standard/83314.html

10. FFMPEG PROJECT. About FFmpeg. Online. [Accessed 3 June 2023]. Available from: https://ffmpeg.org/about.html

11. FFMPEG PROJECT. FFmpeg Formats Documentation. *FFmpeg Formats Documentation*. Online. 31 May 2023. [Accessed 22 February 2023]. Available from: https://ffmpeg.org/ffmpeg-formats.html

12. COPES, Flavio et al. Introduction to Node.js. Online. [Accessed 4 June 2023]. Available from: https://nodejs.dev/en/learn/

13. Express/Node introduction. Online. 24 February 2023. [Accessed 4 June 2023]. Available from: https://developer.mozilla.org/es/docs/Learn/Server-side/Express_Nodejs/Introduction

14. WILSON, Douglas. Express. Online. 13 September 2022. [Accessed 4 June 2023]. Available from: https://expressjs.com

15. MONGODB INC. MongoDB Manual. Online. 26 May 2023. [Accessed 4 June 2023]. Available from: https://www.mongodb.com/docs/manual/core/databases-and-collections/

16. MONGODB INC. Advantages of MongoDB. Online. [Accessed 5 June 2023]. Available from: https://www.mongodb.com/advantages-of-mongodb

17. REACT DEVELOPMENT TEAM. React, the library for web and native user interfaces. Online. [Accessed 5 June 2023]. Available from: https://react.dev

18. VIDEO-DEV. hls.js. Online. 1 June 2023. [Accessed 5 June 2023]. Available from: https://github.com/video-dev/hls.js

19. DASH INDUSTRY FORUM. Dash.js wiki. Online. 15 February 2021. [Accessed 5 June 2023]. Available from: https://github.com/Dash-Industry-Forum/dash.js/wiki

20. MDN CONTRIBUTORS. Web Audio API. Online. 19 February 2023. [Accessed 5 June 2023]. Available from: https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API

21. DOCKER, INC. Docker overview. Online. 6 June 2023. [Accessed 6 June 2023]. Available from: https://docs.docker.com/get-started/overview/

22. ALGAZI, V.R., DUDA, R.O., THOMPSON, D.M. and AVENDANO, C. The CIPIC HRTF database. In : *Proceedings of the 2001 IEEE Workshop on the Applications of Signal Processing to Audio and Acoustics (Cat. No.01TH8575)*. 2001. p. 99–102. DOI 10.1109/ASPAA.2001.969552.

23. LIBRARY OF CONGRESS. Advanced Audio Coding (MPEG-4). Online. 28 April 2022. [Accessed 9 June 2023]. Available from: https://www.loc.gov/preservation/digital/formats/fdd/fdd000114.shtml

24. OPPENHEIM, Alan V., SCHAFER, Ronald W. and BUCK, John R. *Discrete-Time Signal Processing*. . Second. Prentice-hall Englewood Cliffs, 1999. ISBN 0-13-754920-2.

25. OMRI KAZELNIK. Going Behind The Scenes of Docker Networking. Online. March 2020. [Accessed 11 June 2023]. Available from: https://argus-sec.com/blog/engineering-blog/docker-networking-behind-the-scenes/#:~:text=All%20the%20Docker%20containers%20are,communicate%20with%20the%20outside%20world.

26. VALIN, Jean-Marc, VOS, Koen and TERRIBERRY, Timothy B. *Definition of the Opus Audio Codec*. Online. September 2012. RFC Editor. [Accessed 12 June 2023]. Request

for Comments. Available from: https://www.rfc-editor.org/info/rfc6716Published: RFC 6716DOI: 10.17487/RFC6716

27. THE WEBM PROJECT. WebM Container Guidelines. Online. 28 November 2017. [Accessed 12 June 2023]. Available from: https://www.webmproject.org/docs/container/

28. RAINER, Benjamin, LEDERER, Stefan, MUELLER, Christopher and TIMMERER, Christian. A seamless Web integration of adaptive HTTP streaming. In : Online. January 2012. p. 1519–1523. [Accessed 12 June 2023]. Available from: https://www.researchgate.net/publication/236594173_A_seamless_Web_integration_of_adaptive_HTTP_streaming

29. MATROSKA DEVELOPMENT TEAM. Data Layout. Online. 5 April 2023. [Accessed 12 June 2023]. Available from: https://www.matroska.org/technical/diagram.html

30. CAN I USE. Can I use Media Source Extensions? Online. 2023. [Accessed 20 June 2023]. Available from: https://caniuse.com/?search=media%20source%20extensions

31. CAN I USE. Can I use WebM? Online. 2023. [Accessed 20 June 2023]. Available from: https://caniuse.com/?search=webm

ANEXO I: MANUAL DE DESPLIEGUE DEL SERVICIO

Es necesario tener instalado Docker en la máquina en la que se va a desplegar el servicio de streaming. Docker se puede descargar desde su web oficial https://www.docker.com/, donde además se proporciona la documentación necesaria para realizar la instalación en función del sistema operativo del equipo.

Una vez instalado Docker, hay que descargar el repositorio de GitHub https://github.com/jgm00115/TFG_Telema.git.

La descarga se puede hacer desde el mismo navegador web o, si se tiene git instalado, ejecutando el siguiente comando:

% git clone https://github.com/jgm00115/TFG_Telema.git

Una vez clonado, es necesario acceder al directorio del repositorio localmente y ejecutar el siguiente comando:

% docker compose up --build

Con esto, Docker creará las imágenes del back-end y el front-end desde el código fuente del repositorio, descargará las imágenes de MongoDB y mongo-express, creará los volúmenes necesarios, compondrá la red de tipo "*bridge*" e instanciará todos los contenedores necesarios para desplegar el servicio.