UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

Autor
Francisco Yeray Marrero Cabrera

Director
Cayetano Guerra Artal

Titulación
Ingeniería Informática

2018 - 2019

Aplicación web para la comunicación audiovisual en entornos educativos

Proyecto Final de Carrera - Memoria

Índice general

Índice de figuras

Índice de tablas

Índice de códigos

Índice de Diagramas

1. Resumen del proyecto

Este proyecto se centra en el desarrollo de una plataforma virtual para la comunicación audiovisual en tiempo real, principalmente centrada en entornos educativos, realizada, prácticamente en su totalidad, con el lenguaje Javascript.

Se basa en una aplicación web con una interfaz simple, limpia e intuitiva, que sirve como plataforma para impartir docencia y/o realizar presentaciones vía *online*, permitiendo a los usuarios hacer uso de esos componentes que forman parte de cualquier tipo de exposición educativa, o de cualquier otra índole. La plataforma nos permite compartir vídeo, audio, mensajes de texto y presentaciones en formato PDF, teniendo, de ese modo, todos los atributos necesarios para impartir y recibir docencia a distancia. Como si de un aula de cualquier universidad se tratase, un profesor y sus alumnos pueden ingresar en la aplicación e interaccionar entre sí como si de un día de clase normal se tratara, pero sin la necesidad de desplazarse. Con este sistema, un profesor y sus alumnos compartirán asignaturas, que en la plataforma se traducirá en salas en las cuales los alumnos que tengan dicha materia podrán entrar. Además, puede ser usada como una plataforma para llevas a cabo tutorías individuales online.

Para desarrollar la aplicación se hace uso de Mediasoup |1|, un potente WebRTC que usa el modelo SFU, integrable en aplicaciones hechas con Node.js |2|, que será el entorno *backend* utilizado para construir esta plataforma. En cuanto al *frontend*, se usa jQuery |3| y ReactJS|4| para conseguir una interfaz dinámica y de una sola página. Al respecto de las presentaciones, podremos escoger un archivo en formato PDF guardado en el disco duro local y mostrarlo al resto de usuarios de la sala gracias a PDF.js |5|.

2. Introducción

Han pasado casi dos décadas desde que finalizamos el siglo XX, la tecnología no ha parado de evolucionar, y con ello, se han visto incrementadas las opciones para realizar actividades desde casa, ya sea pedir comida a través de una aplicación móvil, desarrollar funciones laborales a través de teletrabajo o realizar videoconferencias, sin necesidad de compartir una misma localización. Extrapolar el uso de estas tecnologías en el mayor rango de campos posibles es un camino en el que se ha de seguir trabajando dados los beneficios que supone el poder realizar cualquier función, laboral o personal, sin la obligación del desplazamiento. En el terreno académico, es realmente importante e interesante, ya que el acceso a los conocimientos no debería depender de ninguna variable, y menos aún del azar, que es la que resulta de las oportunidades que varían en función del lugar de nacimiento.

La docencia acaba siendo una actividad en la que se ven involucrados una serie de factores: uno o varios profesores, varios alumnos, medios de apoyo para realizar las enseñanzas y un espacio común. Por ello, si dispusiéramos de un entorno virtual en el que todos los factores pudieran confluir para realizar dicha actividad, se evitarían muchas restricciones.

De esa necesidad, y en base a esa premisa, se toma la decisión de comenzar con la implementación de un sistema que sirva como entorno virtual para la comunicación audiovisual especialmente enfocado en la educación, pero que, a su vez, sea lo suficientemente flexible como para que pueda tener aplicación en otros sectores.

El desarrollo del proyecto consistirá, principalmente, en la implementación de un sistema de intercambio de información audiovisual, cuya base principal será la tecnología WebRTC, *Web Real Time Communications*, que se usa para el intercambio de datos en tiempo real. Sobre esta base, se introducirán una serie de funcionalidades que permitan a los actores involucrados disponer de las herramientas necesarias para la actividad docente. Para crear dicho entorno, se hace uso de tecnologías como ReactJS, para la creación de aplicaciones web de una sola página; SocketIO, para el intercambio de datos necesarios para el establecimiento de la comunicación; además de, HTML, JavaScript y jQuery, elementos fundamentales para este tipo de sistemas.

3. Tecnologías Web

Con el paso de los años, hemos ido viendo como las aplicaciones web han ido cobrando mayor importancia, en parte por la capacidad hardware y de las conexiones de red, pero mayormente por el crecimiento de las herramientas que se han puesto por y para disposición de los desarrolladores. Gracias a esta evolución, hemos podido afrontar con mayor efectividad problemas costosos, como la fragmentación de los usuarios finales, la diversidad de navegadores, la falta de seguridad, la modularidad del desarrollo de grandes aplicaciones o el coste para afrontar aplicaciones de gran envergadura. Además de esto, se ha ido incrementando la cantidad de usuarios que prefieren acceder al contenido que les interesa a través de aplicaciones web, en vez de hacerlo a través de aplicaciones de escritorio.

En la actualidad, el contenido multimedia tiene un gran peso de cara al interés de los usuarios y, gracias a las capacidades que se tiene hoy en día en ese aspecto para desarrollar aplicaciones web, podemos generar una gran oferta. En este capítulo se presentarán, concretamente, las herramientas y tecnologías que se han usado en nuestra plataforma.

3.1. JavaScript ES6

En el año 1995 Brendan Eich crea un lenguaje al que acabaría dándole el nombre de JavaScript debido a una estrategia de marketing. En 1997 se crea un comité para estandarizar JavaScript por la *European Computer Manufacters Association*, ECMA, en el cual se diseña el estándar del DOM (*Document Object Model*) para evitar incompatibilidades entre navegadores. A partir de entonces, los estándares de este lenguaje se rigen por ECMAScript.

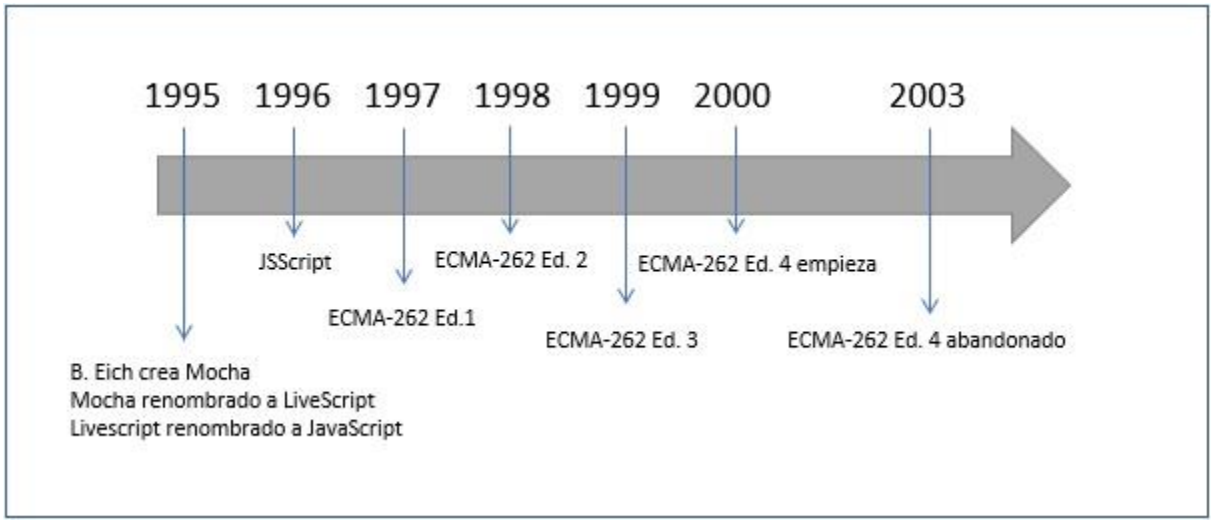


Figura 1. Evolución cronológica del lenguaje JavaScript.

En 1999 aparece la tercera versión del estándar ECMAScript, que se iba a mantener vigente durante muchos años. Hubo pequeños intentos de escribir la versión 4, pero no fue posible, por lo que acabaron abandonando la idea. No fue hasta el 2011, cuando finalmente se aprobó la versión 5 (ES5), que ha sido la versión estándar que soportaban la mayoría de los navegadores hasta hace poco y que sigue siendo, de algún modo, la candidata ideal si lo que se pretende es que el código escrito valga para todos los tipos de sistemas y navegadores.

Una vez llegado el año 2015 aparece la versión que se ha utilizado para desarrollar este proyecto, la ES6 |9|. Una versión que presenta diversas mejoras al lenguaje y nuevas variantes respecto a la programación con JavaScript.

Las características básicas de ES6 son:

- **Función *Arrow*:** conocidas en otros lenguajes (C# y Java) como expresiones lambda, se trata de abreviaciones de funciones utilizando el operador =>. Su principal característica es el mejor control del valor de *this*, ya que, al usar las nuevas funciones flecha, el valor de *this* va a ser igual a la función contenedora.

- **Clases:** si algo se había echado en falta hasta la llegada de la versión ES6 por parte de los programadores que usan este lenguaje, ha sido el poder hacer uso

de clases como en cualquier otro lenguaje orientado a objetos. Ahora, con Javascript, ya podemos hacer uso de clases como si se tratara de programación en Java, con sus herencias, constructores, métodos y atributos.

- ***This*:** el uso de la variable *this* podía llegar a ser una tarea realmente ardua. Con la llegada de ES6 podemos tener control sobre el contexto al que se refiere usando el método *bind*, que nos permite referenciar un contexto u otro.

- ***Let* y *Const*:** para referenciar variables que no sean accesibles más allá de un ámbito podemos usar *let* y para crear constantes podemos usar *const*.

- **Módulos:** es importante organizar el código y hacerlo, modular y reutilizable. Ahora ya es posible crear, importar y exportar módulos como si se tratara de Python o Ruby.

- **Promesas:** la programación asíncrona en JavaScript se había convertido en un caos con el uso de *callbacks* anidados. Con la inclusión de las promesas, objetos que se usan para cubrir las operaciones asíncronas, obtenemos un código limpio.

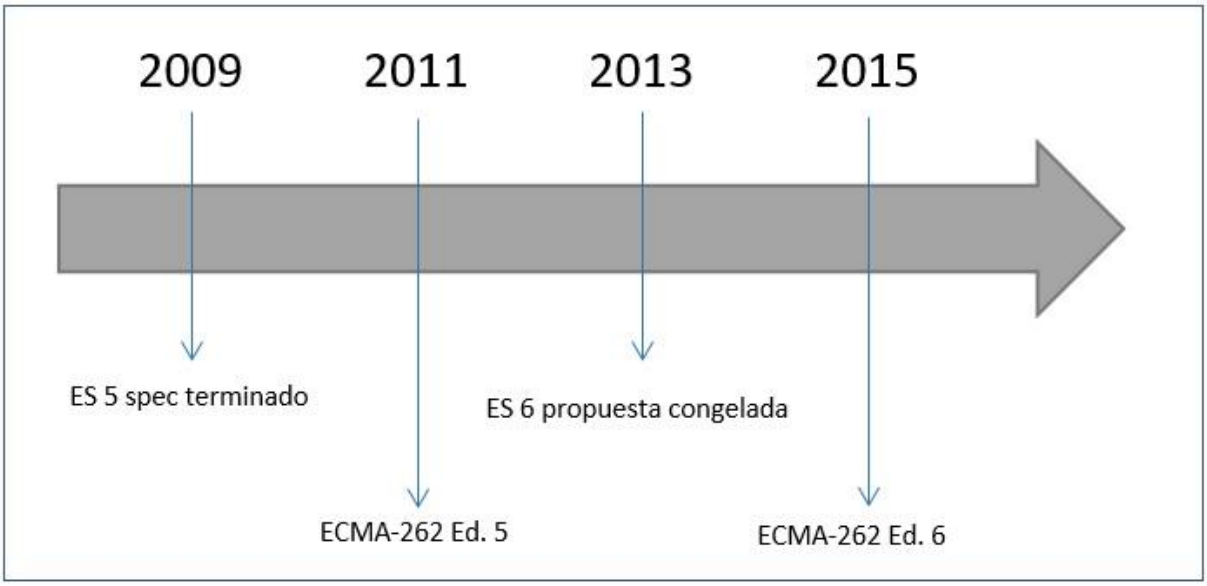


Figura 2. Evolución cronológica de JavaScript. Segunda etapa.

3.2. JQuery

En el año 2006 se presentó jQuery, una biblioteca que trataba de hacer multiplataforma el lenguaje JavaScript en todos los navegadores. Durante una década, jQuery ha sido el estándar de desarrollo de los programadores Web y la biblioteca más utilizada puesto que se encuentra en aproximadamente el 96% de las aplicaciones web.

La librería jQuery es una biblioteca de código abierto y *software* libre que resulta de gran utilidad para simplificar la manera de interactuar con los documentos HTML, manejar eventos y desarrollar animaciones. Otras funcionalidades de esta herramienta, como pueden ser el manejo del árbol DOM o la posibilidad de interaccionar con la técnica de AJAX, no van a ser necesarias, puesto que haremos uso de ReactJS para abordar la creación de la interfaz de usuario y manejo de los elementos HTML, y de Socket.IO para la comunicación en tiempo real con el servidor.

3.3. ReactJS

En el desarrollo de aplicaciones web SPA, de una sola página, son necesarias herramientas que actualicen parte de la vista sin tener que recargar toda la página. En circunstancias como estas es donde *frameworks* como ReactJS tienen su papel. Se trata de una librería desarrollada por Facebook para facilitar la creación de interfaces de usuario en la que los datos pueden cambiar constantemente y deben ser mostrados al usuario sin necesidad de refrescar la página ni recargar partes innecesarias.

Entre las características más importantes de ReactJS está su virtual DOM, un clon del DOM original sobre el cual se trabaja, que nos permite recargar individualmente cada componente cuando sea necesario, lo que ofrece una gran velocidad a la hora de renderizar vistas. También nos ofrece la capacidad de reutilizar componentes, que nos permite mantener una mejor estructura de nuestro proyecto y con ello conseguir un mayor rendimiento.

Para facilitar lo máximo posible la tarea, a la hora de trabajar con esta librería, tenemos que hacer uso de una especie de pseudolenguaje llamado JSX. Este pseudolenguaje permite escribir directamente etiquetas HTML dentro del *render* de ReactJS, quedando todo más simple y fácil de leer. Debido a que el código escrito con JSX no es interpretable directamente como código JavaScript, es necesario hacer uso de Babel, una herramienta con la que podemos transformar código de última generación (o con funcionalidades extras) a código que cualquier navegador o versión de Node.js entienda.

3.4. Socket.IO

Internet se creó a partir del llamado paradigma solicitud/respuesta de HTTP. El sistema de intercambio de datos fue evolucionando y hace ya algún tiempo que existen tecnologías que mantienen una conexión HTTP abierta que permiten al servidor enviar información al cliente en cuanto detecta nuevos datos disponibles, sin necesidad de tener que enviar una solicitud para ello.

Hay un tipo de aplicaciones web en el cual surge la necesidad de tener una comunicación bidireccional en tiempo real con el servidor, es en ese tipo de situaciones, donde herramientas como Socket.io tienen su papel protagonista.

Socket.io es una librería JavaScript para Node.js que permite una comunicación bidireccional en tiempo real entre cliente y servidor. Para ello, hace uso, principalmente, de una tecnología llamada WebSocket.

3.4.1. WebSocket

WebSocket es un protocolo con el que se consigue un canal de comunicación permanente y bidireccional entre cliente y servidor a través de una única conexión TCP, permitiendo enviar mensajes e información binaria en ambas direcciones en el mismo instante de tiempo.

El establecimiento de una conexión comienza desde el cliente a través de una petición *get* HTTP, que, si es respondida favorablemente por el servidor (con un código de estado 101), se actualiza a una conexión de tipo WebSocket. A partir de este punto, se mantiene un canal TCP abierto de forma permanente, que ambas partes utilizarán para enviarse información bidireccionalmente.

```
53  //Se dispara cuando se inicia la conexión
54  socket.on('connection', function(client) {
55    //Vemos el id del cliente que ha creado la conexión
56    console.log('Client ' + client.id + ' connected to socket.');
57    //El cliente nos envía datos al servidor
58    client.on('data', function(data) {
59      console.log(data);
60      //Del mismo modo, también podemos enviar datos al cliente
61      client.emit('data', data);
62    });
63  });
```

Código 1. Ejemplo de WebSocket con Socket.io.

3.5. Node.js

El desarrollo de aplicaciones web suele llevar a los programadores a tener que usar dos tipos de lenguaje, uno para la parte del cliente, normalmente JavaScript, y otro para la parte del servidor, que podría ser PHP, PERL, Python o Ruby, entre otros. Node.js ofrece la posibilidad de programar con JavaScript en la parte del servidor, facilitando la tarea del desarrollo *full stack*.

Node.js es un entorno multiplataforma de código abierto para la capa del servidor, que permite compilar JavaScript en código máquina gracias al motor V8 de Google. Al margen de la facilidad que supone tener la posibilidad de trabajar con el mismo lenguaje en ambos lados, este entorno ofrece ventajas relevantes a la hora de trabajar con aplicaciones en tiempo real.

Las principales ventajas para trabajar con Node.js son:

- Permite mantener conexiones persistentes a través de WebSockets de un modo sencillo.

- Es altamente escalable, debido a su arquitectura dirigida por eventos y con Entrada/Salida asíncrona.

- Tiene un gestor de paquetes llamado NPM que permite mantener las dependencias de cada proyecto separadas en módulos reutilizables. Estos módulos pueden ser publicados en el repositorio que ofrece NPM y así compartirlos abiertamente.

- Existe una comunidad enorme detrás de Node, lo que implica una gran cantidad de contenido al alcance de la mano disponible en la red y en libros.

3.6. Mediasoup

Mediasoup es una librería de libre distribución y uso creada por Iñaki Baz Castillo, que permite construir un sistema de *streaming* de vídeo y audio usando el modelo SFU de WebRTC. Se trata de un proyecto que comenzó en el año 2015 y que actualmente sigue estando activo y recibiendo constantes mejoras. A pesar de que no esté respaldado por una documentación detallada y abundante, mantiene un grupo de Google muy activo donde su desarrollador responde todas las posibles dudas acerca de su uso.

Iñaki Baz Castillo es un ingeniero de telecomunicaciones apasionado por las tecnologías de desarrollo web que permiten la comunicación en tiempo real. Durante los últimos años de su carrera profesional, ha estado profundamente involucrado en VoIP (*Voice Over IP*), colaborando en muchos proyectos de código abierto y escribiendo decenas de artículos. También ha sido coautor en la especificación del protocolo incluido en el RFC 7118 |8|, donde comienzó su transición al mundo de WebRTC y poco después vendría su principal trabajo de código abierto, Mediasoup.

Esta API, que está ya en su versión 2.0, presenta una herramienta perfecta para conseguir desarrollar una arquitectura que permite a los navegadores comunicarse entre sí para compartir contenido multimedia en tiempo real. Lo que hace que te decantes por esta y no por otras librerías que también te permiten crear sistemas WebRTC, es la capacidad de escalado que se consigue gracias al uso del modelo SFU. Puede ser usado de un modo muy sencillo, ya que solo tiene que ser cargado como un módulo de Node.JS. En su totalidad, esta herramienta la forman dos módulos, uno para la parte del cliente y otro para la parte del servidor, que serán usados para crear los elementos necesarios para conseguir el correcto funcionamiento de la comunicación WebRTC de Mediasoup.

3.6.1. Arquitectura de Mediasoup-server

Debido a que se trata de un modelo basado en SFU, la parte principal y la que tiene la mayor parte de la carga de trabajo estará en el servidor. Será allí donde, haciendo uso de la librería, se crearán todos los procesos encargados de manejar las comunicaciones en tiempo real.

A continuación, presentamos las diferentes entidades que conforman la API y su uso en la parte del servidor:

- *Server*: procesos separados del hilo principal, que maneja todos los protocolos de comunicación e intercambio que usa Mediasoup (ICE, DTLS, RTP, RTCP, DataChannel, etc…).

- *Room*: representa una sala con todos los participantes (*Peers*). Todos los que estén en la sala podrán enviar y recibir los datos de audio y vídeo.

- *Peer*: instancia que representa a un participante dentro de la conferencia. Esta entidad tiene asociada instancias de *Transport*, *Producer* y *Consumer*.

- *Transport*: representa el camino de red que comunica cliente y servidor, negociado a través de ICE y DTLS.

- *Producer*: representa y maneja las tramas de audio y vídeo que deben ser enviadas del cliente al servidor. Un *Producer* está conectado a muchos *Consumers*.

- *Consumer*: representa y maneja las tramas de audio y vídeo que deben ser enviadas del servidor al cliente.

Con las entidades anteriores se crea el modelo expuesto en la Figura 3.

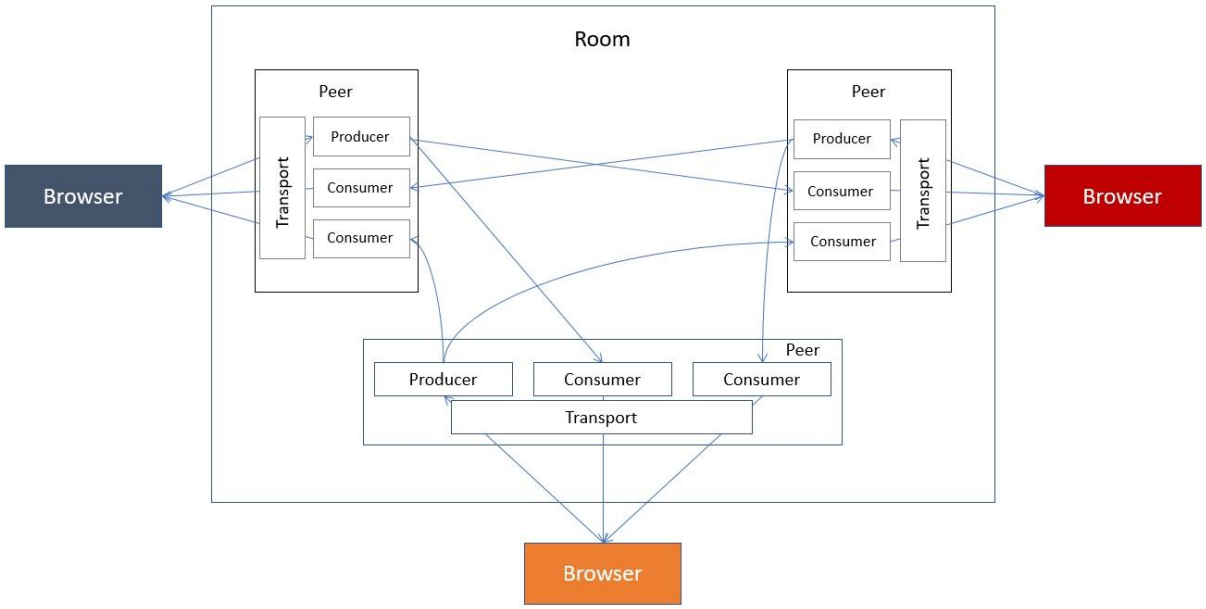


Figura 3. Proceso de comunicación de Mediasoup.

La Figura 3 muestra cómo funciona el proceso de comunicación usando Mediasoup. Tenemos una sala a la que se han unido 3 usuarios. Dichos usuarios compartirán en sus navegadores *streams* de audio y vídeo que serán enviados a través del protocolo de transporte al servidor, una vez que las tramas multimedia llegan al servidor se envían al resto de usuarios conectados a la sala, cuando los usuarios reciben las tramas de los otros usuarios conectados, las envían a sus clientes asociados a través de la capa de transporte. Dicho de otro modo, tenemos una entidad *Room*, a la que se han unido 3 *Peers*. En el cliente, cada usuario crea un *Producer* para enviar los datos necesarios al servidor a través de *Transport*. Una vez que la información llega al servidor, se comparte con el resto de *Peers* conectados, que crearán un *Consumer* por cada *Producer* que se haya generado por otro *Peer*. Estos *Consumers* serán enviados al cliente a través de *Transport*. Para crear este tipo de conexiones, Mediasoup hace uso de ciertas herramientas y protocolos de WebRTC.

3.6.2. WebRTC

En mayo del 2010, Google compró *Global IP Solutions* |9|, una compañía de *software* que se dedicaba a desarrollar aplicaciones basadas en VoIP (*Voice over Internet Protocol*). Poco tiempo después, Google compartió como código abierto todas las tecnologías que habían sido desarrolladas por GIPS. Un año más tarde, creó un proyecto de código abierto basado en las tecnologías de GIPS, conocido como WebRTC, que ha sido respaldado por la W3C |10| y estandarizado a través de la IETF |7|.

WebRTC lo conforman un conjunto APIs y de protocolos de comunicación estandarizados, que permiten a los navegadores web comunicarse entre sí en tiempo real sin la necesidad de usar *plugins*. En la Figura 4 tenemos ordenados por capas los diferentes protocolos utilizados por WebRTC.

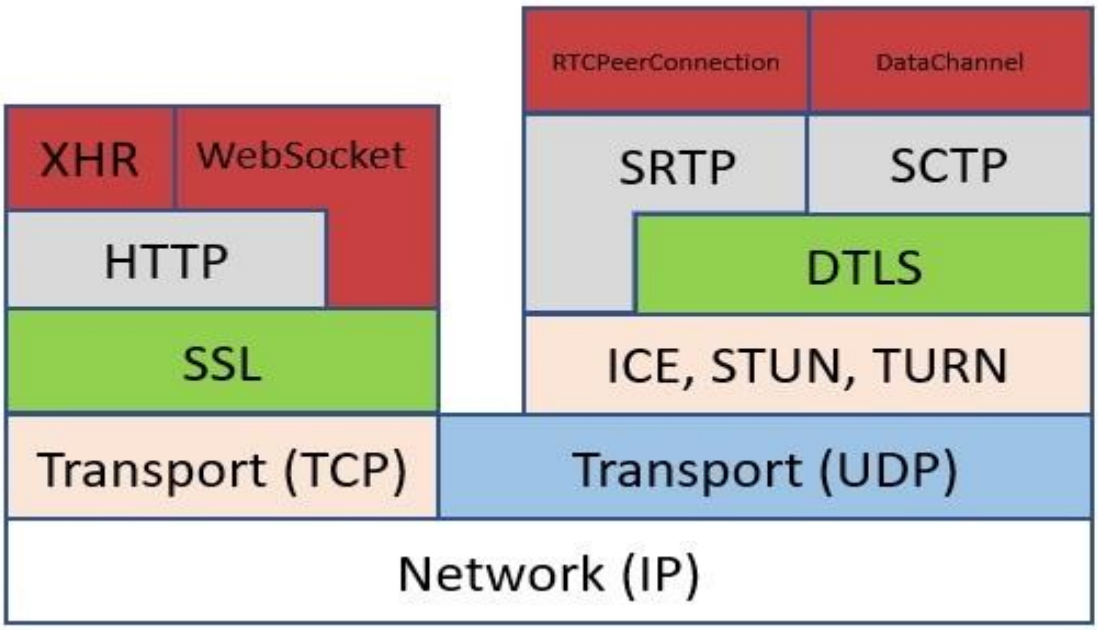


Figura 4. Protocolos de las distintas capas de WebRTC que usa Mediasoup.

RTCPeerConnection

RTCPeerConnection es un componente de WebRTC que permite manejar de un modo eficiente y estable la comunicación de datos continua (*streams*) entre *peers*, es decir, que se encarga de la tarea de conectar directamente dos navegadores. Antes de poder tener los *peers* conectados de este modo, es necesario intercambiar una serie de mensajes que contienen ciertos parámetros para permitir la comunicación. A dicho proceso se le conoce como señalización (*signaling*).

Signaling

Para conectar varios *peers* es necesario conocer la localización de red donde se encuentra cada uno de los usuarios. Ese tipo de información viene dada a través de la dirección IP de cada dispositivo. Una vez que los usuarios sepan localizarse a través de Internet, pueden comenzar a compartir datos a través del resto de protocolos de WebRTC.

En un mundo simple, dos ordenadores podrían realizar el proceso de señalización directamente, intercambiando la información necesaria para proceder con la conexión como se muestra en la Figura 5.

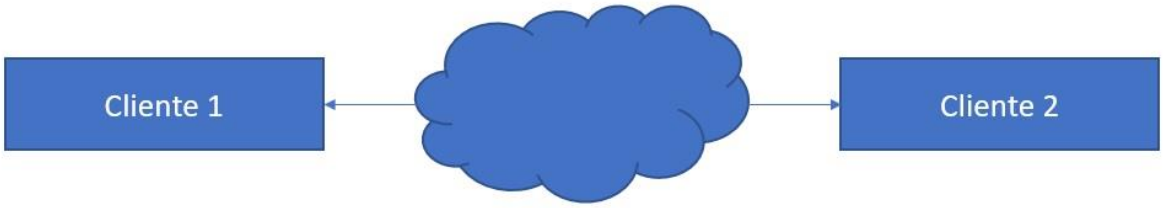


Figura 5. Señalización sin NATs.

Debido a que el número de máquinas conectadas a internet aumentó exponencialmente y con ello se veía demasiado cerca el momento en el que se agotarían las direcciones IPs de 32 bits, se creó el llamado NAT |12| (*Network Address Translation*). La idea detrás esto fue hacer que las redes de ordenadores utilizaran un rango de direcciones especiales, llamada IPs privadas y que se conectaran a internet a través de una única dirección, llamadas IPs públicas. Es por esto que dentro de las herramientas que forman WebRTC está ICE (*Interactive Connectivity Establishment*) para solventar los problemas que las conexiones de red actuales suponen.

ICE

ICE es un *framework* que permite a los navegadores web conectarse con los *Peers* asociados en un mundo en el que existen NATs y *firewalls*. Para conseguir esto, se apoya en dos técnicas llamadas STUN (*SessionTraversal Utilities for NAT*) y TURN (*Traversal Using Relays around Nat*). En primer lugar, ICE, mediante un servidor STUN intenta conectar los *peers* directamente haciendo uso de UDP (ver Figura 6). Si no es posible realizar la conexión, ICE intenta hacerlo mediante HTTP, que usa TCP. En caso de que este vuelva a fallar, normalmente porque uno de los peers está detrás de un NAT simétrico o un firewall, ICE hace uso de un servidor TURN (ver Figura 7).

STUN

STUN (*Session traversal Utilities for NAT*) lo forman un conjunto de métodos estandarizados, que permiten descubrir localización pública de una máquina y evitar así los problemas de conexión generados por la existencia de redes NATs o *firewalls*. Permite descubrir a un host que está detrás de un NAT, descubrir su dirección IP pública, el puerto y el tipo de NAT como se puede ver en la siguiente figura.

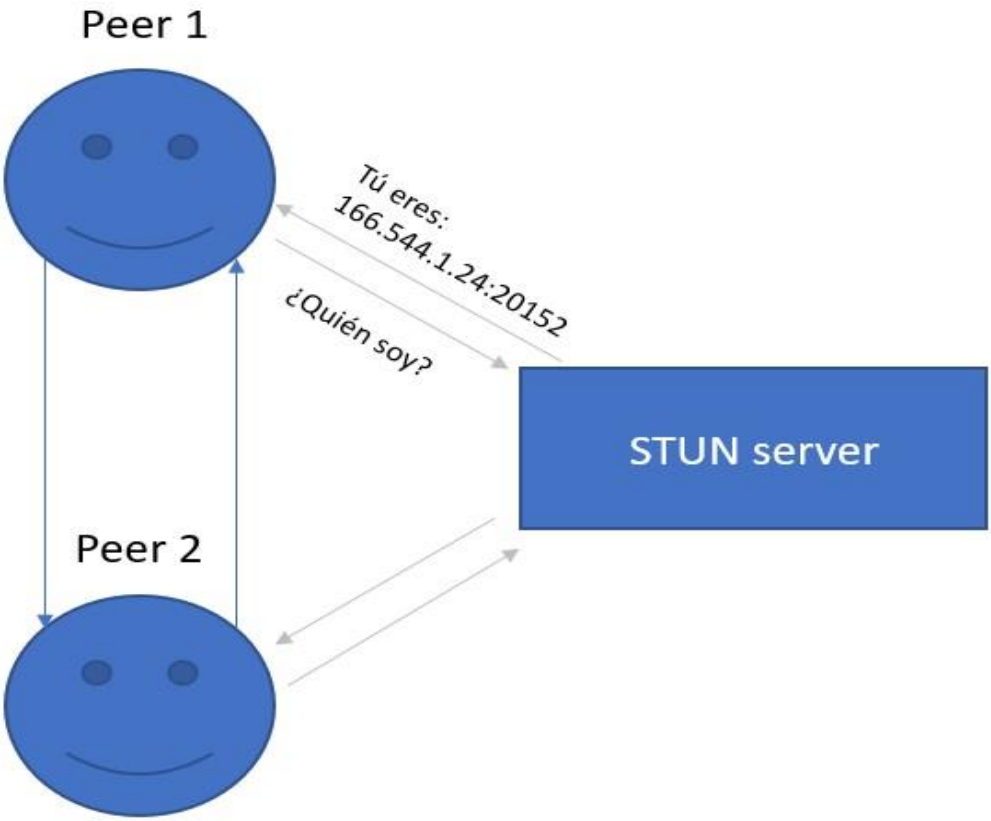


Figura 6. Funcionamiento de STUN en NATs no simétricas.

STUN funciona con los siguientes tipos de NAT:

- ***Full cone* NAT:** mapea dirección y puerto interno a dirección y puerto externo, lo que implica que cualquier paquete que se envíe a la dirección pública con el puerto especificado, será directamente redirigido a la dirección interna y el puerto interno especificados.

- ***Restricted cone* NAT:** igual al anterior, salvo que, en este caso solo se puede hacer llegar un paquete a la dirección interna si previamente se ha enviado un paquete al servidor desde la propia dirección interna. Para este caso no importa el puerto.

- ***Port restricted cone* NAT:** igual al anterior, pero en este caso sí importa el puerto.

TURN

Existe otro tipo de NAT llamada simétrica, que resulta ser la más difícil con la que trabajar. Esto se debe a que el mapeo entre IP y puerto privado contra IP y puerto público no se conserva, es decir, por cada requerimiento saliente se asigna un puerto aleatorio y este varía para cada comunicación. Debido a esto, cuando el host de destino intenta enviar un paquete a la dirección origen local, el *router* NAT rechaza el paquete porque no reconoce al host como autorizado. La parte positiva de este tipo de NATs es que varias máquinas internas pueden establecer comunicaciones, con el mismo puerto origen, al mismo tiempo.

Dada la naturaleza de las conexiones peer-to-peer, cuando un usuario se esconde detrás de una NAT simétrica tendrá que hacer uso de TURN (*Traversal Using Relays around* NAT). Este método funciona creando un servidor TURN al cual se van a conectar todos los usuarios que deseen intercambiar datos a través de WebRTC y que servirá como intermediario, recibiendo y distribuyendo todos los paquetes.

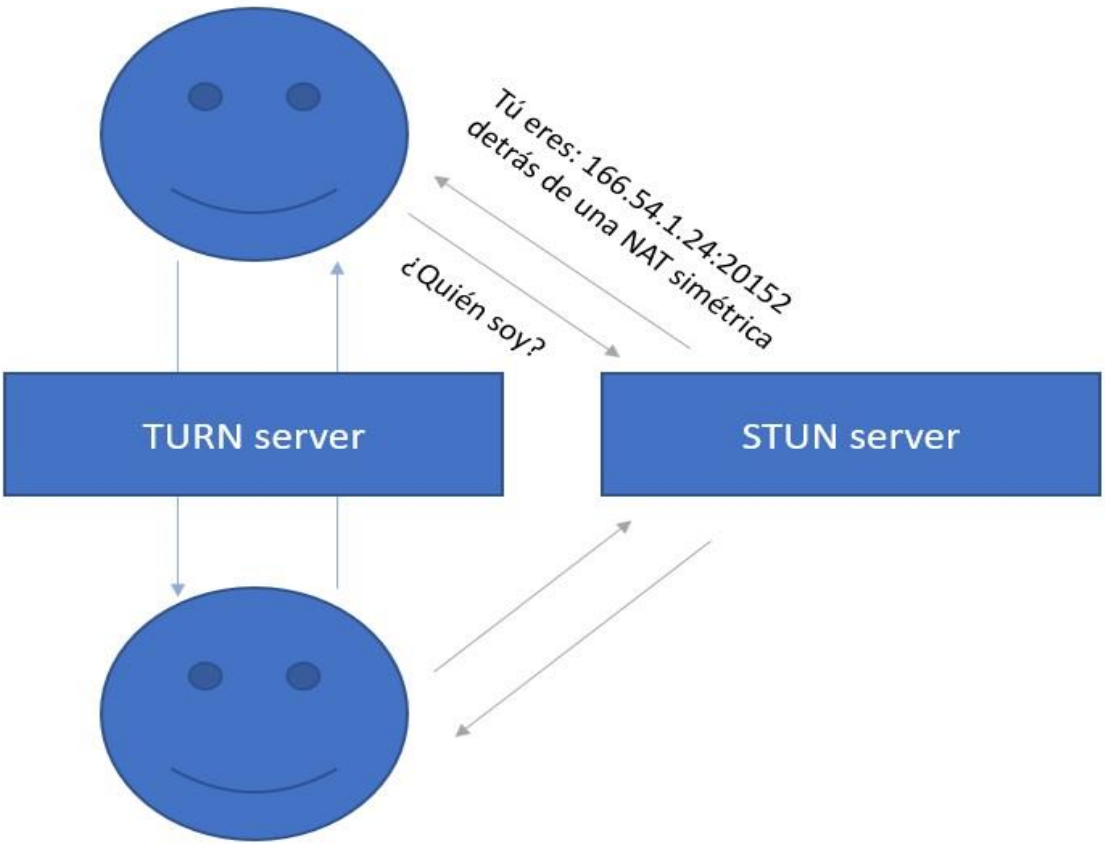


Figura 7. Funcionamiento de TURN con NATs simétricas.

DataChannels

La API DataChannel de WebRTC permite al navegador Web una comunicación de datos bidireccional directamente de un *peer* a otro. Entre los tipos de datos que soportan estos canales están USVString, Blob, ArrayBuffer y ArrayBufferView.

Estos canales pueden ser configurados para operar de un modo confiable (similar a TCP) o no confiable (similar a UDP). La posibilidad de elegir un modo u otro se consigue gracias al protocolo de comunicación que se usa para este tipo de transmisiones, llamado SCTP (*Stream Control Transmission Protocol*).

SCTP es un protocolo orientado a conexiones, que provee confiabilidad, control de flujo y secuenciación como TCP, aunque a diferencia de este, permite envío de datos fuera de orden. Además de esto, es un protocolo orientado a mensajes y no a datos binarios, similar a como funciona UDP. SCTP usa múltiples streams a diferencia de TCP, con esto se evita el problema de retraso en la transmisión debido a una pérdida de paquete o una secuencia de mensajes fuera de orden, algo que perjudicaría gravemente en las transmisiones en tiempo real de audio o vídeo.

Atributo	**TCP**	**UDP**	**SCTP**
Fiabilidad	Fiable	No fiable	Fiable
Gestión de conexiones	Orientado a conexión	Sin conexión	Orientado a conexión
Transmisión	Orientado a bytes	Orientado a mensaje	Orientado a mensaje
Control de flujo	Sí	No	Sí
Control de congestión	Sí	No	Sí
Tolerancia de errores	No	No	Sí
Entrega de datos	Estrictamente ordenada	Desordenada	Parcialmente ordenada
Seguridad	Sí	Sí	Mejorada

Tabla 1. Diferencias entre TCP, UDP y SCTP.

Todos los navegadores que tiene implementado WebRTC hacen uso de la encapsulación de mensajes para tener seguridad, para prevenir las intervenciones de otros usuarios, escuchas ilegales y la falsificación de mensajes. Para ello, WebRTC hace uso del protocolo DTLS, un derivado de SSL, que provee privacidad a los datos que se envían a través de la red con DataChannel.

3.6.3. Topologías WebRTC

En el mundo de los streaming multimedia a través de WebRTC para compartir datos de vídeo y audio en tiempo real existen diferentes topologías, como existen en cualquier tipo de conexiones de red. Estos modelos distribuyen las conexiones de varios modos, ofreciendo ciertas ventajas e inconvenientes.

Los tres tipos de modelos que pueden ser usados en conexiones WebRTC son los siguientes:

- ***FULL MESH*:** es la topología más simple, consiste en crear una conexión entre cada peer y el resto de peers de la sala. El número de conexiones que se crean en este tipo de tipologías es de $N * \frac{N-1}{2}$. Este modelo conlleva una implementación muy sencilla, no necesita de servidor que haga de repetidor y repartidos de datos y en caso de que una peer falle el resto sigue funcionando sin ningún problema. Destacar que es un método que tiene una muy baja latencia. Desafortunadamente, este modelo no es escalable.

- ***MCU*:** en el modelo MCU (*Multipoint Control Unit*) existe un servidor central que se encargará de recibir todos los datos de los peers conectados. Una vez en el servidor, todos los datos serán decodificados, re-escalados, codificados y enviados. La principal ventaja es que solo se tiene una conexión entrante y otra saliente por *peer*. La gran desventaja viene dada por la gran carga de trabajo que supone al servidor realizar toda la tarea de codificación y descodificación, pudiendo generar, además, una latencia añadida.

- **SFU:** en el modelo SFU (*Selective Forwarding Unit*) existe un servidor central que puede enviar los datos en diferentes calidades al resto de *peers* según las necesidades y capacidades de red que tenga cada cliente. Cada *peer* conectado tendrá que enviar los datos únicamente al servidor, pero en diferentes resoluciones, esto se consigue gracias a una herramienta llamada Simulcast. Las principales ventajas destacables son la reducción del número de conexiones, la posibilidad de elegir la calidad de vídeo de los datos enviados a cada peer y la baja carga de trabajo en el servidor y el cliente por no tener que codificar y decodificar la información como en el modelo MCU. Es un modelo óptimo en cuanto a escalado.

4. Estado del Arte

Existe multitud de aplicaciones para realizar videollamadas usando tecnologías de WebRTC. Muchas de estas plataformas se usan del mismo modo, se crea una sala, se aporta un enlace para compartir con el resto de usuarios con los que se quiere realizar una vídeo llamada, los demás participantes entran, todos comparten sus dispositivos y ya se puede disfrutar de una vídeo llamada múltiple.

El funcionamiento básico interno de las distintas plataformas es el siguiente:

1. Se crea un espacio único llamado sala en el cual se asegura que los canales de comunicaciones creados por los *peers* y los datos multimedia compartidos solo pueden llegar a los usuarios que allí se encuentren.

2. Un usuario ingresa en la sala que ha creado, siendo identificado como un *peer*. Comparte sus dispositivos multimedia para que se creen streams de audio y de vídeo.

3. Otros usuarios entran en la misma sala que ha sido creada con anterioridad y son identificados como *peers* únicos. Se crean canales de comunicación entre los posibles usuarios que se encuentren en la sala y los nuevos. Si alguno tenía un *stream* de datos activo, este es recibido por los nuevos usuarios conectados. Como ocurrió en el punto anterior, cada nuevo usuario comparte sus dispositivos multimedia para crear streams de datos que pueden ser enviados a través de los canales de comunicación que ya se hayan creado.

Aunque el funcionamiento de todas las plataformas de videollamadas que se pueden encontrar en internet sea muy similar, las capacidades y posibilidades de cada una de ellas son muy distintas dependiendo del tipo de arquitectura que tengan. Los principales fallos de las arquitecturas que conforman dichas plataformas son:

- Hacen uso de topologías *FULL MESH*, en las cuales se crean conexiones entre todos los *peers* consumiendo un alto ancho de banda, carga de CPU y con una capacidad de escalado nula. Este tipo de topologías puede comenzar a fallar a partir de 5 usuarios compartiendo vídeo llamada en la misma sala.

- Permiten enviar datos de vídeo en una sola calidad y el usuario final no tiene la opción de elegir la resolución según la capacidad de su conexión.

- Tienen incompatibilidad con ciertos navegadores. Debido al uso de diferentes códec de vídeo que tiene cada navegador o simplemente a las distintas funciones que pueden ser utilizadas en cada uno, se acaban creando plataformas con muy poca compatibilidad.

- Están diseñadas con protocolos de comunicación antiguo que no han sido renovados, que ofrecen menos posibilidades.

- No tienen en cuenta la seguridad.

- Siguen haciendo uso de compresores y codificadores de vídeo que tienen peor rendimiento y calidad, como, por ejemplo, el caso de uso de VP8 frente a VP9.

En la actualidad, existen empresas que disponen de plataformas que ofrecen servicios para poder realizar exposiciones gracias a sus sistemas con videollamadas y a otros componentes añadidos.

Entre ellas vamos a centrarnos en dos muy destacadas:

- **Appear.in:** se trata de un equipo de 17 personas que han creado una aplicación web simple e intuitiva para realizar videollamadas y compartir pantalla. Ofrecen una versión gratuita que permite un máximo de 4 usuarios por sala y una PRO que soporta 12 usuarios por sala por 9,99$ al mes. Entre sus principales desventajas está la capacidad de escalado, permitiendo un número máximo de usuarios simultáneos muy bajo y el no tener un desarrollo interno que permita la elección de la calidad de vídeo, debido a la no implementación de conexiones *multistreaming*.

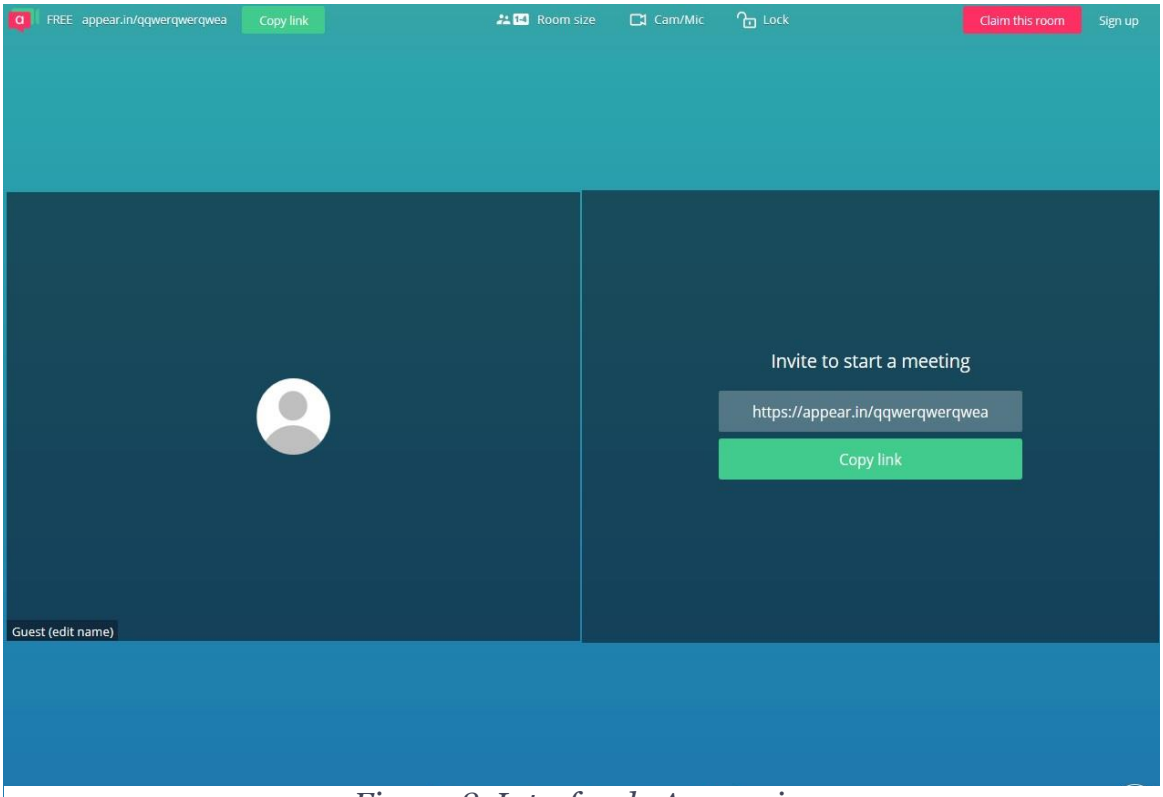


Figura 8. Interfaz de Appear.in.

- **Noysi.com:** se trata de una *startup* española que ofrece una herramienta que facilita la comunicación remota entre distintos usuarios, enfocada principalmente al campo empresarial. Entre los diferentes puntos a resaltar están su capacidad de escalado, permitiendo videollamadas de hasta 25 usuarios con una buena calidad.

Figura 9. Herramientas que ofrece Noysi.com.

Por un lado, tenemos soluciones que ofrece el mercado, como la de *Appear*, con un coste de desarrollo bajo, una cuota de uso reducida, pero con unas carencias más que notables, como son la capacidad de escalado y la calidad de los datos transmitidos. Por la otra parte, tenemos soluciones como la de *Noysi*, que mantiene un mayor y mejor rendimiento pero que desafortunadamente tiene un coste elevado en términos de desarrollo software y mantenimiento. Además de esto, las opciones que se encuentran siguen sin ajustarse del todo a lo que se busca.

La solución que se propone se distingue y mejora en los aspectos descritos a continuación:

- Coste de desarrollo bajo gracias al completo uso de software libre.
- Alta capacidad de escalado, por el uso del modelo SFU.
- Gran compatibilidad con los distintos tipos de dispositivos y navegadores.
- Mantiene una gran seguridad de comunicación usando los protocolos SSL y SRTP.

- Capacidad para compartir presentaciones en formato PDF de un modo muy sencillo.

- Tiene un mejor enfoque de cara al uso docente.

5. Especificación

Como en el desarrollo de cualquier nuevo proyecto, es de suma importancia, que antes de empezar a codificar los módulos que lo constituyen, se tenga una completa y plena comprensión de lo que el software necesita, la funcionalidad que debería tener, realizar una planificación temporal y una estimación de costes.

5.1. Análisis de requisitos

En este punto se van a obtener y definir cuáles son las necesidades que se deben satisfacer, antes de comenzar a desarrollar. Para ello, se realizará un estudio completo de las necesidades que deberá satisfacer la solución escogida finalmente, este estudio define una serie de funcionalidades. A éstas se les conoce como requisitos de un sistema y se dividen en dos tipos: requisitos funcionales y requisitos no funcionales.

Requisitos funcionales

Con los requisitos funcionales se pretende expresar como debe ser el funcionamiento y/o comportamiento del sistema frente a las interacciones con su entorno. Estos requerimientos van a depender de los posibles usuarios de la aplicación.

A continuación, se muestran una serie de tablas con la descripción de los requisitos funcionales más relevantes en este sistema:

REQ-1	Registro de usuario
Descripción	El módulo debe permitir el registro de nuevos usuarios que formarán parte del sistema.
Prioridad	Baja
Acciones iniciadoras	Un usuario nuevo introduce sus datos de registro.
Comportamiento esperado	Se crea un nuevo documento en la base de datos.
Requerimientos funcionales	-
Comportamiento ante errores	Se alerta de un fallo en los datos introducidos y no se crea ningún registro nuevo en la base de datos.

Tabla 2. Requerimiento funcional 1, registro de usuario.

REQ-2	Identificación de usuario
Descripción	El módulo debe permitir la identificación de usuarios ya registrados. Estos usuarios se pueden dividir en tres tipos según su rol en el sistema. Alumno, profesor y administrador.
Prioridad	Baja
Acciones iniciadoras	Un usuario introduce su nombre de usuario y contraseña.
Comportamiento esperado	El usuario pasa a una página que muestra sus salas asociadas.
Requerimientos funcionales	-
Comportamiento ante errores	Se alerta de un error en los datos introducidos y se mantiene en la misma página.

Tabla 3. Requerimiento funcional 2, identificación de usuario.

REQ-3	Mostrar salas
Descripción	El módulo debe mostrar todas las salas disponibles en función del rol y las materias de cada usuario.
Prioridad	Media
Acciones iniciadoras	Usuario se identifica en el sistema.
Comportamiento esperado	Se muestran todas las salas asociadas al usuario.
Requerimientos funcionales	[REQ-2]
Comportamiento ante errores	-

Tabla 4. Requerimiento funcional 3, mostrador de salas.

REQ-4	Acceso a dispositivos multimedia
Descripción	El módulo debe permitir el acceso a los dispositivos multimedia que el usuario tenga conectados para crear streams de datos.
Prioridad	Media
Acciones iniciadoras	Entrar a una sala.
Comportamiento esperado	Se pide acceso a los dispositivos multimedia del usuario. Micrófono por parte de alumnos. Cámara y micrófono por parte de profesores.
Requerimientos funcionales	[REQ-2]
Comportamiento ante errores	-

Tabla 5. Requerimiento funcional 4, acceso a dispositivos conectados.

REQ-5	Señalización
Descripción	El módulo debe crear la comunicación necesaria para compartir datos de conexión entre usuarios.
Prioridad	Alta
Acciones iniciadoras	Activación de dispositivos multimedia.
Comportamiento esperado	Se crean una serie de mensajes de señalización entre diferentes usuarios o el usuario y el servidor.
Requerimientos funcionales	[REQ-4]
Comportamiento ante errores	Se vuelve a intentar la señalización entre usuarios.

Tabla 6. Requerimiento funcional 5, señalización entre usuarios y servidor.

REQ-6	**Creación de productor**
Descripción	El módulo debe permitir crear un flujo de datos saliente a partir de un stream propio.
Prioridad	Alta
Acciones iniciadoras	Obtener el stream de datos local.
Comportamiento esperado	Se crea un productor que enviará el flujo de datos multimedia una vez se haya completado la conexión.
Requerimientos funcionales	[REQ-5]
Comportamiento ante errores	-

Tabla 7. Requerimiento funcional 6, crear productor de datos.

REQ-7	**Creación de consumidor**
Descripción	El módulo debe permitir crear flujos de datos entrantes a partir de flujo de datos externos.
Prioridad	Alta
Acciones iniciadoras	Otros usuarios crean productores y se ha establecido una conexión.
Comportamiento esperado	Se recibe un consumidor con el contenido multimedia de otros usuarios para ser reproducido localmente.
Requerimientos funcionales	[REQ-6]
Comportamiento ante errores	-

Tabla 8. Requerimiento funcional 7, crear consumidor de datos.

REQ-8	**Creación de transporte**
Descripción	El módulo debe crear una conexión permanente entre usuarios para permitir el transporte de datos entre usuarios.
Prioridad	Alta

Acciones iniciadoras	Otros usuarios entran en la sala.
Comportamiento esperado	Se crea una conexión permanente asociada a cada usuario.
Requerimientos funcionales	[REQ-5]
Comportamiento ante errores	Se muestra un mensaje de error de establecimiento de conexión y el motivo.

Tabla 9. Requerimiento funcional 8, crear transporte entre usuarios.

REQ-9	**Reproducción de contenido**
Descripción	El módulo debe reproducir todo el contenido multimedia recibido de los consumidores de otros usuarios.
Prioridad	Alta
Acciones iniciadoras	Se tiene un nuevo consumidor con stream multimedia.
Comportamiento esperado	Se crea un nuevo evento para reproducir el contenido multimedia.
Requerimientos funcionales	[REQ-6, REQ-8]
Comportamiento ante errores	-

Tabla 10. Requerimiento funcional 9, reproducción de contenido multimedia.

REQ-10	**Inclusión de presentaciones**
Descripción	El módulo debe permitir la adición de presentaciones en formato PDF, añadirlo a la pantalla principal y compartirlo con el resto de usuarios de la sala.
Prioridad	Alta
Acciones iniciadoras	Un usuario, de tipo profesor, selecciona una presentación en formato PDF que se encuentre en su disco duro local.
Comportamiento esperado	Se crea una nueva ventana en la pantalla principal para visualizar el contenido de la presentación.
Requerimientos funcionales	[REQ-2]
Comportamiento ante errores	Se mantiene un rótulo avisando de que no se ha cargado presentación.

Tabla 11. Requerimiento funcional 10, inclusión de presentaciones.

REQ-11	**Chat**
Descripción	El módulo debe permitir el intercambio de mensajes de texto entre los usuarios conectados.
Prioridad	Baja
Acciones iniciadoras	Usuarios se conectan a la misma sala.

Comportamiento esperado	Se pueden enviar y recibir mensajes de texto.
Requerimientos funcionales	[REQ-2]
Comportamiento ante errores	-

Tabla 12. Requerimiento funcional 11, chat para miembros de sala.

REQ-12	**Tabla usuarios y salas**
Descripción	El módulo debe permitir mostrar una lista con todo el contenido de la base de datos, los usuarios, las salas y las relaciones que la forman.
Prioridad	Alta
Acciones iniciadoras	Un usuario de tipo administrador ingresa en la plataforma.
Comportamiento esperado	Se crea una lista que muestra todos los usuarios y salas que están registrados en la base de datos.
Requerimientos funcionales	[REQ-2] con restricción de que debe ser de tipo administrador.
Comportamiento ante errores	Se muestra mensaje.

Tabla 13. Requerimiento funcional 12, R para la base de datos.

REQ-13	**Administración de contenido**
Descripción	El módulo debe permitir a un administrador leer, actualizar, crear y borrar contenido de la base de datos desde el propio sistema.
Prioridad	Alta
Acciones iniciadoras	Un usuario de tipo administrador ingresa en la plataforma.
Comportamiento esperado	Se permiten una serie de acciones ingresadas desde el sistema que se vuelcan a la base de datos.
Requerimientos funcionales	[REQ-12]
Comportamiento ante errores	Se muestra mensaje.

Tabla 14. Requerimiento funcional 1, CUD para la base de datos.

Requisitos no funcionales

A continuación, se muestran una serie de tablas con la descripción de los requisitos funcionales más relevantes en este sistema:

REQNF-1	**Diseño *responsive***
Descripción	La interfaz de usuario debe tener un diseño responsive adaptada a las distintas resoluciones.

Tabla 15. Requisito no funcional 1 – responsive.

REQNF-2	**Facilidad de uso**
Descripción	Debe ser un sistema intuitivo y simple para todo tipo de usuarios.

Tabla 16. Requisito no funcional 2 – sencillo.

REQNF-3	**Modular**
Descripción	Debe dividirse en módulos sencillos que puedan ser modificados fácilmente.

Tabla 17. Requisito no funcional 3 – modular.

	Escalabilidad
Descripción	Debe soportar una carga de conexiones y datos alta.

Tabla 18. Requisito no funcional 4 – escalable.

REQNF-5	**Eficiencia**
Descripción	Debe tener un procesado de peticiones rápido y con unos tiempos de espera bajos.

Tabla 19. Requisito no funcional 5 – eficiente.

REQNF-6	**Compatibilidad**
Descripción	Debe ser compatible con el mayor número de plataformas posible.

Tabla 20. Requisito no funcional 6 – compatible.

5.2. Estimación de costes

Todavía existe un paso más que se debe dar antes de comenzar con el desarrollo del proyecto. Este paso trata de informar al cliente sobre los costes tanto temporales como económicos. Una vez se han realizado entrevistas con los clientes, se ha concretado el alcance del proyecto y los objetivos que debe cumplir es momento de llevar a cabo una serie de gestiones que tendrán como objetivo el estudio de la viabilidad del proyecto. Dichas gestiones tratan de la gestión temporal y económica del mismo.

Coste temporal

Es importante realizar una buena estimación de costes para resolver problemas asociados al esfuerzo y tiempo invertidos en el desarrollo, además de para mantener al cliente informado sobre cuánto debe esperar para obtener su producto.

Existen muchos modelos que sirven para llevar a cabo una estimación de costes según una serie de factores. Entre dichos modelos hay unos que son ideales porque se obtienen unas estimaciones muy aproximadas. Estos modelos son conocidos como COCOMO (*COnstructive COst MOdel*). Para nuestro caso concreto vamos a hacer uso de su versión más actual conocida como COCOMO II o COCOMO 2000.

Dentro de COCOMO II existen diferentes modelos que se usarán en función del tipo y cantidad de información disponible, concretamente para nuestro sistema objetivo, el más adecuado es el de diseño temprano, debido a que se han definido los requerimientos funcionales en puntos anteriores y nos encontramos en las primeras etapas del desarrollo.

La fórmula para el cálculo del esfuerzo estimado viene dada por la siguiente ecuación:

$$Esfuerzo = A * (KSLOC^b) * M$$

Donde:

- **Esfuerzo** es la estimación del coste temporal en meses por cada persona.
- **A** es una constante que captura los efectos lineales sobre el esfuerzo de acuerdo a la variación del tamaño (A=2,94).
- **KSLOC** es el tamaño del software expresado en miles de líneas de código. Se calcula en base a los puntos de función no ajustado que veremos más adelante.
- **B** refleja el esfuerzo creciente requerido para incrementar el tamaño del proyecto. Varía entre 1,1 y 1,24 dependiendo de la novedad del proyecto, la flexibilidad del desarrollo, los procesos para la resolución de riesgos, la cohesión del equipo de desarrollo y el nivel de madurez del proceso en la organización.
- **M** esfuerzo nominal de desarrollo. Deben escogerse 7 de los 17 factores para este modelo.

Para calcular los KSLOC tenemos que hacer uso de una tabla de puntos de función no ajustados que nos darán unos valores que indican el tamaño de la funcionalidad y

que tendremos que convertir a KSLOC en función del lenguaje utilizado, en este caso JavaScript.

En primer lugar, tenemos que calcular los componentes involucrados en nuestro sistema en función de las entradas, las salidas, las interacciones, las interfaces externas y los archivos internos.

EI (entradas)	Procesos en los que se introducen datos y que suponen la actualización de cualquier archivo interno.
EO (salidas)	Procesos en los que se envía datos al exterior de la aplicación.
EQ(interacciones)	Procesos consistentes en la combinación de una entrada y una salida, en el que la entrada no produce ningún cambio en ningún archivo y la salida no contiene información derivada.
ILF (archivo)	Grupos de datos relacionados entre sí internos al sistema.
EIF (interfaces)	Grupos de datos que se mantienen externamente.

Tabla 21. Funciones de tipo dato y de tipo transacción para PFNA.

Categoría	Cantidad
Entradas	5
Salidas	4
Interacciones	0
Interfaces Externas	2
Archivos internos	0

Tabla 22. Cantidad de funciones por tipo del sistema a desarrollar.

A continuación, presentamos una tabla con las traducciones a funcionalidades del sistema de cada una de las categorías:

EI 1	Proceso de registro
EI 2	Proceso de identificación
EI 3	Actualización de datos de usuario
EI 4	Eliminar usuario
EI 5	Crear usuario
EO 1	Listar usuarios
EO 2	Listar salas
EO 3	Listar conectados
EO 4	Presentación
ILF 1	Tabla de usuarios
ILF 2	Tabla de salas

Tabla 23. Elementos de los procesos del sistema.

Para calcular los puntos de función no ajustados tenemos que ponderar el valor de cada categoría haciendo uso de la tabla 24. Una vez obtenido el sumatorio final de todos

los valores de los PFNA, podremos traducir a líneas de código según el lenguaje, en la tabla 25 se muestra una equivalencia de líneas de código por punto de función. El resultado que obtenemos como líneas de código estimadas para el sistema según el total de puntos de función que se han obtenido y que se pueden ver en la tabla 26 es:

$$KSLOC = 47 * 55 = 2585$$

PUNTOS DE FUNCIÓN SIN AJUSTAR (PFNA)				
Tipo	Descripción	Complejidad		
		Simple	Media	Compleja
EI	Entradas Externas	3	4	6
EO	Salidas Externas	4	5	7
EQ	Consultas Externas	3	4	6
LIF	Ficheros Internos	4	10	15
EIF	Interfaces Externas	5	7	10

Tabla 24. Tabla de valores de puntos de función sin ajustar.

Lenguaje	SLOC/FP			
--	Avg	Med	Mín	Max
ASP	56	50	32	106
Assembler	209	203	91	320
C	148	107	22	704
C++	59	53	20	178
C#	58	59	51	66
FoxPro	36	35	34	38
J2EE	57	50	50	67
Java	55	53	9	214
JavaScript	54	55	45	63
JSP	59	-	-	-
.Net	60	60	60	60
Perl	57	57	45	60

Tabla 25. Equivalencia de líneas de código por cada punto de función.

En la tabla 26, le damos un peso a cada proceso y lo sumamos para tener el PFNA del sistema a desarrollar.

Proceso	Complejidad	Valor
Proceso de registro	Baja	3
Proceso de identificación	Baja	3
Actualización de datos de usuario	Alta	6
Eliminar usuario	Baja	3
Crear usuario	Media	4
Listar usuarios	Baja	4
Listar salas	Baja	4

Listar conectados	Media	5
Presentación	Alta	7
Tabla de usuarios	Baja	4
Tabla de salas	Baja	4
Total	-	47

Tabla 26. Puntos de función por cada proceso.

Factor	Muy bajo	Bajo	Medio	Alto	Muy alto	Máximo
AEXP	1.12	1.10	1.00	0.88	0.81	-
LTEX	1.20	1.09	1.00	0.91	0.81	-
PEXP	1.19	1.09	1.00	0.91	0.85	-
SCED	1.43	1.14	1.00	1.00	1.00	-
CPLX	0.73	0.87	1.00	1.17	1.34	1.74
TOOL	1.17	1.09	1.00	0.9	0.78	-
RUSE	-	0.95	1.00	1.07	1.15	1.24

Tabla 27. Multiplicadores de requisitos de esfuerzo nominal.

Factor	Descripción
AEXP	Experiencia en la aplicación
LTEX	Experiencia en el lenguaje
PEXP	Experiencia en la plataforma
SCED	Restricciones de tiempo
CPLX	Complejidad del producto
TOOL	Herramientas externas
RUSE	Requerimiento de reusabilidad

Tabla 28. Descripción de factores de esfuerzo nominal.

Factor	Nivel para el proyecto	Valor asociado
AEXP	Bajo	1.10
LTEX	Medio	1.00
PEXP	Bajo	1.09
SCED	Alto	1.00
CPLX	Medio	1.00
TOOL	Alto	0.9
RUSE	Bajo	0.95

Tabla 29. Valores asociados a los esfuerzos para el sistema a desarrollar.

Como último paso queda calcular el factor multiplicador M que mide el esfuerzo nominal del desarrollo según 7 factores que podemos ver en la *tabla 28*. Dichos valores se ponderan según el nivel de cada factor en el proyecto que se pretende desarrollar como se indica en la *tabla 29*, para ello se hace uso de los pesos asociados indicados en

la *tabla 27*. Solo queda aplicar dichos valores a la fórmula del esfuerzo que mide el coste en meses por persona, obteniendo como resultado:

$$A = 2{,}94$$

$$KSLOC = 2{,}585$$

$$B = 1{,}1$$

$$M = 1{,}10 * 1{,}00 * 1{,}09 * 1{,}00 * 1{,}00 * 0{,}9 * 0{,}95 = \ 1.025$$

$$Esfuerzo = 2{,}94 * (2{,}585)^{1{,}1} * 1.025 = 8{,}566\ meses$$

Para valorar el esfuerzo en número de horas debemos pensar en que un programador tiene entre 4 y 5 horas efectivas trabajando a jornada completa, esto implica que, al mes, quitando que se libra sábados, domingos y festivo, nos da como resultado 94,5 horas efectivas al mes que multiplicándolo por el número de meses que hemos estimado en el punto anterior, tenemos un número de horas totales de:

$$\mathbf{Esfuerzo_{horas} = 94{,}5 * 8.566 = 809{,}487\ horas}$$

Coste económico

Una vez se ha estimado el coste en tiempo, según el número de meses y/u horas, es muy sencillo calcular el coste en términos económicos. Vamos a calcularlo de dos formas diferentes, por un lado, según el sueldo mensual de un trabajador en este lenguaje y con las características necesarias para poder implementarlo, por otro lado, por un trabajador *freelance* que cobre por horas.

1. Perspectiva de empresa. En la tabla 30 tenemos el sueldo aproximado de trabajadores por lenguaje de programación según un artículo de SkyLab. En el caso de nuestro sistema, el 75% está hecho con JavaScript (*frontend*) y el 25% con NodeJS (*backend*). Por tanto, ponderando la carga de este proyecto con los sueldos de la tabla 30 tenemos:

$$Coste\frac{medio}{anual} = 0{,}75 * 25.000 + 0{,}25 * 27.000 = \mathbf{25.500€}$$

$$Coste\frac{medio}{mensual} = \frac{25.500}{12} = \mathbf{2.125€}$$

$$Coste_{final} = 2.125 * \ 8{,}566_{neses} = \mathbf{18.202{,}75€}$$

2. Perspectiva de freelance. En la tabla 31 tenemos el coste medio por hora de los trabajadores por cuenta propia según un estudio de Payoneer. En este caso, es obvio que el sistema debe ser más barato, porque se paga según horas efectivas y no meses de trabajo, como ya vimos en la estimación del esfuerzo.

$$Coste = 16{,}3 * 809{,}487 = \mathbf{13.194{,}6}\ €$$

Lenguaje	Salario
Java	20.000€
Javascript	25.000€
PHP	30.000€
C#	40.000€
C	40.000€

Tabla 30. Salarios según una publicación de SkyLab |13|.

Área	€/hora
Programación aplicaciones móviles	17,14 €
Programación web	16,30 €
Desarrollador backend	17,14 €
Programación de juegos	18,86 €
Tester /QA	15,43 €

Tabla 31. Coste en hora programadores de diferentes áreas según MCLENIT |14|.

5.3. Planificación

MÉTRICA Versión 3 es una metodología orientada a la planificación, el desarrollo y el mantenimiento de los sistemas de información promovida por el Ministerio de Hacienda del Gobierno de España para la sistematización de actividades del ciclo de vida de los proyectos software. Usaremos los procesos y actividades adaptando la metodología al tamaño y las condiciones del proyecto que se va a desarrollar.

Procesos y actividades de los dos puntos a usar en el sistema:

- Planificación de sistemas de información
    1. Análisis de las necesidades
    2. Determinación de responsables
    3. Especificación del ámbito y alcance
    4. Definición del plan de trabajo
    5. Catalogación de requisitos
    6. Análisis de los sistemas de información actuales
    7. Selección de la arquitectura tecnológica

8. Definición de proyecto a realizar

- Desarrollo de sistemas de información
    1. Identificación de requisitos de diseño
    2. Especificación del entorno tecnológico
    3. Identificación de mecanismos genéricos de diseño
    4. Identificación de clases asociadas a un caso de uso
    5. Diseño de la realización de los casos de uso
    6. Revisión de la interfaz de usuario
    7. Diseño de clases
    8. Diseño de módulos del sistema
    9. Diseño de comunicación entre módulos
    10. Revisión de la interfaz de usuario
    11. Diseño del modelo físico de datos
    12. Verificación de las especificaciones de diseño
    13. Aceptación de la arquitectura del sistema
    14. Especificación del entorno de pruebas
    15. Especificación de requisitos de documentación de usuario
    16. Presentación y aprobación del diseño del sistema

En las siguientes imágenes, se muestra la planificación de las tareas mediante un diagrama de Gantt:

Proyecto básico con Gantt y dependencias

	Listo	Nombre de la tarea	Fecha de Inicio	Fecha final	Asignado a	% Cumplimiento	Duración	Predecesores
1	☐	**Planificación de sistemas de información**	**28/09/17**	**27/10/17**			**22d**	
2	☐	Análisis de necesidades	28/09/17	02/10/17			3d	
3	☐	Determinación de responsables	03/10/17	04/10/17			2d	
4	☐	Especificación del ámbito y alcance	05/10/17	06/10/17			2d	
5	☐	Definición del plan de trabajo	09/10/17	11/10/17			3d	
6	☐	Catalogación de requisitos	12/10/17	16/10/17			3d	
7	☐	Análisis de los sistemas de información actuales	17/10/17	19/10/17			3d	
8	☐	Selección de la arquitectura tecnológica	20/10/17	20/10/17			1d	
9	☐	Definición de proyecto a realizar	23/10/17	27/10/17			5d	
10	☐							
11	☐	**Desarrollo de sistemas de información**	30/10/17	09/02/18			75d	
12	☐	Identificación de requisitos de diseño	30/10/17	02/11/17			4d	
13	☐	Especificación del entorno tecnológico	03/11/17	08/11/17			4d	
14	☐	Identificación de mecanismos genéricos de diseño	09/11/17	10/11/17			2d	
15	☐	Identificación de clases asociadas a un caso de uso	13/11/17	15/11/17			3d	
16	☐	Diseño de la realización de los casos de uso	16/11/17	20/11/17			3d	
17	☐	Revisión de la interfaz de usuario	21/11/17	24/11/17			4d	
18	☐	Diseño de clases	27/11/17	19/12/17			17d	
19	☐	Diseño de módulos del sistema	20/12/17	05/01/18			13d	
20	☐	Diseño de comunicación entre módulos	08/01/18	09/01/18			2d	
21	☐	Revisión de la interfaz de usuario	10/01/18	10/01/18			1d	
22	☐	Diseño del modelo físico de datos	11/01/18	16/01/18			4d	
23	☐	Verificación de las especificaciones de diseño	17/01/18	19/01/18			3d	
24	☐	Aceptación de la arquitectura del sistema	22/01/18	23/01/18			2d	
25	☐	Especificación del entorno de pruebas	24/01/18	29/01/18			4d	
26	☐	Especificación de requisitos de documentación de usuario	30/01/18	05/02/18			5d	
27	☐	Presentación y aprobación del diseño del sistema	06/02/18	09/02/18			4d	

Figura 10. Lista de tareas de planificación del desarrollo.

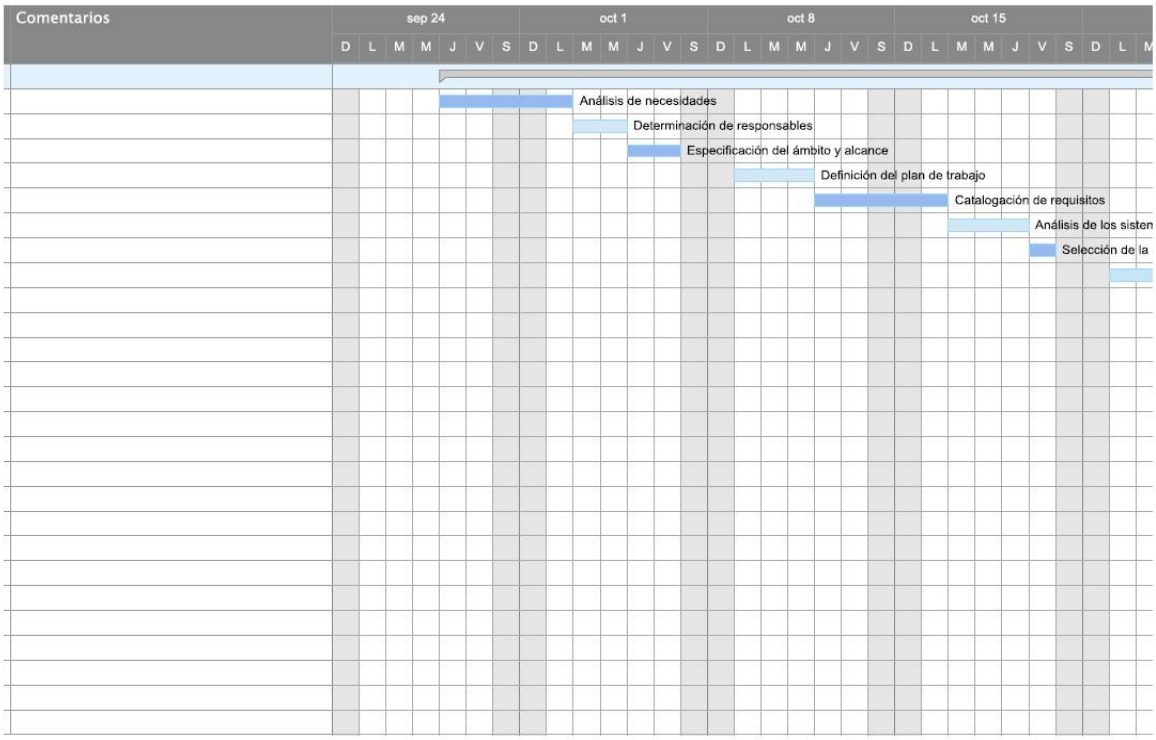


Figura 11. Primer cronograma de planificación temporal.

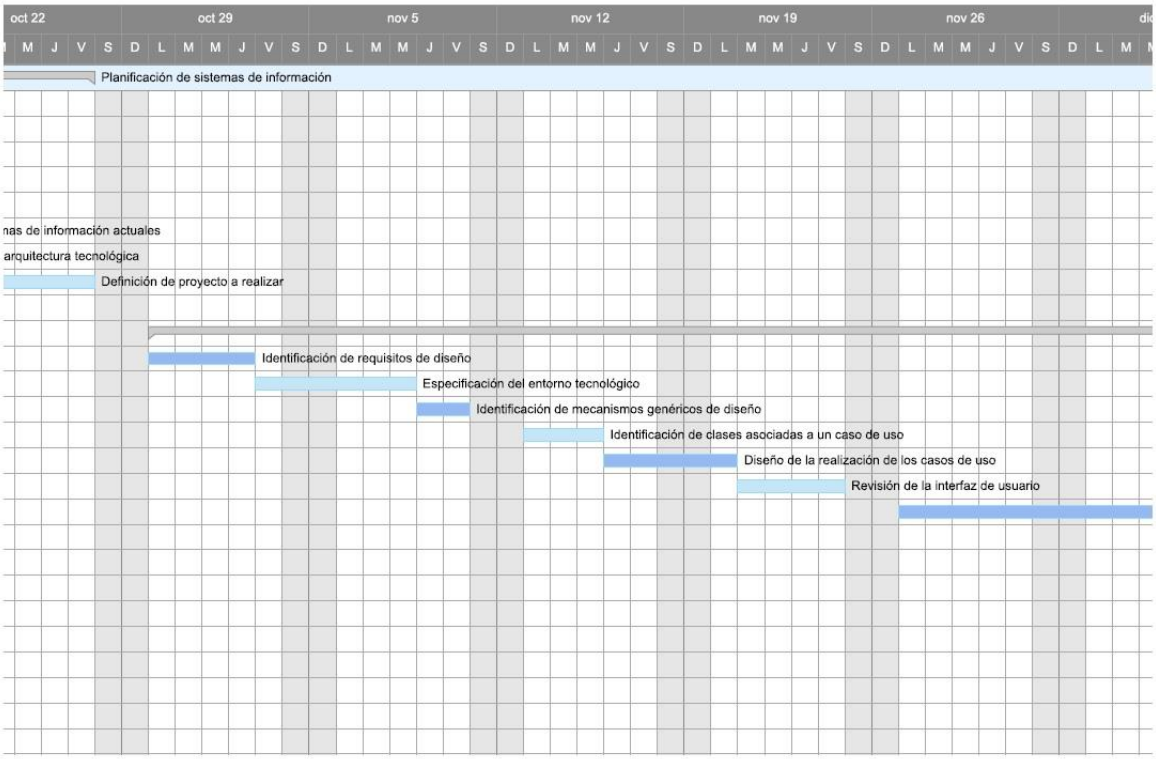


Figura 12. Segundo cronograma de planificación temporal.

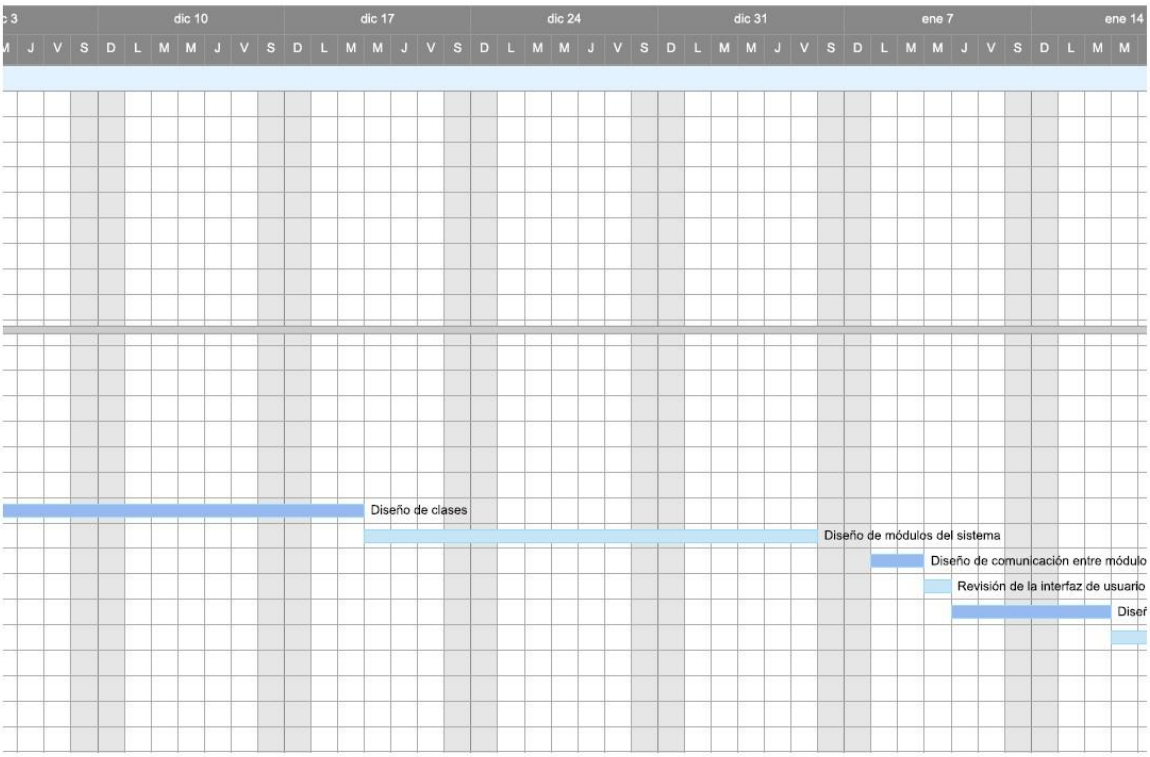


Figura 13. Tercer cronograma de planificación temporal.

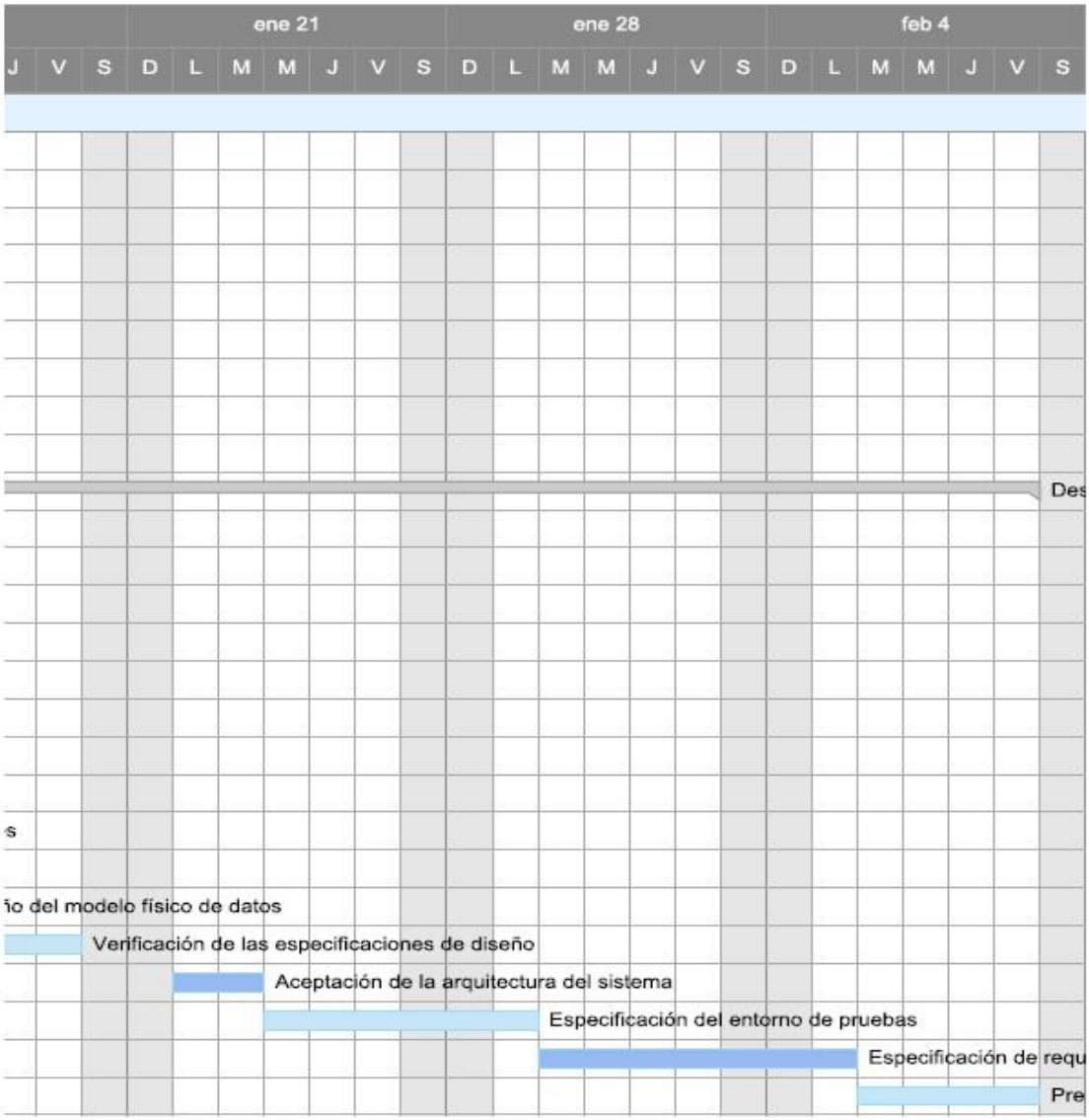


Figura 14. Cuarto cronograma de planificación temporal.

6. Desarrollo

Una vez se ha realizado el análisis de requisitos y tenemos una idea general de lo que el sistema pretende y necesita, es hora de comenzar la fase de desarrollo. En ella, se pretende tener un diseño que especifique como debe ser el sistema, para luego proceder a realizar la programación de las distintas entidades. En primer lugar, se utilizarán los casos de uso para definir la funcionalidad general del sistema. En segundo lugar, se transformarán las entidades definidas en análisis de requisitos a través de los diagramas de clases. En tercer lugar, se usarán diagramas de secuencia

para tener una visión clara de las interacciones temporales de los objetos. En cuarto lugar, se crearán bocetos que sirvan de idea para la interfaz. Y, por último, se procederá a trasladar la información de los puntos anteriores a líneas de código.

6.1. Casos de uso

Con el diseño de los casos de uso se pretende describir las acciones del sistema desde el punto de vista del usuario, eso nos dará una visión de los requerimientos del sistema desde su punto de vista.

A continuación, se muestran los diagramas más relevantes del sistema.

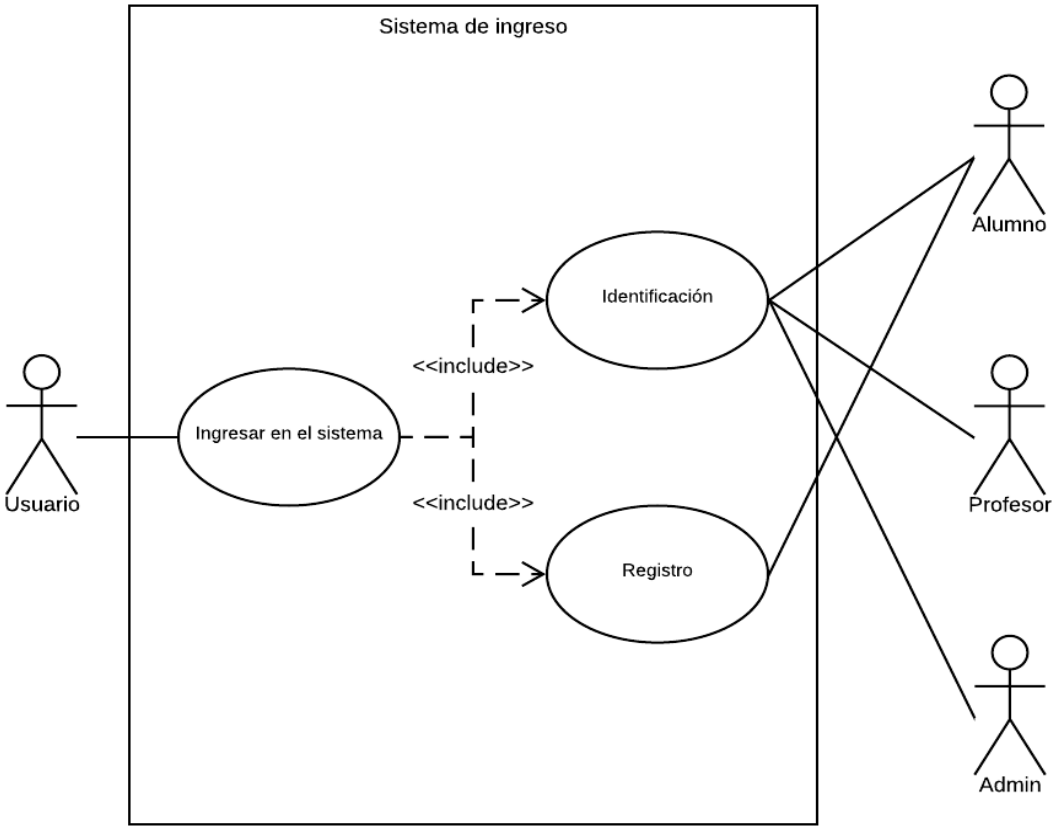


Diagrama 1. Sistema de ingreso.

El usuario puede entrar a la aplicación web registrándose como un alumno o identificándose si ya ha sido registrado por algún administrador

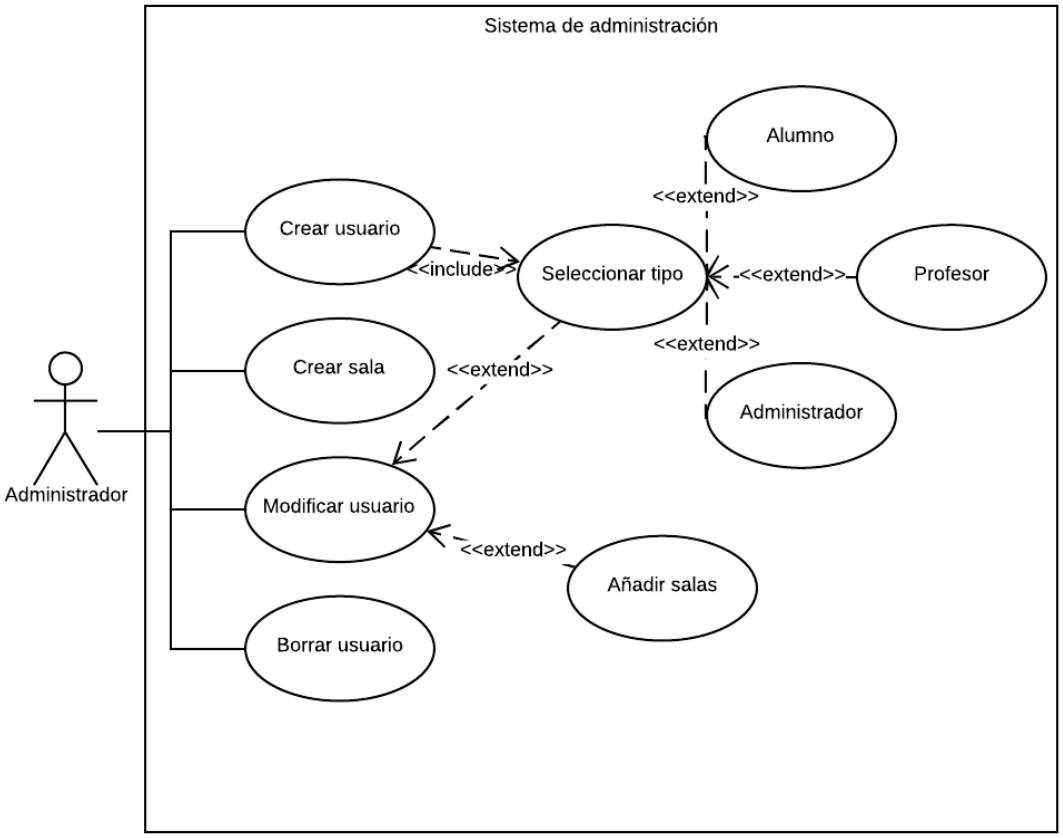


Diagrama 2. Sistema de administración.

Un administrador puede usar el sistema para crear usuarios, modificarlos, borrarlos o crear salas nuevas y añadirlas a estos.

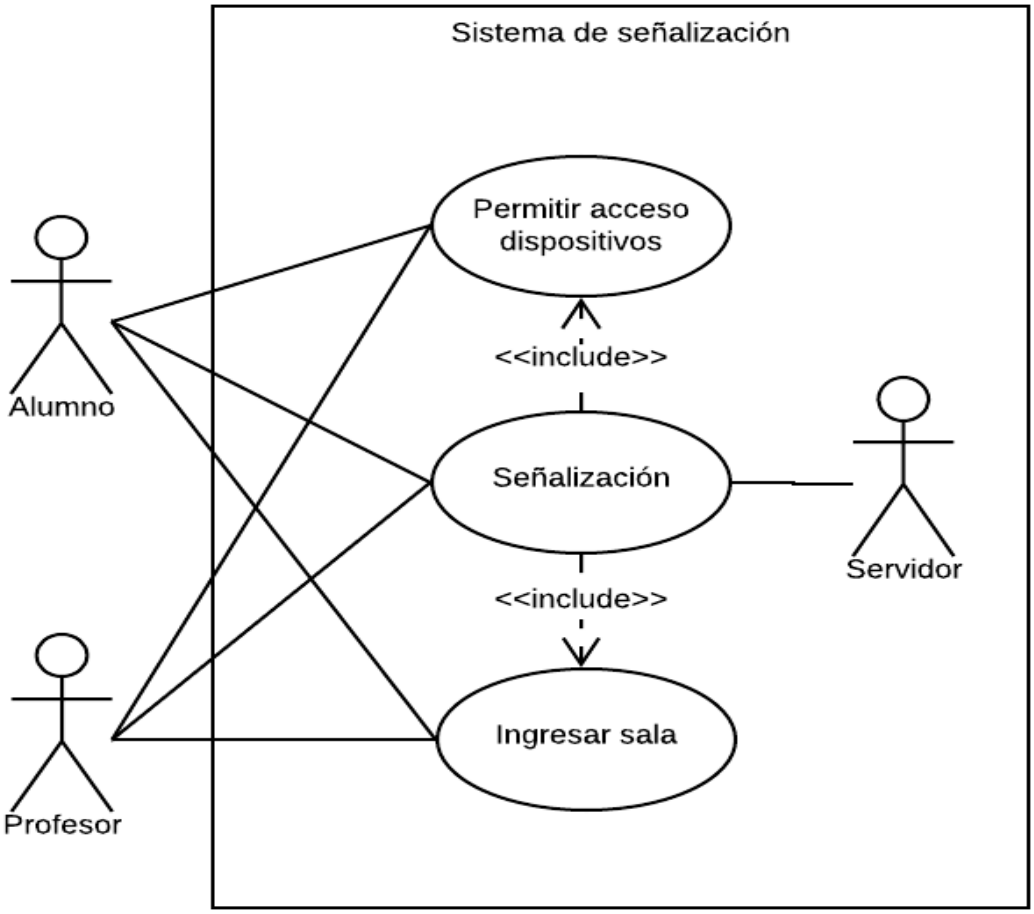


Diagrama 3. Sistema de señalización.

Los usuarios del sistema pueden y deben ingresar a sala y permitir el acceso a dispositivos para comenzar el proceso de señalización.

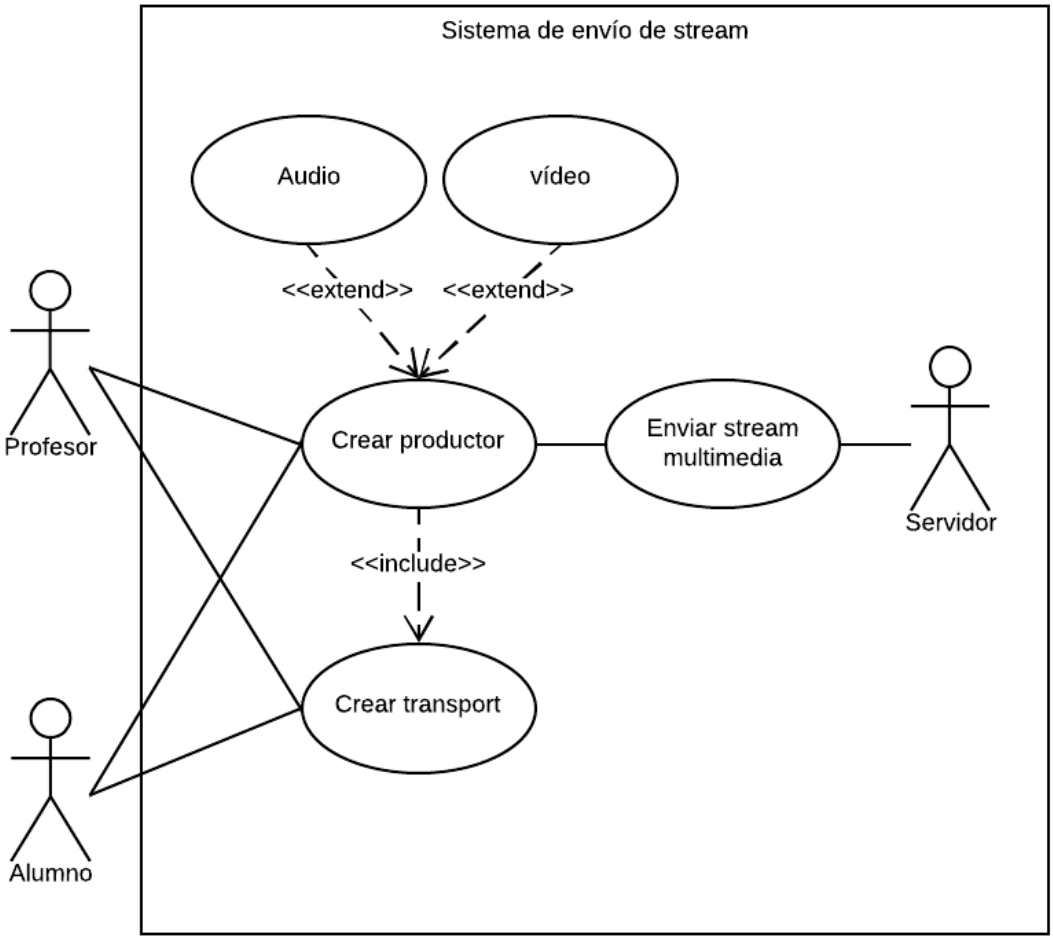


Diagrama 4. Sistema de envío de stream.

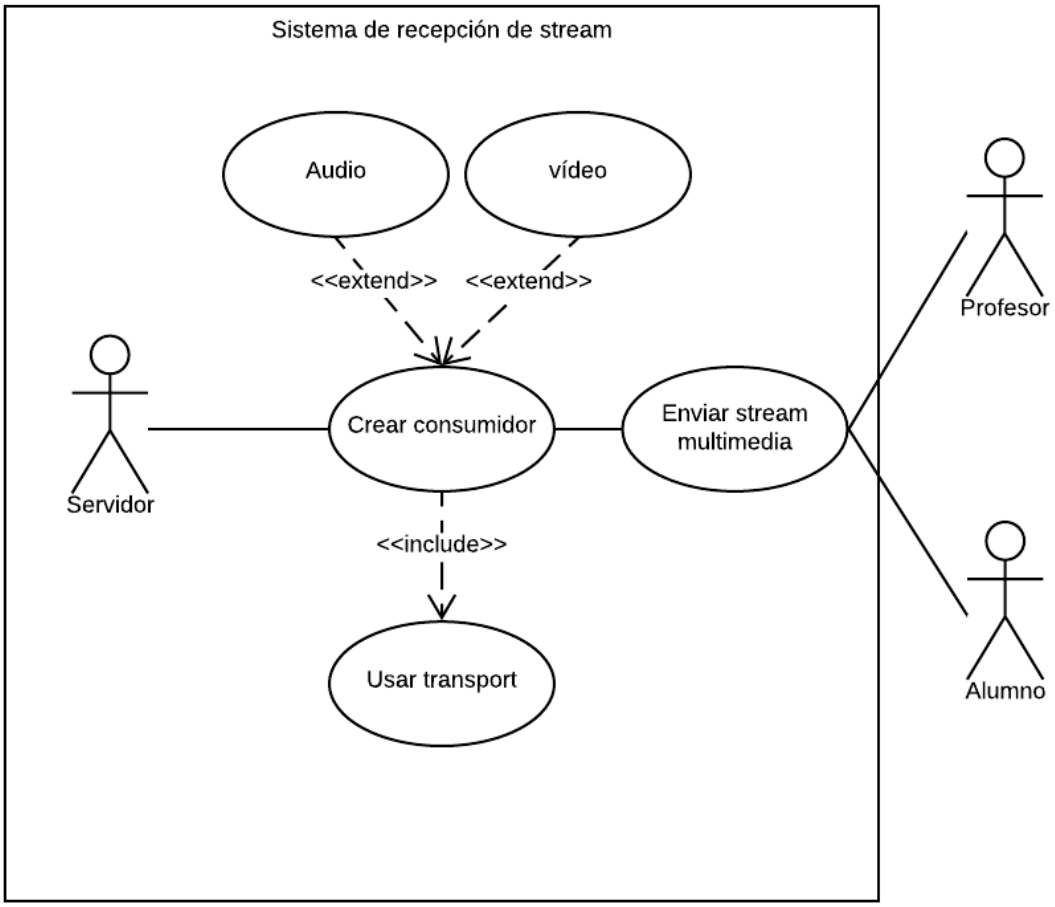


Diagrama 5. Sistema de recepción de stream.

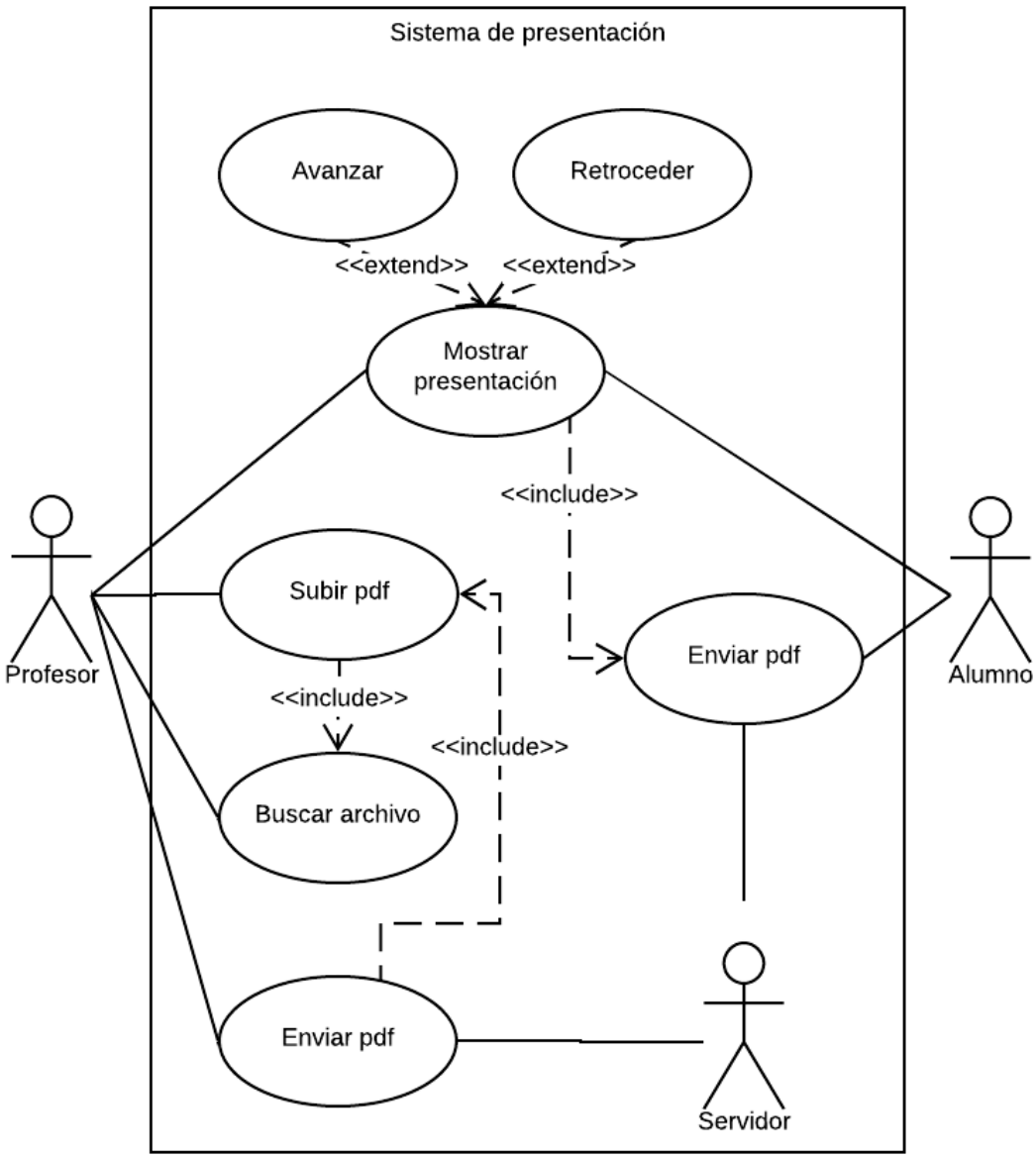


Diagrama 6. Sistema de presentación.

6.2. Diagramas de clases

Una vez se ha definido la funcionalidad general, el siguiente paso es, crear un diseño de las clases fundamentales que necesita el sistema para cubrir los requisitos funcionales que se han presentado en puntos anteriores.

En el diagrama 7, tenemos el diseño de las clases necesarias para representar las interfaces de las distintas pantallas. Dichas clases servirán como *views* y *controllers* de las páginas del sistema.

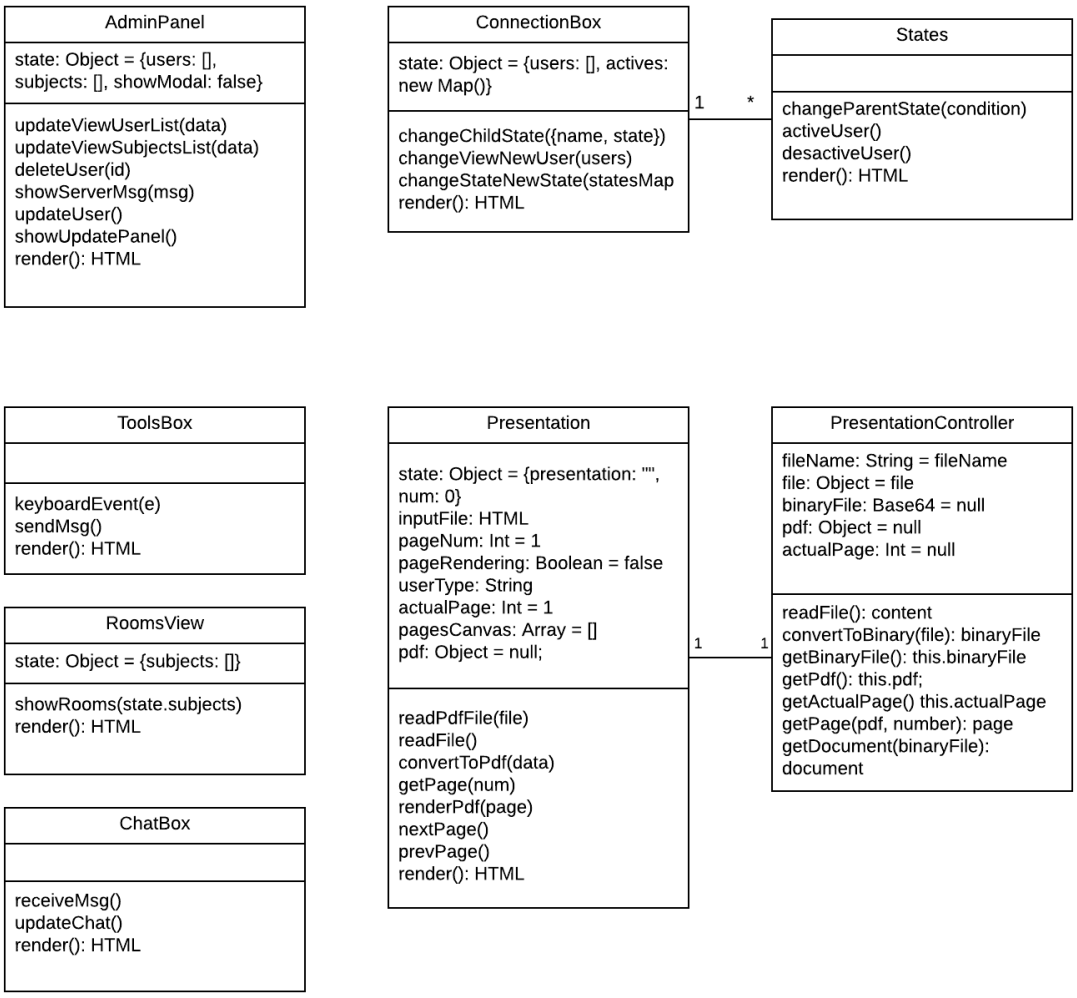


Diagrama 7. Clases de la interfaz.

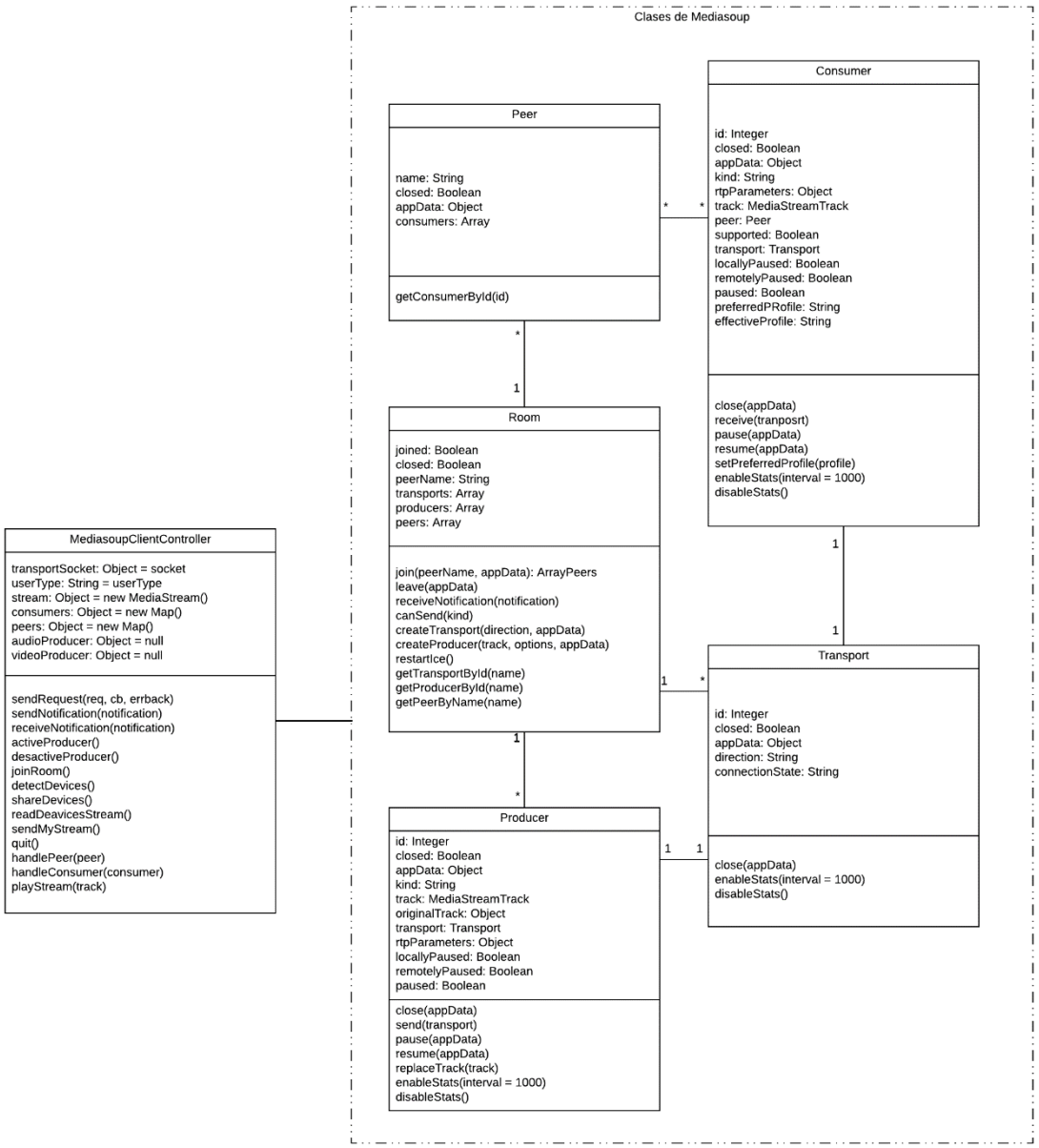


Diagrama 8. Clases del cliente y Mediasoup.

En el diagrama 8, se muestran los diseños de las clases del cliente encargadas de la parte de la comunicación en tiempo real entre *peers*. Enmarcado en un rectángulo con líneas discontinuas está el diseño de las clases pertenecientes a Mediasoup, esto nos servirá para tener una visión clara de cómo éstas están relacionadas con nuestro sistema y de todos sus atributos y métodos, necesaria para el desarrollo de nuestro modelo SFU. Por otro lado, tenemos el diseño de la clase a implementar *MediasoupClientController*, que será la encargada de controlar toda la comunicación

entre *peers* y *server* para conseguir la señalización y el intercambio de datos multimedia en tiempo real.

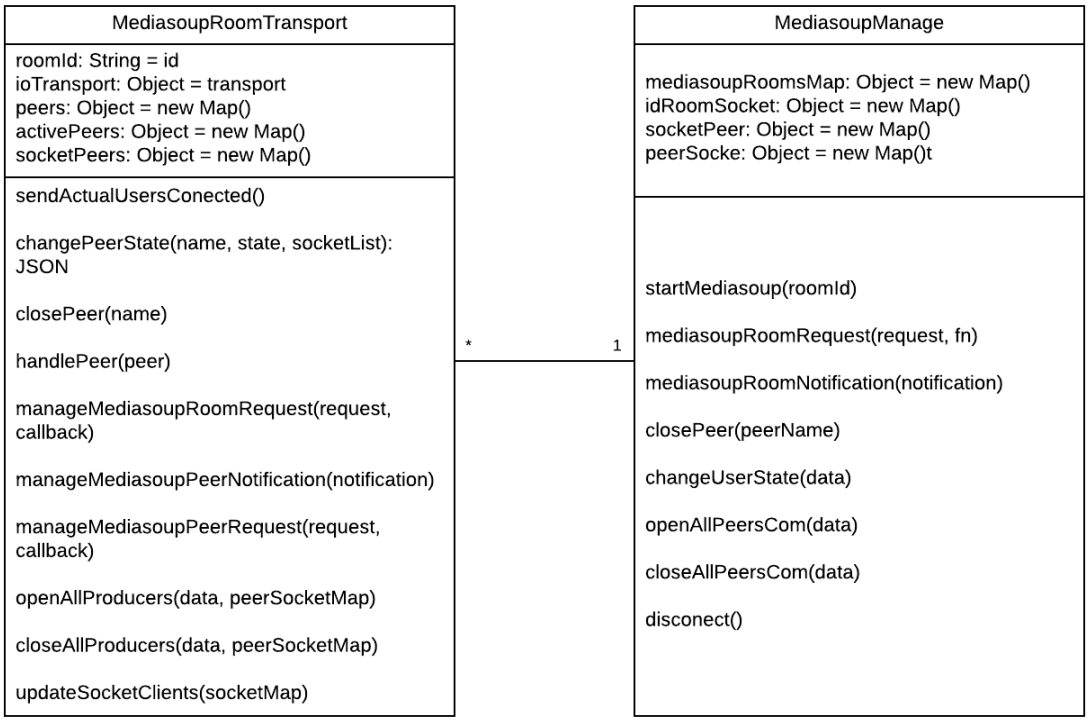


Diagrama 9. Clases del servidor.

En el diagrama 9, se presentan los diseños del *backend*. A la derecha del diagrama, tenemos *MediasoupManage*, que será usada como capa intermedia que enlace y sirva de intercambio de mensajes entre la *Room* del cliente y la *Room* del servidor. A la izquierda del diagrama, tenemos *MediasoupRoomTransport*, donde tendremos todas las entidades y componentes necesarios para manejar una Room en el servidor.

6.3. Diagramas de flujo

En los siguientes diagramas se presentan figuras con los flujos de información que influyen de algún modo en el sistema o que podrían pertenecer a él, presentando de este modo, una secuencia sobre cómo interactúan o podrían interactuar los distintos objetos del sistema a lo largo del tiempo.

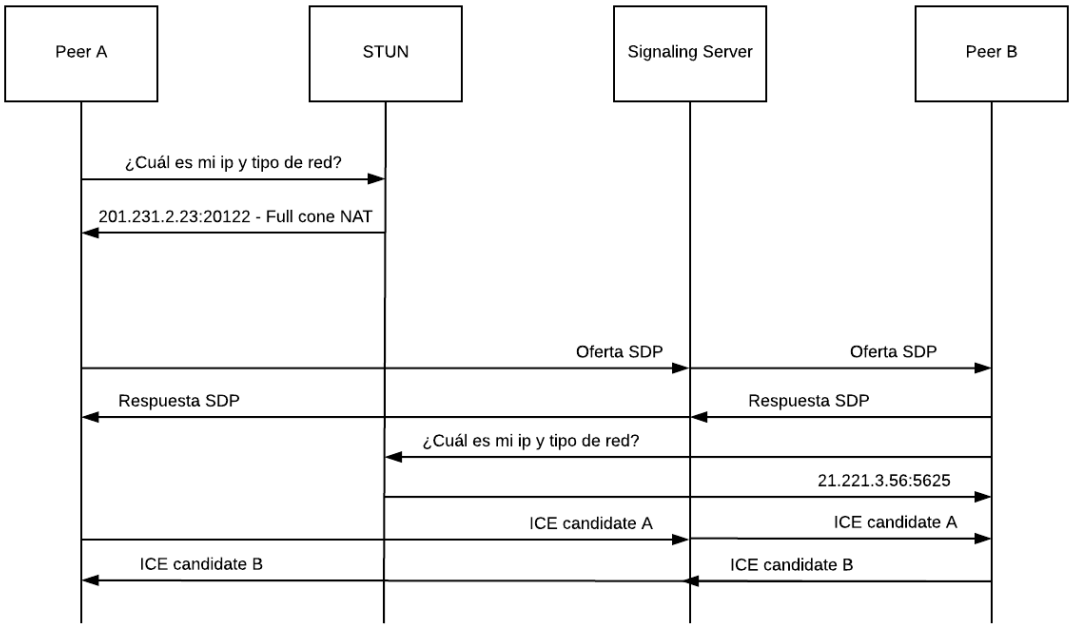


Diagrama 10. Signaling en NATs no simétricas.

Hay una serie de interacciones indispensables que deben darse para el correcto funcionamiento del sistema. Algunas de ellas, se dan con el proceso de señalización, comentado anteriormente, en el punto 3.6.2, junto con el repaso a todas las tecnologías WebRTC. Existen varias posibilidades dentro de la señalización según el tipo de red con el que nos encontremos. Para un escenario en el que los *peers* y/o el servidor, porque en este caso se debe interpretar al servidor como otro *peer*, se escondan tras una red no simétrica, se puede realizar un proceso de señalización como se muestra en el Diagrama 10. Por otra parte, el caso más probable es que tengamos unos *peers* que estén detrás de *firewalls* y/o redes simétricas, en ese caso, tendremos un proceso de señalización como se muestra en el Diagrama 11, en el cual se necesita de un servidor TURN que tenga una dirección IP pública y no se esconda tras alguna red simétrica para que haga de mediador en el intercambio de archivos.

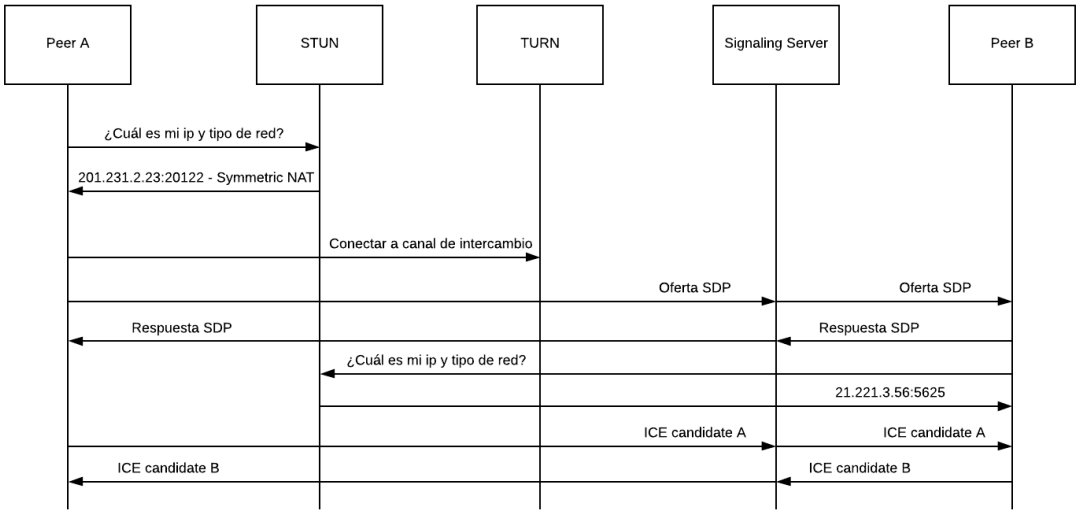


Diagrama 11. Signaling en NATs simétricas.

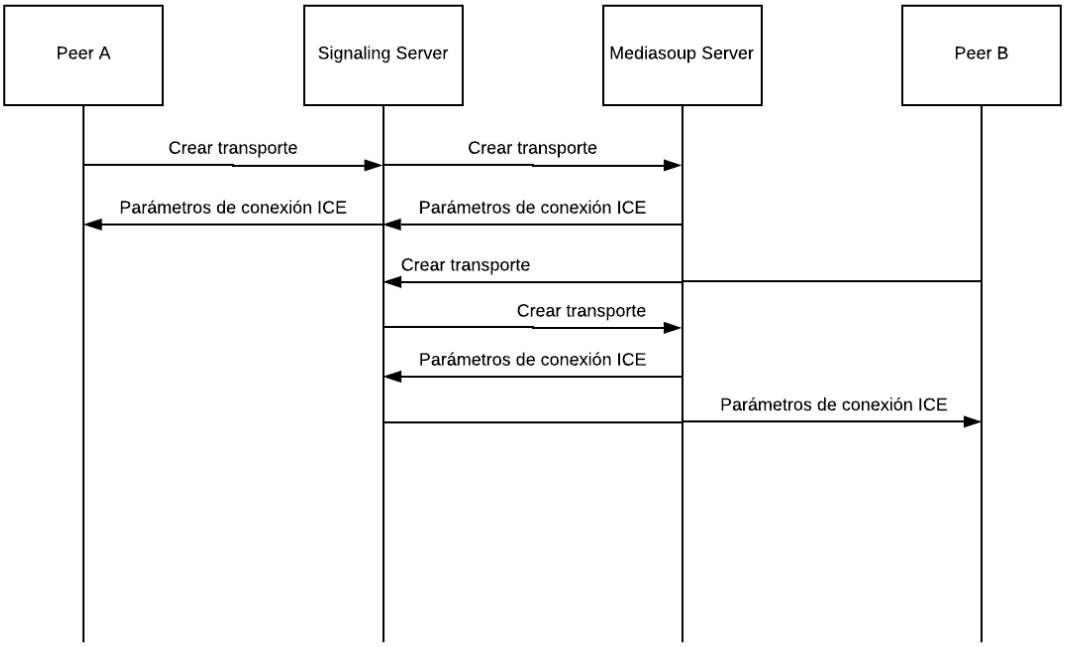


Diagrama 12. Proceso de señalización del sistema.

El caso más común en nuestro sistema lo tenemos en el diagrama 12. Puesto que el sistema a desarrollar con Mediasoup está basado en un intercambio tipo SFU, en el que las conexiones se establecen a partir del servidor y es este el que envía todo el flujo de datos multimedia al resto de peers conectados, se puede decir que éste haría un papel similar al del servidor TURN descrito en el apartado anterior, por ello, si revelamos la

dirección pública y privada de dicha máquina podríamos hacer una conexión sin los problemas con las redes que se han descrito anteriormente.

6.4. Interfaz de usuario

En la interfaz de usuario de nuestro sistema se prioriza claridad y simplicidad por encima de originalidad y el exceso de adornos. Para desarrollarlo se hará uso de Bootstrap 4 |16|, la última versión del *framework* que ofrece una serie de plantillas para el diseño de elementos HTML y CSS.

En la aplicación web tendremos 3 pantallas principales, por lo que, deberíamos tener diseños que compartan atributos y colores similares para todas ellas. También se distribuyen los elementos de un modo coherente e intuitivo. En las figuras 15, 16, 17 y 18 se presentan los bocetos que servirán como idea en la fase de implementación.

En la Figura 15, se presenta el boceto de la página principal, que se encuentra al acceder al sistema, donde los usuarios tendrán diferentes campos para realizar la identificación. En la Figura 16, se presenta el boceto de la página de salas, donde los usuarios tendrán una lista de salas en un panel, acompañadas de unos botones que tendrán diferentes colores según el estado de la sala. En la Figura 17, se presenta el boceto de la página principal para los usuarios de tipo alumnos, en esta se puede apreciar la distribución en varias cajas que contendrán la información que se comparte a través del sistema, chat, contactos, streams de vídeo y presentación. En la Figura 18, se presenta el boceto de la página principal pero esta vez para la parte de los profesores, tendrá una distribución prácticamente similar a la de los alumnos, pero con la diferencia de que estos tienen acceso al control de las presentaciones.

Figura 15. Boceto de la página de ingreso.

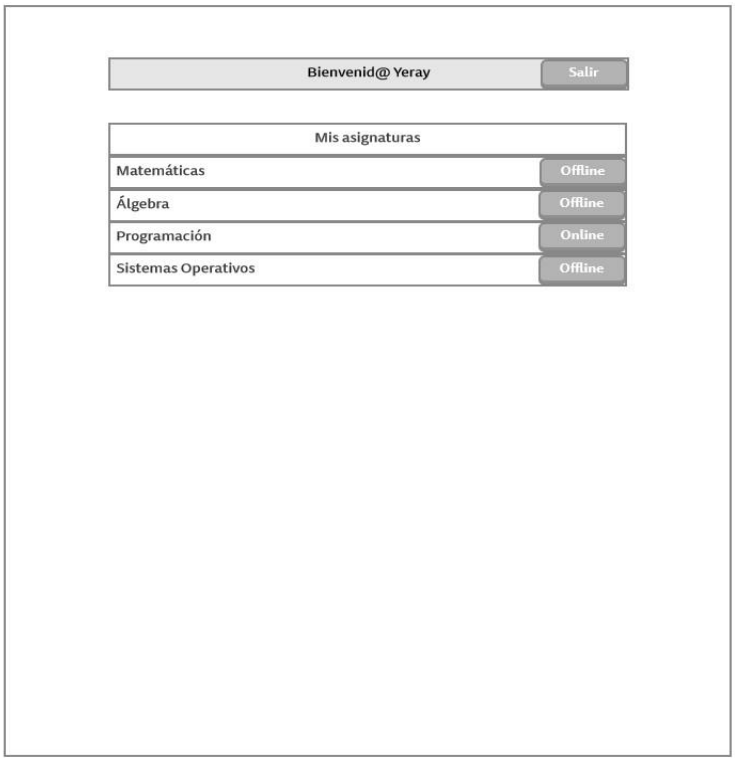


Figura 16. Boceto de la página de salas.

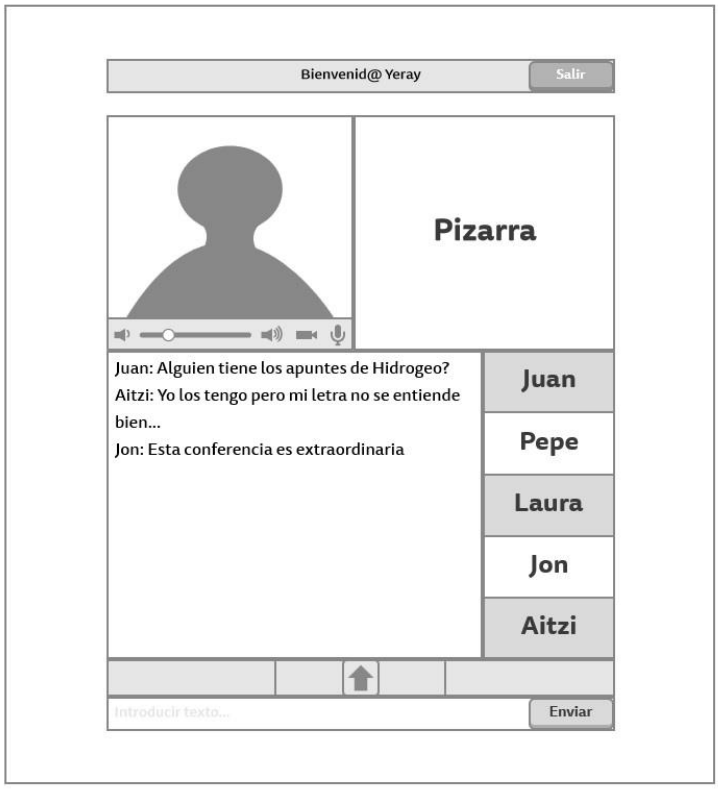


Figura 17. Boceto de la página principal del sistema.

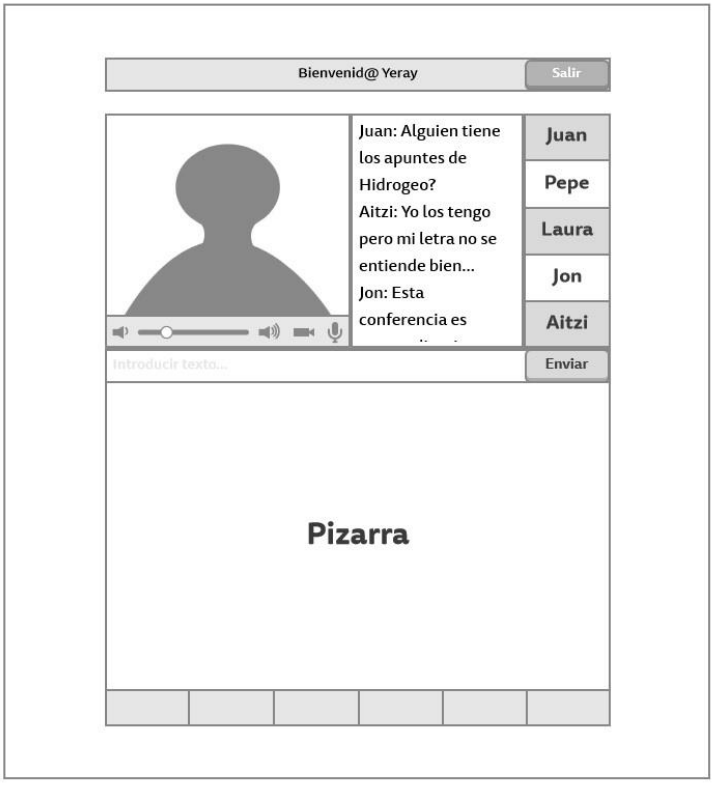


Figura 18. Boceto de la página principal del sistema (profesores).

6.5. Implementación

La implementación del sistema usando Mediasoup consistirá en desarrollar los módulos y partes necesarias para permitir a los usuarios realizar las videollamadas y presentaciones desde el navegador web. Debido a que existe variedad en el uso de navegadores por parte de los usuarios, y que además pueden acceder desde distintos dispositivos, hay que tener en cuenta, a la hora de construir el sistema, que debería ser funcional para el mayor número de objetivos posibles. Por tanto, se pretende realizar un diseño responsive y se hará uso de unas APIs con alta compatibilidad.

6.5.1. Rooms

Las Rooms representan un espacio virtual en el cual van a estar alojados aquellos Peers que desean conectarse entre sí para compartir Tracks de audio y/o vídeo. En la aplicación web se tendrá una lista de salas disponibles, que podrán ser creadas y asignadas por el administrador, cada una de estas salas representará una ruta con un nombre diferente y único que identificará a la Room de Mediasoup que va a ser creada en el servidor. Además de esto, se tendrá una representación local de la Room en la parte del cliente, que servirá para generar los mensajes de intercambio entre cliente y servidor de Mediasoup y con la cual podremos crear objetos de las clases Transport, Producer, Consumer y Peer, de los que haremos uso para que el sistema funcione correctamente.

```
19          this.room = new mediasoupClient.Room(
20                              {
21                                  requestTimeout : 12000
22                              }
23                          );
```

Código 2. Creación de una instancia RoomMediasoup en el cliente.

```
16          //Se intenta crear una Room en el servidor con la configuración de los codecs
17          try {
18              this._mediaRoom = mediasoupServer.Room(config);
19          }
20          catch (error) {
21              this.close();
22              throw error;
23          }
```

Código 3. Creación de una instancia RoomMediasoup en el servidor.

```
71      //Se recibe este mensaje cada vez que un cliente entra en una sala
72      //Se busca la Room por si ya ha sido creada anteriormente
73      //De ser así, incluye al nuevo usuario, sino, crea una nueva Room con el id de la sala
74      client.on('startMediasoup', function(roomId) {
75          console.log(roomId);
76          if (!mediasoupRoomsMap.has(roomId)) {
77              try {
78                  var room = new mediasoupRooms(roomId, mediasoup, io);
79                  idRoomSocket.set(client.id, roomId);
80                  mediasoupRoomsMap.set(roomId, room);
81                  client.join(roomId);
82                  console.log('Creating room mediasoup server');
83
84              }
85              catch (error) {
86                  console.log(error);
87              }
88          }
89          else {
90              if (!idRoomSocket.has(client.id)) {
91                  client.join(roomId);
92                  idRoomSocket.set(client.id, roomId);
93              }
94          }
```

Código 4. Creando tabla clave valor para alojar distintas salas.

Con Mediasoup podemos crear instancias de una Room, como se muestra en el Código 2 para el cliente (que será una representación local de la Room del servidor) y como se muestra en el Código 3 para el servidor. Para conseguir tener activas diferentes Rooms en el sistema, una por cada sala que aparezca representada en la aplicación web, cada una con su espacio virtual, se creará una instancia por sala y se mapeará dicha instancia con el identificador de la sala, como podemos observar en el Código 4, de este modo, cada vez que un usuario entra a una sala el sistema sabrá si esa Room ya existe y por tanto incluiría al nuevo usuario (Peer) o debe crear una instancia nueva.

Mediasoup dispone dos tipos de Room, una para el cliente y otra para el servidor. La que podemos crear con la librería en la parte del servidor, servirá para realizar todas las peticiones al servidor de Mediasoup que será el encargado de generar todas las acciones y procedimientos. Es decir, será la que esté en el núcleo de funcionamiento de todo el sistema. La que podemos crear con la librería en la parte del cliente, será una representación local que nos permitirá crear los objetos para manejar el transporte, la producción y la consumición de *streams* multimedia y los mensajes que deberán ser intercambiados con el servidor Mediasoup cuando queramos que este realice algún tipo de acción. Cuando se crea un Peer en el cliente, lo que se hace es mantener una representación local y virtual de un usuario y una vez quiera entrar en la Room se envía un mensaje al servidor, haciendo uso de cualquier sistema de intercambio de mensajes, para que este incluya al nuevo Peer en la Room del servidor.

A través de la clase Room se pueden crear instancias de Transport y de Producer, con las cuales podremos crear la señalización, la conexión y la creación de los Tracks de los streams de audio y/o vídeo para enviar a los demás usuarios de la sala.

6.5.2. Signaling

Como se ha visto en capítulos anteriores, uno de los elementos básicos y prioritarios a la hora de conseguir establecer un canal de comunicación entre Peers o como en este caso entre Peers y el servidor es llevando a cabo una implementación de señalización. Algunas librerías que hacen uso de WebRTC incluyen funciones con las que podemos llevar a cabo el proceso de señalización a través de WebSocket o SIP, en el caso de Mediasoup, tendremos que desarrollar el método de intercambio de mensajes necesario para transmitir hasta y desde el servidor la información de señalización, así como el resto de mensajes de los cuales hace uso Mediasoup para funcionar.

Mediasoup se fundamenta en el intercambio de una serie de peticiones y/o notificaciones y respuestas entre los clientes y el servidor. Para realizar el intercambio de estos mensajes se va a hacer uso de SocketIO, que hace uso de WebSocket.

```
24          //Enviamos mensaje de petición request al servidor
25          //Recibimos una respuesta y se la enviamos al cliente de Mediasoup
26          this.room.on("request", (request, callback, errback) => {
27              this.transportSocket.emit("mediasoupRoomRequest", { peer: this.name, body: request }, (response) => {
28                  callback(response);
29              });
30          });
```

Código 5. Enviando mensaje al servidor para crear capa de transporte.

En el código 5, se puede observar cómo se envían mensajes a través de WebSocket al servidor con SocketIO |17|, usando la función *emit*. La información que se transmite y el tipo de mensajes que se intercambian en el proceso de señalización lo podemos ver en la Figura 19.

El proceso de negociación para la señalización comienza desde el servidor una vez a MediasoupServer le ha llegado la petición de transporte, cuando esto ocurre, se envía la respuesta con la dirección pública y el puerto asignados para establecer la conexión. Para que no exista ningún tipo de problema por tipos de redes, a MediasoupServer se le indica la IP privada del servidor y en el router se hace una redirección de los puertos a través del Port Forwarding de la NAT |18|. Una vez hecho esto, las conexiones ICE |19| se establecerán sin ningún tipo de problema, encargándose el cliente de Mediasoup de gestionar dicho proceso con la información que le llega en la respuesta. Una vez se tiene toda esta información y se han intercambiado los mensajes pertinentes, ya se puede disfrutar de una conexión estable y permanente a través de la cual se podrá enviar y recibir streams de audio y/o vídeo. Para ello se tendrán dos caminos por cada conexión entre cada peer y el servidor, uno para enviar datos y otro para recibirlo, lo que implica que necesitaremos 2 puertos por cada peer conectado.

createTransport

Create a server-side `Transport`.

Request:

```
{
  method: 'createTransport',
  target: 'peer',
  id: 1111,
  direction: 'send', // 'send'/'recv'.
  options: {},
  dtlsParameters: {}, // Optional.
  appData: Any
}
```

Response:

```
{
  iceParameters: {},
  iceCandidates: [],
  dtlsParameters: {}
}
```

Figura 19. Ejemplo de mensajes intercambiados para la señalización.

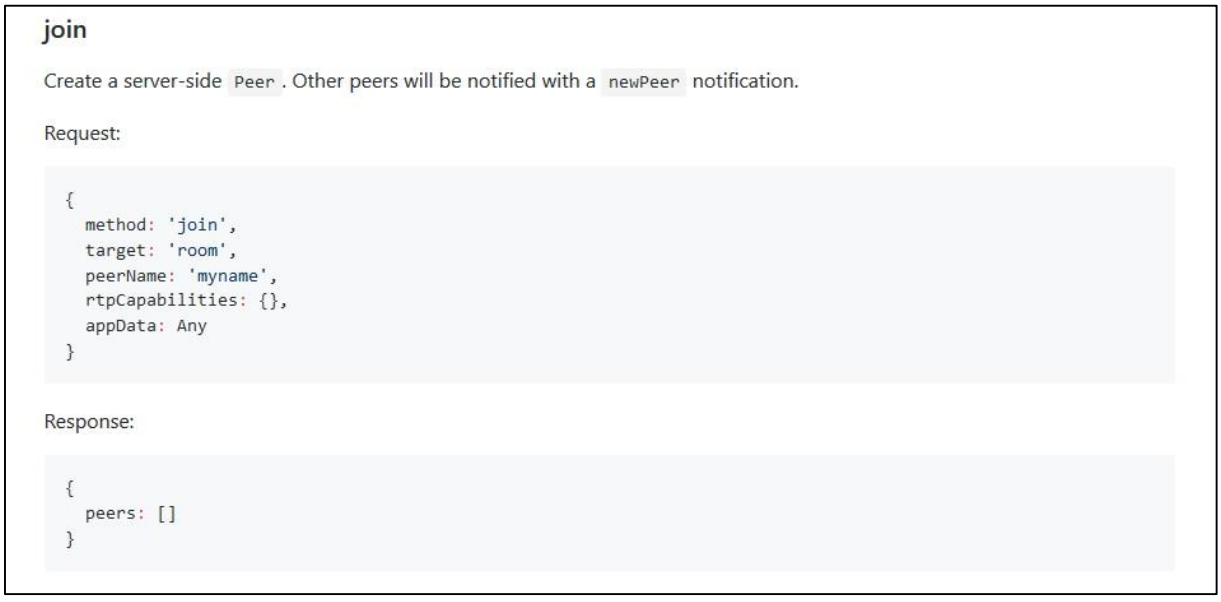
6.5.3. Mensajes Mediasoup

El servidor de Mediasoup es el encargado de distribuir y realizar todo el trabajo interno de conexión, *streaming* y control de flujo de datos entre los diferentes Peers de cada sala. Para poder manejar en cada momento las acciones que se necesita que realice Mediasoup se usan mensajes que este pueda entender, estos mensajes serán enviados desde el cliente al servidor y este llamará a ciertas funciones del servidor para que reciban el mensaje, que puede ser una petición o una notificación, lo que desencadenará una serie de acciones según lo que se pretenda o necesite conseguir. Del mismo modo, el cliente de Mediasoup, necesita la respuesta generada por el servidor de Mediasoup para tener conocimiento de cómo se ha comportado este ante la acción que se le ha pedido y por tanto como debería reaccionar el cliente según la situación que se encuentre.

En el Código 6 se puede observar cómo se realiza el intercambio de mensajes comentado para dos tipos de peticiones hacia el servidor, como son, *QueryRoom*, para obtener información acerca de la Room específica que ha sido creada en el servidor y *Join*, que crea un Peer específico.

```
106     //Maneja mensajes de petición hacia el servidor de Mediasoup
107     manageMediasoupRoomRequest(request, callback) {
108         switch (request.method) {
109             case "queryRoom" : {
110                 this._mediaRoom.receiveRequest(request)
111                 .then((response) => {
112                     callback(response);
113                 })
114                 .catch((error) => console.log(error.toString()));
115                 break;
116             }
117
118             case "join" : {
119                 const { peerName } = request;
120                 this._mediaRoom.receiveRequest(request)
121                 .then((response) => {
122                     var peer = this._mediaRoom.getPeerByName(peerName);
123                     this.handlePeer(peer);
124
125
126                     callback(response);
127                 })
128                 .catch((error) => console.log(error.toString()));
129                 break;
130             }
131         }
132     }
```

Código 6. Peticiones al servidor de Mediasoup.



join

Create a server-side `Peer`. Other peers will be notified with a `newPeer` notification.

Request:

```
{
  method: 'join',
  target: 'room',
  peerName: 'myname',
  rtpCapabilities: {},
  appData: Any
}
```

Response:

```
{
  peers: []
}
```

Figura 20. Parámetros del mensaje JSON de petición Join.

La petición *Join* se realiza con un mensaje como un objeto tipo JSON, que será enviado a través de WebSocket y que contendrá una serie de parámetros que el servidor necesita conocer para crear ese nuevo Peer en el servidor, dichos parámetros pueden ser observador en la Figura 20. Como se puede apreciar en la imagen, tenemos el nombre que identifica al Peer asociado en la parte del cliente y luego existen dos atributos como son *method* y *target* para que se pueda identificar el destino del mensaje y la acción a realizar.

6.5.4. Producción y envío de streams multimedia

Una vez se ha realizado la conexión por medio de ICE, el siguiente paso es crear las producciones de streams de audio y/o vídeo por parte de los Peers y enviarlos al servidor para que este los distribuya a los demás usuarios de la sala. MediasoupClient dispone de una entidad llamada Producer, con la cual podremos manejar los streams multimedia y enviarlos al servidor a través de la capa de transporte.

```
148             .then((devices) => {
149                 let mediaDevices = {audio: false, video: false};
150                 devices.forEach((device) => {
151                     if (device.kind == "audioinput") {
152                         mediaDevices.audio = true;
153                     } else if (device.kind == "videoinput") {
154                         mediaDevices.video = true;
155                     }
156                 });
157                 return mediaDevices;
158             })
159             .then((mediaDevices) => {
160                 this.mediaDevices = mediaDevices;
161                 return navigator.mediaDevices.getUserMedia(mediaDevices);
162             })
163             .then((stream) => {
164                 if (this.mediaDevices.audio == true) {
165                     var audioTrack = stream.getAudioTracks()[0];
166                     this.audioProducer = this.room.createProducer(audioTrack);
167 
168                 }
169 
170                 if (this.mediaDevices.video == true) {
171                     var videoTrack = stream.getVideoTracks()[0];
172                     this.videoProducer = this.room.createProducer(videoTrack);
173 
174                 }
175             });
```

Código 7. Acceso a dispositivos y creación de Producer para audio y vídeo.

El primer paso consiste en permitir acceso a los dispositivos multimedia de las máquinas de los clientes como podemos ver en el Código 7. Se hace uso del método antiguo para pedir permisos de acceso a los dispositivos multimedia porque tiene un mayor rango de versiones compatibles. Para cada uno de los dispositivos permitidos se crea un productor asociado del tipo de datos que contiene el stream, que puede ser de audio o vídeo, por ello se tiene que observar las opciones disponibles por el usuario, en caso de ser del tipo profesor podrá tener productores de audio y vídeo y en caso de ser tipo alumno solo de audio. Dichas producciones quedarán sin enviar hasta que el administrador de la sala, que suele ser de tipo profesor, permita desde la interfaz web, el envío de contenido. Se permite crear productores de audio y de vídeo, independientemente si se es usuario de un tipo o de otro para permitir futuros trabajos enfocados a tutorías en los que tanto alumno como profesor puedan compartir vídeo.

```
158                 this.recvTransport = this.room.createTransport('recv');
159                 this.sendTransport = this.room.createTransport('send');
```

Código 8. Entidades de transporte para enviar y recibir streams.

```
217         //Enviar el stream con la ayuda de las entidades Producer y Transport
218         sendMyStream(producer, transport) {
219             if (!this.room.canSend('audio') && producer.kind === "audio") {
220                 console.log("CANT SEND AUDIO");
221             }
222             else if (!this.room.canSend('video') && producer.kind == "video")
223                 console.log("CANT SEND VIDEO");
224             }
225             else {
226                 producer.send(transport)
227                     .then(() => {
228                         console.log("sending our stream");
229                     })
230                     .catch((e) => {
231                         console.log(e);
232                     })
233             }
234         }
```

Código 9. Enviar productores multimedia a través de transport asociado.

Una vez se tienen los productores y los transportes, ya podemos enviar los streams de audio y/o vídeo haciendo uso de estos objetos como se puede observar en el Código 9. Esto ocurre de la siguiente forma, al llamar a la función para enviar el contenido multimedia que está en el productor a través de la capa de transport indicada como parámetro de la función, se crea un mensaje de petición como el de la Figura 21 que tendrá que ser enviado al servidor para que MediasoupServer realice las acciones

necesarias. Una vez Mediasoup recibe el mensaje y el contenido, este se encarga de distribuir los datos multimedia a los otros usuarios que estén conectados a la misma sala, avisándoles con una notificación como la que se muestra en la Figura 23 de que tienen una nueva consumición de contenido disponible.

Figura 21. Mensaje para crear nuevo productor en MediasoupServer.

6.5.5. Distribución multimedia

La distribución multimedia usa un modelo SFU como el que se ve en la Figura 22. En capítulos anteriores se ha hablado de este modelo, que abandona la idea de conexiones *peer-to-peer* para centralizar todos los intercambios con el servidor, siendo este el centro de recepción y envío de los streams multimedia. De este modo, tendremos que cada uno de los peers conectados enviarán todo el flujo de datos al servidor de Mediasoup y este a su vez, enviará una copia del contenido a cada usuario conectado a la misma sala, donde la resolución del vídeo que se envía desde el servidor al resto de conectados, podrá ser redimensionado para adaptarse a las capacidades de las conexiones de red establecidas entre cada par.

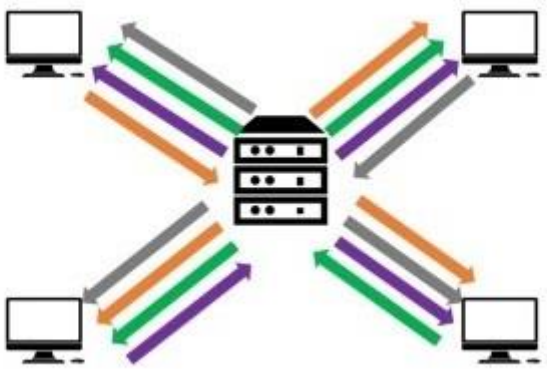


Figura 22. Gráfico modelo SFU WebRTC.

Cuando el servidor Mediasoup dispone de un nuevo *stream* de datos multimedia, recibido a través de la entidad Producer del cliente, este notifica al resto de Peers conectados en la sala con el mensaje que se puede observar en la Figura 23, provocando que salte el evento asociado de cada entidad con un nuevo Consumer que contendrá la información del usuario que ha creado el stream y los datos multimedia en sí mismos. Solo queda reproducir lo que se ha recibido.

newConsumer

A new server-side `Consumer` has been created.

Notification:

```
{
  method: 'newConsumer',
  target: 'peer',
  notification: true,
  id: 3333,
  kind: 'video',
  peerName: 'alice',
  rtpParameters: {},
  paused: false,
  preferredProfile: 'high',
  effectiveProfile: 'medium',
  appData: Any
}
```

Figura 23. Mensaje para notificar de un nuevo Consumer disponible.

6.5.6. Reproducción de contenido

Reproducir contenido multimedia directamente en una aplicación web sin necesidad de recurrir a elementos externos que nos permita manejar este tipo de contenido es posible desde la llegada de HTML5 (*HyperText Markup Language 5*). Esta nueva versión nos ofrece nuevos elementos y atributos que reflejan el uso que los usuarios modernos le dan al navegador. En concreto, a los elementos multimedia, por ello, añadieron dos nuevas etiquetas **<audio>** y **<vídeo>**, que proporcionan funcionalidades multimedia mediante una interfaz estandarizada, haciendo uso de códec para mostrar los contenidos.

En el sistema, haremos uso de ambas etiquetas para reproducir el contenido de los streams que recibirán los Peers a través de los Consumers. En la Figura 24, se puede observar el uso de la etiqueta de vídeo, así como, el posible uso de la etiqueta de audio en un documento HTML, que ha sido comentada porque esta será creada dinámicamente, según el número de Consumers que le lleguen al usuario, información que no se puede conocer a priori.

```
37              <div id="videoCaptureDiv" class="col-md-6 col-lg-6">
38                  <!--<audio id="mediasoupAudio" autoplay></audio>-->
39                  <video id="mediasoupVideo" autoplay controls></video>
40              </div>
```

Código 10. Uso de las etiquetas <audio> y <vídeo>.

En el Código 11, se puede observar cómo se hace uso de las funciones que ofrecen dichos elementos, para añadir el contenido del stream que se recibe a través del Consumer a un nuevo MediaStream y de ese modo poder reproducirlo con la función específica para dicho cometido, como es *play*. Como se ha comentado anteriormente y puede verse en la siguiente imagen, cada vez que llega un nuevo Consumer desde el servidor Mediasoup, se crea un nuevo elemento de <audio> y se añade al contenedor.

```
260         handleConsumer(consumer) {
261             consumer.receive(this.recvTransport)
262             .then((track) => {
263                 if (track.kind === 'audio') {
264                     var audio = document.createElement('audio');
265                     let stream = new MediaStream();
266                     stream.addTrack(track);
267                     audio.src = window.URL.createObjectURL(stream);
268                     audio.play();
269                     $("#videoCaptureDiv").append(audio);
270                 }
271                 if (track.kind === 'video') {
272                     var video = document.getElementById('mediasoupVideo');
273                     console.log(track);
274                     let stream = new MediaStream();
275                     stream.addTrack(track);
276                     video.src = window.URL.createObjectURL(stream);
277                     video.play();
278                 }
279             })
280             .catch((error) => {
281                 console.log(error);
282             })
```

Código 11. Función para reproducir el stream recibido.

6.5.7. Presentación

Uno de los puntos importantes del sistema se basa en la opción de las presentaciones, permitiendo a los usuarios de tipo profesor que hagan uso de la aplicación, no solo compartir contenido de audio y vídeo a través del micrófono y la webcam, sino también, compartir diapositivas que apoyen la exposición.

La mayor parte de las presentaciones se acaban guardando en formato PDF para no tener ningún tipo de problemas de compatibilidad, por ello, se hace uso de la librería *PDFjs*, para leer los documentos que contienen las diapositivas. Para conseguir esto, se hace uso de los métodos de la API File Reader de JavaScript, que nos leerá el fichero y nos representará la información del archivo como una cadena de caracteres codificados en base 64. Esta cadena puede y debe ser decodificada para trabajar con la librería PDFjs y poder leer las páginas contenidas en el documento. Como se puede observar en el Código 12, primero se obtiene la ruta del fichero seleccionado, después se lee el archivo en base 64, tras esto se convierte en binario para poder trabajar con PDFjs, obteniendo de este modo el documento y las páginas, pero sin tener todavía la vista de éstas.

```
34            $('#inputFile').next('.custom-file-label').addClass("selected").html(fileName);
35            this.pdfController = new PdfController(fileName, file.target.files[0]);
36            this.pdfController.readFile()
37                .then((content) => {
38                    var binaryPdf = this.pdfController.convertToBinary(content);
39                    this.props.con.emit('newPdf', {data: content, num: 1});
40                    return this.pdfController.getDocument(binaryPdf);
41                })
42                .then((pdf) => {
43                    this.setState({ presentation: pdf, num: this.actualPage});
44                    return this.pdfController.getPage(pdf, 1);
45                })
46                .then((page) => {
47                    console.log(page);
48                    this.renderPdf(page);
49                })
50                .catch((error) => {
51                console.log(error);
52            });
```

Código 12. Controlador de los ficheros PDF.

Para la visualización dinámica del contenido web, se usa, como se ha comentado anteriormente, ReactJS. Al realizar cualquier modificación en el *state* de una clase creada con React, esta vuelve a llamar a la función *render* pudiendo modificar de ese modo todos los elementos de la vista de los cuales se encarga dicha clase. En el Código 14, se muestra la función render para la clase encargada de la visualización de la parte de la presentación, que no son más que etiquetas HTML que pueden ser escritas directamente como tal gracias a JSX, código que tendrá que ser reconstruido con Babel, compilador que transforma código JSX o EcmaScript 6, en uno que puedan entender los navegadores actuales. Los componentes creados con React tienen un ciclo de vida y estos van ejecutando métodos a lo largo de ese ciclo. Para este caso, se hará uso de uno que es llamado tras acabar de renderizar la vista como se puede observar en el código 13. Cada vez que esto ocurra, borrará los *canvas* con la visualización de las páginas que se han leído anteriormente y en su lugar, añadirá la que toque.

```
75      componentDidUpdate(prevProps, prevState) {
76          if (this.state.num > 0) {
77              if ($('#pdfDiv').has("canvas").length) {
78                  $('canvas').remove();
79              }
80              $('#pdfDiv').prepend(this.pagesCanvas[this.state.num - 1]);
81          }
82      }
```

Código 13. Visualización de página del PDF.

```
    render() {
        var item = this.state.presentation;
        var user = this.userType;
        return (
            <div className="noMargin">
                {item == "" ?
                    <div className="alert alert-info align-middle" display="inline-block" role="alert">
                        No ha cargado ninguna presentación.
                    </div>
                :
                    user == "Profesor" ?
                        <div id="pdfDiv" className="noMargin">
                            <a href="javascript:" onClick={this.prevPage} className="badge badge-dark">&lt;</a>
                            <span className="badge badge-light">
                                Page:
                                <span className="badge badge-light" id="pageNum">
                                {this.state.num}
                                </span>
                                /
                                <span className="badge badge-light" id="pageCount">
                                {this.state.presentation.numPages}
                                </span>
                            </span>
                            <a href="javascript:" onClick={this.nextPage} className="badge badge-dark">&gt;</a>
                        </div>
                        :
                        <div id="pdfDiv" className="noMargin">
                        </div>
                }
            </div>
        )
    }
}
```

Código 14. Renderizado de vista con React.

Una vez tenemos el esquema básico de funcionamiento de las presentaciones, queda compartir y sincronizar las diapositivas con el resto de la sala. Para conseguir esto, simplemente tenemos que enviar el documento en Base 64 al resto de usuarios de la sala, haciendo uso de WebSockets, se envía en este formato porque no se pueden transmitir datos binarios a través de WebSocket. Entonces, cada usuario de la sala hará exactamente lo mismo que el que ha añadido la presentación, obtendrá las páginas del documento con PDFjs y las incluirá en unos canvas para poder visualizarlas. Solo queda enviar el número de página actual al resto de usuarios para que todos puedan ver la misma dispositiva. Únicamente el que ha añadido la presentación, que tiene que ser un usuario de tipo profesor, puede cambiar el número de la página. Además de esto, en el servidor se mantendrá un mapeo de la sala, con la presentación y el número de la página, para los nuevos usuarios que entren.

6.5.8. Panel de administración

En cualquier aplicación web, existe al menos, una pantalla que debe servir para administrar toda la información contenida en la base de datos, sin la necesidad de acceder a esta directamente para modificarla. En programación, existen cuatro funciones básicas de persistencia de bases de datos; crear, leer, modificar y borrar, lo que se conoce con el acrónimo CRUD, que viene de *Create*, *Read*, *Update* and *Delete*

en la lengua inglesa. Este sistema hace uso de un panel de administración, en el que se podrán gestionar los datos de los usuarios que ingresen a la aplicación y de las salas.

Como se ha comentado anteriormente, cada usuario tiene un tipo, para categorizar sus funcionalidades y una serie de salas a las que podrá acceder, dentro de todas las disponibles. Con el panel de administración se podrá cambiar toda la información necesaria perteneciente al ámbito de la aplicación. A continuación, se muestran las pantallas de las que dispone un usuario de tipo administrador que será el que tenga permisos para realizar dichas funciones.

- Crear un usuario

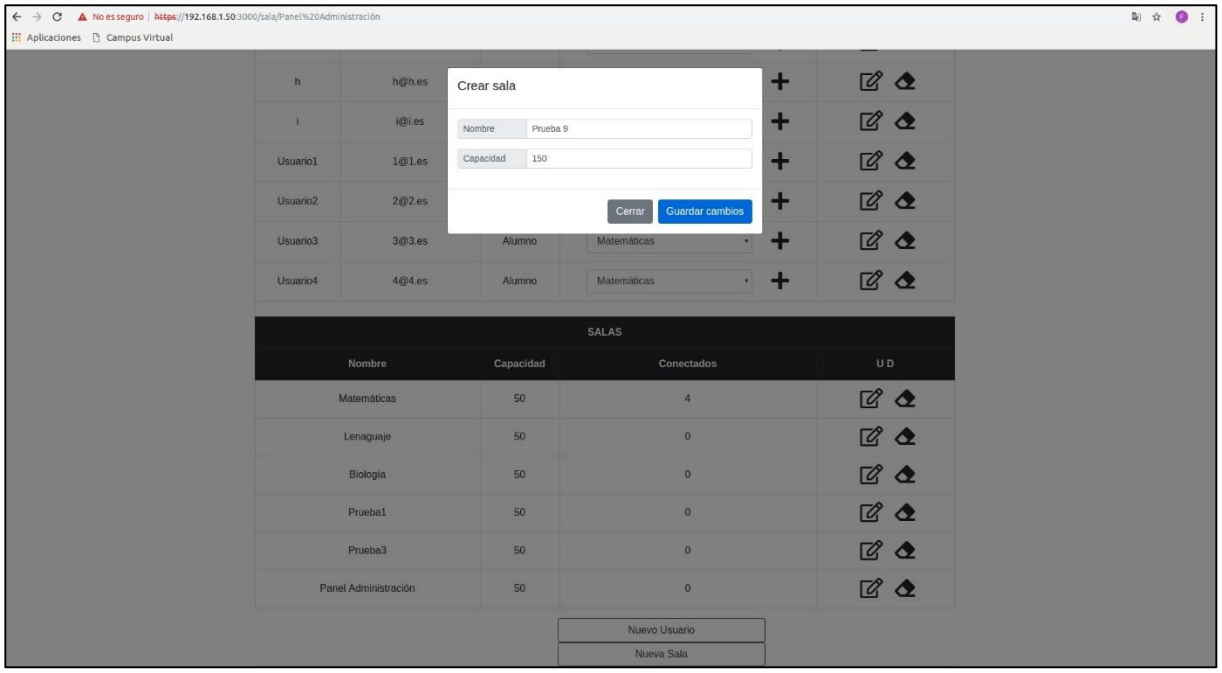


Figura 25. Pantalla para crear nuevos usuarios en el sistema.

- Crear una sala

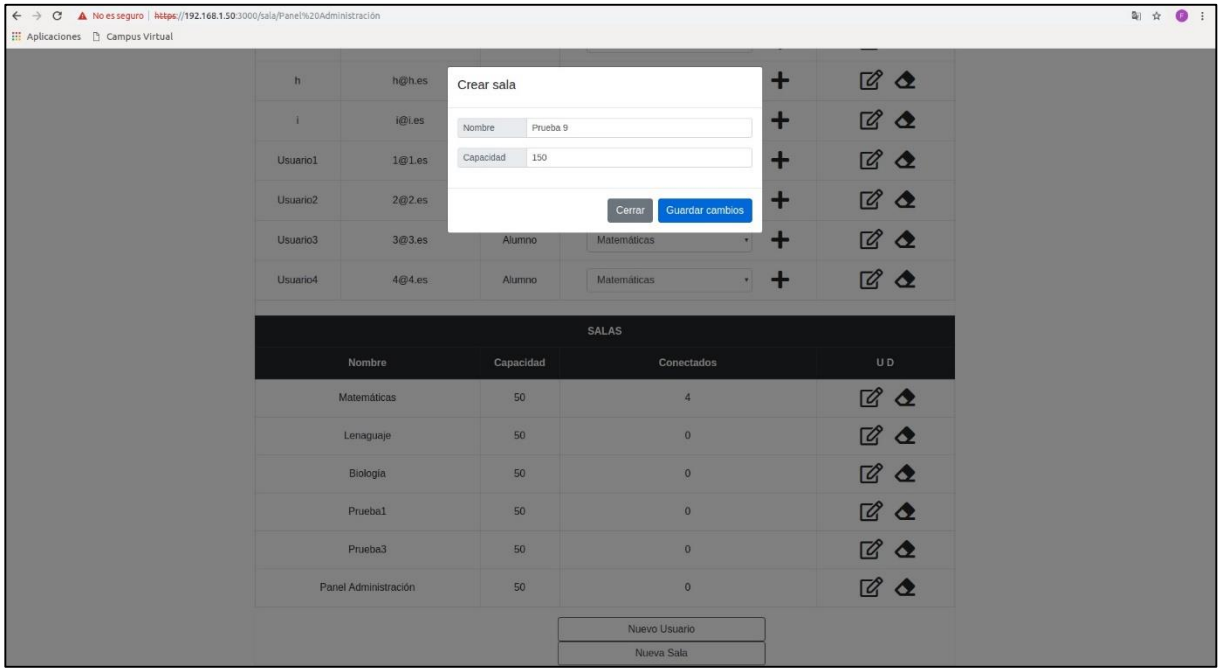


Figura 26. Pantalla para crear nuevas salas en el sistema.

- Leer usuarios

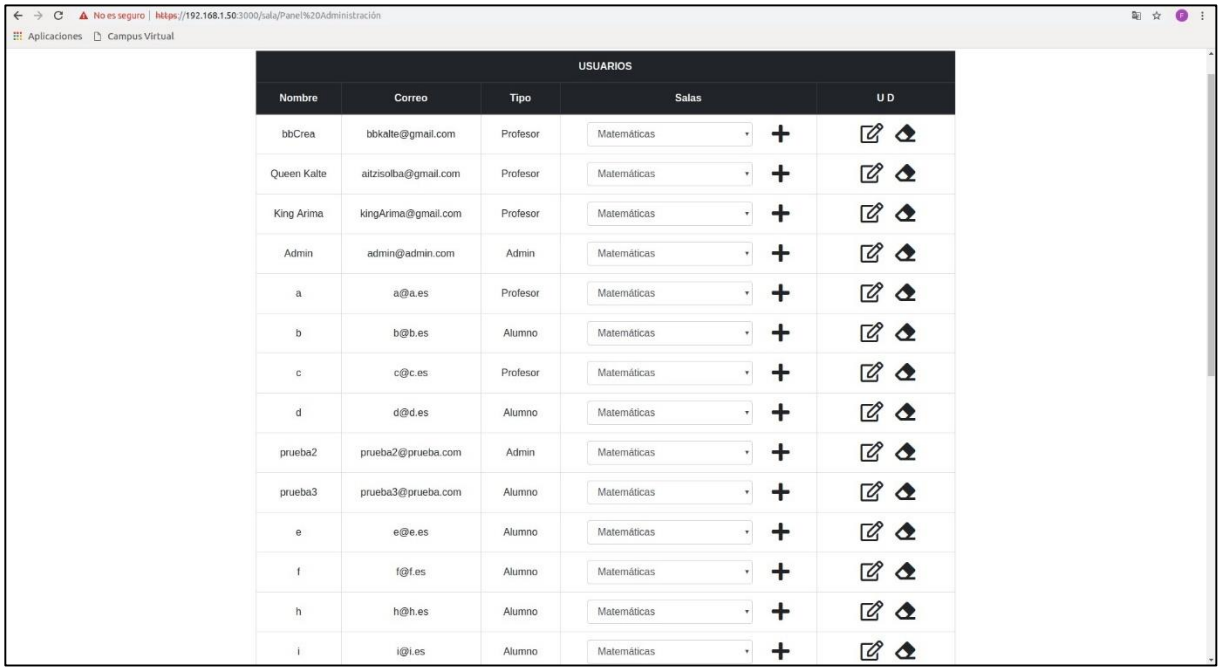


Figura 27. Pantalla donde se muestran todos los usuarios que están registrados en el sistema.

- Leer salas

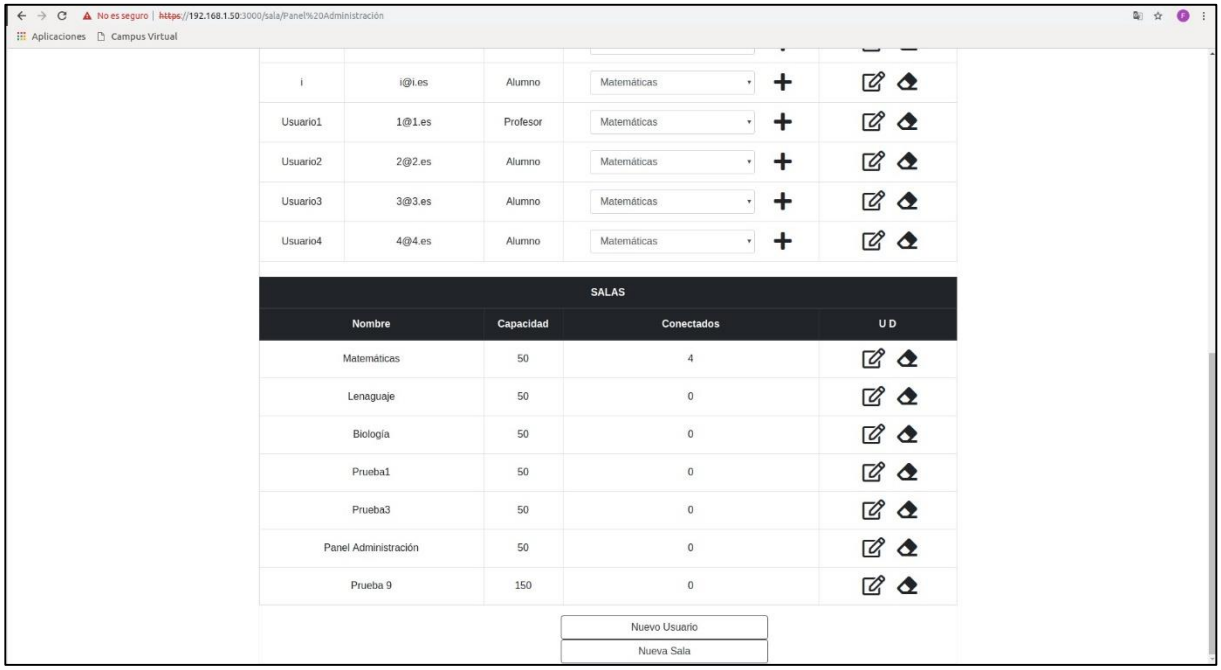


Figura 28. Pantalla donde se muestran todas las salas registradas en el sistema.

- Modificar información básica de usuario

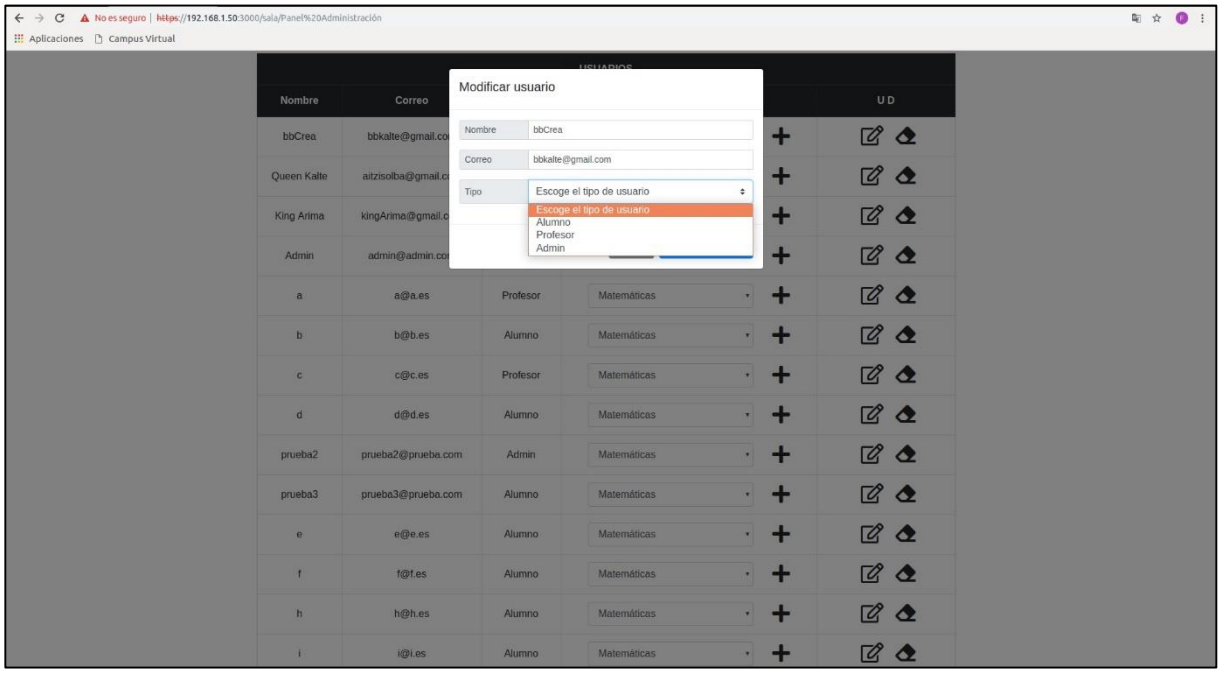


Figura 29. Pantalla para modificar la información básica de un usuario que está registrado en el sistema.

- Modificar salas disponibles para un usuario

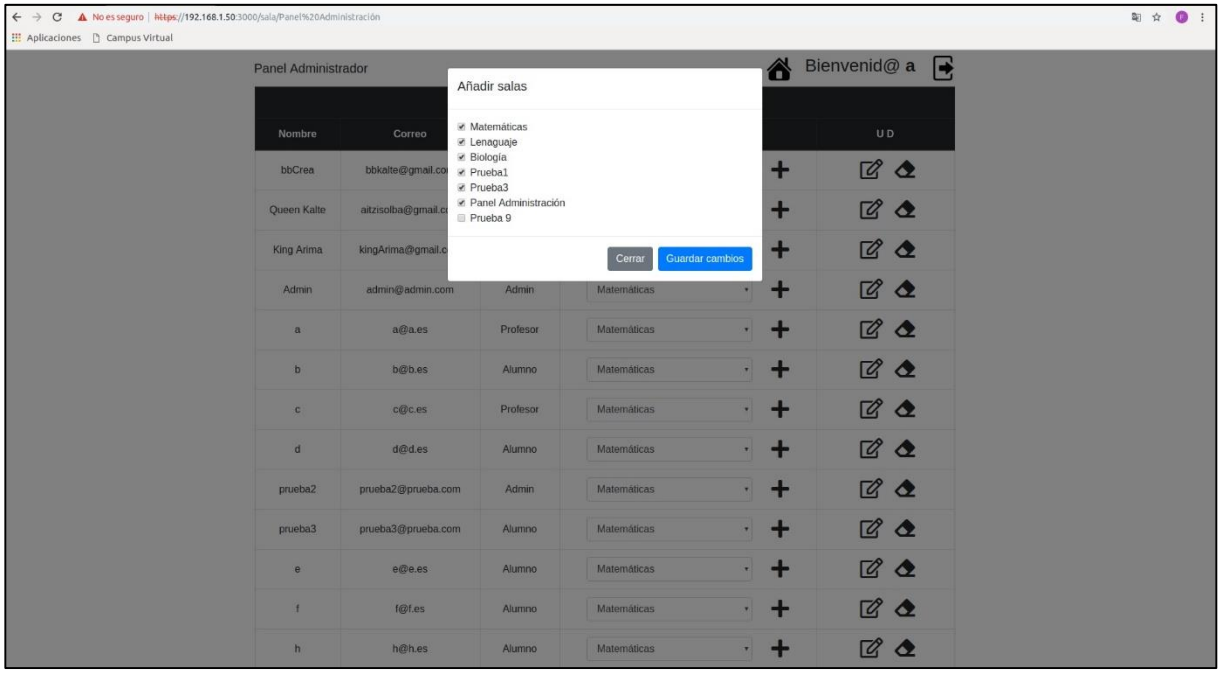


Figura 30. Pantalla para modificar las salas a las cuales puede ingresar un usuario, dentro de las que están disponibles en el sistema.

- Eliminar usuario

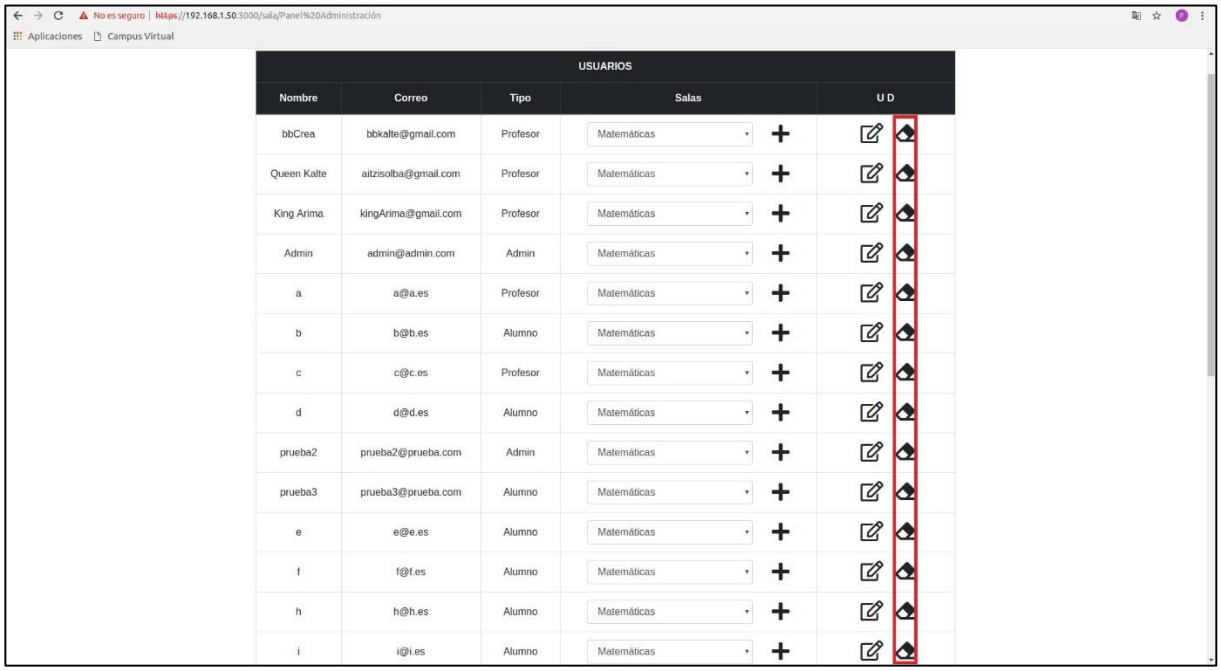


Figura 31. Misma pantalla de la Figura 27. Está remarcado en rojo el botón que sirve para eliminar un usuario del sistema. Aparecerá un mensaje de confirmación para borrar de forma permanente un usuario.

- Eliminar sala

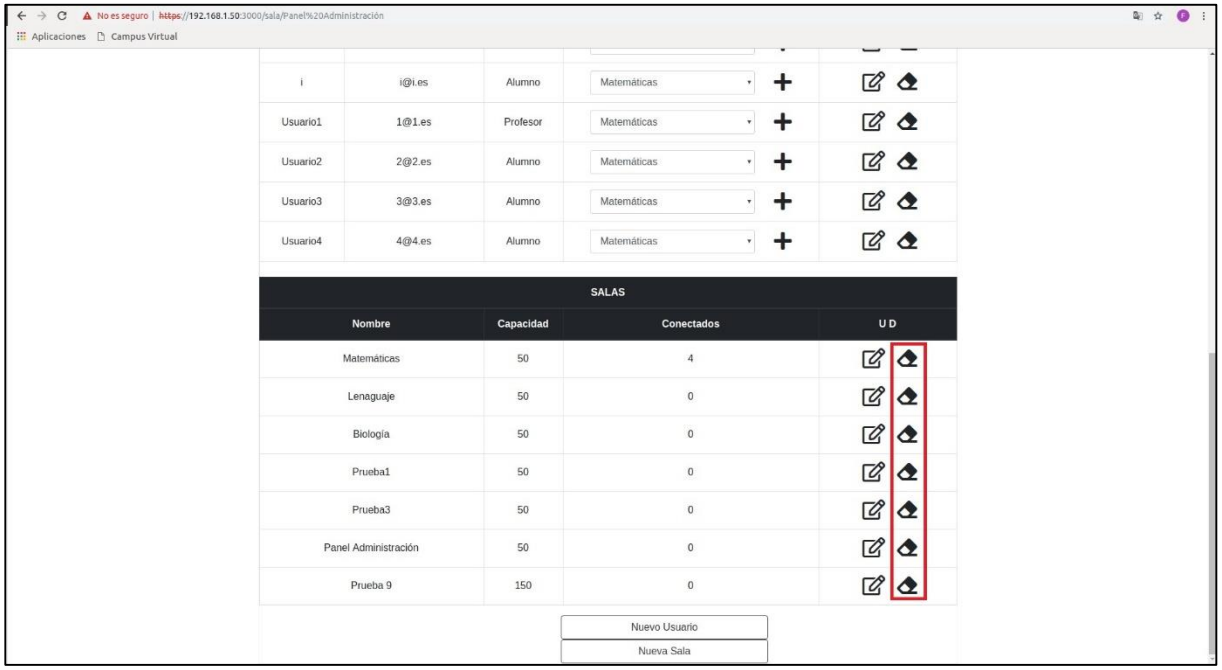


Figura 32. Misma pantalla de la Figura 28. Está remarcado en rojo el botón que sirve para eliminar una sala del sistema. Aparecerá un mensaje de confirmación para borrar de forma permanente una sala.

7. Pruebas y resultados

En este capítulo se van a realizar una serie de pruebas con el objetivo de evaluar el comportamiento de la implementación del sistema que se encuentra en una fase terminada con opción a mejoras futuras. Dichas pruebas consistirán en comprobar el correcto funcionamiento del sistema, su capacidad para adaptarse a distintos entornos, tanto por parte del navegador como del dispositivo donde se ejecuta y finalmente, probar con varios usuarios en una misma sala para encontrar el límite de carga de trabajo.

7.1. Conexión en sala

7.1.1. Prueba

Como se ha comentado anteriormente, los usuarios del sistema podrán ingresar en distintas salas, donde se tiene completa independencia en cada una de ellas, lo que significa que usuarios de una misma sala podrán interactuar entre sí, pero nunca,

podrán hacerlo con los usuarios de otras salas. Para comprobar esto, 4 usuarios entrarán en la sala de Matemáticas y 4 en la de Biología.

7.1.2. Resultado

En la Figura 33, se puede ver en el resultado de ingresar 4 usuarios diferentes en una misma sala del sistema llamada Matemáticas. En la Figura 34, se tienen otros 4 usuarios que se han conectado a una sala llamada Biología. Como se puede observar, cada uno de los 4 usuarios de cada sala pueden ver e interactuar con los que están conectados a la misma sala, pero no con los demás, por lo tanto, se podría concluir que funciona correctamente.

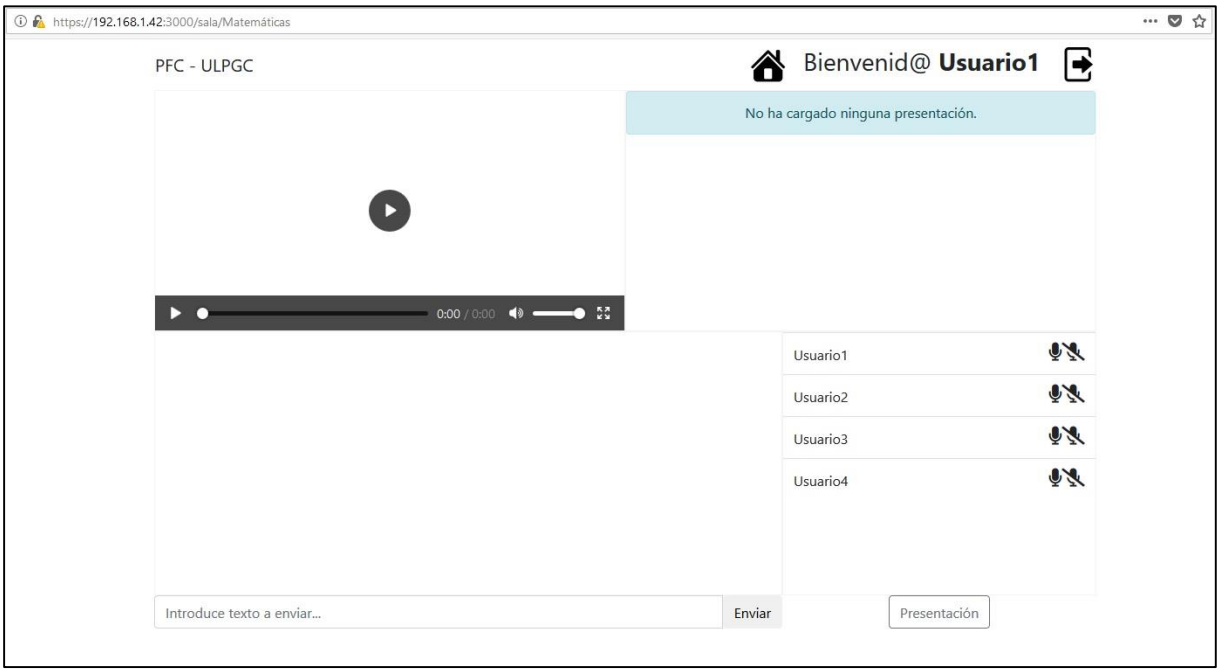


Figura 33. Conexión a sala 'Matemáticas'.

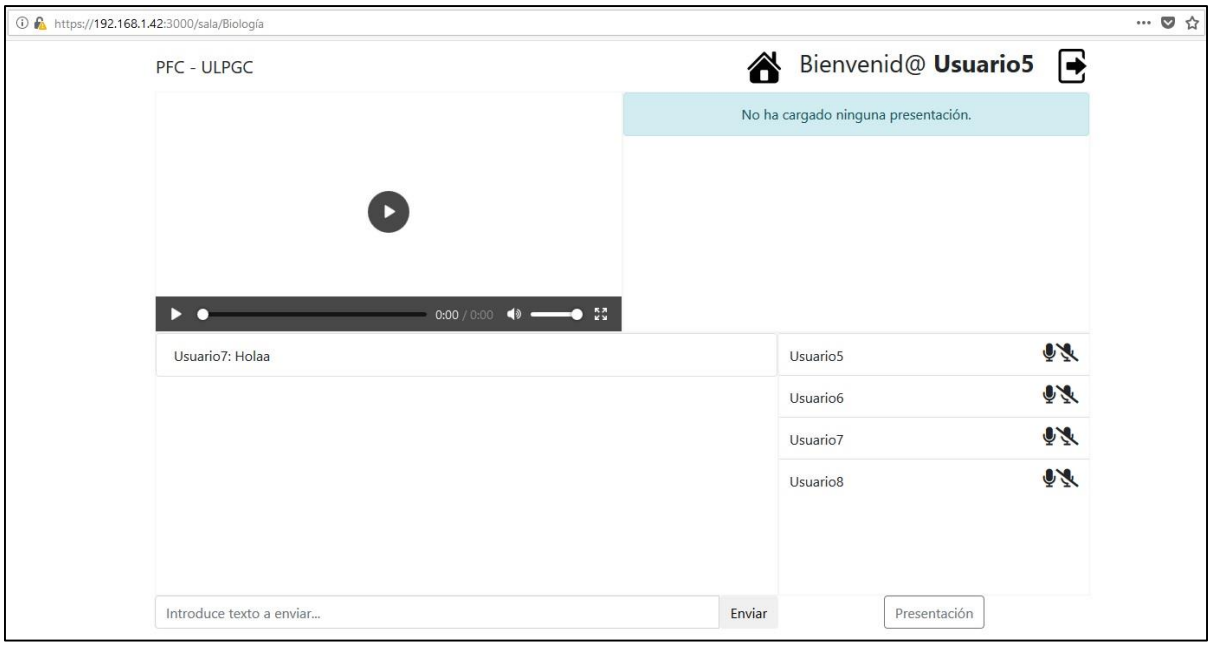


Figura 34. Conexión a sala 'Biología'.

7.2. Streaming de vídeo

7.2.1. Prueba

El streaming de vídeo y audio son las principales características del sistema. En esta prueba se van a conectar varios usuarios, un profesor y varios alumnos para la realización de una exposición sin presentación. Se pretende comprobar el correcto funcionamiento del envío de stream de vídeo y recepción para todos los usuarios conectados en la sala.

7.2.2. Resultado

En la Figura 35, se puede observar como el usuario C recibe el stream de vídeo generado por el profesor que esté realizando la presentación en el sistema. Aunque no se pueda apreciar en la imagen, en la prueba que se ha realizado, se ha comprobado que la fluidez del vídeo es perfecta. En la Figura 36, se tienen en 3 pestañas diferentes del navegador, la recepción del stream por otros 3 usuarios más, todos lo reciben correctamente.

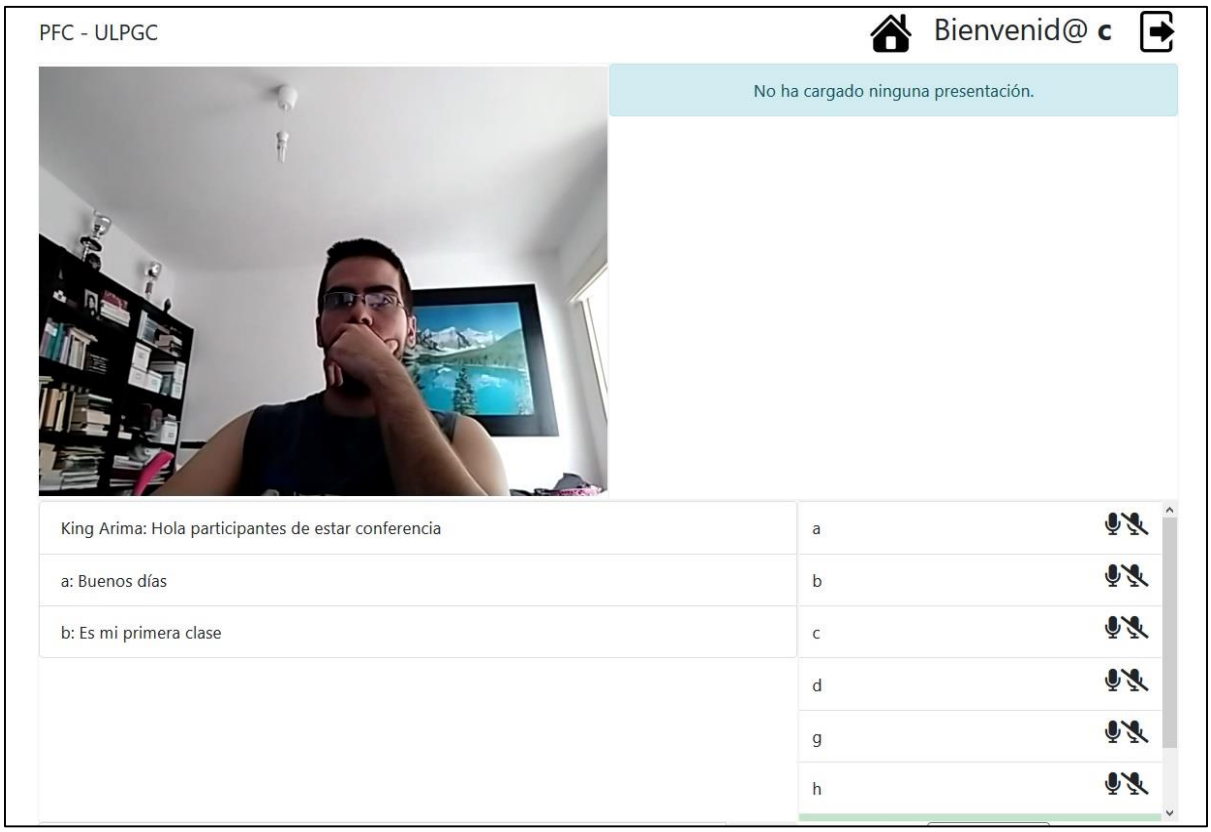


Figura 35. Recepción de Stream de vídeo para el usuario C.

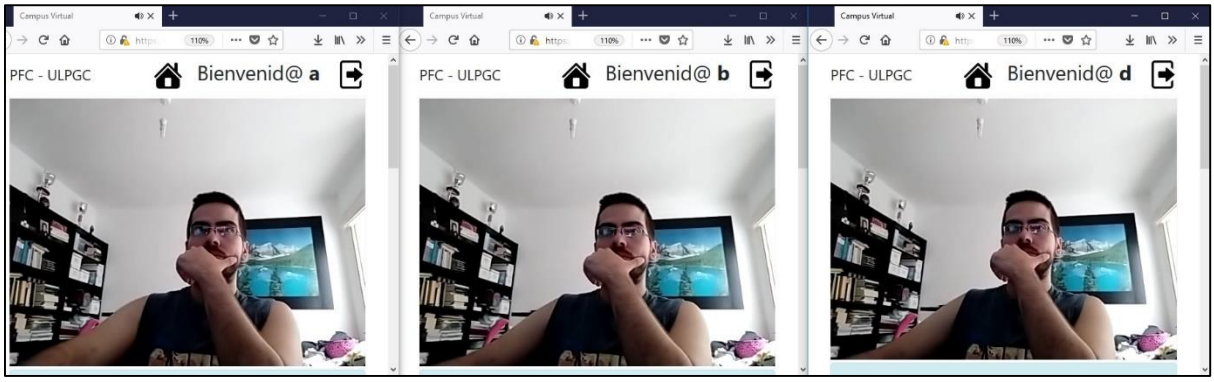


Figura 36. Recepción de Stream de vídeo por parte de 3 usuarios distintos.

7.3. Presentación en sala

7.3.1. Prueba

Otra de las características del sistema, es la capacidad para incluir presentaciones a las exposiciones como se ha comentado en capítulos anteriores. En esta prueba se incluirá una presentación contenida en un archivo en formato PDF alojado en el disco duro local. Para completar la prueba, otros usuarios de tipo alumno entrarán en la sala para comprobar que reciban la presentación correctamente por la página que pretende enseñar el profesor.

7.3.2. Resultado

En la Figura 37, se puede observar el sistema una vez un usuario de tipo profesor ha añadido una presentación. Cada una de las diapositivas se pudo apreciar correctamente, probando distintos tipos de diagramas, letras y dibujos. En la Figura 38, se puede ver como un usuario de tipo alumno que se acaba de conectar a la sala, puede ver la página que el profesor está observando en directo.

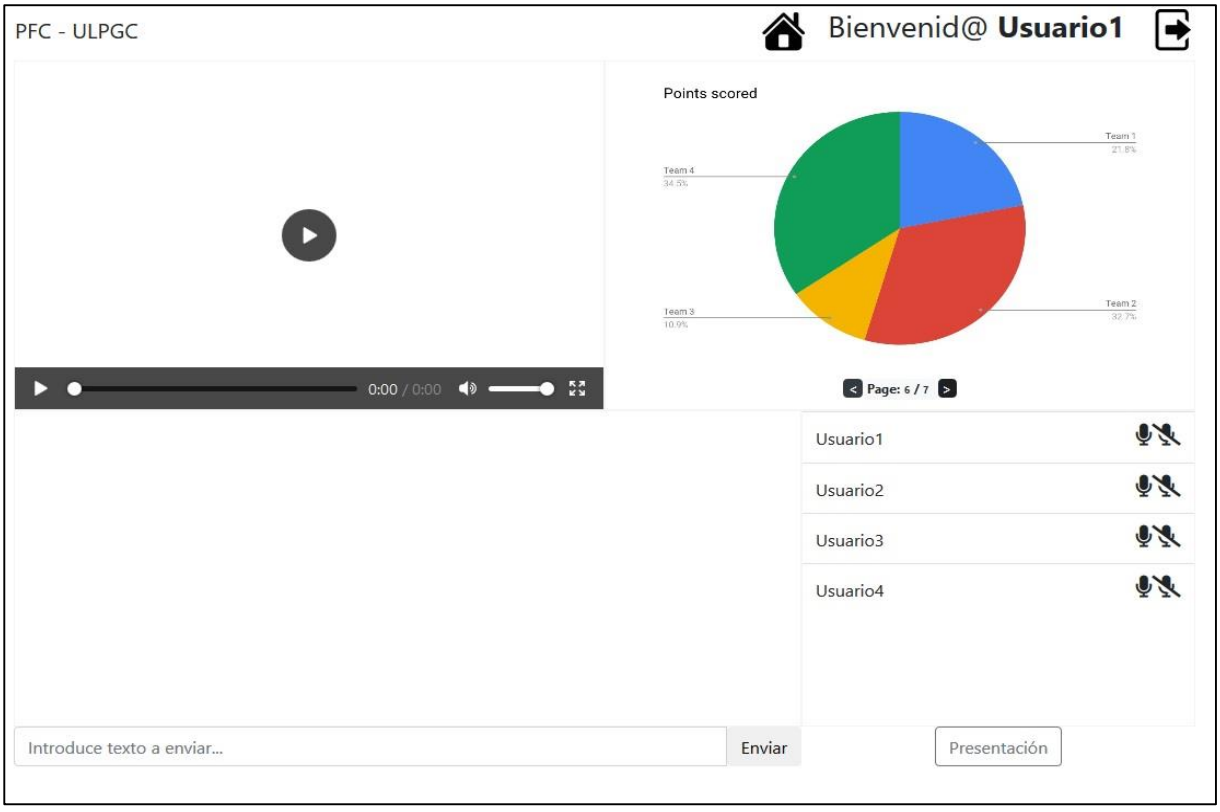


Figura 37. Presentación incluida por usuario de tipo Profesor.

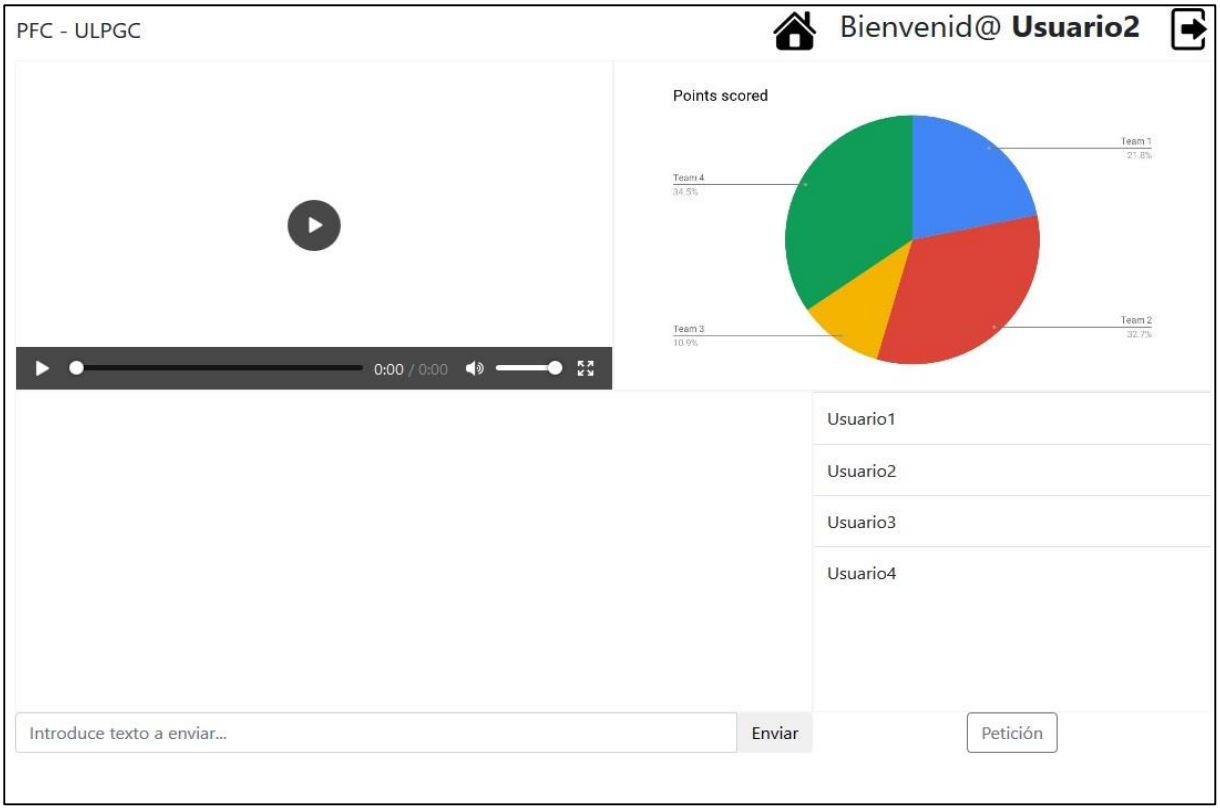


Figura 38. Usuario de tipo alumno conectado a la sala de presentación.

7.4. Compatibilidad

Cuando se desarrolla una aplicación web se ha de tener en cuenta que los usuarios finales pueden conectarse desde una amplia variedad de navegadores, que pueden ejecutarse a través de ordenadores o dispositivos móviles como pueden ser *smartphones* o t*ablets*. Existe una gran variedad de versiones para los 3 principales navegadores de los que se dispone, Microsoft Edge, Mozilla Firefox y Google Chrome desde las cuales cualquier usuario podría estar ingresando al sistema, por ello, es importante trabajar en la compatibilidad del sistema, para que cubra el mayor rango de opciones posibles sin perder la funcionalidad.

Versión	Cualquiera	Hasta 62	Hasta 71
Compatibilidad	Compatible	Compatible	Compatible

Tabla 32. Compatibilidad sistema con navegadores.

Hasta las versiones más actuales de los navegadores web, el sistema tiene un funcionamiento y rendimiento apropiado. Si observamos la Figura 39, donde se muestra una estadística de uso por navegadores, ofrecida por MuyComputer |15|, podemos concluir que el sistema abarca una compatibilidad de alrededor del 76.31% (Chrome: 61.69%, Edge: 4.45% y Firefox: 10.17%), una cifra realmente interesante dada la variedad de opciones que pueden escoger los usuarios.

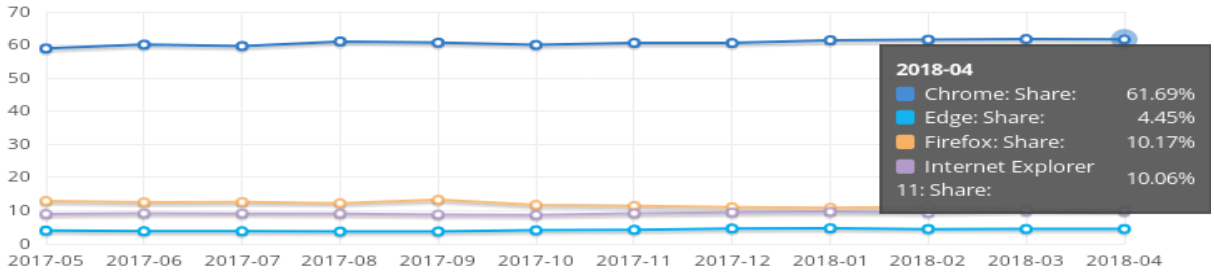


Figura 39. Uso por navegadores 2017 – 2018.

8. Futuros trabajos

El objetivo principal de la aplicación está centrado en entornos educativos, por lo que, una línea de trabajo futuro sería crear unas salas especiales que sirvan como tutorías virtuales entre profesor y alumno y que, además, dispongan de unas ventanas que sirvan como pizarras virtuales, para crear una experiencia lo más cercana a una asistencia real. Dicha opción, no supondría una gran carga de trabajo, dado que, las tecnologías usadas son las implementadas y explicadas en este proyecto.

Otra línea de trabajo futuro, está relacionada con la compatibilidad, puesto que, cuanto mayor sea esta, mayor cantidad de usuarios finales podrán acceder al sistema sin ningún tipo de trabajo por su parte, lo que implica, una mayor capacidad para captar clientes potenciales. En versiones posteriores a Firefox Mozilla v.62 se ha visto un pequeño problema de incompatibilidad con el sistema desarrollado. Adaptar el sistema para que siga siendo funcional para los usuarios en sus versiones actualizadas se traduce en un coste/retribución realmente rentable.

9. Conclusión

El desarrollo de este proyecto pretende servir como base para la implementación de un sistema robusto de comunicación audiovisual centrada en entornos educativos, pero que pueda ser adaptable a diferentes campos con pequeñas modificaciones y ampliaciones. La utilización de sistemas de código abierto y tecnologías en auge, nos han garantizado en el proyecto, seguridad, flexibilidad, rendimiento y bajo coste.

Para conseguir esto, se ha utilizado como pilar fundamental la tecnología WebRTC, que permite las comunicaciones en tiempo real y que destaca por su alta calidad y eficiencia. En la parte del cliente, *frontend*, se ha usado ReactJS para obtener una aplicación en una sola página, sin necesidad de recargas y actualizaciones, como suelen tener las aplicaciones web actuales. Un sitio web de una sola página no puede ser viable sin que lo acompañe un servidor, *backend*, con el que poder transmitir datos en tiempo real, ya sea usando AJAX o usando Websockets como es el caso de este sistema. Para conseguir esto, se ha hecho uso de SocketIO, una librería que transmite datos en formato JSON entre cliente y servidor en tiempo real.

Referencias

|1| Mediasoup. https://mediasoup.org

|2| NodeJS. Entorno en tiempo de ejecución multiplataforma para la capa del servidor. https://nodejs.org/es/

|3| Jquery. Librería JavaScript creada por John Resig para simplificar la manera de actuar con HTML y manipular el árbol DOM. https://jquery.com/

|4| ReactJS. Biblioteca JavaScript para la creación de interfaces de usuario. https://reactjs.org/

|5| PDF.js. Biblioteca JavaScript para la manipulación de documentos en formato PDF. https://mozilla.github.io/pdf.js/

|6| ES6. Nuevas funcionalidades de ECMAScript 6. http://es6-features.org/

|7| IETF. Internet Engineering Task Force. https://www.ripe.net/participate/internet-governance/internet-technical-community/ietf

|8| IETF. The WebSocket Protocol as a Transport for the Session Initiation Protocol (SIP). https://tools.ietf.org/html/rfc7118

|9| Global IP Solutions. Corporación basada en el desarrollo de aplicaciones en tiempo real de procesamiento de voz y vídeo. https://en.wikipedia.org/wiki/Global_IP_Solutions

|11| W3C. World Wide Web Consortium. https://www.w3.org/

|12| NAT. Network Address Translation.
https://es.wikipedia.org/wiki/Traducci%C3%B3n_de_direcciones_de_red

|13| Salario medio de programador por lenguaje de programación. SkyLab. http://www.skylabcoders.com/es/-cu%C3%A1l-es-el-salario-de-un-programador-_13037

|14| Valor hora promedio de un freelance en IT. Mi Carrera Laboral.
https://micarreralaboralenit.wordpress.com/2015/06/03/el-valor-hora-promedio-de-un-freelance-en-it-en-el-mundo-es-de-21-dolares-la-hora/

|15| El mercado de los navegadores según MuyComputer.
https://www.muycomputer.com/2018/05/03/chrome-dominando-navegadores/

|16| Bootstrap v4. Herramienta para el desarrollo front end con diseños responsive. https://getbootstrap.com/

|17| SocketIO. Biblioteca JavaScript que permite la comunicación bidireccional en tiempo real entre cliente y servidor. https://socket.io/

|18| Port Forwarding. Configuración de redirección de puertos para routers**.** http://culturacion.com/como-configurar-port-forwarding/

|19| ICE. Técnica usada en las conexiones peer-to-peer.
https://en.wikipedia.org/wiki/Interactive_Connectivity_Establishment

|20| Topologías usadas en WebRTC. https://webrtcglossary.com/sfu/