# DESARROLLO DE PROTOTIPO WEB FUNCIONAL DE TELEMEDICINA PARA ATENCIÓN DE CITAS MÉDICAS EN LA IPS UNIVERSITARIA

Daniel Santiago Vallejo Ortega

Informe de práctica para optar al título de Bioingeniero

Asesor
Jenny Kateryne Aristizábal Nieto, Magíster (MSc) en Ingeniería Biomédica

Universidad de Antioquia
Facultad de ingeniería
Bioingeniería
Medellín, Antioquia, Colombia
2022

| Cita | Vallejo Ortega [1] |
|---|---|
| **Referencia**<br><br>Estilo IEEE (2020) | [1] D. S Vallejo Ortega, “Desarrollo de prototipo web funcional de telemedicina para la atención de citas médicas en la IPS Universitaria”, Trabajo de grado profesional, Bioingeniería, Universidad de Antioquia, Medellín, 2022 |





Coordinador de prácticas programa Bioingeniería: Javier Hernando García Ramos.

Asesora interna: Jenny Kateryne Aristizábal Nieto.

Asesor externo: Yair Monsalve.

IPS Universitaria, Área de TICS.





Centro de Documentación Ingeniería (CENDOI)

**Repositorio Institucional:** http://bibliotecadigital.udea.edu.co

Universidad de Antioquia - www.udea.edu.co

**Rector:** John Jairo Arboleda Céspedes.

**Decano/Director:** Jesús Francisco Vargas Bonilla.

**Jefe departamento:** Juan Diego Lemos Duque.

## Dedicatoria

Este trabajo está dedicado a mis padres y abuelas, por brindarme los medios para lograr mis aspiraciones y objetivos, por compartir conmigo en los momentos felices y brindarme el apoyo y la fortaleza en los momentos difíciles.


## Agradecimientos

Agradecimientos a mi familia por el inagotable apoyo, al Hospital Alma Máter por haberme dado la oportunidad de realizar mi práctica académica, a la Universidad de Antioquia por haberse convertido en mi segundo hogar y a los docentes del pregrado de Bioingeniería por haber sido parte fundamental de mi crecimiento personal y profesional.

TABLA DE CONTENIDO

LISTA DE TABLAS

LISTA DE FIGURAS

# SIGLAS, ACRÓNIMOS Y ABREVIATURAS

| | |
|---|---|
| **SFTP** | Protocolo seguro de transferencia de archivos |
| **TI** | Tecnologías de la información |
| **ICE** | Protocolo de Establecimiento de Conectividad Interactivo |
| **SDP** | Protocolo de Descripción de Sesiones |
| **STUN** | Utilidades de cruce de sesiones para NAT |
| **TURN** | Transversal usando relés alrededor de NAT |
| **NAT** | Traducción de direcciones de red |
| **TCP** | Protocolo de control de transmisión |
| **IP** | Protocolo de internet |
| **IPv4** | Protocolo de internet versión 4 |
| **RTP** | Protocolo de transporte en tiempo real |
| **OSI** | Modelo de interconexión de sistemas abiertos |

## RESUMEN

El objetivo del presente trabajo fue el desarrollo de un prototipo de telemedicina para la atención de citas médicas en el Hospital Alma Máter de Antioquia debido a la gran importancia que tiene la atención médica basada en herramientas tecnológicas, pues estas contribuyen a sobrepasar las brechas geográficas y a brindar un servicio médico cuando las condiciones externas o internas a las circunstancias de un usuario de una institución de salud requieren de una asistencia médica virtual y a distancia. Para el cumplimento del proyecto se adoptaron técnicas ágiles de desarrollo de software para el levantamiento de requerimientos del prototipo. A partir de los requerimientos, se realizó el diseño de la arquitectura cliente-servidor y la implementación del prototipo acorde a las tecnologías necesarias basada en los roles de los participantes. Finalmente, se implementó una política de tratamiento de datos que contempla las condiciones y seguridad de la información durante la cita médica. Se obtuvo un prototipo web que permite la comunicación en tiempo real entre 2 pares, el cual incluye un consentimiento informado de la política de tratamiento de datos y diferentes funcionalidades como la grabación de la cita y un chat para el intercambio de diferentes tipos de archivos, con una autenticación y autorización segura según los roles. Acorde a los resultados obtenidos es posible planear una segunda etapa para evaluar la aplicación en un escenario con pacientes reales del Hospital Alma Máter.


***Palabras clave*** —**WebRTC, Hospital Alma Máter, Política de Tratamiento de Datos, Telesalud, Telemedicina.**

## ABSTRACT

The aim of this work was the development of a telemedicine prototype for attending medical appointments in the Hospital Alma Máter of Antioquia due to the great importance of medical care based on technological tools, since these help to overcome geographic gaps and provide a service doctor when external or internal conditions to the circumstances of a user of a health institution require virtual and remote medical assistance. For the fulfillment of the project, agile software development techniques were adopted to collect the requirements of the prototype. Based on the requirements, the design of the client-server architecture and the implementation of the prototype according to the necessary technologies based on the roles of the participants were carried out. Finally, a data treatment policy was implemented that contemplates the conditions and security of the information during the medical appointment. A web prototype was obtained that allows real-time communication between 2 peers, which includes informed consent of the data processing policy and different functionalities such as the recording of the appointment and a chat for the exchange of different types of files, with a secure authentication and authorization based on roles. According to the results obtained, it is possible to plan a second stage to evaluate the application in a scenario with real patients of the Hospital Alma Máter.


***Keywords*** — **WebRTC, Hospital Alma Máter, Data Processing Policy, Telehealth, Telemedicine.**

# I. INTRODUCCIÓN

La telemedicina se define como un acto médico que es realizado sin contacto físico directo entre un profesional en salud y un paciente, por medio de un sistema telemático, el cual hace uso de tecnologías de la información y la telecomunicación para poder brindar asistencia médica adecuada, sin tener en cuenta la distancia que pueda separar a los involucrados en un encuentro médico [1]. Al día de hoy el uso de una aplicación que permita la conexión remota entre profesionales en salud y pacientes es un requisito indispensable para realizar telemedicina, pues gracias a circunstancias como la pandemia por Covid-19, en la cual el encuentro físico puede ser riesgoso y complicado, se hace conveniente acceder a una atención en salud remota y sobre todo de calidad en la que las citas médicas puedan ser satisfactorias si la urgencia médica así lo permite.

Actualmente la IPS Universitaria de la Ciudad de Medellín, (ahora llamada Hospital Alma Máter de Antioquia) no cuenta con una plataforma en telemedicina que permita atender citas médicas a través de videollamadas, por lo cual, en su misión de convertirse en un Hospital Inteligente bajo el proyecto de Salud Digital [2], ha contemplado como uno de sus objetivos, el desarrollo y la implementación de una plataforma de telemedicina que posibilite el encuentro médico.

Los miembros del equipo de tecnología de la información del Hospital Alma Máter han visto conveniente que dicha plataforma contemple la portabilidad de la aplicación, debido a la necesidad de que los participantes que vayan a ser beneficiados del servicio de telemedicina, puedan acceder desde cualquier dispositivo que tenga conexión a internet y que a través de un enlace puedan acceder a la teleconsulta.

Hoy en día existen plataformas de videollamada gratuitas como Google Meet o Skype, sin embargo, para el uso de estas suele ser común cumplir con requerimientos como registro a otro tipo de servicios; en este tipo de plataformas, el uso de funcionalidades más avanzadas como la grabación de las videollamadas suelen ser pagadas por parte de la empresa gestora debido a que se necesita garantizar una infraestructura para el almacenamiento de dichos archivos. Además, al ser plataformas ya desarrolladas y dispuestas en producción, la personalización de

esta suele ser un impedimento para adaptarla a las necesidades que requiere una institución prestadora de servicios de salud o sistema de telemedicina.

Un factor importante para considerar es que debido a la pandemia por Covid-19, el Hospital Alma Máter ha tenido una gran afectación económica, pues sus recursos han sido priorizados en la adquisición de personal e infraestructura tecnológica en dispositivos biomédicos, por lo que se ha descartado en primera instancia adquirir software especializado para el desarrollo de prototipo de plataforma.

Por estas razones, se emprendió el desarrollo de un prototipo para teleconsulta con base en los requerimientos del Hospital Alma Máter y sus miembros. Una vez obtenidos y evaluados los requerimientos principales se diseñó e implementó una arquitectura cliente-servidor teniendo en consideración la autenticación por roles en el encuentro médico, y teniendo presente la necesidad de adoptar una política de tratamiento de datos.

Gracias al avance en tecnologías de la información (TI), telecomunicaciones, el advenimiento de HTML5 en la web, y tecnologías de comunicación en tiempo real, se obtuvo un prototipo web de telemedicina entre pares (uno a uno) sin la necesidad de instalar programas adicionales o descargar aplicaciones nativas, en la que se permiten realizar el intercambio de archivos, mensaje de texto, grabación de la videollamada y demás funcionalidades.

# II. OBJETIVOS

## 2.1. Objetivo general

Desarrollar el prototipo funcional de la aplicación web que permita realizar videollamada entre doctor y paciente en la IPS Universitaria.

## 2.2. Objetivos específicos

- Realizar el levantamiento de requisitos del prototipo de la aplicación de telemedicina basado en las necesidades del personal de salud y la IPS Universitaria.
- Diseñar arquitectura cliente-servidor para implementación del prototipo de la aplicación.
- Definir e incorporar los ambientes de desarrollo y las herramientas necesarias para la construcción del prototipo de la aplicación de telemedicina que permita garantizar la autenticación, autorización y distinción de roles del encuentro médico.
- Implementar política de tratamiento de datos de los usuarios para el sistema de telemedicina de acuerdo a la normativa de manejo de datos clínicos.

# III. MARCO TEÓRICO

## 3.1 SERVICIOS MÉDICOS A DISTANCIA

### 3.1.1 TELEMEDICINA

La telemedicina se define como el provisionamiento remoto de servicios de salud en los procesos de promoción, prevención, diagnóstico, tratamiento y rehabilitación, por profesionales de la salud a pacientes, por medio del uso de tecnologías de la información y la comunicación con el objetivo de realizar un intercambio de información en un encuentro médico y garantizar la oportunidad, resolutividad y acercamiento en la prestación de servicios de salud a la población que presenta limitaciones de oferta o acceso [3]. La telemedicina apoya los esfuerzos por mejorar la calidad de la atención médica contribuyendo a aumentar la eficiencia del sistema de salud, a reducir gastos de los integrantes del mismo sistema al evitar la necesidad de transportarse para llegar al lugar del encuentro médico, a brindar apoyo clínico remoto, a superar las barreras geográficas, a ofrecer los medios de comunicación y a obtener mejoras en los resultados de los pacientes. Es por esto que un enfoque digital en las instituciones de salud en el que la Telemedicina se dispone a brindar su potencial, hace posible la disminución de costos del sistema de Salud, contribuye al crecimiento de la integralidad de la unidad básica habilitable del Sistema Único de Habilitación, incrementa el acceso y la oferta a las comunidades marginadas y satisface el cumplimiento de atención médica con la demanda de los beneficiarios de la institución en salud de manera accesible y equitativa [4], convirtiéndose así en una herramienta atractiva e impactante para garantizar el éxito en las misiones de salubridad de la región.

### 3.1.2 TELESALUD

La Telesalud se define como el conjunto de acciones y procesos enmarcadas en el contexto de la salud, servicios y métodos que se ejecutan en la distancia, a través de herramientas y tecnologías de la información [4]. La Telesalud abarca los procesos de Telemedicina y Teleducación en salud, por lo que, al ser más general, abarca la provisión de otros servicios como consejería en salud mental, orientación en nutrición y salud mental, entre otros [5].

### 3.1.3 TELECONSULTA

La teleconsulta se define como un encuentro médico remoto que se realiza a través de tecnologías de la información y telecomunicación entre un paciente y un médico, que se encuentran ubicados en diferentes lugares geográficos y que brinda la posibilidad de interactuar entre sí [6] .

## 3.2 HERRAMIENTAS Y PROTCOLOS DE COMUNICACIÓN EN TIEMPO REAL

### 3.2.1 WebRTC

Web Real-Time Communication (WebRTC) es un marco de trabajo gratuito y de código abierto para la Web (también tiene conjunto de herramientas de desarrollo de software para aplicaciones nativas) que permite desarrollar aplicaciones de par a par o navegador-navegador para realizar encuentros virtuales que contienen audio, video y canales de datos sin ningún software de terceros adicional, como programas complementarios (conectores o instaladores) o aplicaciones nativas, y con la menor latencia posible en la mayoría de los escenarios debido a que no utiliza servidores que medien el trasporte de la videollamada una vez esta ha empezado. El proyecto de WebRTC fue declarado como código abierto por Google en 2011 e incluye los componentes fundamentales para una comunicación de alta calidad en la Web. A través de unas API's (interfaces de programación de aplicaciones) de JavaScript estos componentes pueden ser accedidos del lado del navegador, lo que permite que los desarrolladores puedan crear sus propias aplicaciones web para el intercambio de información multimedia. Google, Mozilla, Opera y los demás navegadores son compatibles con WebRTC y los tres primeros están involucrados en el proceso de desarrollo [7].

WebRTC es una tecnología robusta que contiene un conjunto de características de comunicación de voz (VoIP), en la cual se incluyen programas necesarios para codificar y descodificar (códecs) los datos de un archivo de tipo multimedia y poder realizar la transferencia de este a través de la red. Además, provee funciones avanzadas como la cancelación del eco acústico, ocultamiento de la pérdida de paquetes (capa de red del modelo OSI) por medio de memorias temporales de fluctuación, limpieza de imágenes distorsionadas, control de ganancia automática, soporte de códec VP8 (Formato tipo WEBM, ideal para videollamadas y multimedia en línea) y reducción o supresión del ruido [7].

Para realizar la comunicación entre pares, se utilizan tres API's de WebRTC:

- *getUserMedia:* Esta API es la encargada de obtener los flujos de audio y video del dispositivo local que se usará para realizar la comunicación con el otro par. En esta API será posible obtener los canales de flujo como video y audio de todas las posibles fuentes de audio y video que ofrezca el dispositivo. De igual forma, esta API permite controlar parámetros de la salida como la resolución, la tasa de captura de cuadros o datos por

segundo [7]. En la figura 1 se muestra una representación del funcionamiento de la API de *GetUserMedia* durante una presentación de WebRTC por parte de los desarrolladores de Google.

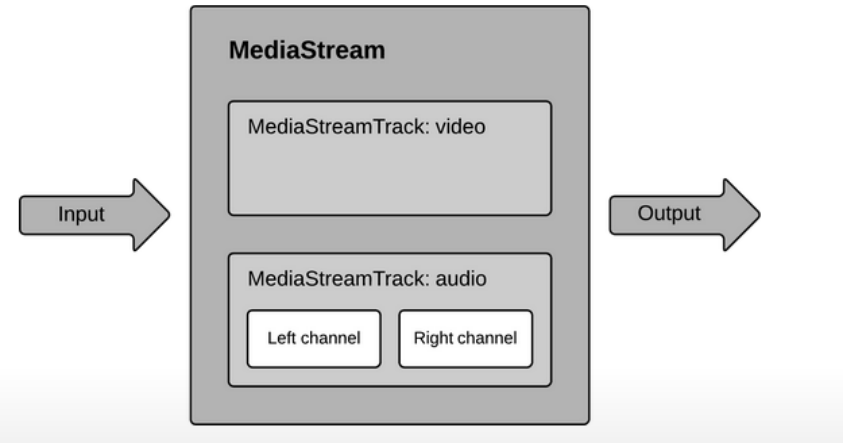


Fig. 1. Funcionamiento de API getUserMedia
Nota: fuente https://io13webrtc.appspot.com/#9 Esta figura fue tomada de la presentación de WebRTC por parte del grupo de desarrollo de Google

- *RTCPeerConnection:* Esta es la API fundamental para la comunicación entre pares. Es una API compleja, pues involucra el uso de varios protocolos y funcionalidades. Con esta API es posible realizar el proceso de comunicación en tiempo real con el otro par al hacer uso de tecnologías de "atravesamiento" de *NAT/firewalls* de los protocolos integrados de ICE, STUN/TURN los cuales son explicados más adelante en las secciones 3.2.3 y 3.2.5 respectivamente. Por medio de esta *API* se envían al otro par los flujos de audio y video locales; y de igual manera, gracias a esta API se obtendrán los mismos flujos del par remoto una vez la comunicación haya sido exitosa. La API se encarga de gestionar el funcionamiento de los motores de audio y video, verificando los códecs, memorias temporales, cancelación de ruido y demás parámetros para obtener la mejor comunicación posible [7]. En la figura 2 se observa el funcionamiento de la API de *PeerConnection* durante una presentación de WebRTC por parte de los desarrolladores de Google.

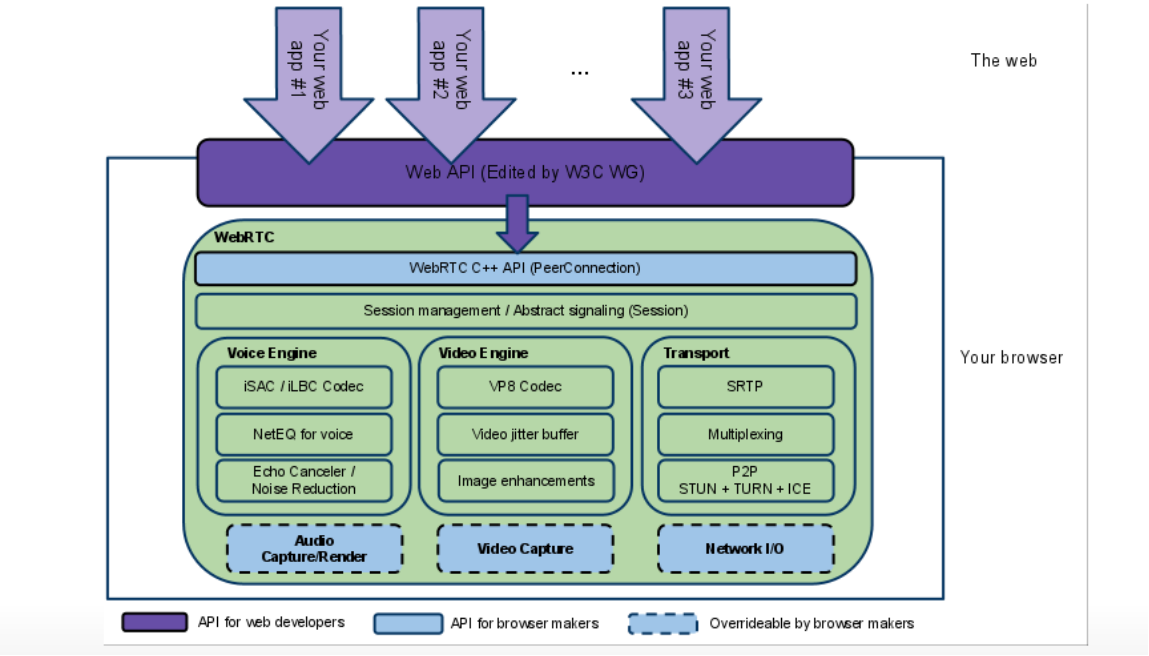


Fig. 2. Funcionamiento de API PeerConnection
Nota: fuente https://io13webrtc.appspot.com/#9 Esta figura fue tomada de la presentación de WebRTC por parte del grupo de desarrolladores de Google

- *RTCDataChannel:* Esta *API* es la encargada de la creación de canales de datos para el envío de mensajes u otro tipo datos crudos en una comunicación bidireccional. Los mensajes se envían con la latencia más baja posible y se pueden utilizar canales confiables o no confiables configurable según el tipo de intercambio de datos que se desee tener [7].

### 3.2.2 PROTOCOLO DE DESCRIPCIÓN DE SESIONES

El Protocolo de Descripción de Sesión (SDP) es una parte fundamental de las herramientas que utiliza WebRTC. Este protocolo es utilizado para negociar opciones de medios multimedia y la información de la sesión mientras se establece una conexión entre pares. Es un protocolo destinado a dar cuenta de las sesiones de comunicación multimedia con fines de anuncio de sesión, invitación a sesión y negociación de parámetros. No se encarga de entregar los datos de la videollamada per se, más bien se usa para la negociación previa al intercambio de datos asociados a multimedia entre pares como formatos, propiedades y opciones, tales como la resolución, encriptación y códecs, todos estos últimos asociados a los metadatos de la videollamada [7].

### 3.2.3 CANDIDATOS ICE

Los candidatos ICE hacen uso del protocolo de Establecimiento de Conectividad Interactiva (ICE) el cual plantea un mecanismo que permite reconocer todas las posibles direcciones de transporte y topologías de red disponibles de un par que son candidatas para realizar una comunicación en tiempo real. Hoy en día, normalmente los usuarios que tengan a su disposición usar sus dispositivos para navegar en la web no tienen una conexión directa con el internet, pues esta conexión se da gracias a dispositivos SOHO u otros dispositivos de red que tienen una IP pública desde donde se hacen las peticiones al internet. Por ello, ICE efectúa otros protocolos y servicios llamados STUN y TURN (definidos más adelante) los cuales proporcionan información acerca de las IP's públicas y puertos como candidatos reflexivos y candidatos de retransmisión respectivamente [7]. Una vez todos los posibles candidatos son descubiertos, ICE ejecuta comprobaciones de conectividad para hallar el camino más corto con el fin de realizar la comunicación con otro par a partir de varios caminos de red descubiertos [8].

### 3.2.4 NAT/FIREWALLS

La traducción de direcciones de red (NAT) es un método mediante el cual las direcciones IP se mapean de un dominio a otro con el fin de proporcionar un enrutamiento transparente a los anfitriones de red. Tradicionalmente, los dispositivos NAT se utilizan para conectar un dominio de direcciones aislado con direcciones privadas no registradas a un dominio externo con direcciones registradas globalmente únicas. Este método surgió debido a la escasez de direcciones de red versión 4 (IPv4), pues permitió mapear direcciones privadas repetidas entre diferentes dominios a diferentes dispositivos conectados a una misma red de área local por medio de un servidor DHCP, el cual se encarga de asignar direcciones IP automáticamente a todos los clientes de una red [9]. Además de solventar el problema de la escasez de IP's, se ha utilizado como un método de protección (*Firewall*) puesto que impide el libre acceso de peticiones de un dominio externo a una dirección de red privada, el cual es el caso de la mayoría de dispositivos que se encuentran conectados a un dispositivo SOHO, el cual ofrece las herramientas para la comunicación y funcionamiento de las redes locales pequeñas. Una de las técnicas más conocidas para poder sobrepasar el NAT y establecer comunicaciones en tiempo real, se conoce como el atravesamiento o recorrido de NAT, el cual permite la comunicación par a par (P2P) [10]. Un NAT posee reglas de transmisión de entrada y salida, con el fin de poder hacer el mapeo de una

red de área local a una red de área amplia y viceversa [10]-[11]. La configuración del NAT varía en cuanto a su seguridad, en donde se clasifican diferentes tipos como:

- *Cono completo:* Este tipo de NAT es el menos estricto, pues permite cualquier transmisión de entrada a un área local por parte de una dirección de transporte (IP y puerto) externo sin considerar previas transmisiones de salida [11]. En la Figura 3 se observa la tabla de topología NAT en cono completo.

```
    {NAT internal side}  |    {NAT external side}  |  {Remote host}
                         |                         |
1. (INT_ADDR, INT_PORT) => [ (EXT_ADDR, INT_PORT) -> (REM_ADDR, REM_PORT) ]
2. (INT_ADDR, INT_PORT) <= [ (EXT_ADDR, INT_PORT) <- (   *    ,    *    ) ]
```

Fig. 3. Figura de tabla de topología NAT en cono completo
Nota: fuente https://doc-kurento.readthedocs.io/en/latest/knowledge/nat.html

- *Cono restringido por dirección:* Este tipo de NAT permite la transmisión de entrada según una específica IP que haya realizado una transmisión de salida previa hacia la dirección de transporte externa. Sin embargo, permite la transmisión de entrada con un puerto distinto al inicial [10]. En la Figura 4 se observa la tabla de topología NAT en cono restringido por dirección.

```
    {NAT internal side}  |    {NAT external side}  |  {Remote host}
                         |                         |
1. (INT_ADDR, INT_PORT) => [ (EXT_ADDR, INT_PORT) -> (REM_ADDR, REM_PORT) ]
2. (INT_ADDR, INT_PORT) <= [ (EXT_ADDR, INT_PORT) <- (REM_ADDR,    *    ) ]
```

Fig. 4. Figura de tabla de topología NAT en cono restringido por dirección
Nota: fuente https://doc-kurento.readthedocs.io/en/latest/knowledge/nat.html

- *Cono restringido por puerto:* Es el NAT tipo cono más restrictivo, pues tanto la dirección como el puerto deben corresponder con la transmisión de entrada de una dirección de transporte externa. Por lo que previamente debe existir una regla de transmisión de salida hacia esa dirección de transporte externa [11]. En la figura 5 se observa la tabla de topología NAT en cono restringido por puerto.

```
    {NAT internal side}  |    {NAT external side}  |  {Remote host}
                         |                         |
1. (INT_ADDR, INT_PORT) => [ (EXT_ADDR, INT_PORT) -> (REM_ADDR, REM_PORT) ]
2. (INT_ADDR, INT_PORT) <= [ (EXT_ADDR, INT_PORT) <- (REM_ADDR, REM_PORT) ]
```

Fig. 5. Figura de tabla de topología NAT en cono restringido por puerto
Nota: fuente https://doc-kurento.readthedocs.io/en/latest/knowledge/nat.html

- *NAT Simétrico:* Este tipo de NAT es el más estricto. Se comporta de la misma manera que un NAT restringido por puerto, pero con una excepción. Cada transmisión de salida hacia una dirección remota asigna un uvo puerto aleatorio en el lado externo del NAT. Esto significa que dos transmisiones consecutivas desde el mismo puerto local a dos direcciones de transporte remotos diferentes tendrán dos puertos de origen externos diferentes, incluso si la dirección de transporte de origen interno en la transmisión de salida igual para ambos [11]. En la figura 6 se observa la tabla de topología NAT simético.

```
    {NAT internal side}  |    {NAT external side}  |  {Remote host}
                         |                         |
1. (INT_ADDR, INT_PORT) => [ (EXT_ADDR, EXT_PORT1) -> (REM_ADDR, REM_PORT1) ]
2. (INT_ADDR, INT_PORT) <= [ (EXT_ADDR, EXT_PORT1) <- (REM_ADDR, REM_PORT1) ]
...
3. (INT_ADDR, INT_PORT) => [ (EXT_ADDR, EXT_PORT2) -> (REM_ADDR, REM_PORT2) ]
4. (INT_ADDR, INT_PORT) <= [ (EXT_ADDR, EXT_PORT2) <- (REM_ADDR, REM_PORT2) ]
```

Fig. 6. Figura de tabla de topología NAT simétrico
Nota: fuente https://doc-kurento.readthedocs.io/en/latest/knowledge/nat.html

En la figura 7 se puede observar el proceso de mapeado y atravesamiento de NAT cuando 2 pares desean mantener una comunicación sin mediación de servidores. En el lado izquierdo se muestra a 2 pares conectados antes de realizar el atravesamiento de NAT. En el centro se observa el proceso de atravesamiento y a la derecha muestra la conexión entre los 2 pares luego del atravesamiento.

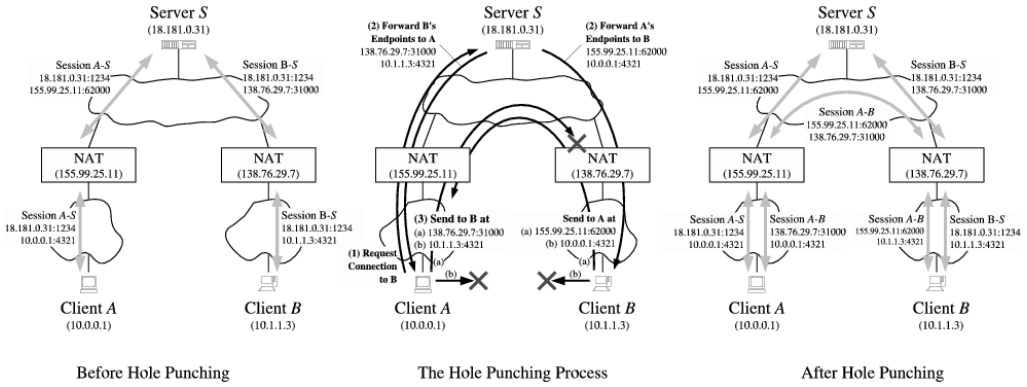


Fig. 7. Figura de atravesamiento de NAT entre 2 pares.
Nota: fuente *https://archive.is/u7His*

### 3.2.5 STUN/TURN

STUN y TURN son protocolos complementarios al marco de trabajo de ICE [11], pues gracias a estos protocolos es posible obtener los candidatos reflexivos y de retransmisión. Para la obtención de estos candidatos se deben implementar servidores que permitan sobrepasar las restricciones NAT y los firewalls y poder establecer la conexión entre los pares. En el caso de un servidor STUN, se utiliza únicamente para obtener los candidatos reflexivos de cada par, este tipo de servidor solo gestiona peticiones STUN [12]. Una vez ha sido realizada esta tarea, no es necesario más su uso en el transcurso de una comunicación en tiempo real [7]. El servidor TURN será utilizado como un retransmisor de la información en caso de que las direcciones de red de alguno de los pares sean simétricas, en este caso el servidor debe ser utilizado el tiempo que dure la videollamada y puede gestionar peticiones tipo STUN y TURN [13]. Según estadísticas dadas por Varun Singh, director ejecutivo de Callstats.io, aproximadamente el 22% de las llamadas deberán hacer uso de este servidor, mientras que el resto de las llamadas podrán ser realizadas sin ningún servidor durante la misma [7].

### 3.2.6 SEÑALIZACIÓN

En el contexto de las comunicaciones en tiempo real, la Señalización es un proceso no estandarizado que se utiliza para el intercambio de los parámetros y datos necesarios para que la comunicación pueda surgir. Este proceso se puede ver como un par ficticio que se encarga de introducir a los pares que desean realizar la comunicación, pues servirá de mediador para dar inicio a este encuentro. Lo realmente importante de este proceso es que puede ser “desechado” pues una vez la comunicación entre los pares inicia, la presencia de este par “ficticio” ya no es necesaria. Debido a que es un procedimiento agnóstico, cualquier tecnología podría ser utilizada,

desde protocolos como HTTP hasta medios como *Whatsapp* o correo electrónico podrían tomar el papel de Señalizadores. Una de las metodologías más utilizadas para suplir la necesidad de la señalización es la implementación de un servidor dedicado a la gestión de diferentes peticiones para sistemas que requieren más complejidad y automatización [14]. En la figura 8 se puede observar el mecanismo de señalización.

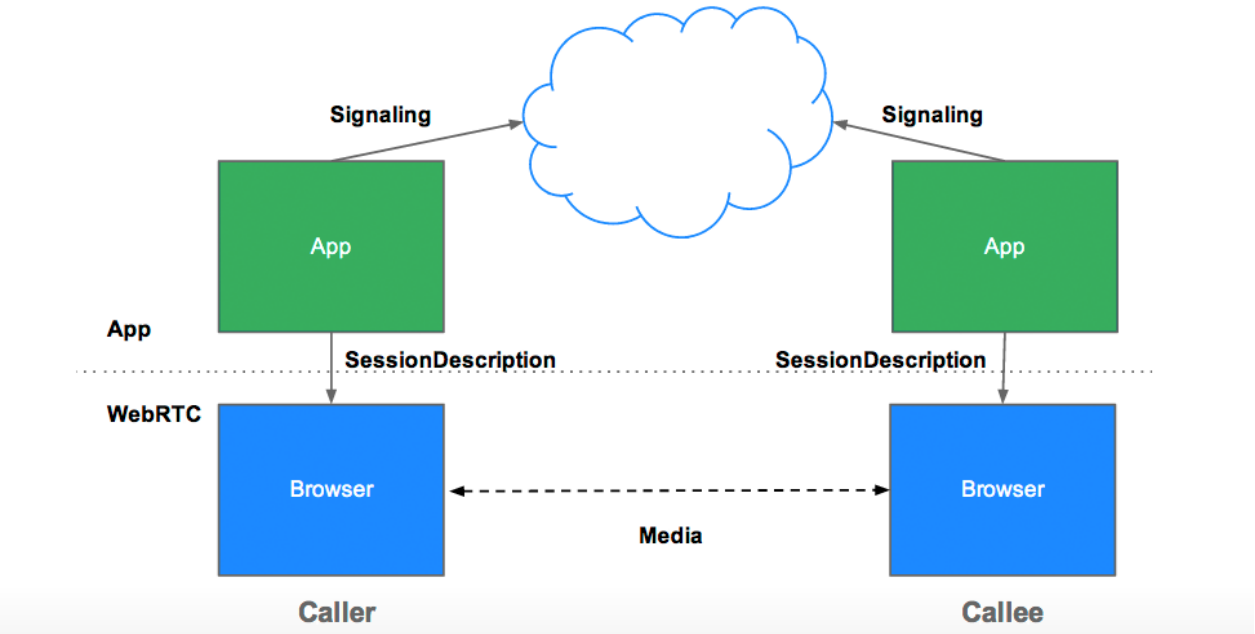


Fig. 8. Mecanismo de señalización
Nota: fuente https://meetrix.io/blog/webrtc/how-to-setup-a-signaling-server.html

### 3.2.7 SERVIDOR DE MULTIMEDIA KURENTO

El servidor de multimedia es una herramienta que se encarga de ofrecer características adelantadas o modernas a aplicaciones que trabajan con datos de tipo multimedia. Kurento por su parte, es un servidor de multimedia que es utilizado para desarrollar aplicaciones avanzadas que utilizan tecnologías como WebRTC. La necesidad de la creación de herramientas como Kurento surge como respuesta a las limitaciones que WebRTC de plano no contempla, como las conferencias virtuales, transcodificación, grabación de videollamadas, visión computarizada, entrre otros, Kurento se construyó basado en la modularización de “tuberías” (*pipelines*) en la que se pueden añadir diferentes elementos de multimedia y conectarlos como se vea conveniente. Uno de esos elementos de multimedia es el *Endpoint* de WebRTC, el cual simulará un par que se encuentra en el lugar de despliegue del servidor. En el caso de una comunicación grupal, Kurento

puede adquirir diferentes arquitecturas según sea requerido, puede actuar como como una *Single-Forward-Unit* (SFU), la cual reducirá el gasto computacional y de red de todos los pares al enviar un solo flujo de video y audio hacia el servidor o puede adoptar una arquitectura *Multipoint-Conferencing-Unit* (MCU), la cual permite que todos los pares envíen su flujo de multimedia y reciban un solo flujo de multimedia dispuesto en una grilla con todos los participantes de la reunión, lo cual reduce aún más el gasto computacional y de red de cada par pero aumenta el mismo del lado del servidor e impide la personalización de la interfaz de usuario de cada par, pues se recibe un único flujo de audio y video de todos los participantes de la reunión. Una de las características más notables es la implementación de la grabación de la videollamada de manera centralizada y remota a través de un elemento de multimedia, lo cual permite la manipulación y exportación de esta a diferentes repositorios donde sea necesitada. Kurento se comunica bajo la arquitectura cliente-servidor por medio de *Websockets* y con el protocolo de comunicación de llamada a procedimiento remoto llamado JSON-RPC 2.0, pues es un protocolo liviano que permite indicarle al servidor las tuberías y elementos de multimedia que se necesitan crear para diferentes escenarios [11]. En la figura 9 se muestran las funcionalidades del servidor de multimedia Kurento.

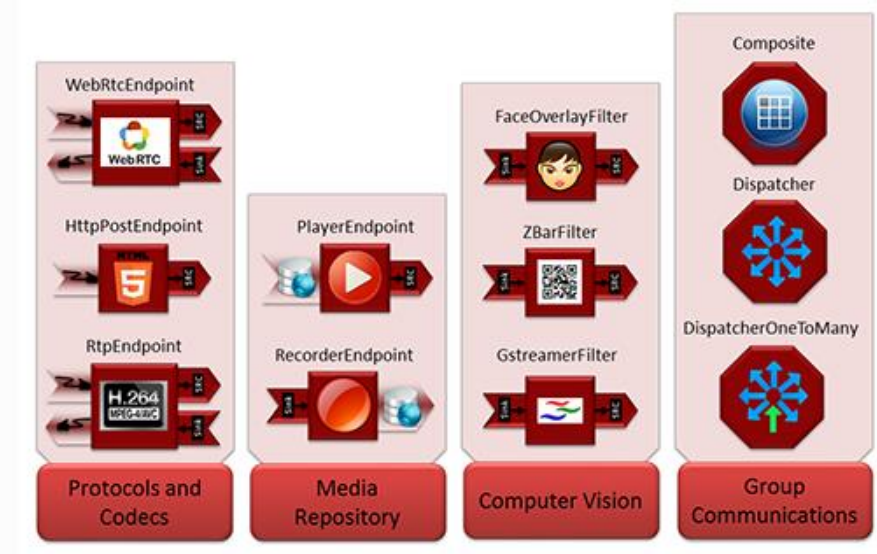


Fig. 9. Funcionalidades de servidor de multimedia Kurento
Nota: fuente https://doc-kurento.readthedocs.io/en/latest/user/intro.html

# IV. METODOLOGÍA

## 4.1 LEVANTAMIENTO DE REQUISITOS

Conforme a la metodología de trabajo del equipo de desarrollo del Hospital Alma Máter y al entorno de entrega y cumplimiento de requerimientos, se adoptó el proceso y marco de gestión de proyectos ágiles conocido como Scrum, el cual gracias a su paradigma de ejecución colaborativa incorpora al proceso de desarrollo de una metodología de trabajo que permite obtener resultados definidos en cortos periodos de tiempo, esto con el fin de poder ir añadiendo tareas, planeando la implementación de los nuevos requerimientos, y añadiendo funcionalidades del prototipo a desarrollar.

Este marco de trabajo permitió realizar el levantamiento de requerimientos de software del prototipo de la aplicación gracias a las historias de usuario primordialmente, pues en estas se condensan los requerimientos y necesidades del equipo de infraestructura de desarrollo del Hospital Alma Máter como el Propietario del Producto o el Arquitecto de Infraestructura. Sin embargo, el levantamiento de requerimientos bajo el proceso de Scrum también se realizó con diferentes técnicas:

- Observación
- Historias de Usuario
- Encuestas
- Entrevistas

La elección de tomar varias técnicas de obtención de requerimientos fue debido a que la aplicación debía ser valorada en diferentes escenarios, modalidades y con usuarios diferentes a los involucrados en el equipo de desarrollo de la aplicación.

Para el caso de la técnica de observación se procedió según la usabilidad y experiencia de usuario en ambiente de desarrollo, atendiendo a las sugerencias y reporte de fallas de ciertas características del prototipo de la aplicación de videollamadas. En el marco de Scrum, cada 2 semanas se realizaron revisiones del cumplimiento de requerimientos tratados en otras técnicas listadas anteriormente como la historia de usuario, este ciclo revisiones en el marco de Scrum se denomina como *Sprint*. El *Sprint* comprende diferentes etapas, una donde se planea y estima el tiempo y actividades que se deben ejecutar para cumplir las tareas programadas para el *Sprint.* El *Sprint* también comprende un seguimiento diario sobre los avances de las tareas que se llevan en

ese momento. Al finalizar el sprint se realizan las últimas 2 etapas, relacionadas con la revisión de las tareas cumplidas y la retroalimentación de los resultados alcanzados. En la figura 10 se observa un diagrama que indica el flujo de un *Sprint* durante 2 semanas.

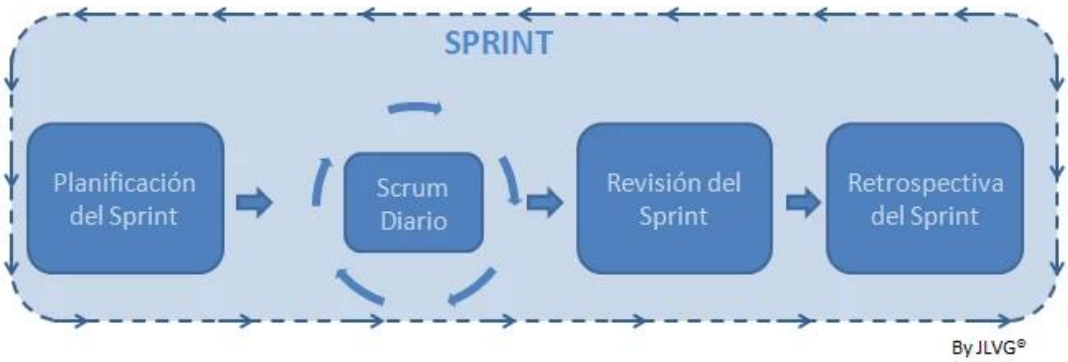


Fig. 10. Flujo de un Sprint
Nota: fuente https://openwebinars.net/blog/que-es-un-sprint-scrum/

El levantamiento de requisitos partió con objetivos claros a priori, sin embargo, debido a la metodología Scrum se fueron adoptando y levantando requisitos de manera paralela al desarrollo del aplicativo. En las revisiones se realizaron las pruebas funcionales con otro participante del equipo en la reunión de revisión. Dicho participante probaba en su dispositivo móvil o computador el funcionamiento del prototipo. A partir de su experiencia con la aplicación se aplicaba la técnica de Observación, pues con este reporte se generaban nuevas tareas para cambiar, mejorar o adaptar cierto requerimiento funcional o visual del prototipo.

En el caso de la historia de usuario, el levantamiento de requisitos se generó de manera más formal. El arquitecto TI de la institución fue el encargado de reportar los requisitos o características necesarias para cumplir durante el *Sprint*. Estos requisitos quedaban registrados en el *backlog* del *Teams Foundation Server* (TFS), el cual cumplía la función de repositorio de estados de las tareas priorizadas a realizar durante el *Sprint*. El día de planeación del Sprint, los requerimientos solicitados fueron seccionados cada uno en pequeñas tareas que permitían cumplir con esos requerimientos.

Dada la necesidad de conocer la usabilidad y la experiencia de usuarios finales (Médicos), fue necesario aplicar otras técnicas de levantamiento de requerimientos que permitieran tener presentes sus conceptos, reportes y sugerencias. Para eso, se llevó a cabo una Entrevista con médicos, personal del proceso de Tecnologías de la Información y la Comunicación (TICS) y área de desarrollo. A partir de esta entrevista, se obtuvieron características a considerar, mejoras y nuevos requisitos por implementar.

Finalmente, se añadió otra técnica de levantamiento de requisitos, la cual fue la Encuesta. En este caso, se concertó con los especialistas y el equipo de desarrollo, sobre el contenido de la encuesta. Esta se realizó con el fin de dejar diligenciado formalmente los requerimientos de los encuestados.

Una vez obtenidos los requerimientos bajo las técnicas mencionadas anteriormente, se procedió a considerar los requisitos más prevalentes y viables para el inicio del desarrollo del prototipo.

## 4.2 DISEÑO DE LA ARQUITECTURA CLIENTE-SERVIDOR

Para diseñar la arquitectura cliente-servidor se tuvo en cuenta el levantamiento de requisitos del proyecto. Al tener claros los requerimientos y funcionalidades iniciales se procedió a realizar una investigación y familiarización en la literatura en donde se observaron implementaciones existentes de las tecnologías de comunicación en tiempo real, basado en la complejidad y modularidad que el prototipo necesitó implementar, con el fin de contribuir en la definición del diseño más conveniente a desarrollar para la plataforma. Después de consolidar la investigación, se procedió a identificar el modelo de arquitectura más adecuado según los requerimientos obtenidos, que permitieran abarcar el manejo controlado y sostenible de las funciones y de la infraestructura del lado del servidor (*Backend)* y del lado del cliente (*Frontend*), eligiendo las herramientas adecuadas para la implementación del prototipo en ambos lados. Para el lado del cliente se realizó mockups para la visualización de la interfaz en dispositivo móvil y para computador. Finalmente, se evaluaron los posibles sitios virtuales o físicos donde se iban a desplegar las aplicaciones acordes a los recursos y disposiciones del Hospital Alma Máter.

## 4.3 SELECCIÓN DE HERRAMIENTAS PARA DESARROLLO Y CONSTRUCCIÓN DEL PROTOTIPO

En esta etapa se escogieron las tecnologías que permitieron la realización de la teleconsulta y las comunicaciones en tiempo real que se utilizan en la web, los criterios elegidos para esto fueron que las herramientas fueran de software libre y pudieran ser portables y compatibles con diferentes dispositivos. Luego de tener en claro eso, se realizó una familiarización con las API’s y funcionalidades que estas ofrecen para manipularlas en el prototipo según los requerimientos obtenidos. Una vez dada esa familiarización, se eligieron los marcos de trabajo y demás tecnologías para desarrollar el prototipo. Posteriormente se dio inicio a la implementación del

diseño de la arquitectura cliente-servidor. Paralelamente se realizó el acople de estrategias tecnológicas que permitieron autorizar el acceso del aplicativo basado en la modularidad y distinción de roles. Durante el proceso de desarrollo se realizaron las pruebas de usabilidad en entorno de desarrollo con tunelamiento de redes para la comprobación de funcionamiento en anfitriones remotos y otros dispositivos en la misma red de área local y también en el entorno de producción a medida que los requerimientos fueron incorporados a la plataforma, debido a la adopción de la metodología Scrum de desarrollo.

### 4.4 IMPLEMENTACIÓN DE POLÍTICA DE DATOS

En primera instancia se realizó una investigación de la política de tratamiento de datos clínicos del Hospital Alma Máter y la normatividad de disposiciones de telesalud, con el objetivo de contemplar los procesos y funcionalidades que destacan estas resoluciones y de esta forma realizar la verificación del cumplimiento y el uso de la política, acorde al levantamiento de requerimientos del proyecto. Luego de haber realizado la investigación y evaluado la política de tratamiento de datos, se procedió a desarrollar la autorización digital en la interfaz de usuario como términos y condiciones del encuentro médico, tal como se dispone en el resto de las aplicaciones virtuales que posee el Hospital Alma Máter , pues es esta autorización la encargada de servir como el mecanismo de consentimiento informado al titular o paciente responsable, y de permitir la consulta de la política de tratamiento de datos, además de restringir o autorizar el acceso al aplicativo según haya aceptado o no las condiciones de la normatividad.

En la figura 11 se puede observar el diagrama metodológico para el desarrollo del prototipo de la aplicación de videollamadas. Las etapas de este desarrollo están relacionadas entre sí debido a que algunas de ellas requerían de los avances de otras etapas, y de igual manera las etapas “primarias” como por ejemplo el levantamiento de requisitos también se realimentaron con nuevas sugerencias o cambios que ocurrieron durante la fase de implementación y diseño de la arquitectura de la aplicación. Esta retroalimentación se justifica en el marco de los objetivos de la metodología de Scrum.

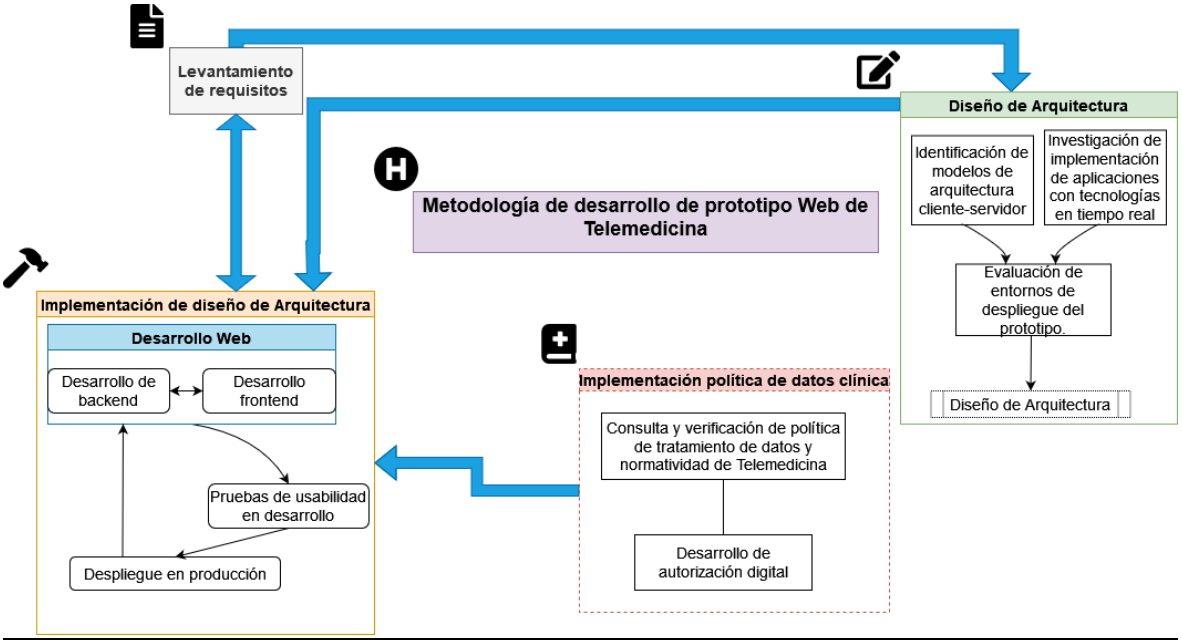


Fig. 11. Diagrama metodológico para la implementación del prototipo

# V. RESULTADOS Y DISCUSIÓN

## 5.1 LEVANTAMIENTO DE REQUISITOS

***Mediante la técnica de Observación***

Por medio de la técnica de observación se adoptaron requerimientos relacionados con la interfaz y la experiencia de usuario como el posicionamiento de ciertos elementos interactivos, la apariencia del prototipo en diferentes plataformas como computadores, celulares, etc.

Los requerimientos más importantes basados en Observación fueron:

- Aplicar diseño web adaptable de la aplicación.
- Permitir realizar el cambio de cámara durante la llamada cuando el dispositivo tenga más de una cámara para ser utilizada.
- Enviar de link de cita médica para acceder a la cita
- Mejorar experiencia de usuario basado en el reporte dado por el equipo de desarrollo durante la revisión del *Sprint*. (Este requerimiento se entiende como el conjunto de todas las reuniones durante la estancia de la práctica académica, puesto que este resultado se consolida como un requerimiento general que engloba todos los pequeños requerimientos específicos relacionados con la experiencia de usuario, por ejemplo, la ubicación de un botón)

***Mediante la técnica de Historias de Usuario***

Esta técnica fue la más utilizada para el levantamiento de requisitos. Cada 2 semanas estos fueron asignados durante el día de la revisión y planeamiento del *Sprint*. Por medio del TFS eran consignados los requerimientos y necesidades propuestas por el arquitecto de TI. Una vez fueron analizados los requerimientos, se crearon tareas pequeñas que permitieron obtener la solución de los requerimientos presentados. En la figura 12, se puede observar un ejemplo de una historia de usuario con las tareas creadas:

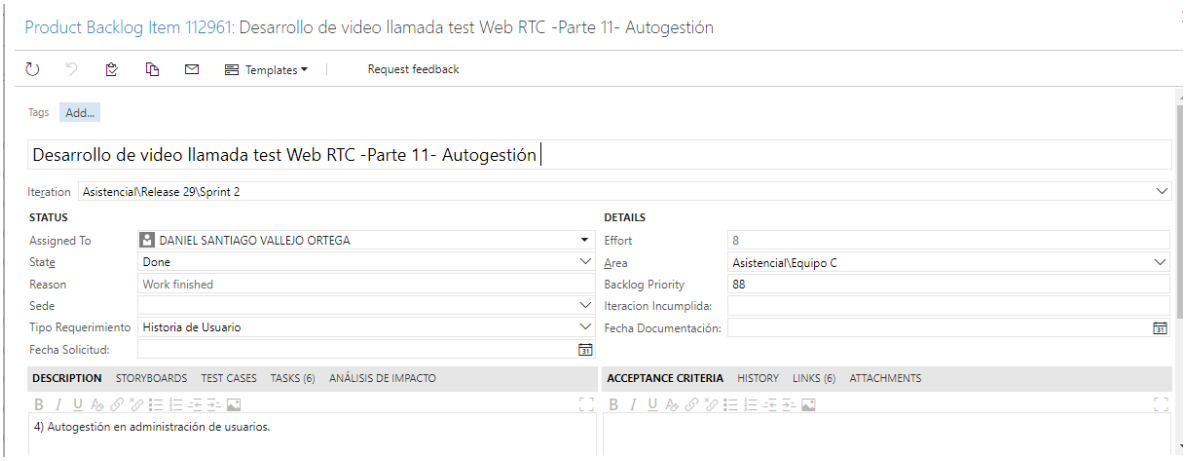


Fig. 12. Historia de usuario del *Backlog* del Producto del proyecto

En la figura 13 se puede observar el estado de las tareas para una Historia de usuario.

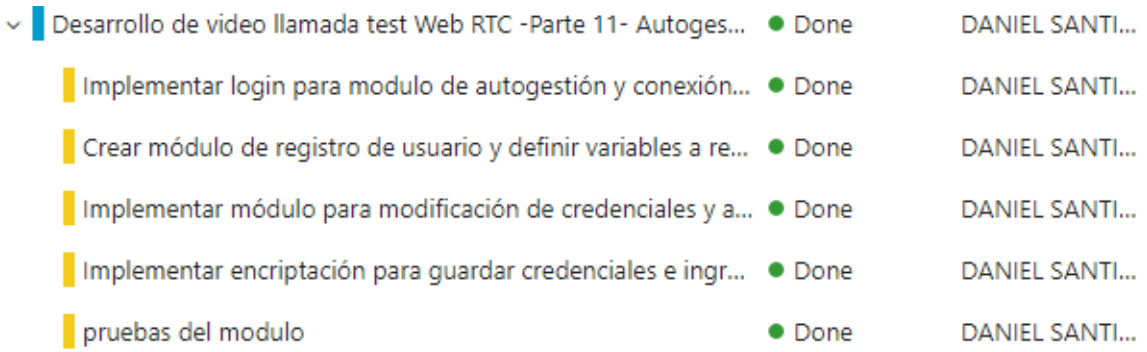


Fig. 13. Tareas creadas para la el cumplimiento de la Historia de Usuario

Debido a la cantidad de requerimientos implementados, aquí se listan algunos de los más fundamentales implementados bajo esta técnica.

- Crear acceso con distinción de roles a la aplicación de videollamadas, uno para paciente y otro para los médicos.
- Implementar autenticación y autorización segura a la hora de ingresar a la videollamada.
- Grabar la videollamada.
- Generar link de acceso a la videollamada.
- Realizar automatización para el envío del link de la cita.
- Compartir pantalla.
- Comprimir videos grabados.

***Mediante la técnica de Entrevista***

Se realizó un encuentro informal con algunos médicos del Hospital Alma Máter, el Arquitecto de Infraestructura y la Coordinadora TICS de la institución. Este encuentro permitió conocer nuevos requerimientos de los usuarios finales (médicos) de la aplicación.

- Guardar el chat.
- Guardar adjuntos.
- Permitir usabilidad en el celular.
- Validar la opción de incluir otro participante

***Mediante la técnica de Encuesta:***

Debido al carácter informal de la entrevista dada con los médicos de la IPS, se tomó la decisión de realizar una encuesta en la cual quedaran diligenciados formalmente los aspectos más importantes a considerar en el desarrollo del prototipo.

En esta encuesta se realizaron las siguientes preguntas mediante un formulario de Google a que fue presentado a tres médicos:

- ¿Tiene alguna predilección sobre el tipo de plataforma en la que la aplicación sea utilizada?
- ¿La aplicación debe permitir la posibilidad de funcionar en diferentes dispositivos digitales? (Computador, Celular, Tablet, etc).
- ¿Considera que la aplicación deba incorporar un chat que permite intercambiar mensajes durante el encuentro médico?
- Si respondió afirmativamente en la pregunta anterior, ¿Qué tipo de mensajes considera que el chat deba implementar?
- ¿Está de acuerdo en que la aplicación deba incorporar la opción de compartir pantalla?
- ¿Considera útil la posibilidad de guardar los archivos intercambiados durante la cita médica con el paciente?
- ¿Considera útil que la aplicación ofrezca la posibilidad de grabar la cita médica cuando esta sea requerida por el paciente?

Al cuestionario respondió un solo doctor y los resultados de la encuesta se presentan en la figura 14.

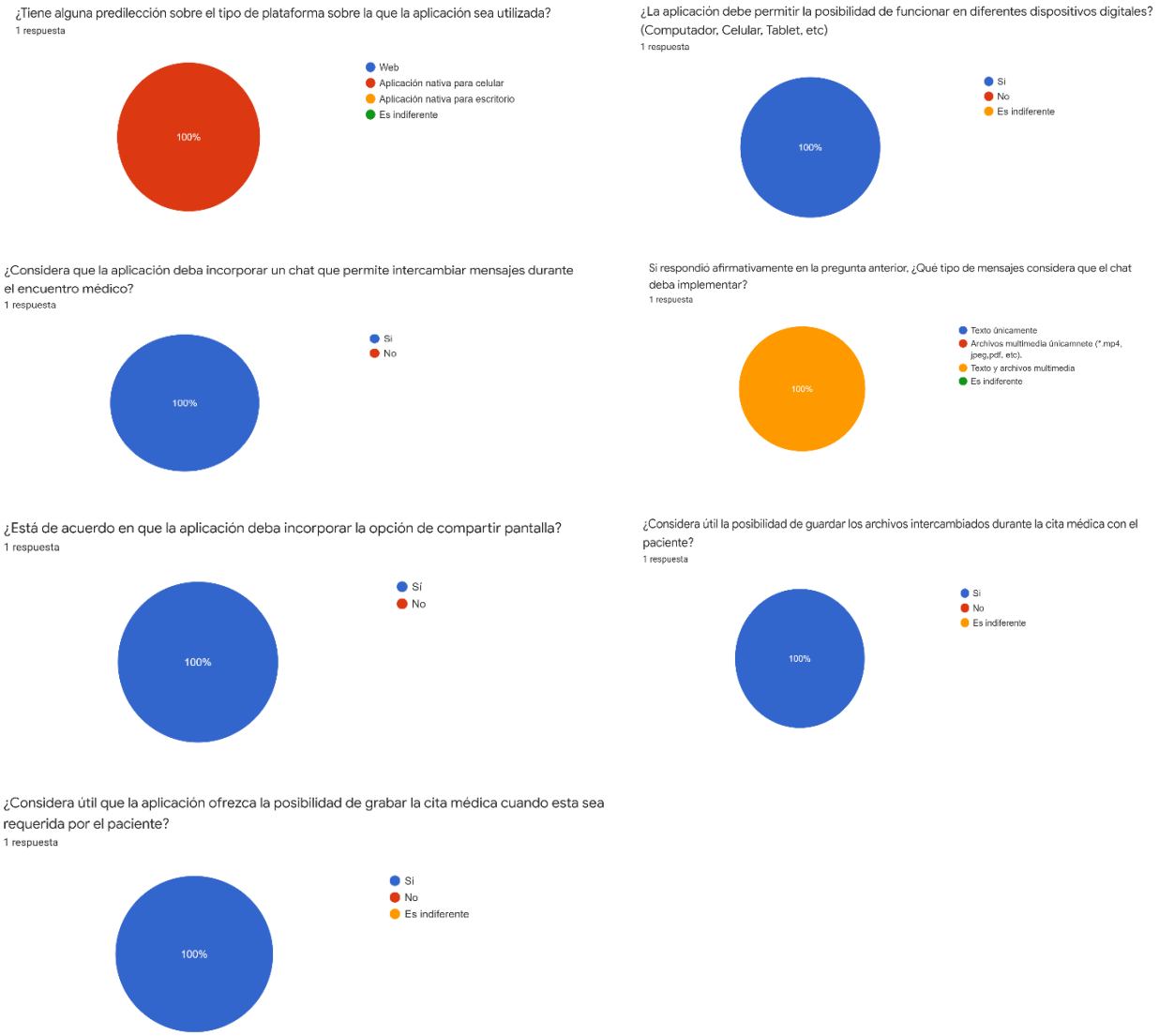


Fig. 14. Resultados de la encuesta por parte de un médico Especialista en Telesalud

Acorde a los resultados se puede ver que bajo las diferentes técnicas de levantamiento de requerimientos, se requirió construir una plataforma de telesalud integral, pues la cita médica virtual debe garantizar la comunicación entre médico y paciente que no solo permita el intercambio de audio y video, sino que permita autenticarse de forma segura basado en los roles respectivos, que garantice una óptima experiencia de usuario puesto que la aplicación será

utilizada por personas que no necesariamente tengan experiencia en el manejo de aplicaciones o el funcionamiento de internet.

Uno de los requerimientos más relevantes estuvo relacionado con la portabilidad de la aplicación. Al tener en cuenta las diferentes plataformas para “servir el cliente”, tales como una aplicación nativa para dispositivo móvil, para escritorio o web, justifica el requerimiento de que la aplicación sea web inicialmente, pues de otra manera se deberían destinar diferentes desarrollos para abarcar diferencias entre los diferentes sistemas operativos, ajustes de programas complementos o API’s dedicadas a implementar una versión de la tecnología a utilizar, lo que conllevaría más tiempo de realizar. Por esta razón el prototipo fue desarrollado como una aplicación web como se tuvo previsto en un inicio, pues todos los dispositivos con conexión a internet poseen un navegador por el cual pueden acceder al aplicativo y por ende al encuentro virtual con el médico.

La necesidad de añadir un chat dentro de la videollamada hace parte de un sistema de telesalud integral, pues por este medio se brinda la posibilidad de compartir no solo texto plano, sino también de compartir adjuntos u archivos relevantes durante un diagnóstico o control que uno de los dos participantes necesiten durante la sesión médica.

Poder grabar la sesión es un recurso útil para pacientes y médicos debido a que, a diferencia de la cita presencial, se pueden revisitar consultas o controles médicos cuando uno de los participantes del encuentro lo vea conveniente. Sin embargo, debido a que las citas médicas suelen superar más de los 15 o 20 minutos de tiempo el video puede llegar a ser de un tamaño considerable en el almacenamiento, por lo que es necesario comprimirlo.

El requerimiento de enviar un link a la cita médica está relacionado con el acceso al encuentro médico, ya que, de esta manera, los participantes de la cita podrán ingresar fácilmente a través de un navegador; es importante considerar la automatización en este caso, porque permite agilizar procesos y evitar el envío manual del link por parte de algún personal del Hospital Alma Máter.

Realizar la aplicación con distinción de roles es fundamental puesto que el prototipo abarca dos tipos de ingresos: uno basado en los médicos y el otro en sus pacientes.

Otras funcionalidades encontradas en los requerimientos tales como compartir pantalla, cambiar de cámara, son importantes debido a que les ofrece a los participantes mejores posibilidades de comunicación, añade integralidad a la plataforma, y una mejor experiencia de usuario.

Con base en los resultados y requerimientos obtenidos se eligieron los más imprescindibles para realizar el prototipo como un producto mínimo viable durante el desarrollo de la práctica académica:

- Plataforma de videollamadas con portabilidad (Funcionalidad en diferentes dispositivos digitales).
- Permitir grabar la videollamada.
- Incluir chat con posibilidad de intercambiar archivos de diferentes tipos.
- Integrar almacenamiento persistente de chat y eventual grabación de la llamada.
- Autenticación y autorización con base en los roles de los participantes de la cita médica.
- Agregar automatización para el envío del link de la videollamada.
- Compartir pantalla.
- Añadir compresión para una eventual cita grabada y automatización de la exportación del archivo al repositorio de datos persistentes.

Un requisito importante que se considera para futuras versiones de la aplicación fue la posibilidad de incorporar a otro especialista durante la cita médica, debido a que agregar más de 2 participantes involucra un cambio en la arquitectura de la aplicación y el modelo P2P.

## 5.2 DISEÑO DE ARQUITECTURA DEL PROTOTIPO

La arquitectura cliente-servidor se ha considerado como una de las arquitecturas de software distribuido más utilizada, en especial sobre aplicaciones web. Esta arquitectura puede tener variaciones a la hora de su construcción, las cuales están clasificadas en modelos que dependen de la complejidad de la aplicación. Para el desarrollo de este prototipo se hizo uso del modelo de arquitectura de N niveles, puesto que en esta aplicación se consideró necesario la separación de funcionalidades que permitan la sostenibilidad, mantenimiento y escalabilidad de la misma; además que algunos de estas funciones requieren un entorno dedicado, tales como una base de datos.

En el caso del prototipo desarrollado, las capas se dividieron en la interfaz de usuario (cliente), la capa de servicios de autenticación y señalización (agentes intermedios de funcionalidades concretas), capa de procesamiento de video y dos almacenamientos dedicados.

El uso de aplicaciones, la transformación digital y los servicios de internet han experimentado un crecimiento abrumador en los últimos años, por lo que presentar servicios ágiles se ha convertido en una necesidad. Por esta razón, el Hospital Alma Máter cuenta con una suscripción para el despliegue y control de aplicaciones en la nube de Azure, por lo que se tomó la decisión de usar el servicio de Infraestructura como Servicio (IaaS) y de Plataforma como Servicio (PaaS), pues gracias a estos el consumo de aplicaciones es más rápido y la configuración y mantenimiento de esta infraestructura es prácticamente nula, lo cual permite enforcarse en el desarrollo de la aplicación y no al mantenimiento de estaciones de trabajo o máquinas físicas.

Acorde a la forma de realizar estas aplicaciones, se evidenció un paradigma claro a la hora de realizar comunicaciones en tiempo real. El lado del cliente era servido por protocolo http y se realizaba un proceso de señalización a través de protocolos como *Long-Polling*, Eventos Emitidos por Servidor, *Websockets* [7]. Este último proceso representa una parte fundamental a la hora de implementar las comunicaciones en tiempo real, pues esta etapa resulta ser la "presentación" entre los participantes de una videollamada. Según la literatura investigada [7], [14]-[15], se observó que el protocolo más usado para la señalización es Websockets, debido a que proporciona una comunicación bidireccional y simultánea a través de una sola conexión por medio del Protocolo de Control de Transmisión (TCP) entre un servidor y la aplicación del cliente, es ligero y funciona en todos los navegadores [15]. En la figura 15 se puede observar la arquitectura de comunicaciones en tiempo real.

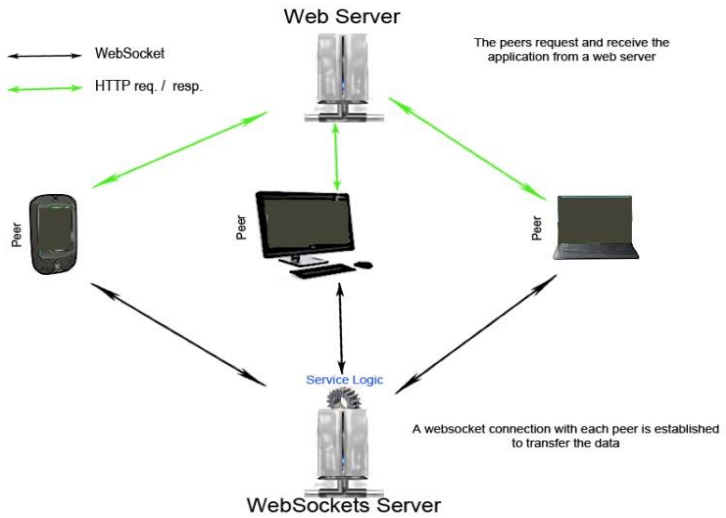


Fig. 15. Arquitectura de comunicaciones en tiempo real
Nota: fuente http://article.sapub.org/10.5923.j.web.20130201.01.html

Al ser una aplicación empezada desde cero, se planearon y diseñaron inicialmente tres aplicaciones que satisfacen los requerimientos del proyecto en la arquitectura cliente-servidor:

- *Backend* de autenticación, autorización y automatización: Primero se planeó un servidor http con incorporación de una arquitectura REST-API definida como una API de Transferencia de Estado Representacional; esta arquitectura se encarga de gestionar los diferentes tipos de peticiones de la aplicación. Bajo este diseño de *backend* se crearon los modelos de las estructuras de tipos de datos que fueron almacenados en la base de datos; estos modelos de datos contienen los atributos que identifican a los participantes del encuentro médico. Una vez creado estos modelos se procedió a implementar las rutas de autenticación y autorización que se encargan de gestionar las peticiones de la aplicación del lado del cliente por medio de una estrategia persistente debido a que esta evita almacenar sesiones del lado del *backend* en memoria (*stateless*), además que, en un eventual escenario de necesitar escalar horizontalmente esta aplicación, se vería comprometida a añadir una base de datos de tipo Redis, la cual se encarga de gestionar sesiones en memoria con un alto rendimiento, pues así puede garantizar la misma sesión ante diferentes puntos de acceso de la REST-API, lo cual aumentaba la infraestructura del prototipo. Luego de gestionar la autenticación y autorización se procedió en esta misma aplicación a implementar la automatización de tareas. Esta automatización surgió con la necesidad de realizar el envío de link de citas médicas programadas haciendo consultas periódicas a la base de datos con fin de obtener información de las citas médicas de los siguientes días de todos los pacientes. *Backend* para comunicación vía *WebSockets*: Esta aplicación se planeó para recibir los puntos de conexión en tiempo real de los participantes de la videollamada cumpliendo la funcionalidad de servidor de señalización, pues en esta instancia se podrán ubicar en salas virtuales para el intercambio de parámetros y sesiones.
- *Frontend* de la videollamada: Esta es la aplicación a la que acceden los pacientes y médicos. Es una aplicación que sirve los archivos estáticos comunes de una página web, los cuales comprenden un lenguaje de marcado para dar forma al sitio web (HTML), un lenguaje de estilos (CSS) y un lenguaje de programación para controlar la funcionalidad de la aplicación (Javascript). En este espacio los participantes podrán interactuar en tiempo real para realizar el encuentro médico. Esta aplicación se encarga de comunicarse

con las aplicaciones descritas anteriormente. Con el *backend* de autenticación se comunica a través del protocolo HTTP mientras que con la otra se realiza a través del protocolo de WebSockets.

La implementación de dos *backends* distintos se realizó debido a que no es conveniente saturar un mismo servicio en el que se podrían gestionar tareas y eventos de distinta naturaleza como las peticiones tipo HTTP, consulta a base de datos y de control de estados en tiempo real, pues estos sirven a distintos propósitos y funcionalidades para el proyecto, lo cual evita la congestión del manejo transaccional de eventos I/O y procesamiento de diferentes peticiones.

Debido a que en este prototipo se consideró el almacenamiento de datos de pacientes, médicos y citas, se planeó la creación de una base de datos que permita contener estos datos y hacerlos disponibles al *backend* de autenticación, pues el acceso al encuentro médico debe ser respaldado bajo la verificación de la existencia de usuarios que existan en dicho repositorio de datos. La base de datos se planeó bajo el modelo no relacional, debido a que, al ser un prototipo, este modelo permite flexibilidad en la creación dinámica de atributos que se van considerando importante anexarlos en cada documento durante la etapa de desarrollo, pues a diferencia de una base de datos de tipo relacional, esta ofrece más rigurosidad con los atributos definidos a la hora de generar un prototipo. En este caso se crearon usuarios ficticios para hacer las pruebas del prototipo.

A medida que se investigaba el uso de la implementación de las tecnologías para cumplir con los requerimientos establecidos se debió añadir nuevos servicios al prototipo de la aplicación. Las tecnologías de comunicación en tiempo real de audio y video requerían del uso de un servidor STUN/TURN y un servidor de multimedia para grabar la cita médica en caso de ser utilizada. Estos dos servicios se desplegaron en uno solo a través de una máquina virtual, dividiendo los puertos del grupo de seguridad de la máquina. Posteriormente, se planeó la adición de un servicio dedicado al almacenamiento de archivos y adjuntos de tipo Blobs (Objetos binarios de gran tamaño), pues a diferencia de una base de datos convencional, esta es optimizada para la manipulación de archivos los cuales pueden ser intercambiados durante la consulta médica y también podrán ser almacenados de forma permanente en la videollamada.

Con eso, los servicios adicionales que se agregaron al diseño de la arquitectura de la aplicación fueron:

- Almacenamiento en Blob o archivos en binario: Almacenamiento dedicado a guardar datos como Blobs, imágenes, videos y otro tipo de archivos no predefinidos.
- Base de datos no relacional: Aquí se almacenan los diferentes modelos de las colecciones relacionadas a la autorización y autenticación de un médico o un paciente (variables definidas en la sección 5.3) para acceder a la aplicación. Aquí se guarda la información de estos y sus credenciales. Se eligió la basa de datos CosmosDB, debido a que provee la infraestructura para almacenar datos de forma no relacional en Azure.
- Servidor de Multimedia: Aquí se procesa y redirecciona el flujo de video y audio de los participantes para ser almacenados temporalmente y luego enviados al almacenamiento tipo Blob.
- Servidor STUN/TURN: Entrega información relacionada a la IP desde donde se comunica el participante de la llamada y en caso de encontrarse en un traductor de direcciones de Red (NAT) estricto, actuar como un servidor de retransmisión de flujo de audio y video.

La arquitectura servidor-cliente quedó diseñada como se muestra en la figura 16.

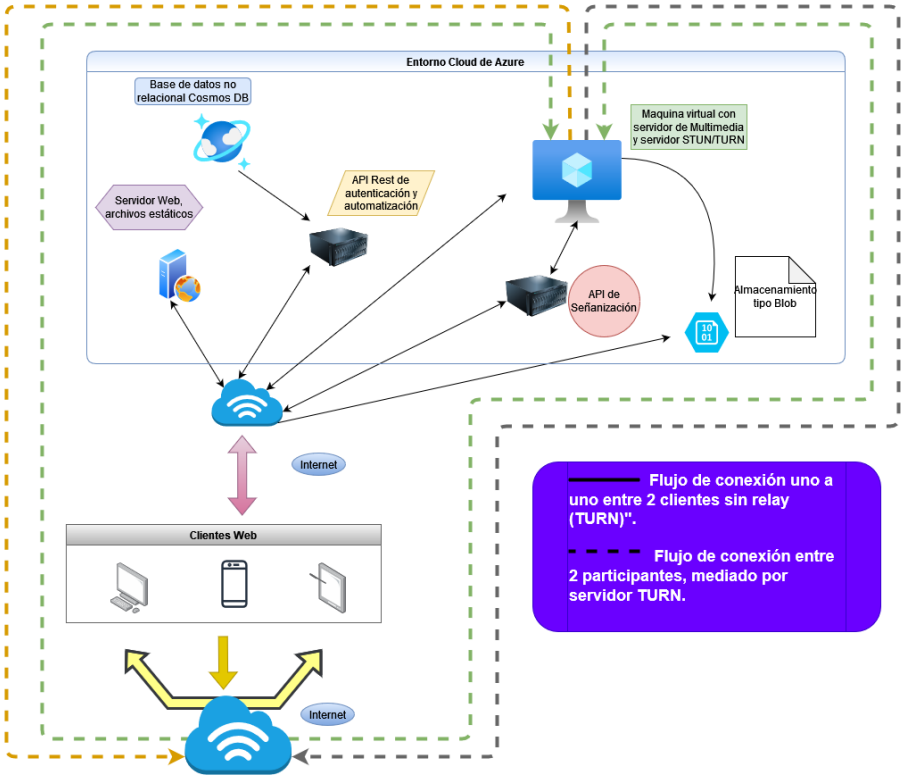


Fig. 16. Diseño de Arquitectura cliente-servidor

Como se observa en la figura 16, se ve que claramente no es un modelo de arquitectura simple, pues las funcionalidades necesitaron de diferentes entornos para la ejecución de sus tareas y evitar la congestión de tráfico de datos. En el diagrama se obtuvieron diferentes recorridos, pues el prototipo funciona diferente según la traducción de direcciones de red (NAT) y firewalls de los participantes de la videollamada, ya que en algunos hogares se tiene un dispositivo SOHO con NAT simétrico (líneas punteadas indican el recorrido una vez se realizó la señalización) el cual impide la comunicación con menos latencia, pues es necesario realizar la comunicación a través de servidor de retransmisión (TURN). En el otro camino (líneas negras sin puntear) refleja la arquitectura de una comunicación sin retransmisión debido a la ausencia de NAT y firewalls estrictos. Teniendo en consideración la importancia de conservar la autonomía de las aplicaciones, se puede ver que existe una aplicación dedicada a servir archivos estáticos (Servidor Web), la cual se encarga de enviar los archivos de la aplicación ante una petición HTTP mediante el acceso al enlace expuesto que gestiona la API de automatización. La API-REST de autenticación se comunica con el usuario final y con la base de datos, pues esta API-REST actúa como un agente intermedio que necesita garantizar la activación, verificación y existencia de una sesión en una cita. En el caso de la API de señalización, se ve que actúa como un servicio de la interfaz de usuario, pero a la vez funciona como un cliente para la máquina virtual. Esto se debe a que esta API funciona como un agente intermedio estricto de eventos, pues para mediar la etapa de “presentación” o “introducción” a la llamada permite saber la conexión de los participantes del encuentro médico y el intercambio de parámetros y sesiones que posibilitaran el inicio de la teleconsulta; sin embargo, si se desea realizar la grabación de la videollamada, el módulo de captación de flujo para centralizar los flujo de RTP de los dos participantes se deben crear otros *endpoints* en la máquina virtual que también se comunican con la API de señalización a través de protocolo de *Websockets*. También se puede evidenciar 2 caminos de entrada en el almacenamiento de Blobs, esto se debe a que el almacenamiento permanente de una eventual grabación de video deberá encontrarse aquí, de igual manera, el envío de archivos además de ser disponibles para el otro participante en el tiempo de la videollamada, fue reconocido como un requisito que estos archivos también fueran almacenados permanentemente en el Blob de la nube.

En la figura 17 se puede observar los mockups generados se para visualizar módulo de videollamada del lado del cliente en un celular y un computador.

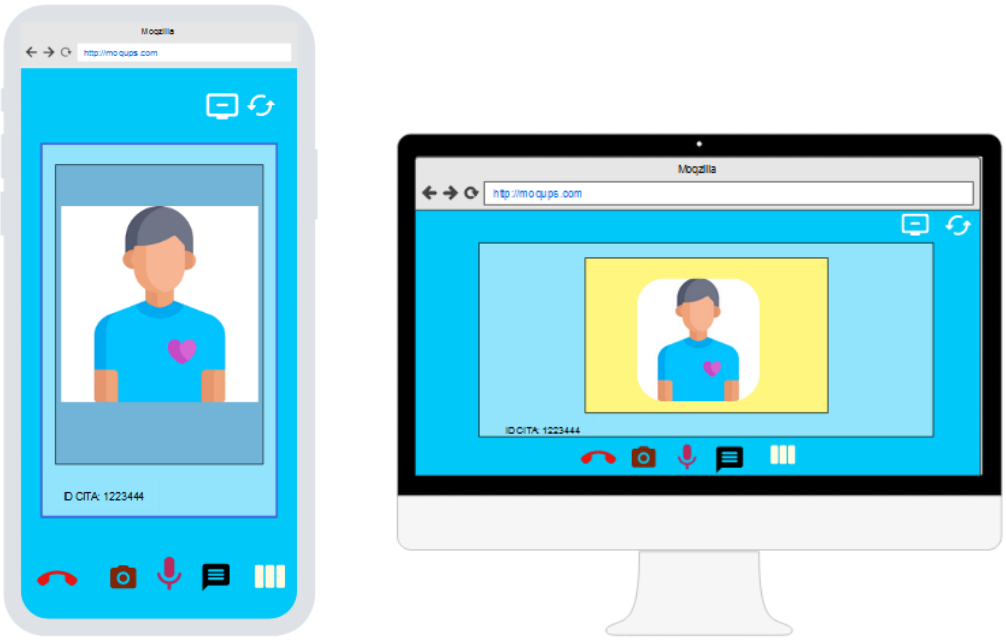


Fig. 17. Mockups del módulo de videollamada en un dispositivo móvil y un computador.

### 5.3 IMPLEMENTACIÓN DEL DISEÑO

Al considerarse el prototipo como web, las tecnologías que soportaban la comunicación de audio y video se reducían a un único candidato sobresaliente. La tecnología elegida fue WebRTC debido a que ofrece la comunicación entre pares por medio de diferentes dispositivos digitales en la web sin necesidad de instalar programas o herramientas adicionales que, de otro modo, habrían sido necesarias pero indeseables para el encuentro médico. Además, de las razones expuestas, cabe resaltar que la tecnología fue diseñada primeramente para las comunicaciones P2P, es decir, para comunicar a dos “personas”, lo cual cumple con el requerimiento de una cita médica, pues en esta se encuentra dos individuos.

Para la implementación de estas tecnologías se tuvo que realizar un estudio sobre lo fundamental de la tecnología. Como ya se mencionó en párrafos anteriores, fue necesario crear un servidor de señalización. En la figura 18 se puede ver este servidor realizando la mediación entre 2 pares que intentan establecer una comunicación. Los pares intercambian las SDPs y los candidatos ICE para poder establecer una comunicación uno a uno y transmitir sus respectivos flujos de audio y video .

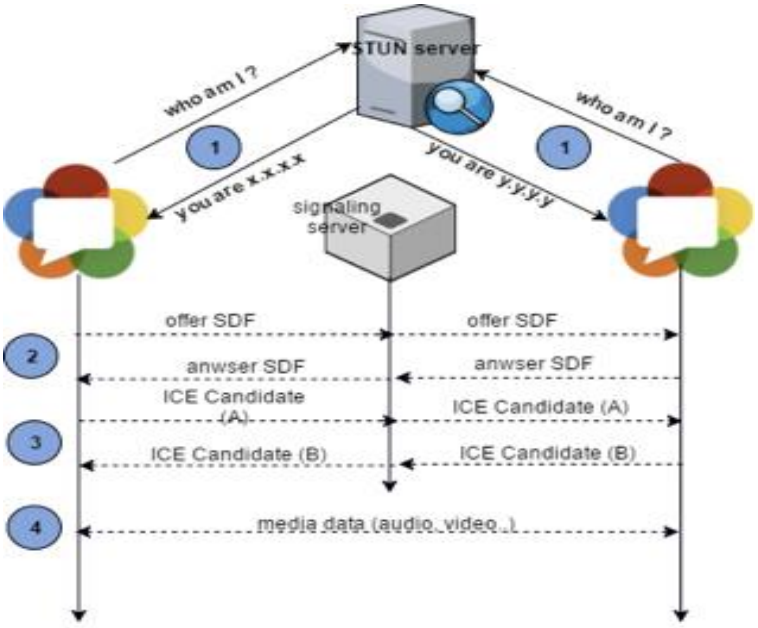


Fig. 18. Servidor de señalización para prototipo de la aplicación de WebRTC
Nota: fuente https://www.linkedin.com/pulse/webrtc-basics-components-hamit-demir

Antes de realizar una comunicación donde se intercambie audio y vídeo entre 2 pares, fue necesario realizar una "presentación" previa de estos. En términos de la tecnología implica que debe existir un intercambio y negociación previa entre los pares sobre los parámetros fundamentales que permiten a un dispositivo digital tener la suficiente información para poderle transmitir la información de audio y video al otro y de igual forma poder recibir la información homologa del par. El intercambio previo de la comunicación se realizó por medio del servidor de señalización. Para la construcción de este, se utilizó *Websockets* por razones anteriormente expuestas; sin embargo, se utilizó una versión extendida llamada Socket.IO, pues esta librería añade nuevas funcionalidades como el trabajo con sesiones o "salas", lo que posibilitó el intercambio de parámetros entre un doctor y un paciente indicado, y previno la posible confusión de intercambio de parámetros con otras citas médicas concurrentes; además, Socket.io es recursivo y no solo brinda conexión via *Websockets* (aunque en la mayoría de los casos es así ya que está soportado en prácticamente en todas las plataformas digitales que soporten una comunicación TCP) si no que en caso de que por algún motivo no se pueda, se utilizará la tecnología de *Long-Polling* como respaldo. Otra de las ventajas por las cuales se eligió esta herramienta es debido a que proporciona un mecanismo de "latido", el cual verifica periódicamente la conexión entre el cliente y servidor y realiza la reconexión en caso de existir

una desconexión. Finalmente, relacionada a la ventaja anterior, Socket.io también implementa un buffer de paquetes, que, en una eventual desconexión, almacenará datos ahí hasta que la reconexión se restablezca dentro de un tiempo límite y posterior a eso serán enviados.

En la figura 18 se observa que existen 2 clientes, los cuales se comunican con el mismo servidor de señalización. En este caso, el paciente y el doctor ocupan los roles de clientes del prototipo en el que se comunican por eventos transaccionales de socket.IO con el uso de *JSON Strings* debido a que permite intercambiar información de forma liviana. Esta comunicación se logró a través del ID generado de la cita médica encontrado en la colección de Citas de la base de datos, el cual es único y da el nombre a la sesión en la que los participantes de una cita médica se encuentran reunidos. Una vez los 2 participantes están allí se realizó el proceso de señalización, el cual, en el marco de WebRTC, consistió en el intercambio de SDP y de los candidatos ICE que se generan en la aplicación del cliente.

Los eventos captados en el servidor de señalización para el buen funcionamiento del prototipo fueron los siguientes:

- Desconexión: Evento que registra la salida de un par de la sesión (se configura el manejo de estados en el otro par), la salida se puede ocasionar por un cierre explícito de la aplicación o la pérdida de señal de internet.
- Notificación de presencia del par en el módulo de llamada: Este evento notifica a un par que está ingresando al encuentro médico, la presencia del otro miembro de la cita médica en el módulo de llamada.
- Evento de mensaje: Aquí se envían algunos mensajes especiales, candidatos ICE, SDP para la grabación y de comunicación punto a punto.
- Evento de creación o unión: En este evento se registran con el ID de la cita la unión a la sala registrada con ese ID el último par en entrar o la creación de la misma para el primer par.
- Evento de Inicio de grabación de llamada
- Evento de Detención de grabación de llamada

En este mismo servidor de señalización se implementó el módulo de comunicación para grabar la videollamada en caso de ser necesario. Para este módulo se utilizó la tecnología de código abierto de Kurento, pues es un servidor de multimedia que permite trabajar con aplicaciones avanzadas

de WebRTC y las telecomunicaciones. Como se explicó antes, en este caso el "cliente" es el servidor de señalización que implementó socket.io, el cual actúa como puente entre la aplicación que conocen los pares y el servidor de Kurento. En la figura 19 se puede observar las comunicaciones entre las aplicaciones que hacen uso de Kurento.

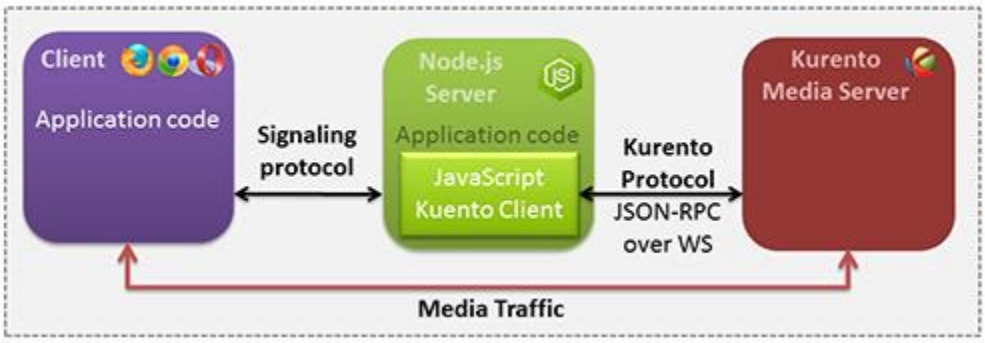


Fig. 19. Conexión entre las aplicaciones de la aplicación del cliente, el servidor de señalización y el servidor de Multimedia de Kurento

Nota: fuente https://doc-kurento.readthedocs.io/en/latest/user/intro.html

En la figura 19 se puede evidenciar la necesidad de añadir el servidor de multimedia. Para tal fin se realizó la implementación de una máquina Linux Bionic basado en la literatura de Kurento [10], haciendo uso de IaaS en Azure. Una vez implementada esta infraestructura se realizó la comunicación con el servidor de señalización para la verificación de la funcionalidad con JSON-RPC mediante *Websockets*.

Acorde a unos de los requerimientos planteados sobre la austeridad en el tamaño de los archivos en almacenamiento y la automatización, se implementaron otros servicios en este servidor de multimedia. El primero relacionado a la reducción del tamaño del video en el almacenamiento persistente de Azure Blob. Para el caso se consultaron diferentes servicios en Linux que permitieran reducir considerablemente el tamaño del archivo de la eventual grabación de la llamada. A partir de la búsqueda, se encontró la herramienta de FFMPEG, la cual es caracterizada por realizar cambios de formato de video y manipulación de archivos. Una de sus funciones más relevantes es la compresión de archivos de video. Al realizar las pruebas se obtuvo que la compresión con la configuración estándar podía ser un poco costosa y lenta en términos computacionales (aumento del uso del procesador), por lo que se tuvieron que adicionar unos parámetros para mejorar esos resultados al proceso de compresión. Se configuró el factor de rata constante (CRF) en 28 para reducir al máximo el tamaño de video manteniendo la misma altura y

ancho de video original y se añadió el parámetro de *speed* en 8 para formato de video tipo WEBM (Ideal para trabajar con HTML5), el cual se encarga de alivianar cargas de procesador y ejecutar la compresión en el mínimo tiempo posible. Teniendo en consideración la duración promedio de las citas en teleconsulta se hicieron pruebas funcionales con videos de alrededor de 25 minutos de duración, los cuales originalmente resultaban en un tamaño de alrededor de 330 MB, algo que es bastante pesado. Sin embargo, luego de aplicar la compresión por FFMPEG el mismo video llegaba a pesar alrededor de 30 MB, lo que redujo considerablemente su tamaño; además, para disminuir aún más el tamaño del video comprimido, se hizo uso de la compresión de GZIP mediante el servicio de TAR de Linux, aunque esta compresión no redujo significativamente el tamaño del video. Una vez finalizada la prueba de compresión, se procedió a investigar en los en los archivos de eventos de Kurento las notificaciones acerca de la detención de la grabación de una videollamada. Para eso se implementó una automatización con un script de *Bash* que hacía uso de una herramienta de notificación de cambio de ficheros. La herramienta en cuestión se llama INOTIFY, y esta se encarga de monitorear los eventos de detención de grabación de manera automática. Al haber realizado las pruebas de funcionamiento de la compresión y la detección del evento de grabación, se incorporó en el script los comandos para compresión y exportación del video al almacenamiento remoto. Gracias a los diferentes métodos de manipulación y subida de archivos al Almacenamiento tipo Blob de Azure, se incorporó la exportación de la videollamada grabada y comprimida por el protocolo SFTP, pues gracias a este se realizan transferencias de archivos de una máquina a otra de manera segura; para este último paso se realizó la transferencia con el dominio del blob, el usuario y la contraseña provistas por el Contenedor de ficheros creado en el portal de Azure.

Como se había mencionado anteriormente, la tecnología de WebRTC hace uso del protocolo de STUN/TURN para la obtención de las IPs de los dispositivos que se involucran en la videollamada. Por eso se implementó un proyecto de código abierto llamado Coturn, el cual hace uso del protocolo de STUN/TURN para garantizar el correcto funcionamiento de la aplicación. Este servidor fue utilizado en la misma máquina donde se encuentra Kurento, pues Coturn es un proceso que en la mayoría de los casos no requiere un uso constante del procesador y memoria de la máquina, por lo que no se vio la necesidad de implementar otra máquina que incorpore este servicio; además, en el caso de una eventual grabación, Kurento también necesita del mismo servicio de Coturn, pues como se mencionó anteriormente, Kurento simula la creación de puntos

de acceso que también utilizan los mismos protocolos que uno de WebRTC real, por lo que tener los 2 servicios resultó beneficioso, pues la latencia en el flujo y proceso de la grabación se reduce. Para comprobar el funcionamiento de Coturn se hizo uso de la página web de TrickleICE.com, la cual entrega información de los tipos de protocolos activos. En la figura 20 se ve la obtención de candidatos de tipo *srlfx* y *relay*, lo cual implica que la implementación de STUN y TURN, fue correcta.

| Time | Component Type | Foundation | Protocol Address | Port | Priority | Mid | MLine Index | Username Fragment |
|---|---|---|---|---|---|---|---|---|
| 0.004 | rtp host | 0 | udp 18c376fa-b439-495a-96bb-25c29a8fec42.local | 63776 | 126 \| 32512 \| 255 | 0 | 0 | 31195050 |
| 0.007 | rtp host | 3 | tcp 18c376fa-b439-495a-96bb-25c29a8fec42.local | 9 | 125 \| 32704 \| 255 | 0 | 0 | 31195050 |
| 0.007 | rtcp host | 0 | udp 18c376fa-b439-495a-96bb-25c29a8fec42.local | 63777 | 126 \| 32512 \| 254 | 0 | 0 | 31195050 |
| 0.007 | rtcp host | 3 | tcp 18c376fa-b439-495a-96bb-25c29a8fec42.local | 9 | 125 \| 32704 \| 254 | 0 | 0 | 31195050 |
| 0.470 | rtp srflx | 1 | udp 179.32.196.112 | 63776 | 100 \| 32543 \| 255 | 0 | 0 | 31195050 |
| 0.471 | rtp relay | 2 | udp 216.39.252.14 | 43426 | 5 \| 32543 \| 255 | 0 | 0 | 31195050 |
| 0.514 | rtcp srflx | 1 | udp 179.32.196.112 | 63777 | 100 \| 32543 \| 254 | 0 | 0 | 31195050 |
| 0.516 | rtcp relay | 2 | udp 216.39.252.14 | 39386 | 5 \| 32543 \| 254 | 0 | 0 | 31195050 |
| 0.516 | | | | | Done | | | |

Fig. 20. Obtención de candidatos srlfx y de relay para servicio de Coturn
Nota: https://webrtc.github.io/samples/src/content/peerconnection/trickle-ice/

En la figura 21 se puede observar el resultado de un video grabado con el servidor de Kurento, una vez ha sido comprimido y exportado el Blob de Azure.

Nota: Las imágenes que incluyen personas en las siguientes secciones se les aplicó un filtro por políticas de privacidad.

Fig. 21. Fotografía de una grabación de una videollamada. La videollamada fue realizada y grabada durante un Revisión de un Sprint con la Propietaria del Producto del equipo de TICS del Hospital Alma Máter

Para el caso de la aplicación del lado del cliente se consideraron varios ambientes de desarrollo de trabajo. Finalmente se escogió el uso de Angular, pues este permite crear aplicaciones modularizadas basadas en el patrón de software de modelo-vista-controlador, el cual es útil para poder separar la lógica de negocio de las vistas; además, se eligió este ambiente debido a que incorpora librerías útiles como RXJS para el manejo de estados de la aplicación. Gracias al modularidad que este marco de trabajo ofrece, fue posible, como se tenía previsto, la obtención de componentes funcionales basados en los roles de los participantes de la cita médica, una aplicación que ve el paciente y la otra, el doctor. Para ello, fue necesario destinaron 2 módulos de autenticación del lado del cliente. Uno en donde el doctor ingresa con credenciales de correo y contraseña, mientras que, para facilitar un acceso fácil y seguro, el paciente ingresa con su tipo y número de identificación. En la figura 22 se observa los módulos de ingreso para el aplicativo de telemedicina, para el médico y el paciente.


Fig. 22. En el lado izquierdo se encuentra el inicio de sesión del médico, en el derecho, el del paciente.

Una vez dado el ingreso al aplicativo, se muestra una ventana donde se le da una información relacionada a la cita y a la preparación de la misma. Este módulo se creó para que el paciente y el doctor, en el momento que estén preparados para acceder (configuren su cámara y micrófono) puedan ingresar a la cita con su otro par. En este módulo se configura la API de *getUserMedia* del marco de WebRTC, para posibilitar el acceso (a estos dos instrumentos de multimedia). En las figuras 23 y 24 se puede ver el módulo de preparación de la teleconsulta en un dispositivo móvil y un computador.

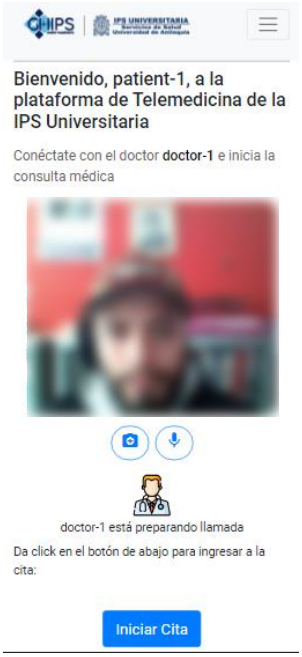


Fig. 23. Módulo de preparación en un teléfono móvil para el paciente.

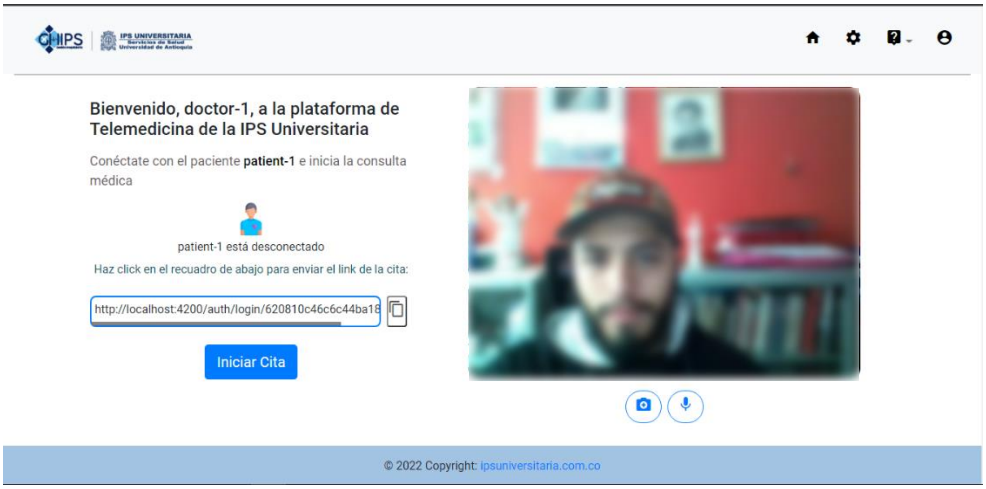


Fig. 24. Módulo de preparación en un computador de escritorio para el doctor.

Como se observa en las figuras anteriores, se observa el uso del servidor de señalización pues aquí ya se puede notar los estados en tiempo real del otro par del encuentro médico (desconectado, preparando llamada y en llamada). Se puede ver un estado de “desconexión” cuando aún el par no ha ingresado al aplicativo, otro de “preparación” el cual significa que el par se encuentra en el mismo módulo verificando su cámara y micrófono antes de ingresar, y por último el estado de “llamada”, que es el módulo donde se realiza la cita y se intercambian flujos de audio y video gracias a la API de PeerConnection de WebRTC. Cabe aclarar que en esta

aplicación del cliente se implementaron todos los homólogos de emisores y receptores de eventos y alertas correspondientes del servidor de señalización.

Para dar inicio a la comunicación entre los 2 pares, se tuvo que implementar eventos de creación de oferta y respuesta de SDP por medio de los métodos de PeerConnection. Para esto se tomó en cuenta al último participante en entrar a la videollamada, pues será este el que genere la oferta con su SDP y la envíe al otro par. Una vez el otro par la reciba, este configura la SDP recibida como SDP remota y se procede a generar la respuesta de SDP. Cabe aclarar que se decidió implementar el uso de *TrickleICE*, el cual es un proceso basado en el protocolo de ICE, pues aquel permitió disminuir el tiempo de negociación para dar inicio a la comunicación en la cita virtual, ya que son enviadas en primera instancia, las SDP que aún no incorporan todos los candidatos ICE, pero que por aparte se envían estos al otro par a mediada de que sean descubiertos y finalmente son añadidos a la SDP de manera dinámica. En la figura 25 se pueden observar los candidatos ICE y la SDP.

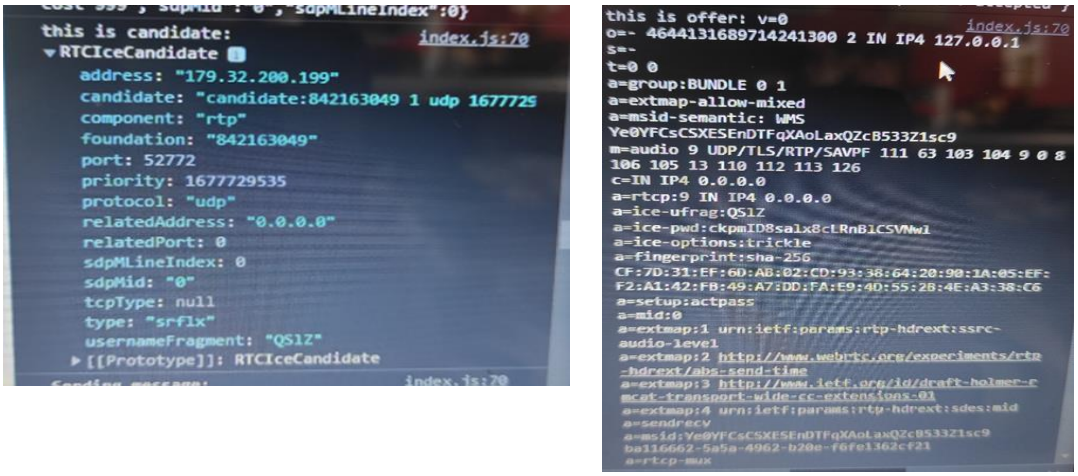


Fig. 25. A la izquierda, obtención de los candidatos ICE. A la derecha, recepción de la oferta de SDP.

Otro de los requerimientos expresados fue la creación de un chat que permitiera el envío y recepción de mensajes de texto y de otros tipos de archivos. Para este caso se hizo uso de canales de datos de WebRTC. Estos canales fueron creados bajo la propiedad *Reliable* o “confiable” la cual se encarga de enviar los datos con el protocolo de red SCTP, pues aquí es menester enviar en orden cada uno de los bytes y garantizar que cada uno de estos fragmentos que componen el mensaje, lleguen bien al otro par. Para la consistencia y aislamiento del chat durante una eventual grabación y tratando de conservar la menor latencia posible y sobrecarga durante una grabación de la llamada, se creó otra PeerConnection a parte que se encarga de lidiar con la creación de

nuevos canales de datos según se envíen diferentes archivos, y dejar que únicamente, en la grabación de llamada, se gestionen los flujos de audio y video por parte del servidor de Kurento. Una de las limitaciones que se encontró en un inicio fue para el envío de archivos diferentes a texto simple. En el caso de texto simple, son enviados archivos como textos de tipo *JSON String*, los cuales son muy livianos. Sin embargo, para archivos más pesados, se tuvieron que realizar modificaciones al canal para hacer el envío como un *arrayBuffer*, el cual contiene los datos binarios en bruto del archivo a enviar. Se halló que los canales de datos soportan un buffer máximo de aproximadamente 265KBs según las pruebas que se realizaron, por lo que en primera instancia existió la limitación del tamaño máximo del archivo a enviar. A pesar de este percance, se decidió buscar alternativas para poder enviar archivos de mayor tamaño, considerando que en este medio se puedan transmitir PDFs o archivos clínicos que pueden tener un tamaño considerable. La solución fue implementar un algoritmo de sección a trozos del archivo basado en el evento de saturación del buffer que ofrece el canal de datos creado para dicho archivo. Para ello, se envían los datos hasta que se llene el buffer y se acumulan temporalmente en el lado del cliente hasta que el buffer deje de saturarse. El otro par se encarga de recibir estos datos y almacenarlos en un arreglo temporal hasta que llegué el caracter de EOF (Fin del archivo), pues este actúa como bandera para que el par reconstruya el archivo, se proceda a la conversión de *arrayBuffer* a tipo Blob y así poder mostrarlo en el chat como un link de descarga.

Una vez solucionado este inconveniente, se vio que la interfaz de usuario sufría dilaciones y se bloqueos cuando se enviaban y recibían archivos muy pesados (por encima de 300MB). Esto es debido a que las aplicaciones en el navegador están basadas en Javascript, el cual maneja un solo hilo principal en el motor del navegador para cada aplicación. Esto fue muy relevante, ya que computacionalmente era muy costoso aplicar el algoritmo de sección a trozos y enviar esos fragmentos, y de igual manera reconstruir el Blob del archivo recibido en *arrayBuffer*. Debido a la importancia de la experiencia de usuario se implementó un *WebWorker*, pues este se encarga de gestionar operaciones dispendiosas por fuera del hilo principal de Javascript, lo cual evitó que el envío y reconstrucción del mensaje alteren negativamente la experiencia de usuario y su interfaz durante la videollamada.

Se obtuvo una interfaz sencilla basada en iconos para que los usuarios pudieran identificar fácilmente las funcionalidades que aquellos iconos ofrecen, pues fue un requisito que se

mencionó anteriormente. En las figuras 26, 27 y 28 se pueden observar las interfaces del módulo de videollamada en computador y en celular.

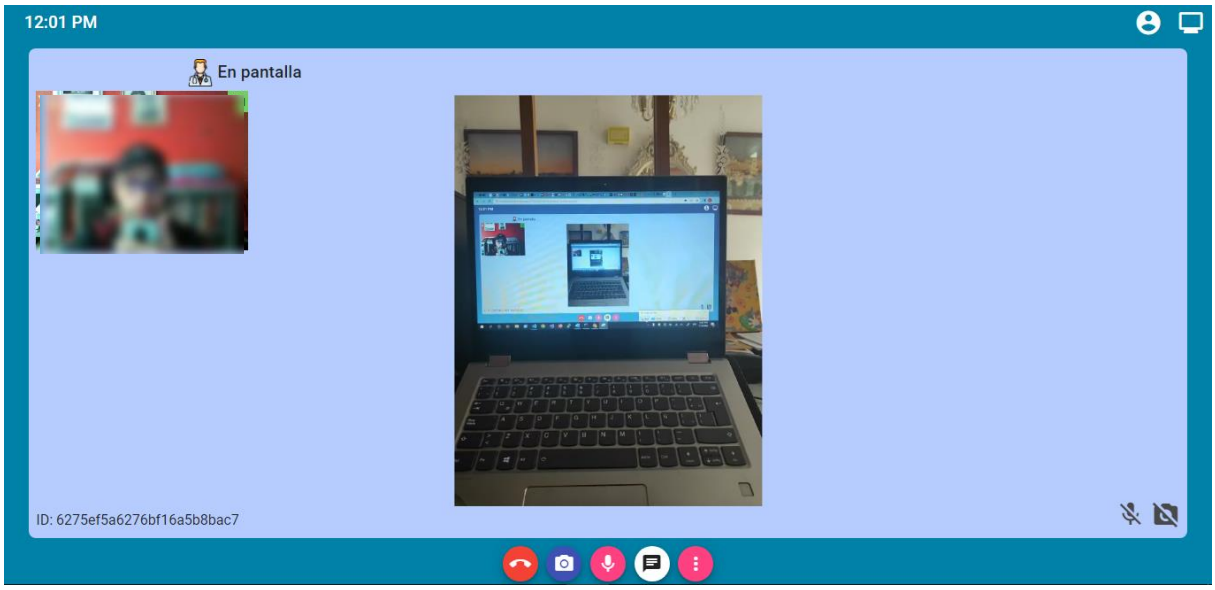


Fig. 26. Módulo de videollamada. Desde un computador de escritorio se puede observar el flujo de video de un celular desde la cámara trasera.

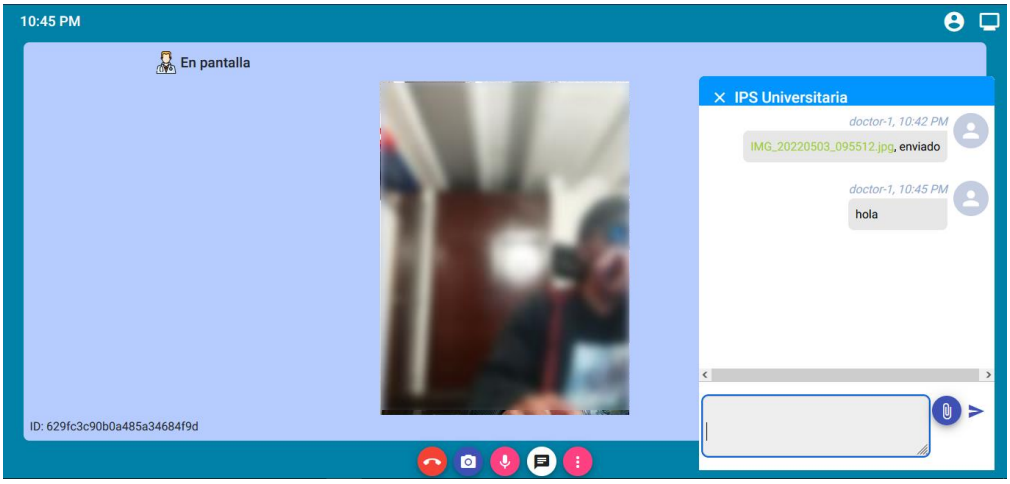


Fig. 27. Módulo de videollamada. Desde un computador se observa los botones y el chat de la interfaz.

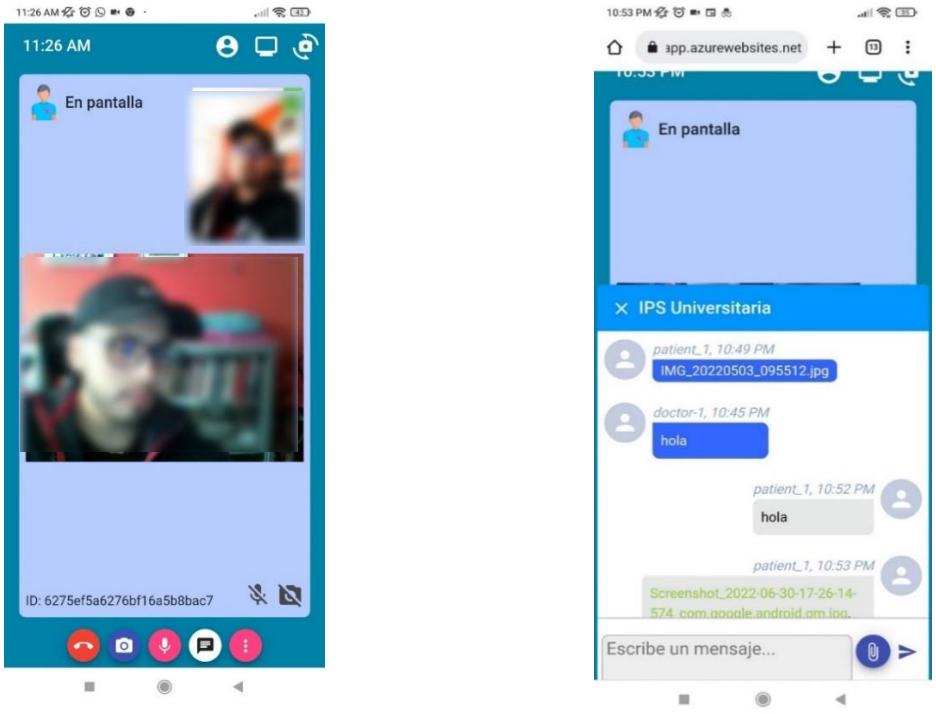


Fig. 28. Módulo de videollamada. En el lado izquierdo se observa desde un celular el flujo de video de un par en un computador. A la derecha, se observa los botones y el chat de la interfaz desde un celular.

Otro de los requerimientos relacionados al anterior fue garantizar la persistencia del chat en un almacenamiento que se encargue de servir de registro de documentos y archivos de las citas de los pacientes. El almacenamiento predilecto fue un Azure Blob en el cual se guardan por ID de la cita en un directorio todos los mensajes de textos sostenidos durante la conversación al igual que los archivos transferidos durante la misma. En este mismo folder se almacena una eventual grabación de la video llamada. Para el almacenamiento de los archivos durante la videollamada se hizo uso de APIS del Azure Blob en conjunto con un token de seguridad, las cuales se encargan de subir los archivos bajo el ID de la cita. En la figura 29 se muestra la estructura de almacenamiento para una cita médica en el Blob de Azure.

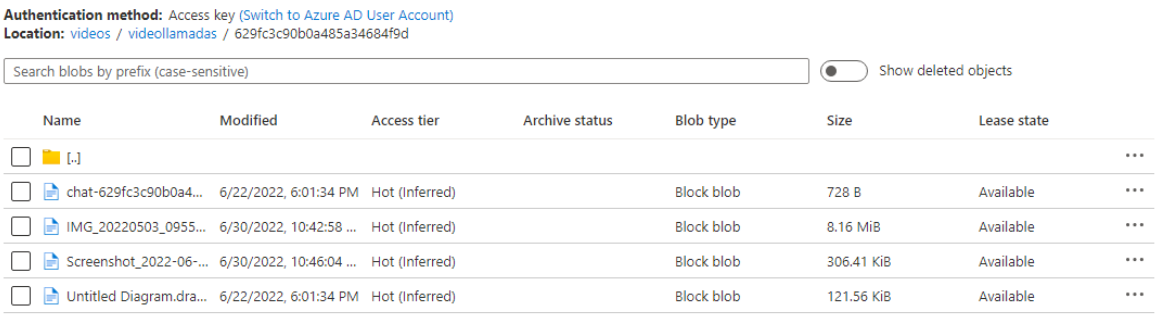


Fig. 29. Contenedor de archivos en jerarquía de carpetas. Se observa una carpeta con el ID de la cita la cual guarda el registro de archivos, chat y grabación de la llamada de dicho encuentro.

Otros requerimientos importantes fueron el poder compartir pantalla y la posibilidad de que, si se tienen más cámaras activas, poder intercambiarlas si es deseado. Para esto se utilizaron 2 APIS. Para compartir pantalla se utilizó *getDisplayMedia*, la cual permite elegir si compartir pantalla completa o únicamente la ventana del navegador en específico. Para el caso de cambiar de cámara se detectó las cámaras disponibles y en las opciones de *getUserMedia* se referencia el ID asociado a las cámaras disponibles del dispositivo digital donde se realiza la videollamada. En la figura 30 se puede observar el flujo de video de la cámara trasera de un celular.

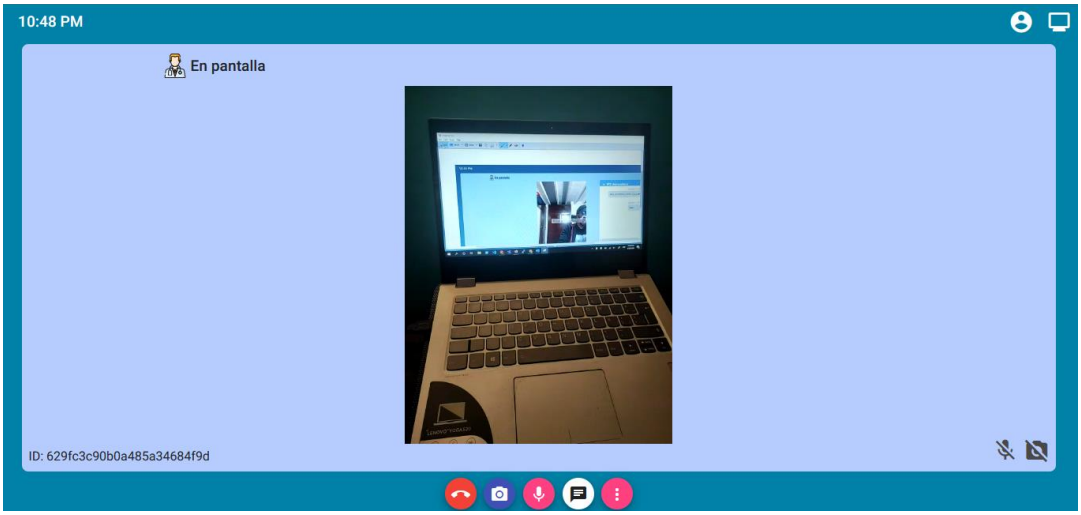


Fig. 30. Flujo de video que se obtiene al cambiar a la cámara trasera del celular y que recibe el otro par, un computador de escritorio.

En la figura 31 se observa el flujo de video de compartir la pantalla desde un computador a un dispositivo móvil.

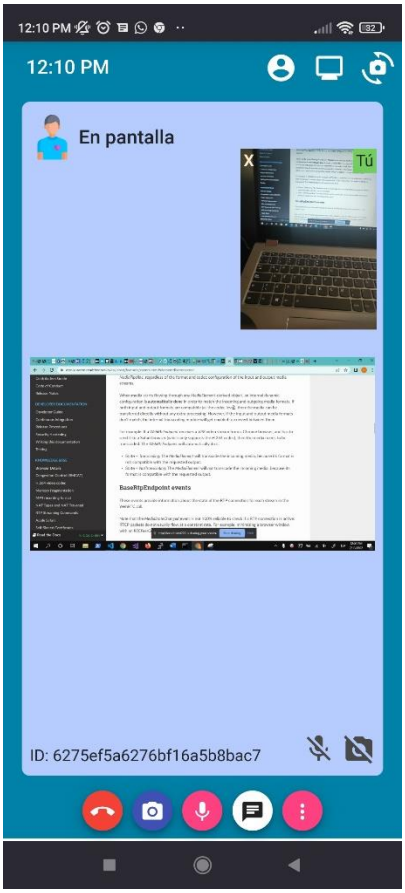


Fig. 31. Flujo de video que recibe un par al celular cuando el otro par comparte la pantalla desde un computador.

Una parte crucial en el desarrollo de software son las pruebas que se la hacen a las aplicaciones. En el caso del desarrollo de este prototipo web de telemedicina se tuvieron que hacer pruebas de funcionalidad un poco diferentes a las pruebas de desarrollo de aplicaciones convencionales, debido a que para saber si la aplicación funcionaba según la metodología planteada se necesitaba de 2 usuarios o dispositivos que mostraran el correcto funcionamiento de las APIS y la implementación de los protocolos que utiliza WebRTC. Para realizar y simular la conexión entre 2 usuarios a la aplicación se hizo uso del servicio de NGROK, el cual, gracias a su mecanismo de redirección de peticiones por medio de un túnel de red, permite servir los archivos estáticos de la aplicación del cliente en un ambiente local, hacia agentes externos de la red de área local sin la necesidad de hacer un despliegue en un sitio de hosting o en el entorno de producción de Azure. Sin embargo, esta no fue la única razón por la que se eligió NGROK como herramienta de pruebas, pues según el funcionamiento de las APIS de WebRTC, estas solo podrán ser utilizadas bajo conexiones que tengan un certificado SSL por razones de seguridad. Por eso fue necesario

configurar NGROK para que los dominios no solo generaran un túnel a las aplicaciones en ambiente de desarrollo, sino que además lo hicieran bajo el protocolo HTTPS. Con esta configuración, fue posible hacer las pruebas de la aplicación.

En el caso de la aplicación de autenticación se creó una API REST basada en el entorno de desarrollo de Express. Una de las consideraciones para usar Express fue que las aplicaciones explicadas anteriormente utilizaban Javascript como lenguaje de programación. Express funciona sobre el ambiente de NodeJS, por lo que fue evidente utilizar esta herramienta porque evita tener que integrar otra tecnología o lenguaje de programación al que ya se ha utilizado anteriormente para desarrollar la aplicación; además, Express provee funcionalidades adicionales como el enrutamiento y modularidad de componentes y agentes intermedios que facilitan la creación de la API Rest. Con el fin de garantizar la distinción de roles se generaron diferentes rutas para el acceso a la aplicación. Sin embargo, antes de crear estas rutas se generaron los campos que el documento debía tener en la colección de la base de datos. En la fase de desarrollo se utilizó un cluster gratuto de MongoDB para almacenar los documentos. Para relacionar este *Cluster* con la REST-API se utilizó Mongoose, pues esta librería permite trabajar fácilmente con consultas a bases de datos no relacionales basado en los modelos de colecciones. En esta base de datos se crearon 3 colecciones, una para almacenar los datos de doctores, otra para los pacientes y por último una colección con las citas ficticias. Para las 2 primeras colecciones se crearon campos con información básica personal y con sus credenciales. En el caso de los documentos de las citas, se consideró el uso de un ID único que hace referencia a la cita creada, además de que posee la referencia al doctor y paciente involucrados para esa cita, los cuales se encuentran en sus respectivas colecciones. Para garantizar seguridad en el acceso a la cita, se implementó la autorización por JWT (JASON-WEB-TOKEN), pues este se encarga de definir, de manera *stateless*, el acceso a los módulos a partir de una correcta autenticación de credenciales basado en la distinción de roles; además, para garantizar una correcta autenticación se implementó una tecnología de encriptación de credenciales llamada *Bcrypt*, la cual impide almacenar estas como caracteres planos visibles en la base de datos, y en vez las guarda como largos caracteres encriptados bajo el cifrado de Blowfish.

TABLA I

Colección de Modelo de pacientes en Base de Datos

| Campo | Tipo | Descripción |
|---|---|---|
| _id | ObjectID | Número de identificación autogenerado en base de datos |
| id_tipo | String | Tipo del número de identificación del paciente |
| id_numero | Number | Número de identificación |
| nombres | String | Nombres del paciente |
| numero_telefono | Number | Número telefónico de paciente |
| email | String | Correo electrónico de paciente |

TABLA II

Colección de Modelo de Doctores en Base de Datos

| Campo | Tipo | Descripción |
|---|---|---|
| _id | ObjectID | Número de identificación autogenerado en base de datos |
| email | String | Correo electrónico de Doctor |
| password | password | Contraseña del doctor para acceder a la videollamada |
| numero_telefono | Number | Número de celular doctor |
| nombres | String | Nombres del Doctor |

TABLA III

Colección de Modelo de Citas en Base de Datos

| Campo | Tipo | Descripción |
|---|---|---|
| _id | ObjectID | Número de identificación autogenerado en base de datos e identifiacdor de cada cita |
| email_paciente | String | Correo electrónico de paciente |
| email_doctor | String | Correo electrónico de doctor |
| numero_telefono_doctor | Number | Número de celular de doctor |
| numero_telefono_paciente | Number | Número de celular paciente |

Finalmente, en la API REST se añadió un módulo de automatización, el cual fue encargado de registrar las citas de los pacientes para determinada fecha. Para esto se hizo uso de la herramienta Cron para asignar Jobs basado en las fechas de las citas. Gracias a esta automatización se obtienen las citas en un archivo JSON. A cada cita se le asignó un Job en específico, el cual envía la información de la cita al paciente y al doctor por 2 medios de uso común de mensajería. Cuando el Job detecta la hora estipulada para la cita, se genera un proceso automático de envío por medio de *Whatsapp* y correo electrónico. Para poder hacer el envío a través de estos medios, se utilizaron las librerías de *Nodemailer* con la cual se realiza el envío del link de la cita a los correos de los participantes a través del protocolo SMTP haciendo uso de un correo ficticio del Hospital Alma Máter. Para el caso de *Whatsapp*, se hizo uso de la librería de *WhatsappWebJS*, con la cual se obtuvo los archivos necesarios para almacenar una sesión de *Whatsapp* Web en la aplicación y desde ahí enviar el link de la cita a los números de teléfono de los participantes del encuentro médico. En la figura 32 se observa los mensajes por el proceso de envío de cita en *Whatsapp* y Correo electrónico.

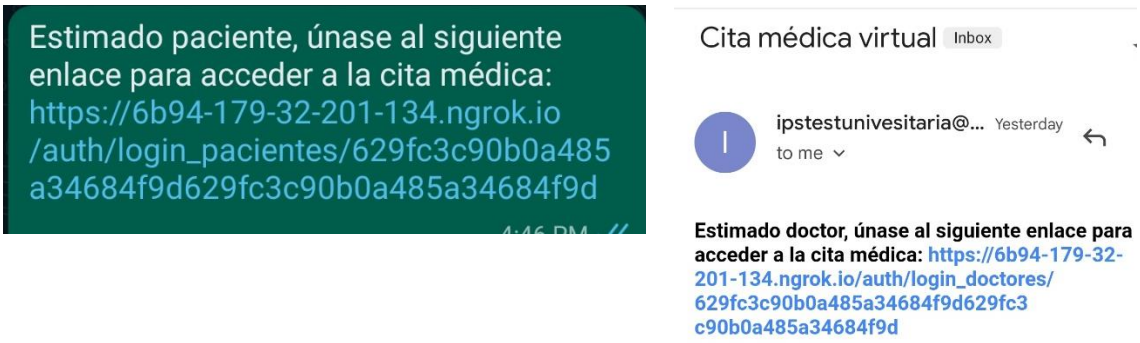


Fig. 32. A la izquierda, mensaje de Whatsapp con link de la cita. A la derecha, recepción de correo electrónico con link de la cita médica.

A medida que las funcionalidades de las aplicaciones iban siendo aprobadas por el equipo de TICS, fueron realizándose los despliegues de las aplicaciones a Azure a través de diferentes servicios que ofrece esta nube. Para el servidor de archivos estáticos, el backend de Autenticación y el de Señalización se hizo uso del Servicio de Aplicaciones, pues este asigna un recurso a partir de una tecnología escogida y su funcionalidad. Como todas estas aplicaciones mencionadas fueron construidas con Javascript, se utilizó NODEJS como la herramienta constructora de las aplicaciones en la nube. Para el caso de los servicios de Kurento y Coturn, fue necesario implementar el servicio de Máquinas Virtuales, pues estas requieren de los recursos generales de un computador y de igual manera gestionan varios puertos para la comunicación con el resto de aplicaciones. Para la base de datos en producción se hizo uso del servicio de Azure Cosmos DB, el cual permitió el almacenamiento de las colecciones creadas en la fase desarrollo del *Cluster* de MongoDB. Finalmente, para el almacenamiento de archivos se hizo uso del servicio Almacenamiento Blob de Azure, el cual almacena archivos en base a contenedores de ficheros.

Acorde a los resultados se puede ver que la aplicación fue constituida por varias herramientas que transmiten el flujo de la comunicación durante una cita médica virtual. Esto se debe a que una arquitectura de aplicación de videollamadas en el ámbito de la telesalud requiere diferentes módulos que posibiliten la integralidad de los servicios prestados por el Hospital Alma Máter.

Para la aplicación del *frontend* se obtuvo una aplicación versátil que no solo permite intercambiar flujos de audio y vídeo, sino que además expone servicios adicionales (compartir pantalla, envío de mensajes y archivos, cambiar de cámara, grabar la llamada, etc) pues es necesario brindar los medios necesarios para que la cita médica se pueda dar con la mayor integridad.

JWT fue elegido para la autorización de sesiones debido a su comportamiento *stateless* que permitió realizar las transacciones fácilmente para el acceso a los módulos de preparación y de videollamada una vez la autenticación fuera realizada. Gracias a que el token se guarda en el almacenamiento local de cada par, fue posible hacer una revalidación del token a medida que el usuario navegara por los módulos y evitar alojar sesiones en memoria del lado del servidor que aloja la aplicación de autenticación.

Se utilizó el correo electrónico y WhatsApp para poder enviar el link de la cita médica, pues estos son comúnmente utilizados como medios para recibir información importante e instantánea, lo que facilita el acceso a la cita médica.

FFMPEG permitió comprimir significativamente los vídeos de una cita médica de alrededor a 25-30 minutos, pues una CRF moderada como la elegida comprime bien sin sacrificar la calidad de video aparente, (CRF para códecs VP8, tipo WEBM). Sin embargo, si se desea obtener una mayor disminución del tamaño del video se puede cambiar el factor de ratio, es decir las medidas originales del video, lo cual terminará en últimas, reduciendo su tamaño.

### 5.4 IMPLEMENTACIÓN POLÍTICA DE DATOS

Debido a la naturaleza del prototipo que se desarrolló, fue necesario implementar una normatividad de tipo clínico que estuviera acorde a los estatutos de la ley en telesalud, la cual corresponde a la Resolución 2654 de 2019 [4]. Por eso, se solicitó al Hospital Alma Máter su política de tratamiento de datos, pues al ser una institución dedicada a la prestación de servicios de salud de alta complejidad, garantiza que el tratamiento de la información tratada durante una cita de teleconsulta se encuentra protegida bajo un enfoque clínico, como una Historia Clínica.

Una vez obtenida la política de datos, se inspeccionó esta normatividad, en donde se encontró que en el apartado 4, llamado *ALCANCE DE LA POLÍTICA,* se expresa que la política de tratamiento de datos del Hospital Alma Máter abarca a toda la información registrada en las bases de datos y la información recolectada por otros medios como la página web y/o aplicaciones desarrolladas por el Hospital Alma Máter. Debido a que el prototipo desarrollado corresponde a una aplicación desarrollada por el Hospital Alma Máter, fue posible incorporar su política de datos a través de una autorización digital, tal como se expresa en el apartado número 12 de dicho documento.

Para hacer evidente el consentimiento informado y cumplir con el derecho de los titulares a ejercer su libre autonomía, en este caso para el acceso a la teleconsulta y tratamiento de la

información en la misma, se implementó la autorización digital mediante un *checkbox* de aceptación de términos y condiciones del encuentro, lo cual es mandatorio según el Artículo 12.1 de la Resolución 2654 de 2019.

En la figura 33 se puede observar que al tartar de entrar en la cita virtual y no aceptar los términos y condiciones se restringe el acceso del usuario y se muestra el mensaje que indica que se debe aceptar dichos términos para acceder.

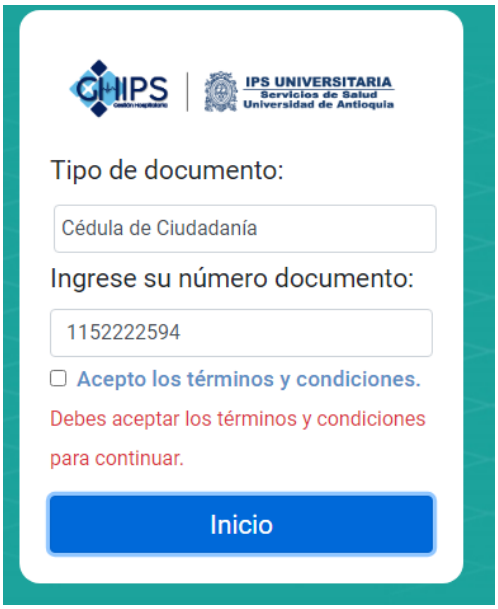


Fig. 33. Inicio de sesión del paciente denegado por no aceptar términos y condiciones

Al presionar sobre los términos y condiciones se implementó una ventana que indica el mensaje de la autorización digital, el cual expresa el Consentimiento Informado y un enlace a la política de tratamiento de datos del Hospital Alma Máter si el usuario desea consultarla. En la figura 34 se puede observar esta ventana de autorización digital.

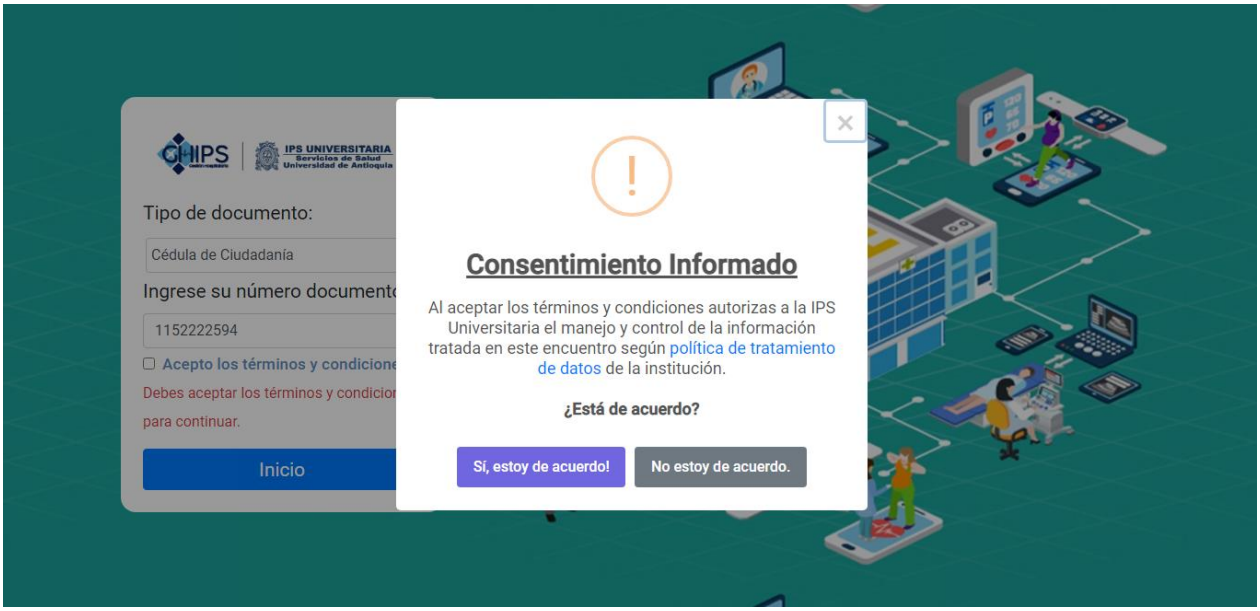


Fig. 34. Ventana de autorización digital

Al presionar sobre las palabras *política de tratamiento de datos* en la figura 34, se abre otra ventana en el navegador donde se puede leer la política de tratamiento de datos del Hospital Alma Máter. La política de tratamiento de datos se puede observar en la figura 35.

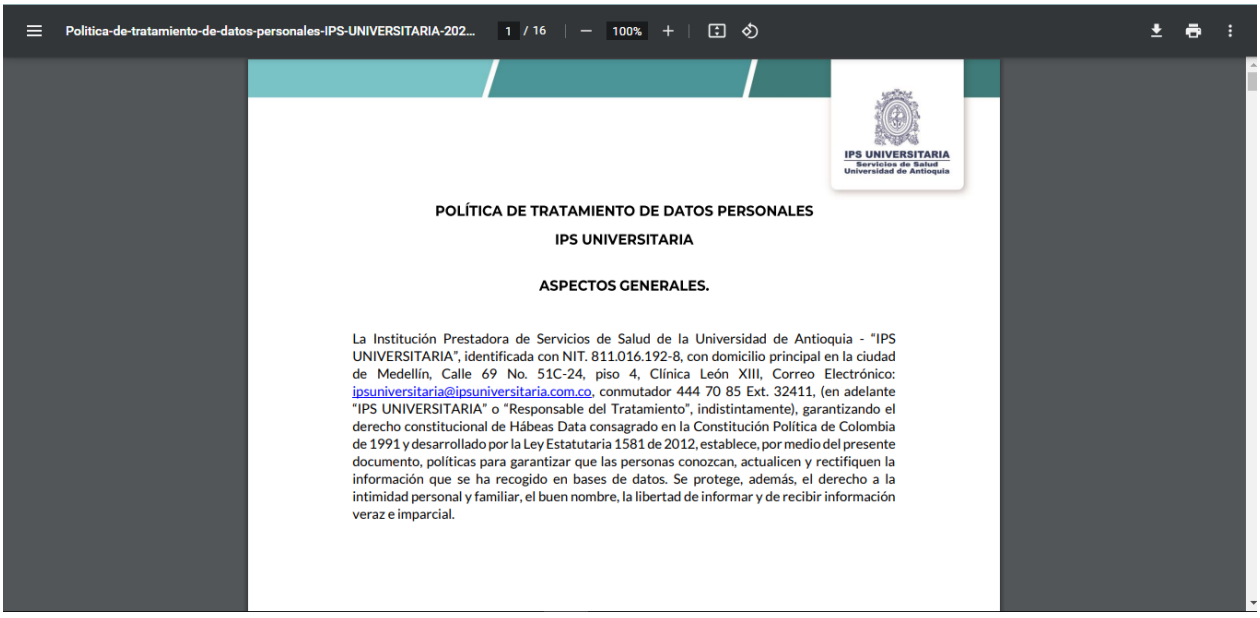


Fig. 35. Política de tratamiento de datos clínicos del Hospital Alma Máter

Uno de los requerimientos que se hicieron al prototipo fue la grabación de la videollamada. Según el parágrafo del artículo 15 Telemedicina Interactiva de la Resolución 2654 de 2019 indica la necesidad de hacer explícito al paciente el inicio de la grabación de la llamada por parte del profesional en salud. Es por esto que se implementó un aviso sobre el inicio del servicio de grabación en donde se muestra expresamente el mensaje para autorizar o declinar el inicio de la misma y expone el enlace para la visita de la política de tratamiento de datos del Hospital Alma Máter. En la figura 36 se observa el informe de aviso antes de iniciar la grabación.

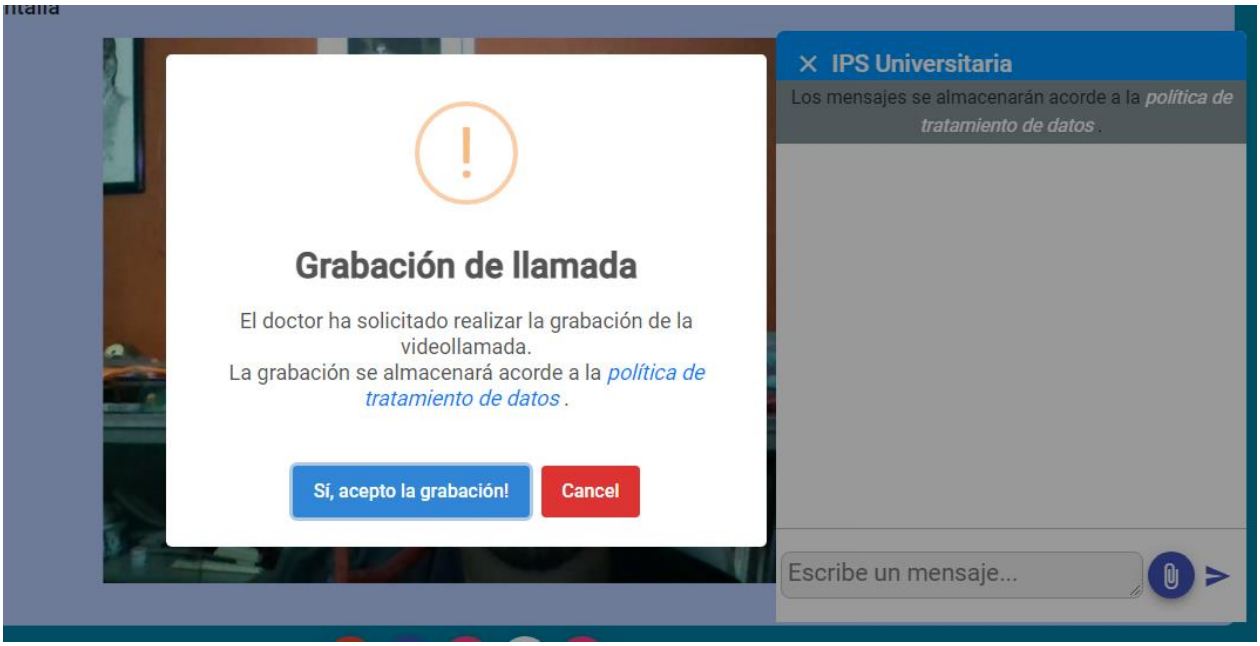


Fig. 36. Aviso de inicio de grabación de videollamada

Con estos avisos implementados, se consolidó en el ámbito de prototipado, un aval que cumple con la política de tratamiento de datos y los mandatos mostrados en los artículos de la Resolución 2654 de 2019 relacionados a la funcionalidades y requerimientos implementados en la aplicación.

# VI. CONCLUSIONES

En este trabajo de práctica académica se planteó el desarrollo de un aplicativo para realizar citas médicas por teleconsulta entre un profesional de la salud y un paciente del Hospital Alma Máter. Este objetivo fue cumplido debido a que se adoptó la metodología Scrum, la cual permitió realizar el levantamiento de requisitos de forma paralela al proceso de diseño e implementación del aplicativo. Con esta metodología, se pudo obtener un prototipo web funcional con diferentes funcionalidades, por lo que se considera oportuno pasar a una segunda etapa en donde se implemente esta aplicación, no solo como prototipo, sino como una aplicación acoplada a los sistemas de gestión de citas que tiene el Hospital Alma Máter.

El uso de diferentes técnicas de levantamiento de requisitos permitió recopilar los requerimientos desde varios enfoques, lo cual fue útil para planear el diseño y las funcionalidades que adopta un prototipo de software.

Para el desarrollo del prototipo en un tiempo límite es necesario priorizar los requerimientos más indispensables para la obtención de un producto mínimo viable.

Una aplicación de teleconsulta de videollamadas debe incorporar múltiples características y funcionalidades, más allá de la transacción de flujo de audio y vídeo; este tipo de aplicaciones debe contar con más herramientas que hagan posible una comunicación integral, como la incorporación de un chat para enviar mensajes simples o archivos, incorporar una eventual grabación de la videollamada, compartir pantalla, etc.

Para las aplicaciones en tiempo real se deben manejar diferentes módulos encargados de diferentes funcionalidades, pues esto permite crear y mantener aplicaciones con mayor flexibilidad y destreza durante el tiempo.

La arquitectura cliente servidor es un modelo de implementación bastante útil para el uso de aplicaciones. Sin embargo, dependiendo de la complejidad de esta, será necesario implementar un nivel o capa más grande que contribuya a suplir las demandas que el proyecto en cuestión exija.

En el prototipo de una aplicación de videollamadas el flujo o recorrido del proceso no está obligado a seguir un camino concreto (STUN/TURN). Sin embargo, el nivel o capa de la

arquitectura cliente servidor no cambia si este es el caso, pues los middlewares implementados siguen siendo los mismos sin importar que los participantes se encuentren bajo NAT o firewalls muy estrictos.

El uso de la computación en la nube permite desplegar y manipular software fácilmente, pues a través de APIS es posible conectarse a recursos de cómputo, hacer despliegues rápidamente sin la necesidad de tener una infraestructura física o hardware que mantener, además de que es agnóstico en cuento al tipo de arquitectura para servir una aplicación, pues en este caso se accedió a servicios como la infraestructura como servicio para las máquinas virtuales y plataforma como servicio para el despliegue de aplicaciones.

WebRTC es un marco de trabajo que posibilita la creación de aplicaciones de videollamadas en tiempo real destinadas a prestar un servicio en salud como una teleconsulta, pues al ser una herramienta agnóstica con su usabilidad, permite enfocar sus funcionalidades en un ambiente o proyecto en el marco de una utilidad médica o social que se desee brindar.

JWT es un método de autorización que se encuentra del lado del cliente, por lo que autoriza a los participantes de una cita médica al acceso de los módulos dispuestos para cada rol, sin la necesidad de almacenar sesiones en la base de datos o en la memoria del servidor de la aplicación.

La compresión de una eventual grabación de video mediante la herramienta de FFMPEG reduce considerablemente el tamaño de este mismo casi al 10% del mismo cuando se consideran casos como citas de aproximadamente 25-30 minutos, mientras que para videollamadas cortas la compresión se reduce a aproximadamente a un 60-70% por ciento del video original.

La política de tratamiento de datos es una normatividad esencial que requieren implementar todas las aplicaciones que incorporan la manipulación y gestión de datos personales, pues en esta están dispuestos los términos, condiciones y mecanismos a los que se somete un usuario o titular de una empresa o institución bajo una suscripción de servicios.

En el contexto de un sistema de telemedicina es muy importante tener en cuenta las normatividades que rigen esta materia a la hora de implementar servicios como la grabación de la videollamada, dado que en dichas normatividades se encuentran artículos y parágrafos que tienen un impacto sobre la autorización de funcionalidades y el curso de la cita, pues a través de esta

advertencia se le permite al paciente ejercer el principio de la libre autonomía al indicarle la posibilidad de grabar o no su cita.

# VII. RECOMENDACIONES

Se recomienda evaluar e implementar una fase de la aplicación que permita incorporar más de 2 participantes, teniendo en consideración el requerimiento planteado en el método de levantamiento de requisitos de entrevista, pues fue un requisito planteado por un médico especialista en telemedicina.

# REFERENCIAS


[1] K. Nader, "¿Qué es la telemedicina?" https://www.elhospital.com/temas/Que-es-la-telemedicina+8082249 (accessed Jul. 14, 2022).

[2] Luisa Correa, Paula Velásquez, Weimar Arango (2022). "Programa Hospital Inteligente" [PowerPoint]

[3] W. Giovanni, J. Barbosa, J. Sareth, and A. Gómez, "Avances en telesalud y telemedicina: estrategia para acercar los servicios de salud a los usuarios," *Acta Odontológica Colomb. Enero-Junio*, vol. 5, no. 1, 2015.

[4] Colombia - Congreso de la República. Ley 1419, Por la cual se establecen los lineamientos para el desarrollo de la Telesalud en Colombia. Bogotá: Diario Oficial, No. 47.922 (13 de diciembre 2010)

[5] T. A. C. Society, "Telemedicina y telesalud ¿Qué son la telemedicina y la telesalud?".

[6] M. R. Besio *et al.*, "Teleconsulta médica. Análisis y recomendaciones del Departamento de Ética del Colegio Médico de Chile," *Rev. Med. Chil.*, vol. 149, no. 11, pp. 1614–1619, Nov. 2021, doi: 10.4067/S0034-98872021001101614.

[7] LORETO, Salvatore y ROMANO, Simon. "Real-Time Communications with WebRTC". 1 ed. Estados Unidos. O'Reilly Media, Inc. 2014. 163p. ISBN 978-1-449-37187-6.

[8] J. Rosenberg, "RFC 5245 - Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols." https://datatracker.ietf.org/doc/html/rfc5245.html#section-2 (accessed Jul. 14, 2022).

[9] M. Holdrege, P. Srisuresh, "RFC 2663 - IP Network Address Translator (NAT) Terminology and Considerations." https://datatracker.ietf.org/doc/html/rfc2663 (accessed Jul. 14, 2022).

[10] Dan Kegel, Bryan Ford, Pyda Srisuresh, "Peer-to-Peer Communication Across Network Address Translators." https://archive.is/u7His (accessed Jul. 14, 2022).

[11] Kurento, "Kurento Documentation Release 6.16.1-dev Kurento," 2022.

[12] Internet Engineering Task Force, "RFC 3489: STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)." https://www.rfc-editor.org/rfc/rfc3489 (accessed Jul. 14, 2022).

[13] Internet Engineering Task Force, “RFC 5766 - Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN).” https://datatracker.ietf.org/doc/html/rfc5766#section-3 (accessed Jul. 14, 2022).

[14] A. Sergiienko, “WebRTC Blueprints Develop your very own media applications and services using WebRTC,” 2014, Accessed: Jul. 14, 2022. [Online]. Available: www.packtpub.com

[15] S. Panagiotakis, K. Kapetanakis, A. G. Malamos ,“Architecture for Real Time Communications over the Web.” http://article.sapub.org/10.5923.j.web.20130201.01.html (accessed Jul. 14, 2022).