Escuela Politécnica Superior

Sistema de telemedicina OneHealth

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Beatriz Asensi Prieto

Tutor/es:
Sergio Orts Escolano

Universitat d'Alacant
Universidad de Alicante

Septiembre 2018

MOTIVACIÓN

Este proyecto se puso en marcha gracias a la motivación de hacer algún sistema que tuviese como punto fuerte la seguridad. Un sistema de telecomunicación médico-paciente requiere mucha seguridad ya que los datos que se manejan son muy sensibles, tales como historiales médicos, datos personales tanto de paciente como médico, etc. Además de esto, con esta idea damos una gran facilidad a la hora de comunicación en el ámbito de la medicina, ya que muchas personas no tienen la facilidad, ya sea económica, de movilidad o familiar para poder ir presencialmente a un centro de salud.

Personalmente, este proyecto pienso que nos ha servido para aprender muchos conceptos nuevos de informática, descubrir tecnologías y enfrentarnos a lo desconocido.

AGRADECIMIENTOS

Muchas gracias a mi tutor Sergio Orts, el cual ha tenido mucha paciencia, nos ha aconsejado, guiado y nos ha cedido una parte de su tiempo para ayudarnos.

Además de esto, agradecer a todos mis amigos y familia que me han apoyado todos los años de estudio.

DEDICATORIA

Quiero dedicar este proyecto a Larry Rider por su paciencia,
apoyo y ayuda durante todos estos meses de desarrollo.
Además quiero dedicárselo a mis amigos
Sergio, Luis y André los cuales siempre me han sacado
una sonrisa y han hecho que las tardes de estudio
sean más amenas durante todos estos años de carrera.

ÍNDICE DE CONTENIDO

1. INTRODUCCIÓN 9
2. OBJETIVOS 10
   a. OBJETIVOS ESPECIFICOS 10
   a. RELACION CON ASIGNATURAS 10
2. ESTADO DEL ARTE 12
   a. SISTEMAS SIMILARES 12
   b. FRAMEWORKS 14
3. MATERIALES Y MÉTODOS 16
   a. HERRAMIENTAS SOFTWARE 16
      i. DOCUMENTACIÓN 16
      ii. DISEÑO 16
      iii. DESARROLLO 16
   b. HERRAMIENTAS HARDWARE 18
   c. DIAGRAMA DE GANTT 18
4. ESPECIFICACIÓN DE REQUISITOS 20
   a. CASOS DE USO 21
   b. DIAGRAMA DE CASOS DE USO 23
   c. DIAGRAMA DE CLASES 23
   d. ESQUEMA ENTIDAD RELACIÓN 24
   e. REQUISITOS FUNCIONALES 25
   f. REQUISITOS NO FUNCIONALES 27
   g. REQUISITOS DE INTERFAZ 29
      i. MOCKUPS 29
5. ARQUITECTURA E IMPLEMENTACIÓN 35
   a. CLIENTE 36
      i. REACT 36
      ii. WEBRTC 37
      iii. SOCKET.IO 42
   b. SERVIDOR 43
      i. NODEJS 43
      ii. SOCKET.IO 44
6. SEGURIDAD 47
7. ESTRUCTURA Y FUNCIONALIDADES 49

a. ESTRUCTURA DEL PROYECTO 49
b. FUNCIONALIDADES 50
8. CONCLUSIONES 55
a. REVISION DE OBJETIVOS 55
b. CONCLUSIÓN FINAL 55
c. TRABAJOS FUTUROS 55
9. BIBLIOGRAFÍA 57
10. ANEXOS 60
a. KURENTO 60

ÍNDICE DE ILUSTRACIONES

Ilustración 1. Janus WebRTC Gateway 14
Ilustración 2. Licode 14
Ilustración 3. Medooze 15
Ilustración 4.Mediasoup 15
Ilustración 5. Kurento 15
Ilustración 6. Diagrama de Gantt 20
Ilustración 7. Diagrama de casos de uso 23
Ilustración 8. Diagrama de clases 24
Ilustración 9. Esquema Entidad-Relación 25
Ilustración 10. Mockup comprobación de código cita 30
Ilustración 11. Mockup información cita videollamada 31
Ilustración 12. Mockup listado de pacientes conectados 32
Ilustración 13. Mockup información de una videoconsulta 32
Ilustración 14. Mockup vista videollamada 33
Ilustración 15. Mockup vista de una llamada entrante 34
Ilustración 16. Arquitectura de OneHealth 35
Ilustración 17. Arquitectura de WebRTC 38
Ilustración 18. Estructura del proyecto 49
Ilustración 19. Vista comprobar código 51
Ilustración 20. Vista ver información de cita 51
Ilustración 21. Vista listado de pacientes 52
Ilustración 22. Vista videollamada 53
Ilustración 23. Vista guardado de video 53
Ilustración 24. Ver información de videollamada 54

ÍNDICE DE TABLAS

Tabla 1. Tareas y tiempos 20
Tabla 2. Caso de uso 1.....21
Tabla 3. Caso de uso 2.....22
Tabla 4. Caso de uso 3.....23
Tabla 5. Requisito funcional 1.....25
Tabla 6. Requisito funcional 2.....26
Tabla 7. Requisito funcional 3.....26
Tabla 8. Requisito funcional 4.....26
Tabla 9. Requisito funcional 5.....27
Tabla 10. Requisito funcional 6.....27
Tabla 11. Requisito funcional 7.....27
Tabla 12. Requisito no funcional 1.....28
Tabla 13. Requisito no funcional 2.....28
Tabla 14. Requisito no funcional 3.....28
Tabla 15. Requisito no funcional 4.....29
Tabla 16. Requisito no funcional 5.....29

1. INTRODUCCIÓN

La videoconferencia en sus inicios, era una herramienta o sistema que solo unos pocos podían disfrutar, ya que esta tenía un coste muy elevado, empresas empoderadas con grandes presupuestos y conexión a Internet media podían permitirse ese lujo.

En la actualidad, la videoconferencia de alta calidad está al alcance de casi todo el mundo. Las mejoras en el acceso a Internet, las diferentes tecnologías, software y hardware, que nos permiten una mejor experiencia y facilidad para el acceso a Internet y a los diversos sistemas.

La videoconferencia ayuda a diversos sectores de la sociedad como a la educación, sanidad, informática, soporte y servicio, reuniones de negocios, etc… Esta herramienta hace que las empresas tengan menos costes a costa de un servicio barato como es esta tecnología.

En nuestro caso, vamos a aplicar esta tecnología a la medicina. La telemedicina o teleconsulta está creciendo cada día más, según la consultora Mordor Intelligence prevé que se facture 66.000 millones de dólares en 2021, con un desarrollo más elevado en el ámbito de la conectividad. La telemedicina tiene grandes beneficios tanto para las empresas como para los pacientes, ya que se puede dar atención sanitaria especializada a personas que viven en zonas lejanas a un centro de salud, ahorrar en el gasto de desplazamientos de los vehículos sanitarios, urgencias y personal médico. Gracias a estos datos, muchos especialistas y empresas del sector de la medicina confíen en la telemedicina.

2. OBJETIVOS

a. OBJETIVOS ESPECIFICOS

La telemedicina médico-paciente hoy en día aún no tiene una gran importancia. Muchas personas tienen dificultades a la hora de ir al médico, ya sea para sacar una cita con su médico o para una consulta. Estas dificultades pueden ser físicas, un persona mayor, o de tiempo, una persona que trabaja de lunes a viernes. Es por ello que uno de los objetivos de esta aplicación es facilitar de alguna manera que estas personas puedan tener una atención rápida. Además de dar un servicio rápido y cercano con el paciente, esta aplicación tiene que ser dotada con métodos de seguridad altos, ya que la información intercambiada entre paciente y médico es sensible. Esta aplicación tiene el objetivo de estar implantada en entornos web, los cuales son accesibles desde diferentes dispositivos y desde casi cualquier sitio. También, para la parte del médico, este podrá grabar la vidodeollamada para posibles comprobaciones a posteriori, además tendrá a su disposición la conversación por texto que tuvo con el paciente. Finalmente, este sistema estará integrado a una web e-Health, la cual permitirá toda la gestión de citas, consultas, etc.

a. RELACION CON ASIGNATURAS

En este apartado enumeraremos las asignaturas impartidas durante el Grado de Ingeniería Informática, las cuales han tenido una aportación directa o indirecta al desarrollo de este sistema.

Fundamentos de las bases de datos y diseño de base de datos

Estas dos asignaturas nos dieron los conocimientos básicos para la creación de una base de datos desde cero, viendo relaciones sencillas y no tan sencillas, cardinalidades y buenas prácticas para tener una base de datos robusta.

Análisis y especificación de sistemas software

Esta asignatura nos proporcionó material y herramientas para analizar una aplicación o sistema de cero y de esta manera sacar los requisitos principales y con ellos hacer diversos diagramas para visualizar mejor los objetivos y funcionalidades del sistema.

Seguridad en el diseño de software

En esta asignatura nos enseñaron las pautas básicas para proteger una aplicación o sistema, diferentes cifrados, como resolver vulnerabilidades y las diferentes catástrofes que han pasado a lo largo de la historia por no tener un sistema seguro.

Aplicaciones distribuidas en internet

En esta asignatura aprendimos tecnologías como nodejs y react, que en la actualidad son de los lenguajes más usados, además de cómo hacer un servidor rest con nodejs.

2. ESTADO DEL ARTE

a. SISTEMAS SIMILARES

Médicos y pacientes

Esto es una página web para la consulta clínica y comunicarse con el médico de forma telemática. Es una plataforma web, la cual tiene diversas características:

- Inicio de sesión con clave o certificado digital.
- Consulta de tarjeta sanitaria.
- Consulta de centro de salud (mapa, números, etc).
- Hospital de referencia.
- Médico y enfermero de cabecera.
- Citas electrónicas.
- Ver historial clínico.
- Datos de facturación farmacéutica.
- Consulta online al médico de determinadas dudas sobre la asistencia sanitaria.
- Ver información del paciente.

Policlínica IUMET

El instituto Universitario de Medicina Telemática IUMET surge para dar a los Centros de Reconocimiento de Conductores (CRC). Este sistema permite hacer consultas instantáneas de exploración psicofísica a los pacientes, dar apoyo formativo para ampliar las habilidades de los profesionales del Centro de Reconocimiento, asesoraminento y promueve la investigación en el ámbito de la seguridad vial. Además tiene un apartado de teleoftalmología.

Vida

Este es una aplicación para plataformas móvil o Tablet y está disponible para los sistemas operativos Android e IOS. Esta aplicación tiene diversas características:

- Video consultas para:
    - Preocupaciones generales de salud
    - Alergias
    - Asma

    - Infecciones sinusales
    - Herpes labial
    - Resfriados
    - Infección del tracto urinario
- Reserva de citas para la video consulta
- El médico podrá recetar al paciente

Este sistema es de pago, el cual tiene un precio de 20€ la consulta.

Sanitas

Sanitas es un seguro médico muy conocido, el cual tiene un servicio de videoconsulta llamado blua. Este sistema web también tiene lo básico de una aplicación eHealth.

- Video consulta médica con especialistas
- Programas de asesoramiento por videoconsulta
- Información de centros
- Reserva de citas
- Consulta de pruebas

Esta aplicación tiene unas cuotas mensuales o trimestrales, las cuales todas contienen el servicio de videoconsulta.

HealthTap

Este es un sistema para plataformas web y móvil con las siguientes características:

- Información inmediata de conocimientos médicos
- Respuestas inmediatas de los especialistas
- Consejos
- Noticias relevantes
- Valoraciones de medicamentos
- Consultas por chat de texto y video
- Asesoramiento de especialistas
- Tratamientos y recetas
- Consultas de laboratorio y resultados.
- Listados a medida

- Recordatorios médicos

Es una aplicación gratuita.

b. FRAMEWORKS

Janus

Janus es un software creado con el propósito de ser una puerta de enlace. Este framework permite una comunicación WebRTC entre navegadores, pudiendo intercambiar mensajes JSON y transmitir RTP/RTCP. Toda la lógica de este sistema está implementada en la parte del servidor. Janus usarse para implementar aplicaciones como priebas de eco, conferencias, grabadoras, pasarelas SIP y similares.

La licencia de este sistema es gratuita y está distribuido bajo los términos de la Licencia Pública General GNU versión 3. Este sistema está creado por la empresa Meetecho.

Ilustración 1. Janus WebRTC Gateway

Demo del programa: https://janus.conf.meetecho.com/videocalltest.html

Licode

Licode es una plataforma de comunicaciones WebRTC de código abierto creada por Lynckia.

Esta plataforma está basada en las tecnologías de WebRTC, es 100% compatible con las últimas versiones estables de Google Chrome. Al estar basado en WebRTC cualquier usuario puede utilizar esta herramienta en sus navegadores, sin la necesidad de instalar ningún plugin ni extensión. Permite incluir sala de videoconferencia, la transmisión, la grabación de cualquier cosa multimedia.

Ilustración 2. Licode

Para poder trabajar con este framework una de las restricciones es que tenemos que tener Ubuntu 14.04 LTS o Mac OS X.

La licencia de este software está bajo los términos de CC BY 3.0.

Medooze

Medooze es un servidor multimedia que te permite recibir y enviar videos mediante la tecnología de WebRTC. Este servidor esta soportado por los sistemas operativos Linux, Mac OS X y Raspberry Pi.

Ilustración 3. Medooze

Este sistema está creado por Medooze.

Licencia MIT.

MediaSoup

MediaSoup tiene como objetivo ser un SFU WebRTC, el cual recibe trasmisiones de audio y video de los participantes de una sala de conferencias y retransmite a todos los demás. Ser un módulo nodejs en el servidor, ser un SDK en la parte de cliente, ser minimalista, que todos los navegadores y ORTC existentes sean compatibles y que no tenga un protocolo específico.

Ilustración 4.Mediasoup

Este software está hecho por Iñaki Baz Castillo y José Luis Millán.

Demo:

https://demo.mediasoup.org/?roomId=xoqaerz3

Licencia ISC.

Kurento

Kurento es un servidor multimedia WebRTC, el cual permite comunicaciones grupales, transcodificación, grabación, mezcla, transmisión y enrutamiento de flujos audiovisuales. Además de eso proporciona indexación de video, realidad aumentada, simplifica la integración de algoritmos de procesamiento de terceros, haciendo más cómodo para los desarrolladores

Ilustración 5. Kurento

utilizar Kurento. Este sistema está distribuido bajo la licencia de Apache 2.0.

3. MATERIALES Y MÉTODOS

a. HERRAMIENTAS SOFTWARE

Para la realización de este proyecto hemos utilizado diferentes herramientas software para crear la documentación y almacenar los diferentes archivos.

i. DOCUMENTACIÓN

Para la realización de la documentación hemos utilizado:

- Microsoft Word 2013

  Microsoft Word es un programa informático orientado al procesamiento de textos. Este fue creado por la empresa Micorsoft y está incluido en la suite informática Microsoft Office.

ii. DISEÑO

- Draw.io

  Draw.io es un sistema de código abierto para crear diagrama de todo tipo. Este está creado por Gaudenz Alder y David Benson. En nuestro caso hemos lo hemos utilizado para la realización de diagramas de casos de uso, da clase, E-R y mockups.

iii. DESARROLLO

- Sublime

  Sublime Text es un editor de texto y editor de código fuente, el cual está escrito en C++ y Python para los plugins que tiene integrados. Este programa no es de código abierto y se debe de adquirir una licencia para su uso continuado, aunque existe una versión de evaluación que no tiene fecha de caducidad que es la que nosotros estamos utilizando.

- Visual Studio Code

  Visual Studio Code es un editor de código fuente desarrollado por Microsoft para distintos sistemas operativos. El programa es de código abierto y gratuito, aunque la descarga oficial está bajo software propietario.

- GitHub

GitHub es un sistema de control de versiones, se utiliza normalmente para la creación y almacenamiento de código fuente de programas. El software está escrito por Ruby on Rails, pero actualmente es propiedad de Microsoft.

- Ubuntu 16.06

  Ubuntu es un sistema operativo de código abierto, distribuido por Linux y basado en una arquitectura Debian. Fue creado por Mark Shuttleworth, director de la compañía Canonical. Ubuntu es distribuido de manera gratuita y es financiado mediante diversos servicios vinculados al sistema y mediante servicio técnico.

- Oracle VM VirtualBox

  Oracle VM VirtualBox es un software de virtualización para arquitecturas x86/amd64. Es un producto multiplataforma, el cual permite ser ejecutado en varios sistemas operativos (Mac OS, Windows, Linux u Oracle Solaris). Pertenece a Oracle y tiene la versión privativa y la versión Open Source.

- Node.js

  Node.js es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor basado en el lenguaje de programación ECMAScript, asíncrono, con I/O de datos en una arquitectura orientada a eventos y basado en el motor V8 de Google. Fue creado por Tyan Lienhart Dahl. Está bajo una licencia MIT.

- React

  React js es una líbreria de Javascript creada por Facebook, el cual permite crear interfaces de usuario, aplicaciones SPA (single page application) más eficientes. Está librería funciona tanto en el lado de cliente como en el de servidor. Utiliza el patrón MVC (Modelo-Vista-Controlador). Fue creado por Jordan Walke y está bajo una licencia MIT.

- Google Keep

  Google Keep es una aplicación para organizar de manera sencilla información a través de tarjetas. Está disponible para Android y en Google Drive como aplicación web. Ha sido desarrollada por Google Inc.

- Google Drive

  Google Drive es una aplicación de almacenamiento de archivos. Está disponible a través de la web y aplicaciones Android e IOS. Tiene 15GB gratuitos para almacenar diferentes

archivos ampliables mediante diferentes planes de pago. Fue creado Google Inc bajo la licencia EULA.

- SplitCam

  SplitCam es un programa para la división de señal de video. Con este programa se pueden añadir efectos visuales, sustituir la cámara por videos o fotos. Es un sistema gratuito.

- Xampp

  Xampp es un programa de software libre, con el cual se gestiona principalmente bases de datos MySQL, el servidor web Apache y los inérpretes para lenguajes de script PHP y Perl

- Workbench

  Workbench es un programa de diseño de bases de datos, que integra desarrollo software, gestión, mantenimiento y diseño de bases de datos MySQL.

b. HERRAMIENTAS HARDWARE

Para la realización de este Trabajo de Fin de Grado hemos tenido que utilizar dos dispositivos muy importantes:

- Cámara

  Hemos usada la cámara integrada en el ordenador portátil, BisonCam, NB Pro.

- Micrófono

  Hemos usado el micrófono integrado del ordenador portátil, Realtek High Definition Audio.

c. DIAGRAMA DE GANTT

En esta apartado mostraremos el diagrama de Gantt, el cual es una herramienta para programar y planificar las tareas que se van a hacer en el proyecto. Este diagrama es un gráfico de barras horizontales, diferenciadas por las tareas a realizar en secuencias de tiempo concreto. Tendremos en la parte superior de la gráfica (eje X) las fechas entre inicio y fin del proyecto y en el eje Y el listado de tareas.

Las tareas realizadas son las siguientes:

Nombre de la tarea	Fecha de Inicio	Fecha Final	Duración (Días)
Creación de base de datos	05/11/2017	13/11/2017	8
Creación de mockups	22/06/2018	02/07/2018	10
Investigación de WebRTC	16/06/2018	11/07/2018	25
Demo webRTC	18/06/2018	25/06/2018	7
Creación de diagramas	04/07/2018	07/07/2018	3
Investigación de frameworks	06/07/2018	29/07/2018	23
Investigación de Kurento	06/07/2018	15/08/2018	40
Desarrollo del servidor	12/07/2018	15/07/2018	3
Integración de Kurento	15/07/2018	22/07/2018	7
Investigación de errores Kurento	22/07/2018	15/08/2018	24
Desarrollo de cliente	05/08/2018	24/08/2018	19
WebRTC con Socket.io	20/08/2018	28/08/2018	8
Grabación de stream con WebRTC	28/08/2018	30/08/2018	6

Memoria	28/06/2018	05/08/2018	38

Tabla 1. Tareas y tiempos

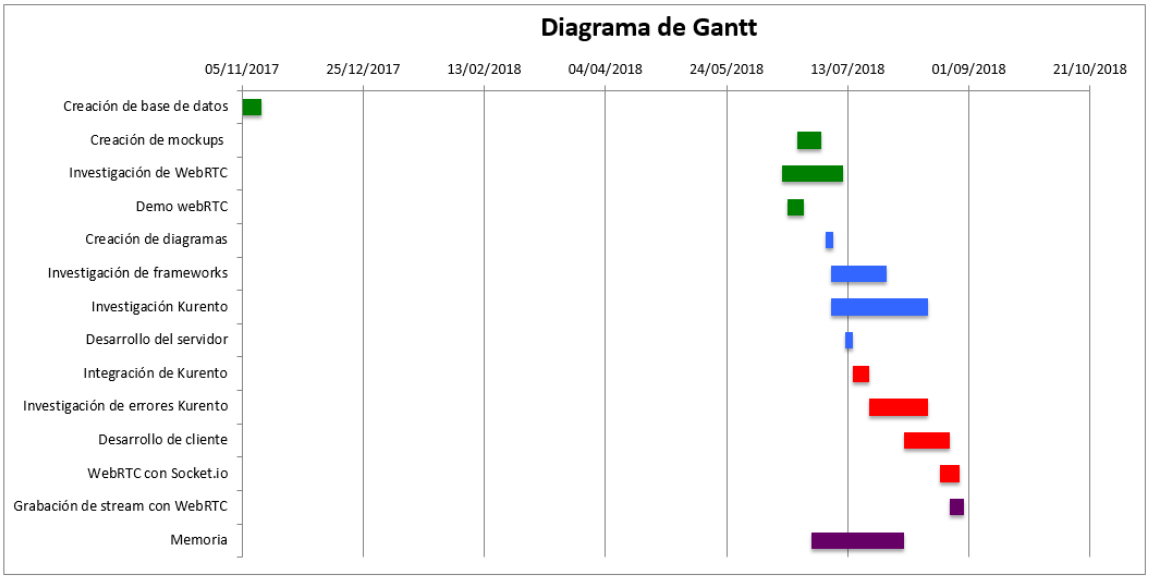


Ilustración 6. Diagrama de Gantt

Como podemos ver en la tabla anterior, la mayor parte del trabajo se hizo desde el 16 de Junio hasta el 5 de Septiembre, excepto la tarea de creación de base de datos. La tarea con mayor duración ha sido la investigación sobre Kurento, este es un framework que nos permitía implementar la llamada y grabación de este de manera sencilla, pero durante la integración de este al proyecto se crearon muchos conflictos, errores y problemas que hicieron que el proyecto se retrasará de manera considerable. La tarea más larga después de Kurento ha sido la investigación de WebRTC, ya que ha sido una tecnología nueva y compleja, que nunca hemos usado y como era la principal característica de nuestro proyecto hemos tenido que trabajar mucho con ella.

4. ESPECIFICACIÓN DE REQUISITOS

En este apartado definiremos los requisitos funcionales, no funcionales, casos de uso, y diferentes diagramas, gracias a todo esto tendremos una idea bien definida de nuestro sistema y cualquier usuario exterior podrá entender de manera fácil cual es el objetivo de nuestro sistema, funcionalidad y especificaciones.

a. CASOS DE USO

A continuación definiremos los diversos casos de uso de nuestra aplicación. Los casos de uso nos ayudan a expresar el comportamiento deseado del sistema, además podemos establecer las necesidades según la necesidad de cada usuario (actor). En nuestra plataforma tendremos dos tipos de actores que interactuarán con las funcionalidades: médico y paciente.

- Ver pacientes disponibles para videollamada

Caso de Uso: **Ver pacientes disponibles para videollamada**	
Actor: **Médico**	
Curso Normal	**Alternativo**
1. El médico inicia sesión en el sistema.	1.1. Si el médico no está registrado, se le informará que debe de hacerlo.
2. El médico va al apartado de videollamada.	
3. El médico buscará por el DNI del paciente y tendrá la lista de los conectados.	
4. El médico llamará al paciente con el que quiere hablar.	
5. El sistema avisará al paciente	
1. El paciente aceptará o denegará la llamada iniciada.	

Tabla 2. Caso de uso 1

- Ver videollamada anterior

Caso de Uso: **Ver videollamada anterior**	
Actor: **Médico**	
Curso Normal	**Alternativo**
1. El médico iniciará sesión en el sistema.	1.1. Si el médico no está registrado, se le informará que debe de hacerlo.
2. El médico irá al apartado de historiales de paciente.	
3. El médico buscará la fecha cuando ocurrió la videollamada.	
4. El médico podrá visualizar la videollamada en el sistema.	

Tabla 3. Caso de uso 2

- Iniciar videollamada

Caso de Uso: **Iniciar videollamada**	
Actor: **Paciente**	
Curso Normal	**Alternativo**
2. El paciente inicia sesión en el sistema.	2.1. Si el paciente no está registrado, se le informará que debe de hacerlo.
3. El paciente va al apartado de videollamada.	
4. El paciente introducirá el código de su cita.	
5. El paciente llamará al médico con el que tiene la cita	

6. El sistema avisará al médico.	
7. El médico aceptará o denegará la llamada iniciada.	

Tabla 4. Caso de uso 3

b. DIAGRAMA DE CASOS DE USO

Representación gráfica de los casos de uso.

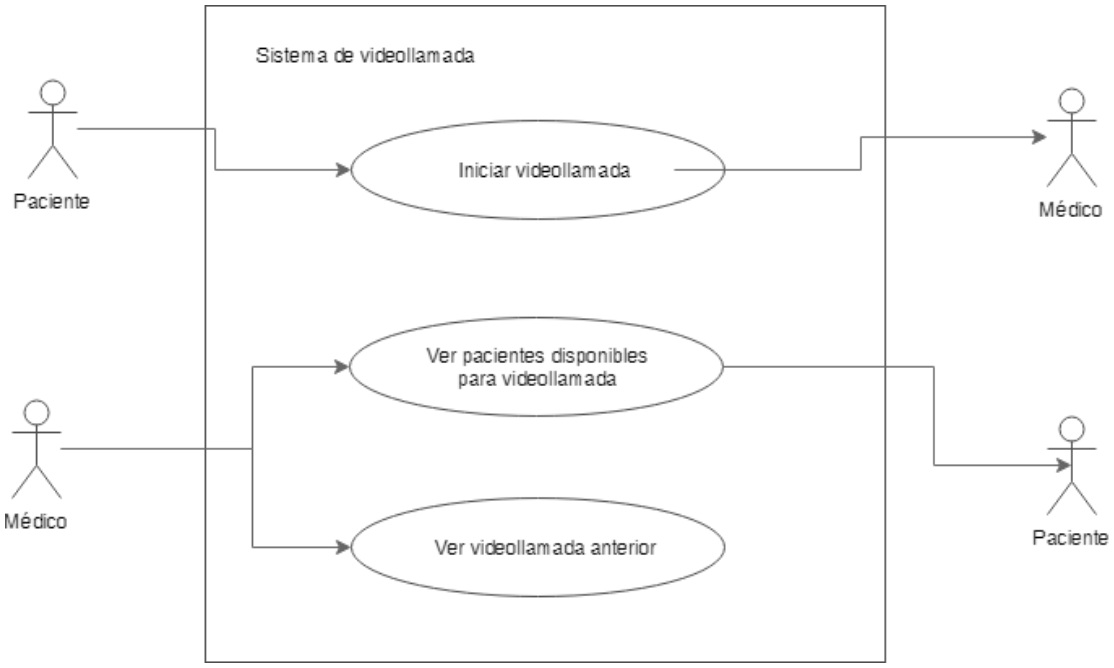


Ilustración 7. Diagrama de casos de uso

c. DIAGRAMA DE CLASES

En este apartado mostraremos nuestro diagrama de clases. Estos diagramas se caracterizan por ser estáticos, por tanto no describen acciones y muestran las entidades que existen y las relaciones

entre ellas. Este diagrama de clases se muestra completo, es decir, con la integración de la plataforma e-Health.

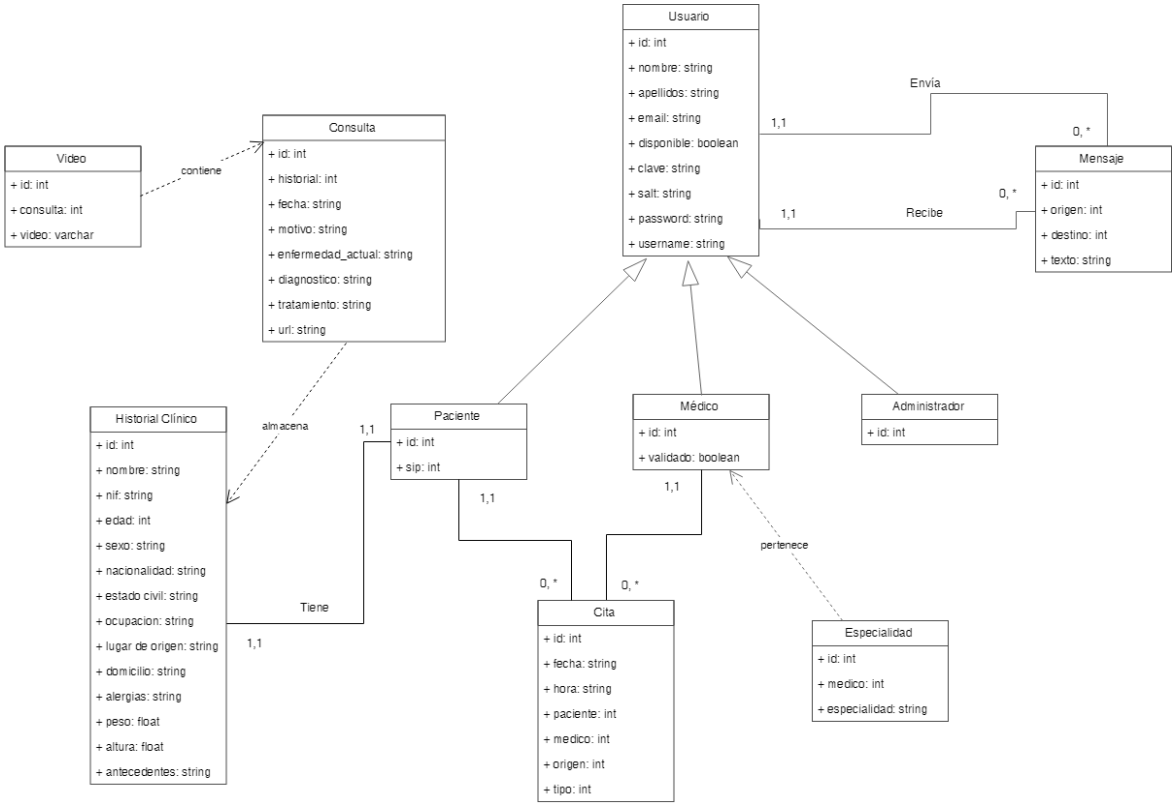


Ilustración 8. Diagrama de clases

d. ESQUEMA ENTIDAD RELACIÓN

A continuación vamos a definir el esquema Entidad Relación de nuestra base de datos. Con este tipo de esquema podemos mostrar las diferentes entidades, relaciones y atributos que nuestra base de datos posee. La base de datos se ha realizado con MySQL con la herramienta de Xampp y Workbench. El esquema de base de datos está entero, se muestra tanto la parte de videollamada como la parte de sistema e-Health integrado.

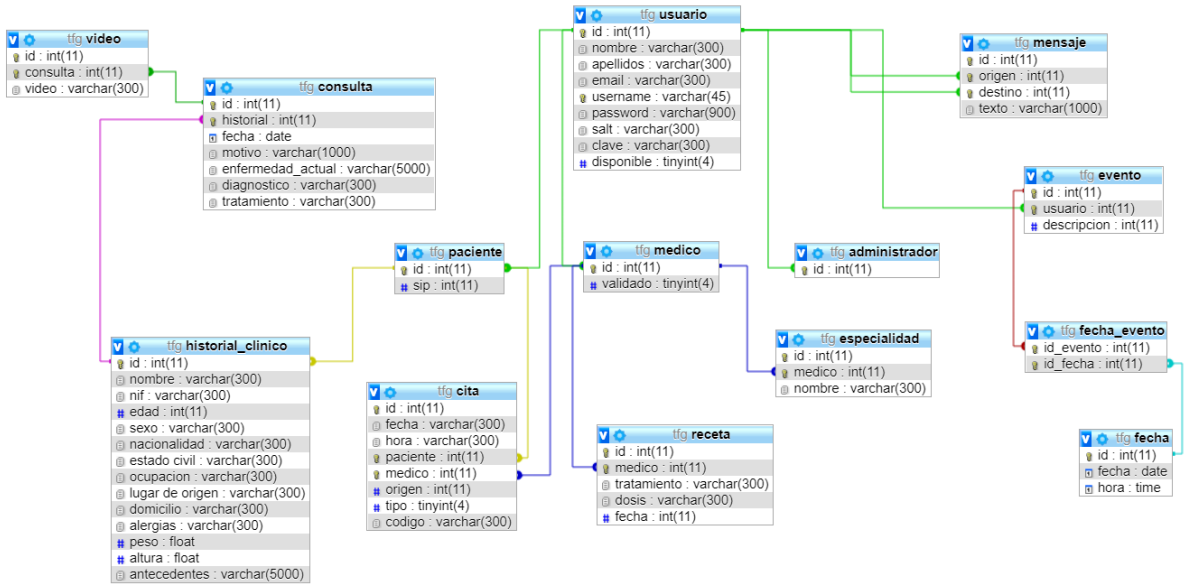


Ilustración 9. Esquema Entidad-Relación

e. REQUISITOS FUNCIONALES

En este apartado enumeraremos los diferentes requisitos funcionales. Estos son declaraciones de los servicios que debe proporcionar el sistema, la forma en que deben reaccionar a las entradas y cómo se debe comportar en situaciones particulares. Definiremos a cada uno con un identificador, nombre, una breve descripción y una prioridad, la cual puede tomar los valores: baja, media, alta.

Identificador:	**RF1**
Nombre:	Llamar a un médico
Descripción:	Un paciente podrá llamar a un médico teniendo una cita previa concertada.
Prioridad	Alta

Tabla 5. Requisito funcional 1

Identificador:	**RF2**
Nombre:	Llamar a un paciente
Descripción:	Un médico podrá llamar a un paciente sin necesidad de tener ninguna cita concertada con antelación. El médico podrá buscar al paciente por su Tarjeta Sanitaria Individual SIP.
Prioridad	Alta

Tabla 6. Requisito funcional 2

Identificador:	**RF3**
Nombre:	Contestar a una llamada
Descripción:	Cualquier usuario ya sea médico o paciente podrá contestar a una llamada entrante. La notificación de la llamada se mostrará en una ventana superpuesta en la página actual.
Prioridad	Alta

Tabla 7. Requisito funcional 3

Identificador:	**RF4**
Nombre:	Denegar una llamada
Descripción:	Cualquier usuario ya sea médico o paciente puede denegar cualquier llamada entrante.
Prioridad	Alta

Tabla 8. Requisito funcional 4

Identificador:	**RF5**
Nombre:	Colgar una llamada
Descripción:	Cualquier usuario ya sea médico o paciente puede colgar una llamada en la que es participe.
Prioridad	Alta

Tabla 9. Requisito funcional 5

Identificador:	**RF6**
Nombre:	Listar videos de un paciente
Descripción:	Un médico podrá ver un listado de videos grabados en anteriores video consultas de un paciente.
Prioridad	Media

Tabla 10. Requisito funcional 6

Identificador:	**RF7**
Nombre:	Ver videos
Descripción:	Un médico podrá visualizar un video concreto de una video llamada hecha anteriormente con un paciente.
Prioridad	Alta

Tabla 11. Requisito funcional 7

f. REQUISITOS NO FUNCIONALES

Ahora definiremos los requisitos no funcionales, estos van a definir las restricciones o funciones ofrecidas por el sistema, aspectos técnicos. Estos requisitos siguen la misma estructura definida en el apartado anterior.

Identificador:	**RNF1**
Nombre:	Fiabilidad del sistema
Descripción:	El sistema en su totalidad debe de dar una sensación de fiabilidad al cliente, teniendo una recuperación de fallos rápida y una ocurrencia de fallos baja.
Prioridad	Alta

Tabla 12. Requisito no funcional 1

Identificador:	**RNF2**
Nombre:	Seguridad
Descripción:	El sistema debe de contar con un nivel de seguridad, para que toda la información personal del paciente y médico esté protegida.
Prioridad	Alta

Tabla 13. Requisito no funcional 2

Identificador:	**RNF3**
Nombre:	Tiempos
Descripción:	El sistema debe de poder notificar una llamada al usuario que se quiere llamar rápidamente, entre 30 segundos y 1 minuto.
Prioridad	Alta

Tabla 14. Requisito no funcional 3

Identificador:	**RNF4**
Nombre:	Usabilidad
Descripción:	El sistema tiene que ser fácil de entender e intuitivo.
Prioridad	Alta

Tabla 15. Requisito no funcional 4

Identificador:	**RNF5**
Nombre:	Recursos
Descripción:	El sistema debe de poder acceder a una cámara y micro interno o externo del dispositivo con el que se está accediendo a la web.
Prioridad	Alta

Tabla 16. Requisito no funcional 5

g. REQUISITOS DE INTERFAZ

i. MOCKUPS

Este apartado está orientado a los mockups. Los mockups son diseños de alta fiabilidad, donde podemos representar toda la información que va a contener una vista, pudiendo establecer colores, tipos de letra y demostrar alguna funcionalidad básica. A continuación están los diferentes mockups diseñados, organizados por el usuario.

1. Parte del paciente

- Vista de comprobación de código de cita

En esta vista el usuario conectado podrá introducir su código de cita anteriormente generado en el apartado de crear citas. Si el código es correcto, el sistema procederá a enviarle otra vista. Para que el código sea válido tiene que estar ese código al paciente, ser el día de la cita y que sea la hora, con un intervalo de 10 minutos menos y 20 minutos más.

Ilustración 10. Mockup comprobación de código cita

- Vista de información de cita de videoconsulta

En esta vista será la siguiente a la de comprobación de código y podemos encontrar en ella toda la información sobre la cita, como la hora, fecha, datos del paciente y médico. Además de esto, esta vista permitirá llamar al médico con el que tiene la cita o cancelar dicha cita.

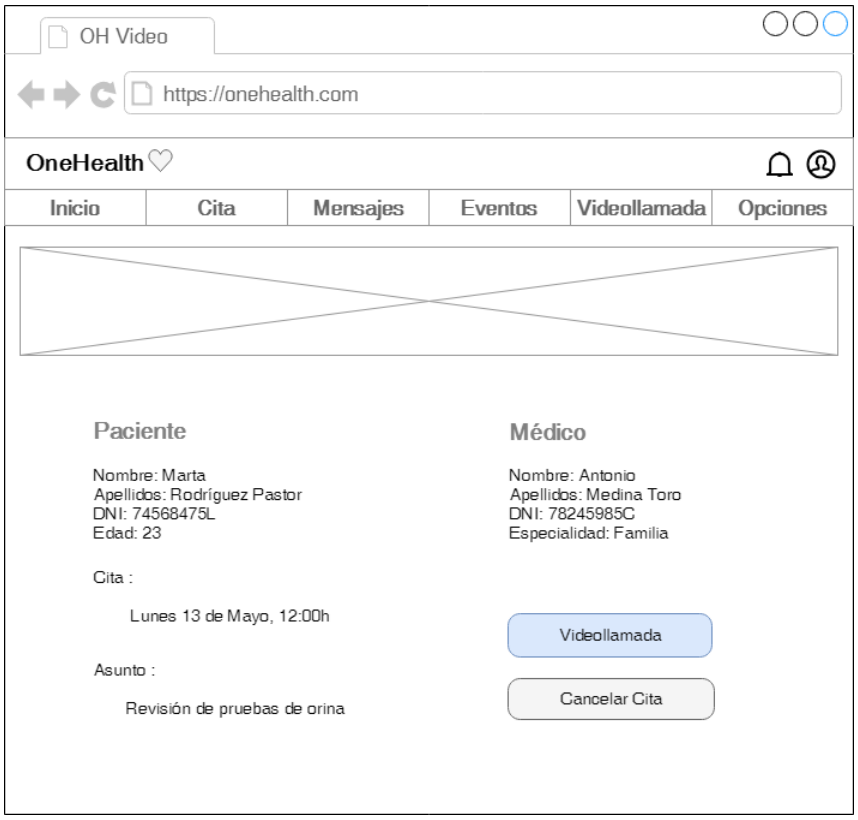


Ilustración 11. Mockup información cita videollamada

2. Parte del médico

- Vista de listado de pacientes conectados

Esta vista permitirá al médico conectado ver los pacientes que hay ahora mismo conectados a la plataforma, para hacer una videollamada con ellos. Este listado permite hacer una búsqueda de paciente a través de su número de tarjeta sanitaria individual SIP.

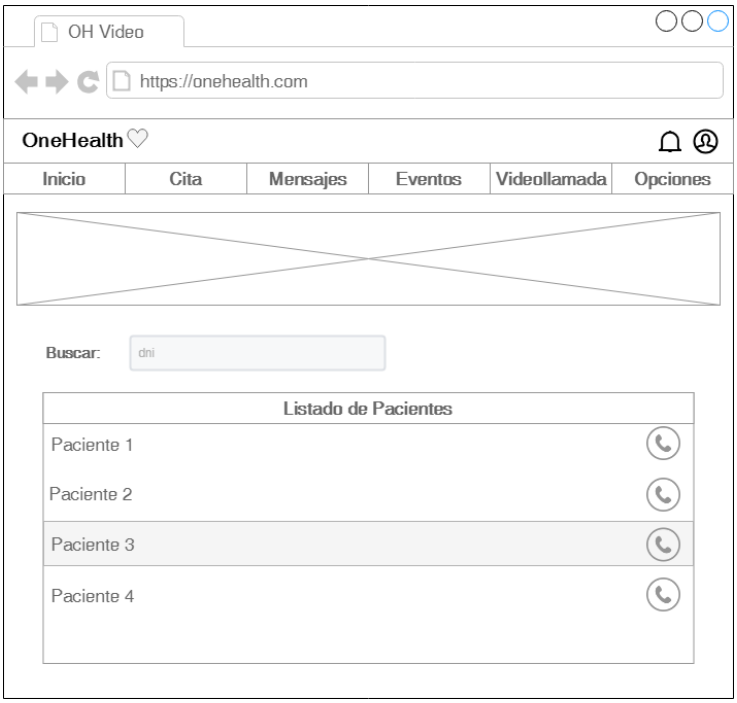


Ilustración 12. Mockup listado de pacientes conectados

- Vista de reproducción de una videoconsulta

Esta vista permite al médico conectado ver un video grabado en una videollamada hecha anteriormente. Este apartado estará situado en la parte de consultas de un paciente.

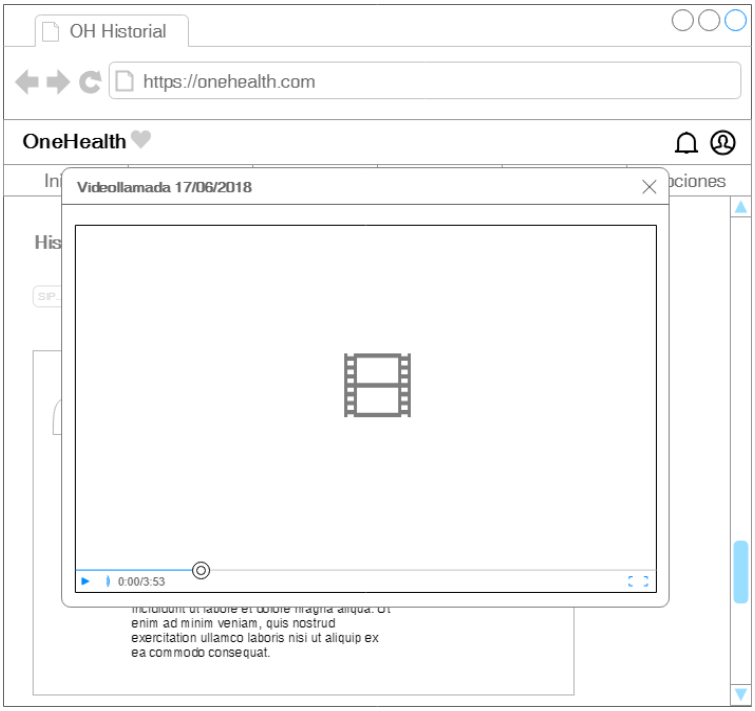


Ilustración 13. Mockup información de una videoconsulta

3. Parte genérica

- Vista de videollamada

Esta vista será genérica, esto quiere decir que cualquier usuario tanto paciente como médico verá lo mismo en una videollamada. Tendremos la imagen de usuario con el que estamos en grande y una imagen pequeña de nuestra imagen.

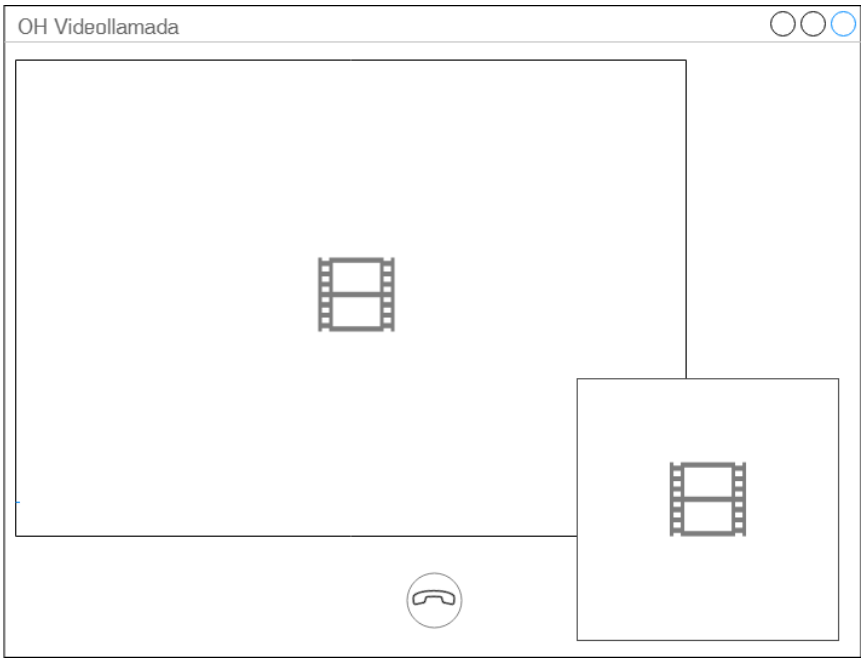


Ilustración 14. Mockup vista videollamada

- Vista de una llamada recibida

Esta vista refleja cómo se le notifica a un usuario que tiene una llamada entrante. Este usuario tendrá a su disposición un botón para contestar y otro para denegar la llamada.

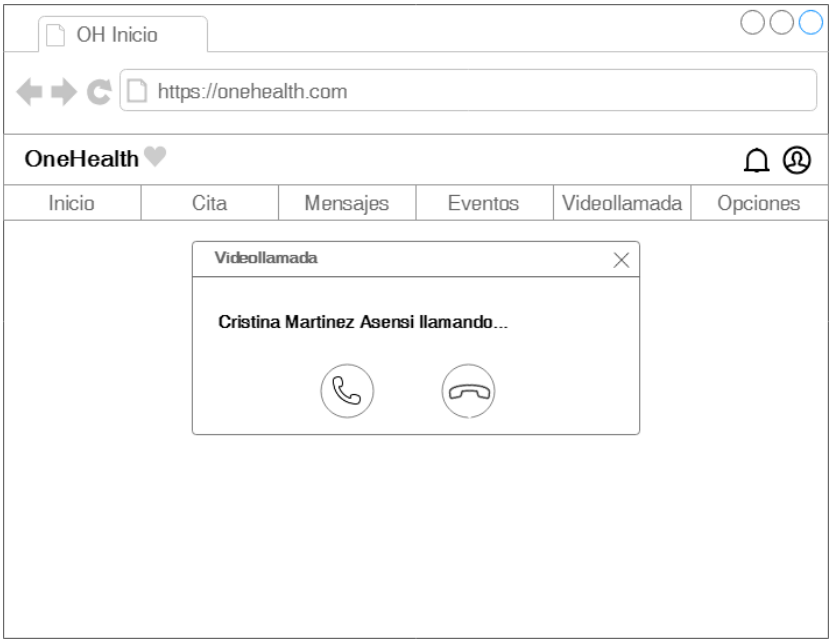


Ilustración 15. Mockup vista de una llamada entrante

5. ARQUITECTURA E IMPLEMENTACIÓN

En este apartado describiremos la arquitectura planteada para nuestro proyecto, la cual estará partida entre los componentes que la componen.

Nuestra arquitectura es cliente-servidor, el cual es un modelo de diseño de software en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Un cliente realiza peticiones a otro programa, el servidor, quien le da respuesta. En nuestro caso, los clientes además de hacer peticiones https, emite mensajes a través de socket.io al servidor. A continuación se ve representada la arquitectura.

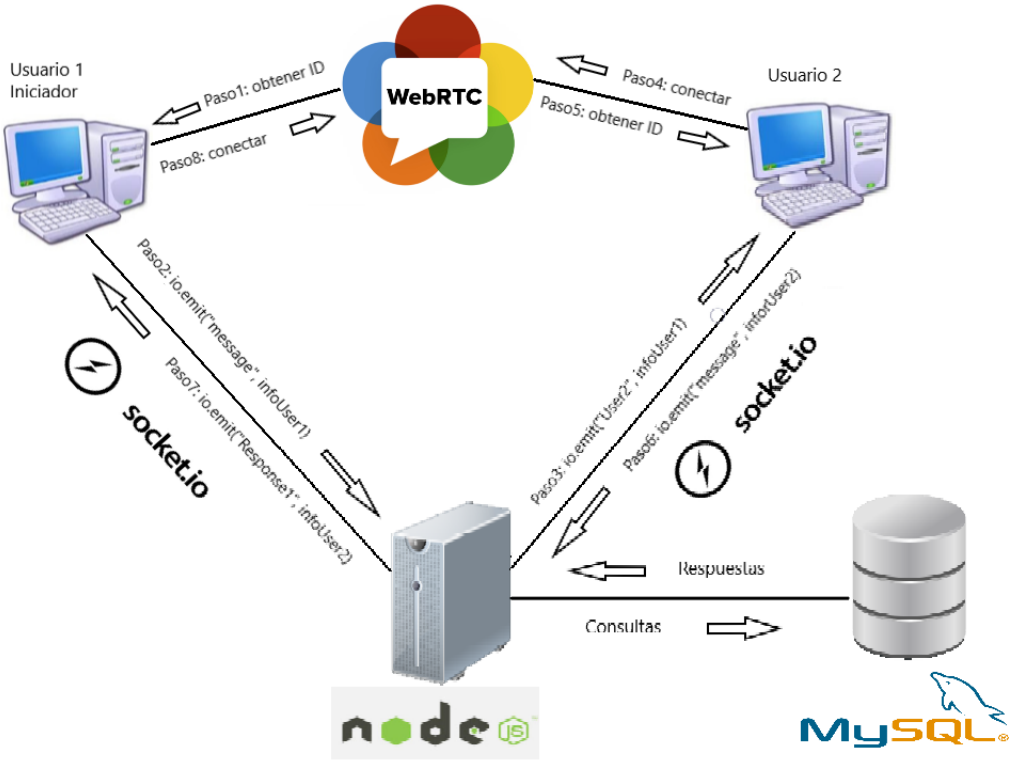


Ilustración 16. Arquitectura de OneHealth

a. CLIENTE

Nuestro cliente está desarrollado con ReactJS, WebRTC y Socket.io. Estas dos características vamos a explicarlas a continuación.

i. REACT

React (también llamada React.js o ReactJS) es una biblioteca Javascript de código abierto focalizada en el desarrollo de interfaces de usuario con el objetivo de animar al desarrollo de aplicaciones SPA (Single Page Application). Es mantenido por Facebook, Instagram y una comunidad de desarrolladores independientes y compañías.

React sirve para desarrollar aplicaciones de manera más ordenada y con menos código que si usas JavaScript puro o librerías como JQuery centradas en la manipulación del DOM. Las vistas se asocian con datos (estados), por lo tanto, si cambian los datos, la vista cambia. Su objetivo es ser sencillo, declarativo y fácil de combinar. React está construido bajo el patrón de diseño MVC (Modelo-Vista-Controlador), y puede ser utilizado con otras librerías de JavaScript. También puede ser utilizado con las extensiones de React-based que se encargan de las partes no-UI (no gráficas) de una aplicación web.

1. WEBPACK

Para hacer un desarrollo del cliente rápido y sencillo hemos optado por utilizar WebPack, el cuál es un servidor de desarrollo que publicará los recursos de webpack para que sean accesibles desde el navegador. En cierta medida se puede considerar un Browserify avanzado ya que tiene muchas opciones de configuración.

Esta plantilla nos permite hacer cambios en ejecución, el servidor se recargará automáticamente la página en el navegador.

WebPack dispone de muchas opciones de configuración, las cuales están listadas en esta página. Alguna de estas configuraciones nos ha servido para nuestro proyecto, como por ejemplo:

- devServer.https

  Esta opción la hemos usado para tener un cliente con seguridad con el protocolo HTTPS. Esta opción puede definirse a través del archivo webpack.config.js activando la opción de https, además se pueden incluir certificados o que estos se generen automáticamente, o podemos añadirle un parámetro más al comando de lanzamiento –https.

- devServer.port

  Esta opción la hemos usado para cambiar el puerto por el que corre nuestro cliente. La configuración predefinida del WebPack lanza el cliente por el puerto 8080, en nuestro caso lo lanzamos en el 8443. Esta cambio puede hacerse en el archivo webpack.config.js o bien, en el comando de lanzamiento con el parámetro –port.

ii. WEBRTC

WebRTC, también conocido como Web Real-Time Communications, es un proyecto de código abierto, promovido por Google, Mozilla y otros, que permite comunicaciones en tiempo real sin plug-ins a través de una API JavaScript. Facilita las aplicaciones de llamada de voz, chat de video y compartimiento de archivos entre navegadores. El códec soportado actualmente para WebRTC es VP8. WebRTC utiliza un servidor denominado Servidor de Conferencias Web que en conjunto con un Servidor STUN es requerido para proveer la página inicial y sincronizar las conexiones entre los dos nodos WebRTC. La razón por la cual se creó WebRTC fue para atacar los problemas de privacidad que aparecen al exponer capacidades y flujos locales.

La seguridad y el cifrado en WebRTC no son opcionales, ya que esta tecnología tiene características integradas nativas que se encargan de los problemas de seguridad. Además WebRTC ofrece cifrado punto a punto entre extremos sobre cualquier servidor garantizando las comunicaciones en tiempo real, privadas y seguras.

WebRTC requiere del usuario para permitir explícitamente el acceso a su cámara y micrófono. Esto garantiza que el usuario sea consciente de que su cámara y micrófono se encenderán. Cuando el usuario permite el acceso, un punto rojo aparecerá en esa solapa, proporcionando una clara indicación, de que la solapa tiene acceso a los medios.

Para que WebRTC transfiera datos en tiempo real, los datos primero se cifran utilizando el método DTLS (Datagram Transport Layer Security). Este es un protocolo integrado en todos los navegadores soportados por WebRTC desde el principio (Chrome, Firefox y Opera). En una conexión DTLS cifrada, el espionaje y la manipulación de información no pueden tener lugar.

Aparte de DTLS, WebRTC también encripta los datos de vídeo y audio a través del método SRTP (Secure Real-Time Protocol) garantizando que las comunicaciones IP – su voz y el tráfico de vídeo – no puedan ser escuchados o vistos por terceros no autorizados.

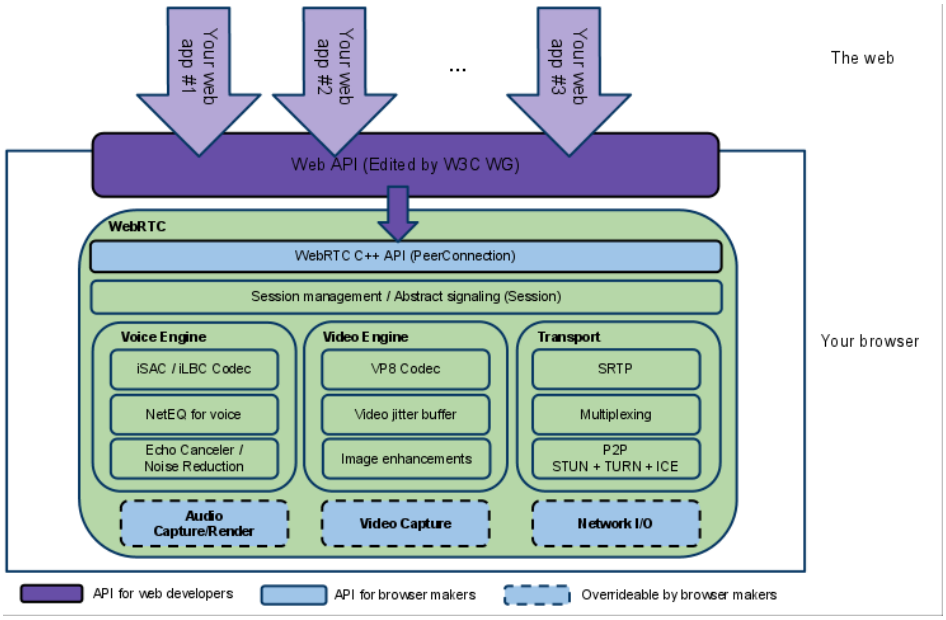


Ilustración 17. Arquitectura de WebRTC

Cómo hemos visto en la imagen de la arquitectura de nuestro proyecto, el primer paso que se hace es de WebRTC, el cual obtener el id del usuario iniciador de la llamada. El código que permite esto es:

```
initiater() {

      var idUsu = localStorage.getItem('id');

      var username = localStorage.getItem('username');

      var mythis = this

      peer = Peer({trickle: false, initiator: true,
      stream:this.state.stream})

      peer.on('signal', (data) => {

            mythis.setState({myID:JSON.stringify(data)})

            var message = {

                  "id": "userToken",

                  "name": username,

                  "iniciador": idUsu,

                  "userRemote": mythis.state.idPac,
```

```
                "token": mythis.state.myID
            }
            mythis.sendMessage(message)
        })
}
```

Cuando la información del iniciador se ha enviado al servidor, este emitirá a través de sockets un evento User(id), siendo id el identificador del usuario al que quiere llamar. Una vez allí, el cliente se conectará a webrtc con el token del iniciador y generara el suyo propio. Una vez se ha hecho todo esto correctamente, el usuario enviará el ID que acaba de generar con WebRTC y lo enviará al usuario iniciador como hemos dicho anteriormente. Esté cogerá el ID del otro usuario y se conectará. El código de conexión es el siguiente:

```
    connect() {
      var idUsu = localStorage.getItem('id');
      var username = localStorage.getItem('username');
      var mythis = this
      var message = null
      var auxID
      if (peer === null) {
            peer = Peer({trickle: false, stream: this.state.stream})
            peer.on('signal', (data) => {
                  if(mythis.state.TypeUser == "Response") {
                        mythis.setState({myID:JSON.stringify(data)}
                        message = {
                          "id": "reponseToken",
                          "name": username,
                          "iniciador": mythis.state.aux.iniciador,
```

```
                    "userRemote": idUsu,
                    "token": mythis.state.myID
                }
            }
        })
}

var data = mythis.state.aux.token
peer.signal(data)
peer.on('connect', () => {
    console.log('peer connected')
})
peer.on('data', (data) => {
    const message = data.toString('utf-8')
    this.setState({messages:
    this.state.messages.concat("\n"+this.state.aux.name+"
    "+message)})
})
peer.on('stream', (stream) => {
    console.log("Send stream")
    this.handleVideoRemote(stream)
})
peer.on('error', (error) => {
    console.error('peer error', error)
})
peer.on('close', () => {
    console.log('peer connection closed')
```

```
            this.state.stream.getTracks().forEach( (track) => {
                  track.stop();
            });
            peer.destroy()
      })
}
```

Con la conexión hecha, paciente y médico pueden comunicarse a través de video, audio y texto. La parte de video y audio pasa por el evento de stream, y el de texto por el evento data, los cuales se pueden ver el anterior código.

Con WebRTC además de ayudarnos con la conexión de la llamada, también nos ha permitido grabar en la parte del médico el video del paciente. Esto se ha realizado con MediaRecorder, esto es una interfaz de MediaStream Recording API, la cual permite grabar fácilmente. Para poder grabar, al constructor del MediaRecorder hay que pasarle como primer parámetro el stream, en nuestro caso le hemos pasado el stream del paciente y como segundo el MIME. Por problemas de soporte en hemos grabado con el formato webm.

```
let options = {mimeType: 'video/webm,codecs=vp9'};
this.mediaRecorder = new MediaRecorder(this.state.otroStream,
options);
```

Una vez que la conexión se ha terminado entre paciente y médico, en la parte del médico se le dará la opción de descargar el video anteriormente grabado. Esto se ha realizado con una librería externa, FileSaver. A la función de descarga le pasamos el objetivo Blob, el cual contiene almacenado el video grabado y como segundo parámetro el nombre que queremos ponerle al video.

```
const superBuffer = new Blob(this.recordedBlobs, {type:
'video/webm'});
saveAs(superBuffer, nombreVideo , true)
```

iii. SOCKET.IO

Socket.io es una biblioteca de JavaScript para manejar eventos en tiempo real mediante una conexión TCP. Esta librería está disponible tanto para la parte del servidor en Nodejs, como para la parte del cliente, siendo estas dos APIs muy parecidas.

Socket.io se basa principalmente en WebSockets pero también puede usar otras alternativas como sockets de Adobe Flash, JSONP polling o long polling en AJAX, seleccionando la mejor alternativa para el cliente justo en el tiempo de ejecución. Además puede ser usado como contenedor para WebSockets, ofreciendo muchas más características, incluida la transmisión de múltiples sockets, el almacenamiento de datos asociados a cada cliente, etc. Aunque esta librería este basada en WebSockets, no es una librería de esta con opciones de respaldo para otros protocolos en tiempo real, sino que es un protocolo de transporte personalizado en tiempo real. También tiene la capacidad de implementar análisis en tiempo real, transmisión binaria, mensajería instantánea y colaboración de documentos.

La ventaja que tiene Socket.io es que maneja la conexión de forma una transparente.

En nuestro sistema hemos integrado esta librería para permitir la inicial comunicación entre los dos puntos. Para poder emitir y escuchar eventos en la parte del cliente primero nos hemos conectado al servidor, esto se hace con la siguiente sentencia:

```
this.socket = io.connect("https://localhost:3000", connectionOptions);
```

La variable connectionOptions define las características que debe de cumplir el socket, como el timeout, reconexiones, etc:

```
var connectionOptions = {
        "force new connection": true,
        "reconnectionAttempts": "Infinity",
        "timeout": 10000,
        "transports": ["websocket"]
}
```

Todo este código anterior lo hemos situado en la función componentDidMount(), la cual es una función interna de React y se invoca inmediatamente después de que el componente (vista) se haya montado. Esto quiere decir que una vez que cuando componte, en este caso ComponenteInicio, se cargue por primera vez, el socket se conectará al servidor y estará listo para recibir y enviar peticiones.

Para enviar y recibir eventos hemos partido los sockets en 2 componentes. Para la iniciación de la llamada, necesitábamos que el socket enviase los datos al servidor desde nuestro componente de la videollamada, ComponenteWebRTCSimple. En este hemos definido la escucha en la función componentDidMount() y en el envió en una función.

Escucha de socket:

```
this.state.socket.on('Response'+ idUsu, function(message) {

        mythis.setState({aux: message})

        mythis.connect()

})
```

Envió de eventos:

```
sendMessage(msg) {

        this.state.socket.emit("message", msg)

}
```

Para la recepción de la llamada la escucha la tenemos en el ComponenteInicio, a continuación de la conexión con el servidor, de esta manera el usuario puede navegar libremente por el sistema y si recibe alguna llamada, este será avisado.

b. SERVIDOR

El servidor esta implementado con Nodejs bajo SSL/TLS y con sockets con seguridad, este ha sido implementado en un sistema Ubuntu 16.05 x64 virtualizado con el programa Oracle VM VirtualBox Administrator anteriormente descrito.

i. NODEJS

Nodejs es una librería y entorno de ejecución dirigida por eventos y por lo tanto asíncrona que se ejecuta sobre el intérprete de JavaScript creado por Google V8. Esta tecnología funciona en el lado del servidor y es de código abierto. Una de las grandes características de node es la capacidad de soportar gran cantidad de conexiones simultáneas a un servidor (escalabilidad). Esto lo hace empleando un solo hijo y un bucle compuesto de diferentes eventos asíncronos.

Nuestro servidor además de estar desarrollado en nodejs, está dotado de seguridad bajo SSL/TLS. Esto hace que toda nuestra conexión cliente-servidor este cifrada, para evitar sniffer. Para poder hacer un servidor con el protocolo HTTPS tenemos que crear unos certificados e importarlos.

El servidor implementado nos permite hacer diferentes acciones sobre las entidades definidas en apartados anteriores, estas acciones vamos a enumerarlas a continuación:

- Buscar paciente por SIP: está funcionalidad buscará un paciente concreto a través de una tarjeta SIP, la cual ha sido pasada por una petición https desde el cliente por el médico. Esta funcionalidad permite una búsqueda eficiente, es decir, busca aunque la tarjeta sip no esté completa.
- Listar pacientes: está funcionalidad será para enumerar todos los pacientes que en este momento están conectados a la plataforma e-Health. Enviará el número de tarjeta SIP, identificador de base de datos, nombre y apellidos de cada uno de los pacientes.
- Ver cita: enviará al paciente toda la información que contiene una cita para videollamada. Los datos enviados con nombre, apellidos y email del paciente, nombre, apellidos, especialidad e identificador (ID de base de datos) del médico y fecha y hora de la cita.
- Comprobar código de cita: esta funcionalidad comprobará a través de un código generado en la creación de la cita, si el paciente tiene una cita para una videollamada, en la fecha actual y hora actual, con un rango de validez de -10 minutos y +20 minutos.
- Cambiar estado: se le enviará una petición a esta función cuando un usuario inicie y cierre sesión en la plataforma, de esta manera, cambiara su estado en la base de datos a su estado actual (conectado= 1, desconectado=0).
- Nuevo video: guardará el nombre del video grabado y la conversación de texto de una videollamada. Esto se guardará en la tabla de video y se asociará a una consulta.
- Ver video: esta funcionalidad nos dará los datos guardados de una videollamada, nombre del video y conversación de texto.

ii. SOCKET.IO

En la parte del servidor tenemos la creación del servidor socket, al cual el cliente se conectará para emitir y recibir eventos. Para poder crear el servidor anteriormente hemos instalado la librería de socket.io desde npm.

La creación de este lo hacemos a partir del servidor https de nuestro servidor principal de nodejs.

```
var io = require('socket.io')(https).listen(server1)
```

De esta abriremos nuestros sockets de maneras segura, y todos los datos que viajen a través de estos estarán cifrados.

Para la escucha de eventos hemos optado por coger un switch, de esta manera cada evento que llega examinamos su parámetro id, este nos definirá la acción que quiere hacer sobre el servidor.

```
socket.on('message', function(msg){
    var message = msg
        switch (message.id) {
            case 'userToken':
                getToken(message);
                break;
            case 'reponseToken':
                resposeToken(message);
                break;
            default:
               var error = {
                    id : 'error',
                    message : 'Invalid message ' + message
                }
                break;
        }
    });
```

Como vemos en el anterior código solo hay disponible un evento al que llamar, el cual es message y el manejo del evento se hace dentro del switch. Dentro de ese switch solo hay dos funciones a las que se le pueden llamar, una es getToken y otra es responseToken.

- getToken: esta función se encarga de coger la información del usuario iniciador de la llamada y enviarla al usuario con el que quiere contactar o hacer la videollamada.

- responseToken: esta función es responsable de coger la información del segundo usuario de la llamada (no iniciador) y enviarla de nuevo al usuario iniciador, de esta manera en el cliente con WebRTC se realizara la videollamada.

6. SEGURIDAD

Uno de los objetivos más importante de este proyecto de fin de grado, es que todos los datos transmitidos y guardados estuviesen protegidos en todo momento para evitar filtraciones y daños. Por ello, en cada una de las partes tenemos diferentes métodos, funciones y protocolos, que nos aseguran a nosotros y a los usuarios que sus datos están seguros.

- Servidor
    - Comunicación

    El servidor está dotado de una protocolo seguro como es HTTPS, esto permite una comunicación segura además de garantizar la protección de la vida del usuario y la integridad de los datos transmitidos.

    - Almacenamiento

    Toda la información que se guarda en la base de datos es cifrada con AES 256 con el modo CBC.

    - Socket.io

    Los sockets hacen uso de wss (WebSockets Secure) gracias a que están creados con el servidor https principal. Además de esto, le hemos puesto una propiedad, websocket, que permite hacer una conexión mejor.

```
var httpsOptions = {

      cert: fs.readFileSync(path.join(__dirname,'security',
      'server.crt')),

      key: fs.readFileSync(path.join(__dirname,'security',
      'server.crt'))

}

var server1 = https.createServer(httpsOptions, app).listen(3000,
function() {

      console.log("Servidor arrancado")

})

var io = require('socket.io')(https).listen(server1)

io.set('transports', ['websocket']
```

- Cliente
    - Webpack

    El cliente tiene en su configuración de webpack.conf las dependencias hacia unos certificados SHA-2, con los cuales podemos acceder al cliente de forma segura, con el protocolo HTTPS.
    - WebRTC

    WebRTC nos permite una comunicación segura, ya que nos obliga a tener una comunicación SSL/TLS entre el servidor y cliente además de integrar distintos protocolos. Utiliza SRTP (Protocolo de transporte en tiempo real seguro), el cual encripta y autentifica los diferentes flujos que se envían en tiempo real. Además de esto, los datos de video y audio que se envían deben de ser validados por el usuario, por eso, cada vez que se inicia una llamada, el navegadores le pedirá permisos al usuario. Finalmente, para aportar un nivel más de seguridad, WebRTC utiliza el protocolo DTLS (Datagram Transport Layer Security) para encriptar los datos enviados entre los usuarios de la llamada.

7. ESTRUCTURA Y FUNCIONALIDADES

a. ESTRUCTURA DEL PROYECTO

En este apartado presentaremos la estructura que hemos creado para nuestro proyecto.

El proyecto a líneas generales esta partido en dos partes, cliente y servidor, en cada una de estas están las diferentes carpetas con los archivos para su funcionamiento.

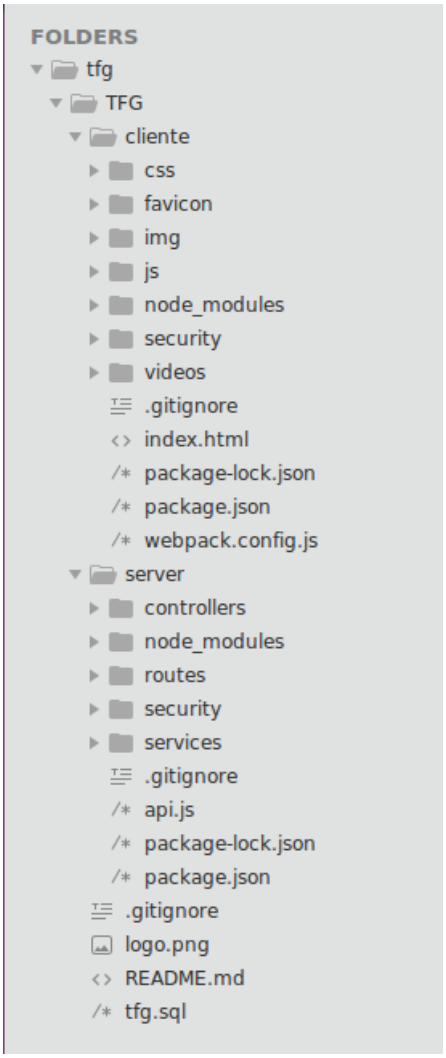


Ilustración 18. Estructura del proyecto

En la parte del servidor tenemos una distribución por carpetas donde nos encontramos la carpeta */server/controllers*, en la cual nos encontramos toda la lógica de la plataforma eHealth. En la carpeta */server/routes* nos encontramos todas las peticiones disponibles de nuestra api, en */server/security* se encuentran los certificados para el servidor y socket seguros, en la carpeta de

/server/services se encuentran las funciones de encriptado y desencriptado y por ultimo fuera de todo tenemos el archivo principal, *api.js* donde se inicia el servidor y socket.

En la parte del cliente tenemos una distribución dividida por estilo, imágenes, seguridad y componentes. La parte más importante del cliente se encuentra en la carpeta de */cliente/js* donde tenemos todos los componentes o vistas del sistema y dentro de ella en */cliente/js/servicios* tenemos todas las llamadas al servidor, las cuales están implementadas con el fetch.

b. FUNCIONALIDADES

En esta sección presentaremos y describiremos las funcionalidades finales de la aplicación e-Health con videollamada. El número de funcionales es bajo ya que este proyecto va sobre una videollamada segura, pero cada función tiene diferentes formar según la persona que lo está utilizando, y en nuestra aplicación hay dos tipos de usuarios, que son médico y el otro paciente.

- Videollamada

Está funcionalidad está para varios, ya que este sistema quiere darle al paciente el acercamiento y la ventaja de gestionar sus propias citas.

Para que el paciente pueda realizar una videollamada, anterioremente debe de haber sacado una cita tipo videollamada en el apartado de crear citas. En el momento que el paciente tenga la cita programada podrá ir al apartado de videollamada y el programa le pedirá un código, el cual le han dado al crear la cita. Este código sólo será válido si la cita es en ese concreto día y hora, pudiendo ser diez minutos antes y 20 minutos después de la videollamada.

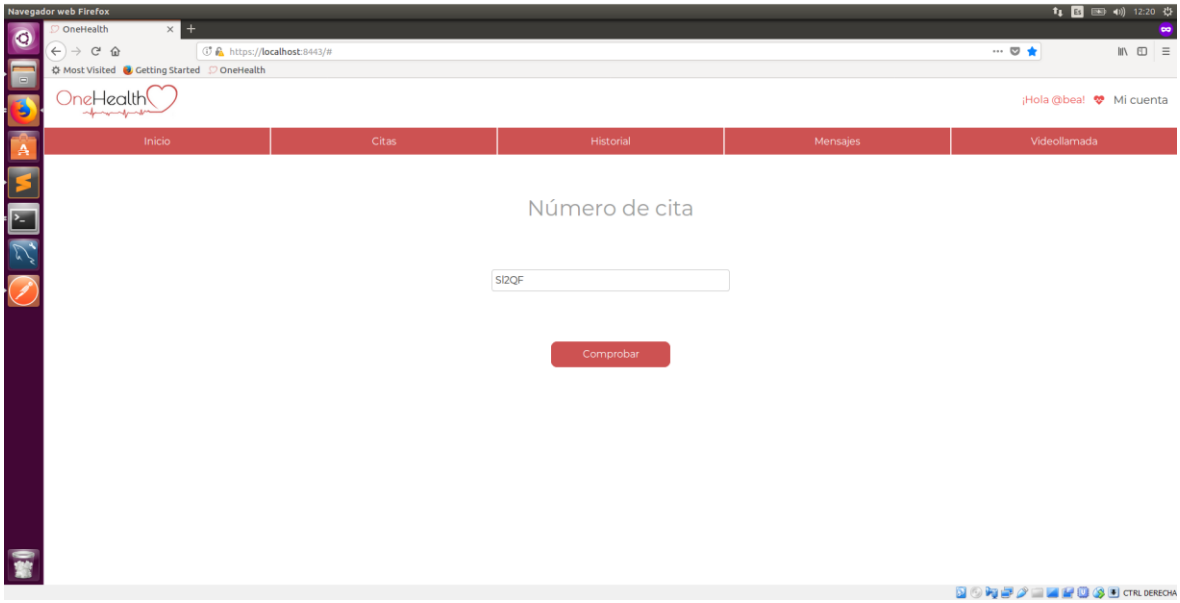


Ilustración 19. Vista comprobar código

Si el paciente introduce un código correcto para su cita, por pantalla se le mostrará los datos de su cita. Además de mostrarle eso, tendrá la opción de empezar la videollamada, la cual se explicará más detalladamente posteriormente, o en su defecto cancelarla si lo ve oportuno.

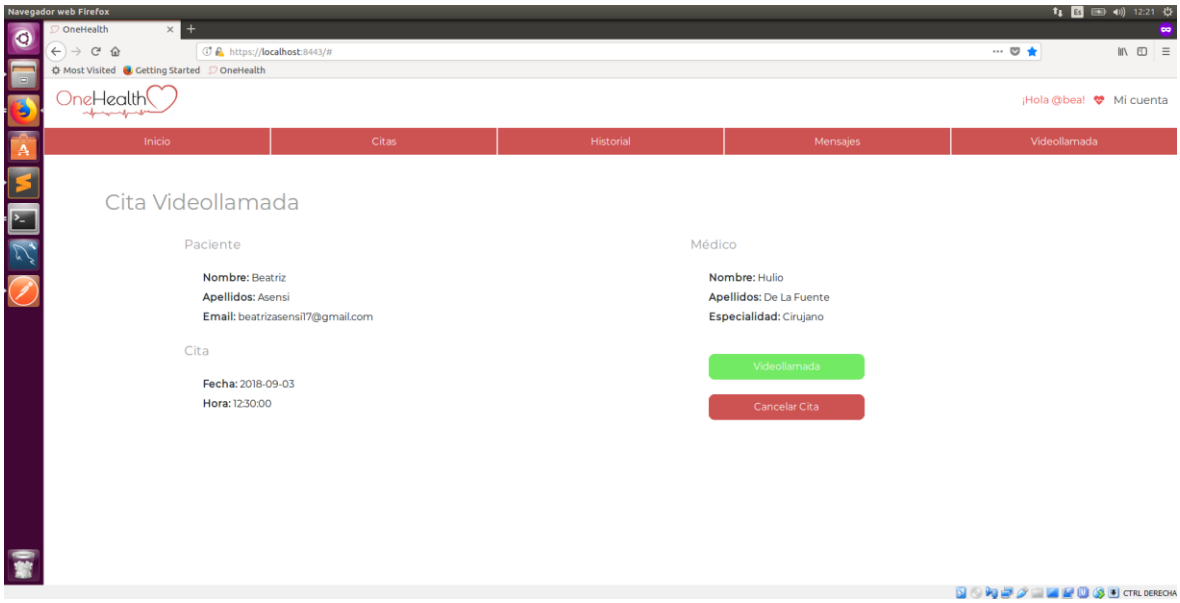


Ilustración 20. Vista ver información de cita

Pasando a la parte del médico, si este quiere llamar a un paciente no necesita saber ningún tipo de código, el tendrá total libertad para poder llamar en caso de cita o urgencia a un paciente, teniendo

como restricción que ese paciente esté conectado a la plataforma, si el paciente no está conectado no aparecerá en la lista de disponibles.

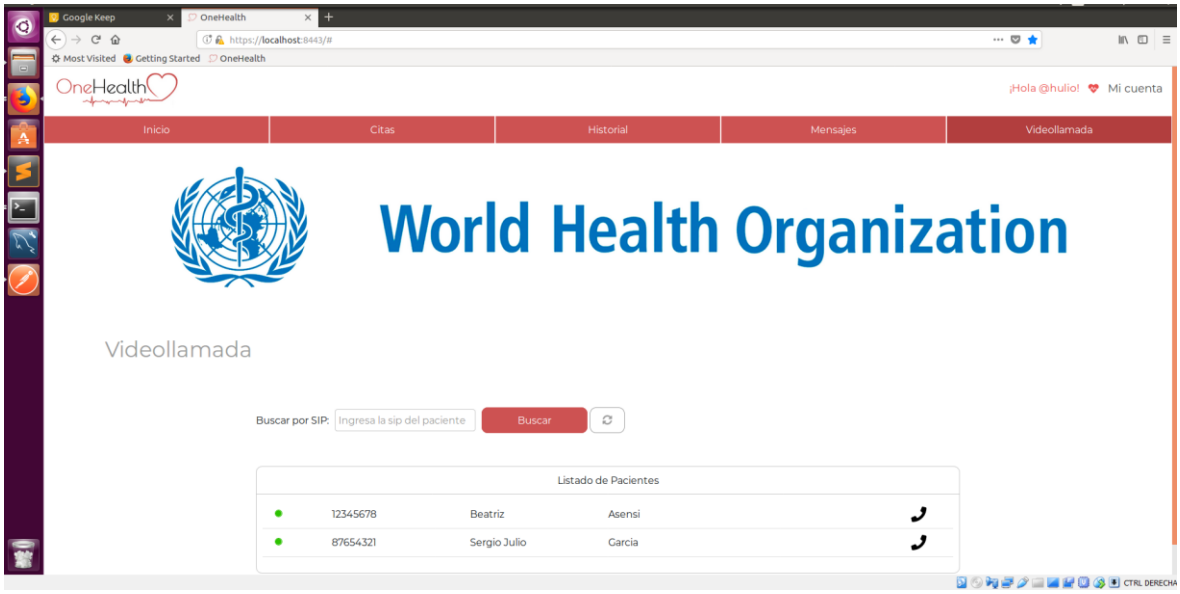


Ilustración 21. Vista listado de pacientes

Para poder hacer una videollamada con cualquier paciente, simplemente tendrá que hacer click en el icono de llamada y se mostrará la vista de la videollamada.

Esta vista será la misma tanto para el paciente como para el médico, la única diferencia es que el médico tendrá una opción la cual es obligatoria para grabar la videollamada. Estando en este paso, el usuario que vaya a iniciar la llamada solo tendrá que aceptar los permisos de video y audio y darle un click al botón verde de llamar, de esa manera al usuario que se le quiere llamar se le notificará en la pantalla que tiene una llamada entrante. Cuando el usuario acepte, ambos se conectarán gracias a WebRTC. En esta conexión además de enviar video y audio de cada uno de los participantes, pueden enviarse mensajes en directo, haciendo así un chat en línea.

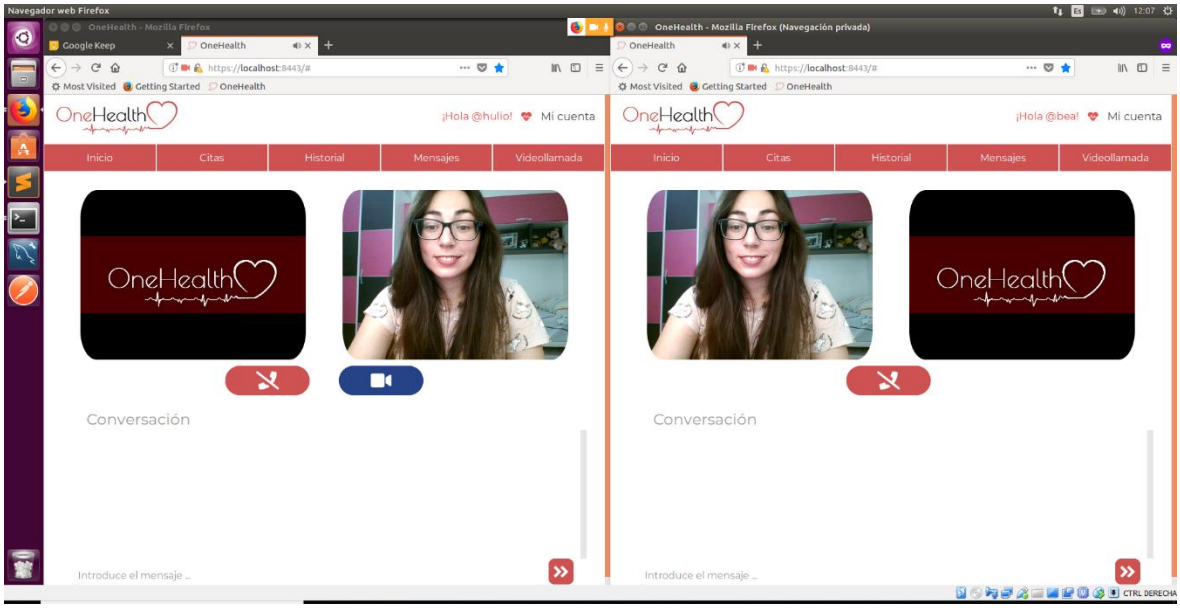


Ilustración 22. Vista videollamada

Cuando alguno de los usuarios quiera colgar la videollamada simplemente tendrá que hacer click sobre el botón rojo de colgar y se cerrará dicha conexión.

Una vez esté cerrada, solo al médico se le dará la opción de guardar la grabación de la videollamada y a continuación rellenar un formulario de consulta.

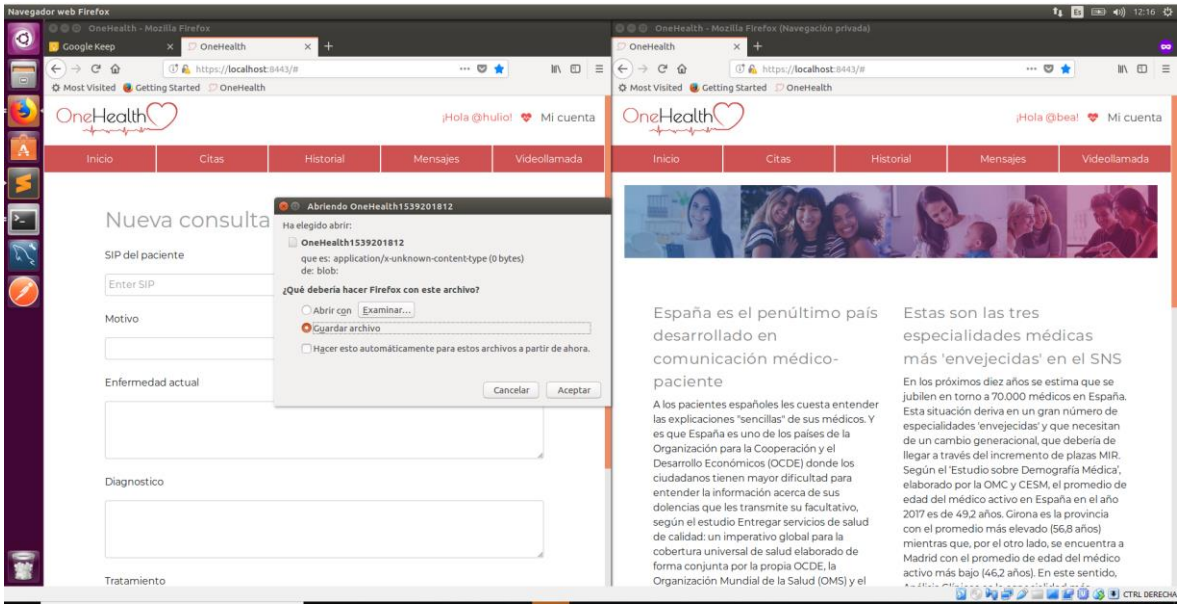
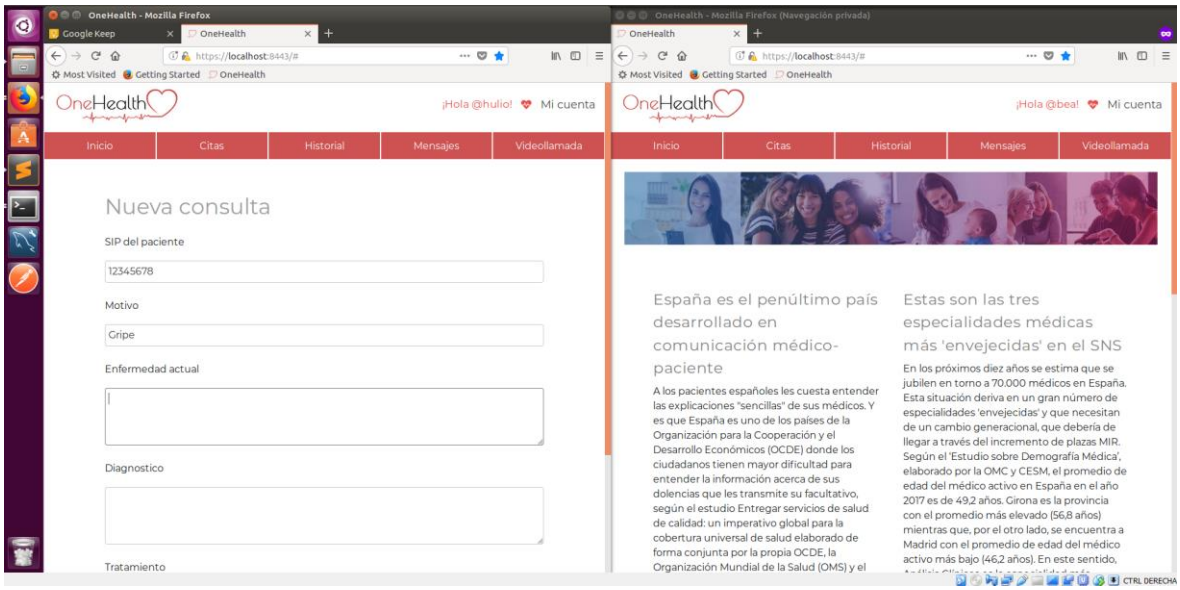


Ilustración 23. Vista guardado de video



- Ver información de videollamada

Esta funcionalidad solo está disponible para el médico. Se llegará a ella a través del apartado de historial de un paciente -> consulta. En esta vista se mostrará el nombre del archivo de video y la conversación que hubo entre paciente y médico durante la videollamada.

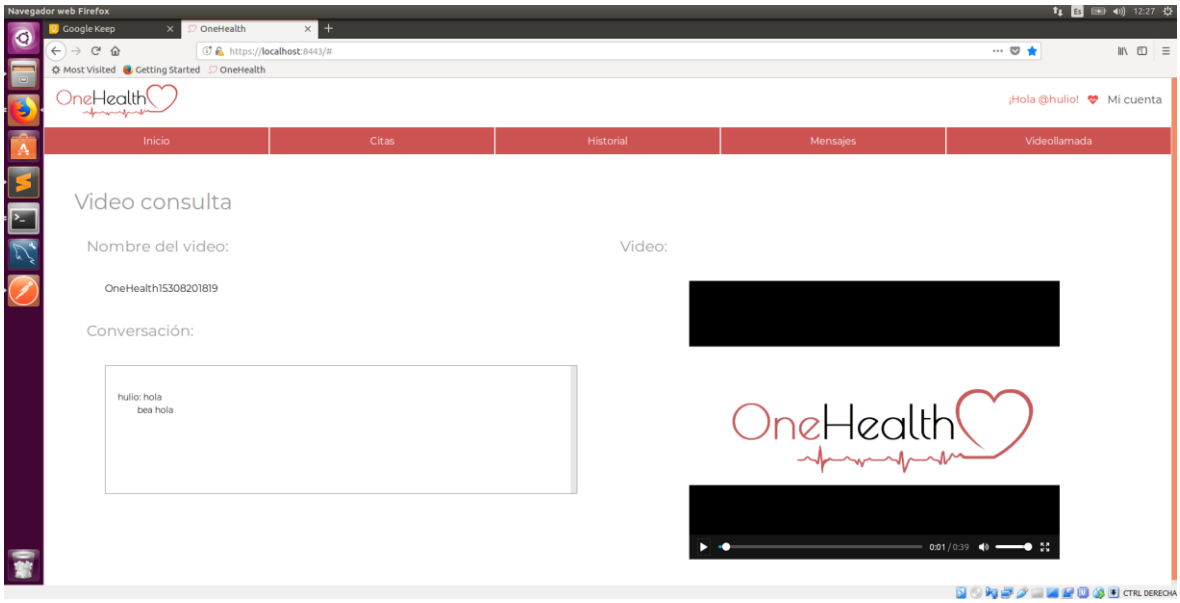


Ilustración 24. Ver información de videollamada

8. CONCLUSIONES

a. REVISION DE OBJETIVOS

Haciendo un repaso del desarrollo final de la aplicación y los objetivos que se propusieron al principio, podemos decir que ha sido un camino difícil, con muchos problemas por el medio, conociendo tecnologías nuevas e intentando adaptar lo mejor posible estas a nuestra aplicación.

Uno de los objetivos que teníamos iniciales fue hacer una grabación entera de la videollamada, con una codificación y cifrado seguro del video, esto no se ha podido hacer al completo, ya que al no poder integrar un servidor multimedia para gestionar las videollamadas, se ha dificultado en su totalidad la tarea de la grabación de ambos usuarios, al igual que la codificación h264, la cual no ha sido soportada por el navegador. Además de esto, al no disponer del tiempo suficiente para la investigación y desarrollo para el guardado de video, no hemos podido manejar del todo bien el almacenamiento del video.

El resto de objetivos del proyecto, en su mayoría han sido cumplidos y realizados de la mejor manera en el tiempo posible.

b. CONCLUSIÓN FINAL

En conclusión, este fue propuesto de manera ambiciosa para aprender nuevas tecnologías, mostrar un sistema dotado de seguridad, con funcionalidades relacionadas con el video, las cuales no hemos hecho nada a lo largo de la carrera y ponernos a prueba con la integración de nuevos conceptos, herramientas y tecnologías. Viendo todo esto, podemos decir que hemos aprendido nuevas tecnologías como WebRTC o socket.io, nos hemos enfrentado a muchos problemas de manera individual, estos han sido resueltos de la mejor manera posible y finalmente hemos conseguido funcionalidades, que al principio del proyecto parecían inalcanzables, como por ejemplo la funcionalidad principal de la videollamada. Con todo esto, solo queremos decir que ha sido un camino largo, lleno de dificultades, pero que al final se ha logrado y aunque hayan cosas que puedan ser mejoradas o puedan haber alternativas mejores, con el conocimiento que teníamos y tiempo, se ha terminado con el sistema de videconferencia paciente-médico.

c. TRABAJOS FUTUROS

Durante el desarrollo de la aplicación hemos tenido varios problemas para la integración de herramientas o funcionamientos óptimos, los cuales por dificultad y tiempo no han podido

resolverse. A continuación se enumerarán los diferentes aspectos de la aplicación a resolver o mejorar:

- Servidor multimedia. Una de las herramientas más fáciles para manejar videollamadas es un servidor multimedia, con el cual se hace más sencillo el manejo de streams, llamadas y grabado de estas. Una de nuestras dificultades fue Kurento un servidor multimedia con el cual tuvimos muchos problemas para comunicarlo con nuestro sistema.
- Grabación. La grabación es una de las funcionalidades de nuestra aplicación a optimizar, ya que ahora mismo la grabación está hecha gracias a WebRTC sólo, por tanto, solo tenemos la grabación de una de las partes de la videollamada. Esta mejora está enlazada con la primera.
- Calidad de video. Ahora mismos la calidad de video que obtenemos de la grabación es de webm, el cual es una calidad baja. Como objetivo teníamos el codificador h264, pero por problemas de compatibilidades y tiempo no ha podido hacerse.
- Seguridad. La parte a mejorar de seguridad de nuestro sistema es en el video, el cual no está cifrado. Esta funcionalidad por tiempo no ha podido completarse.

9. BIBLIOGRAFÍA

Médicos y pacientes (11/12/2013). Obtenido de http://www.medicosypacientes.com/articulo/un-nuevo-portal-permitira-consultar-la-historia-clinica-y-contactar-con-el-medico

Policlínica IUMET (2018). Obtenido de http://www.iumet.es/servicios-telemedicina.php

Aplicación Vida (2018). Obtenido de https://play.google.com/store/apps/details?id=es.vida.android&referrer=adjust_reftag%3DceZwEgCrXcwSj%26utm_source%3DVIDA.es%26utm_campaign%3Dfirstpage-app-promo

Videoconsulta Sanitas (2018). Obtenido de https://www.sanitas.consulting/videoconsulta_sanitas/

HealthTap (2018). Obtenido de https://www.healthtap.com/what_we_make/overview

Desarrollo de un módulo webrtc de HTML5 para la implementación de mensajería instantánea con llamadas de voz y video en los cursos AVA de la UNAD. Obtenido de http://stadium.unad.edu.co/preview/UNAD.php?url=/bitstream/10596/5878/1/1057784190.pdf

Wikipedia (2018). Obtenido de https://es.wikipedia.org/wiki/Wikipedia:Portada

NPM (2018). Obtenido de https://www.npmjs.com/

Servidor https en Node.js (8/11/2015). Obtenido de https://programarivm.com/pon-en-marcha-un-servidor-https-en-node-js-con-express/

2P2 Video Chat with JavaScript/WebRTC (22/07/2015). Obtenido de https://www.youtube.com/watch?v=ieBtXwHvoNk

Getting Started with WebRTC (12/08/2014). Obtenido de https://www.html5rocks.com/en/tutorials/webrtc/basics/#toc-rtcdatachannel

WebRTC Made Simple (16/10/2014). Obtenido de https://blog.carbonfive.com/2014/10/16/webrtc-made-simple/

React 2P2 chat (2015). Obtenido de https://github.com/Codaisseur/react-p2p-chat

React js y el ciclo de la vida de los componentes (14/08/2015). Obtenido de https://medium.com/@pedroparra/react-js-y-el-ciclo-de-vida-de-los-componentes-5d083e5089c6

StackOverflow (2018). Obtenido de https://stackoverflow.com/

Conociendo WebSocket (15/11/2015). Obtenido de https://victordiaz.me/websocket

Node.js y Websockets (3/04/2017). Obtenido de https://manuais.iessanclemente.net/index.php/Node.js_y_Websockets#Introduccion_a_Websockets

Kurento (2018). Obtenido de https://doc-kurento.readthedocs.io/en/stable/

Janus (2018). Obtenido de https://janus.conf.meetecho.com/videocalltest.html

Licode (2018). Obtenido de http://lynckia.com/licode/install.html

Medooze (2018). Obtenido de https://github.com/medooze/media-server-demo-node

MediaSoup (2018). Obtenido de https://github.com/versatica/mediasoup-demo/

Actualización node (26/03/2018). Obtenido de https://askubuntu.com/questions/426750/how-can-i-update-my-nodejs-to-the-latest-version

Errores node (1/02/2017).Obtenido de https://github.com/JeffreyWay/laravel-mix/issues/264

Socket.io (2018). Obtenido de https://socket.io/

Certificado kurento (2017). Obtenido de https://gist.github.com/juhamust/855e5a93f8e2c7abcdc345c5ac8cc9a9

La videoconferencia en la medicina y sus beneficios (09/06/2014).Obtenido de http://med-home.es/?p=1356&lang=es

Aplicaciones de la videoconferencia. Obtenido de http://datateca.unad.edu.co/contenidos/MDL000/ContenidoTelematica/aplicaciones_de_la_videoconferencia.html

Evolución de la videoconferencia (1/12/2014). Obtenido de https://www.digitalavmagazine.com/2014/12/01/evolucion-de-la-videoconferencia-ya-no-es-solamente-para-los-grandes-consumidores/

La telemedicina y la teleconsulta reducen los costes de la asistencia sanitaria y mejoran la atención a los paciente (17/03/2017). Obtenido de http://laesalud.com/2017/esalud/esalud-asturias-asistencia-sanitaria-paciente-teleconsulta-telemedicina/

Generador de Favicon (2017). Obtenido de https://www.favicon-generator.org/

Licencia MIT. Obtenido de https://opensource.org/licenses/MIT

Licencia ISC. Obtenido de https://opensource.org/licenses/ISC

Fontawesome (2018). Obtenido de https://fontawesome.com/

SplitCamera (2017). Obtenido de http://splitcamera.com/

Primeros pasos con WebPack (22/09/2016). Obtenido de https://carlosazaustre.es/primeros-pasos-con-webpack/

Webpack. Obtenido de https://webpack.js.org/configuration/dev-server/

¿Qué es WebRTC? Obtenido de https://www.3cx.es/webrtc/que-es-webrtc/

Seguridad en WebRTC (3/12/2017). Obtenido de http://www.aratecnia.es/seguridad-informatica-webrtc/

Kurento/WebRTC (2014). Obtenido de https://www.naevatec.com/es/soluciones/kurento-webrtc

Draw.io (2018). Obtenido de https://www.draw.io/

Gihub (2018). Obtenido de https://github.com/

Análisis y especificación de requisitos (Tema 2.1). Obtenido de materiales de la UA.

Análisis y especificación de requisitos (Tema 2.2). Obtenido de materiales de la UA.

Casos de uso (Tema 3.2). Obtenido de materiales de la UA.

Diagrama de clases (Tema 3.3). Obtenido de materiales de la UA.

10. ANEXOS

a. KURENTO

Kurento Media Server era la herramienta que se iba a utilizar para el desarrollo de este sistema, pero por varios problemas e incompatibilidades finalmente no se pudo integrar. Antes de explicar los problemas encontrados, vamos a hacer una pequeña explicación de Kurento, la cual esta sacada de la web de NaevaTec.

Kurento Media Server constituye la infraestructura ideal para desarrollar de forma sencilla, y de proporcionar de manera eficiente y escalable servicios de comunicaciones de voz o de videocomunicaciones en tiempo real muy avanzados:

- Video multiconferencias privadas entre múltiples redes.
- Soluciones de compartición y grabación de pantalla. Esto permite construir servicios avanzados de colaboración y customer care en entornos web y móvil.
- Pasarelas entre las redes actuales (IMS, SIP, RTB,…) y redes basadas en WebRTC, esto permite que las apps móviles de su empresa puedan embeber la capacidad de comunicarse con los servicios de call center IVR, de atención al cliente.
- (Video)Call centers accesibles desde apps móviles, tablets, web, …
- (Video) atención de emergencias, también accesible desde apps móviles, tablets, web …
- Video vigilancia, integrando servicios de visión computacional y de realidad aumentada en tiempo real a partir de cámaras fijas, webcams, teléfonos móviles, …
- Y en general cualquier servicio que involucre comunicaciones de audio y video en tiempo real.

A lo largo del desarrollo nos hemos topado con diferentes problemas con Kurento, pero al ser una herramienta muy útil para este proyecto se le destinó gran parte del tiempo a su investigación. El primer problema que nos encontrábamos, fue con los socket que tiene Kurento en su implementación. Al integrar kurento al nuestro proyecto nos dimos cuenta que la conexión de los sockets entre servidor y cliente no funcionaba. Kurento utiliza la librería de ws de npm para los sockets, investigando el por qué funcionaba la conexión, muchos de los post de stackoverflow comentaba que esos sockets no funcionaban del todo bien para la parte del cliente, y recomendaban socket.io. Al ver varios post recomendando lo mismo, modificamos el código de Kurento para integrar socket.io. Una vez que la integración iba y la conexión entre cliente y servidor de kurento funcionaba nos dimos cuenta que al realizar la llamada, cuando nuestro servidor llamaba al servidor de Kurento, el cual estaba instalado y para la demo funcionaba correctamente, siempre daba un error de "reconnect to server 0 100 undefined". Este error quería decir que no encontraba nunca servidor de Kurento, el cual estaba activo. Con esto nos dimos cuenta que estábamos

llamando al servidor de Kurento de manera no segura, cuando en todo momento queremos que nuestras conexión tanto entre cliente como servidor sean seguras. Configuramos todo para que fuese una conexión segura, ya que podría dar error si intentas conectarte de manera segura sin tener el servidor de Kurento configurado. Vimos que de nuevo pasaba lo mismo, y nunca podía conectarse con el servidor. Volviendo a investigar sobre configuraciones y Kurento, realizamos diferentes cambios sobre Kurento, sobre nuestro servidor y seguirá ocurriendo lo mismo. Volvimos a instalar Kurento en una versión diferente de Ubuntu, ya que este framework solo está disponible para desarrollo en Ubuntu 14.04 (Trusty) y Ubuntu 16.04 (Xenial) de 64 bits. Teniendo el mismo fallo, pasamos a ver las dependencias de librerías, probamos diferentes maneras de llamar a la librería sin tener ningún éxito, descargamos la librería para llamarla directamente y tampoco. Finalmente, nos pusimos en contacto con Kurento para ver si podían guiarnos de alguna manera o saber dónde estaba el problema. Viendo que no había ninguna respuesta se optó por eliminar Kurento del proyecto ya que gran parte del tiempo de desarrollo se fue en investigación y pruebas de Kurento sin ningún éxito.