UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Desarrollo de una aplicación de comunicación multimedia a través de internet

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Molinaro , Javier Alexander

Tutor/a: Oliver Gil, José Salvador

CURSO ACADÉMICO: 2021/2022

Resumen

La forma en que nos comunicamos ha cambiado mucho en los últimos años, una de las causas fundamentales se debe a la aparición y perfeccionamiento de la comunicación online, la aparición de este medio de comunicación nos ha permitido crear puentes con gente afín a nosotros, gracias a la existencia de diferentes aplicaciones de comunicación, sin importar la distancia a la que nos encontremos unos de otros.

El proyecto aquí expuesto tiene como objetivo el desarrollo e implementación de una aplicación que permita la comunicación en tiempo real entre distintos ordenadores a través del intercambio de datos multimedia, audio y video, en tiempo real. Esta aplicación será desarrollada utilizando el lenguaje de programación Java y el protocolo TCP.

Palabras clave: audio, video, *webcam-capture*, TCP, Java, JavaFX, comunicación, diseño

Abstract

The way we communicate has changed a lot in recent years, one of the fundamental causes is due to the appearance and improvement of online communication, the appearance of this means of communication has allowed us to create links with people like us, for the existence of different communication applications, regardless of the distance that separates us.

The project presented here aims to develop and implement an application that allows real-time communication between different computers through the exchange of multimedia data, audio and video, in real time. This application will be developed using Java and the TCP protocol.

Keywords: audio, video, *webcam capture*, TCP, Java, JavaFX, communication, design

Resum

La forma en què ens comuniquem ha canviat molt en els últims anys, una de les causes fonamentals es deu a l'aparició i perfeccionament de la comunicació en línia, l'aparició d'aquest mitjà de comunicació ens ha permés crear ponts amb gent afí a nosaltres, gràcies a l'existència de diferents aplicacions de comunicació, independentment de la distància a la qual ens trobem uns dels altres.

El projecte aquí exposat té com a objectiu el desenvolupament i implementació d'una aplicació que permeta la comunicació en temps real entre diferents ordinadors a través de l'intercanvi de dades multimèdia, àudio i vídeo, en temps real. Aquesta aplicació serà desenvolupada utilitzant el llenguatge de programació Java i el protocol TCP.

Paraules clau: àudio, vídeo, *webcam capture*, TCP, Java, JavaFX, comunicació, disseny

Tabla de contenidos

Tabla de ilustraciones

1. Introducción

El presente apartado pretende exponer el problema global a tratar de manera sencilla, teniendo como finalidad realizar una breve introducción del trabajo que describiremos a lo largo del documento.

1.1 Motivación

Desde la aparición del primer teléfono en 1876 nuestra forma de comunicarnos no ha parado de evolucionar, la aparición de Internet ha supuesto una contribución muy importante en esta evolución.

Hoy en día, las distancias ya no son un problema, podemos comunicarnos con personas que se encuentran del otro lado del mundo para tratar los más diversos temas, sin preocuparnos de cosas como del clima, el día de la semana o los servicios postales.

Actualmente existen diversas alternativas para realizar una llamada online, dependiendo de las necesidades que queramos cubrir pueden usarse diferentes alternativas (*Skype, Zoom, Discord,* etc.). Al final lo que el usuario busca es la opción más cómoda y que mejor se adapte a sus necesidades.

La motivación principal que ha llevado a elegir esta temática se basa en el deseo de aportar una nueva alternativa de comunicación a las ya existentes, que se amolde a las necesidades de los potenciales usuarios, no solo a través del envío de audio, como si de una llamada telefónica normal se tratase, sino también a través del envío de imágenes en vivo.

1.2 Objetivos

El objetivo principal del presente trabajo será la descripción y documentación de los pasos realizados para el desarrollo de una aplicación de comunicación multimedia.

Dicha aplicación será desarrollada utilizado el lenguaje de programación *Java* y el protocolo de comunicación *TCP*.

Por tanto, las condiciones que se tienen que cumplir para dar el trabajo por terminado son las siguientes:

1. Obtener imagen desde la webcam disponible en el dispositivo en que estemos ejecutando el programa.
2. Obtener audio desde el micrófono disponible en el dispositivo en que estemos ejecutando el programa.
3. Establecer una conexión entre los dispositivos que se vayan a comunicar a través del protocolo *TCP*.
4. Enviar la información de audio y video de un dispositivo a otro.
5. Reproducir dicha información recopilada de manera correcta, es decir, enseñar por pantalla el video del dispositivo opuesto con el que nos estamos comunicarnos y, a su

vez, reproducir los paquetes de audio por la salida de audio por defecto de nuestra máquina.

6. Crear una interfaz que resulte agradable y sencilla de usar para el usuario final.

1.3 Metodología

Para alcanzar los objetivos planteados se hará uso de la metodología de desarrollo centrado en el usuario (DCU), esta metodología consiste en centrar el diseño de nuestra aplicación en el usuario final.

Comenzaremos realizando una investigación cuantitativa, relacionada con el uso de internet y las aplicaciones de comunicación online, para luego analizar cualitativamente las aplicaciones de comunicación online más relevantes consideradas en la primera investigación.

Con esta información definiremos un perfil de usuario y un posible escenario de uso, los cuales nos ayudarán a diseñar un prototipo que resuelva las necesidades planteadas.

Para finalizar desarrollaremos la aplicación final, previa evaluación del prototipo planteado, y la someteremos a pruebas de funcionamiento.

1.4 Estructura

La presente memoria está estructurada en diferentes apartados que describen el proceso llevado a cabo para conseguir completar el desarrollo de la aplicación. Cada apartado que se vaya presentando tendrá relación con los anteriores, siendo por tanto esta memoria un registro y documentación de un trabajo progresivo.

El contenido que encontraremos en los diferentes capítulos es el que se describe a continuación:

1. **Introducción:** se trata de una exposición global del problema a tratar en el presente documento.
2. **Estado de la cuestión:** se presenta el contexto tecnológico y social actual.
3. **Análisis del problema:** se presentan las soluciones existentes actualmente y la solución que se propone.
4. **Tecnología utilizada:** se listan y describen los tecnologías y herramientas relacionadas y utilizadas en el presente trabajo.
5. **Definición de requisitos:** se listan los requisitos funcionales y no funcionales que deberá intentar alcanzar nuestra solución resultante.
6. **Diseño de la aplicación:** se describirán los pasos seguidos para diseñar nuestra solución y evaluar su diseño.
7. **Desarrollo de la aplicación:** se describirá el proceso realizado para trasladar el prototipo diseñado a una aplicación funcional.
8. **Implantación:** se describe el entorno donde se implantará la solución resultante para la realización de pruebas.
9. **Pruebas:** se expondrán los resultados de las pruebas realizadas a la aplicación.
10. **Actualización de la interfaz:** posterior a las pruebas se realizará una actualización final al apartado gráfico de la aplicación resultante.

11. **Conclusiones:** se exponen las conclusiones alcanzadas con la finalización de este trabajo.
12. **Bibliografía:** referencias bibliográficas.

1.5 Convenciones

Este documento presenta las siguientes normativas de marcado:

- Las palabras extranjeras se representarán en cursiva (ej. *chating*).
- Las citaciones a otros capítulos de la memoria o nombres de estudios estarán en cursiva y con color gris (ej. *1.1 Motivación*).
- Las citaciones textuales de textos u obras externas se entrecomillarán e irán en cursiva (ej. *"El 24 de febrero de 1815 […]").*
- Los nombres de los programas, lenguajes de programación y funciones se escribirán en cursiva (ej. *Java*).

2. Estado de la cuestión

En la presente sección se abordará el contexto tecnológico y social actual en el momento de desarrollo de este trabajo.

2.1 Uso de Internet en España

La rápida implantación en el uso de Internet en nuestro día a día es algo que nadie puede negar, atendiendo a los datos compartidos por el INE (Instituto Nacional de Estadística) en su estudio *6.4 Población que usa Internet (en los últimos tres meses). Tipo de actividades realizadas por Internet* [Ins19] podemos observar que el 93,9% de la población ha usado Internet en los últimos tres meses en el momento que se publicó dicho estudio (2 de diciembre de 2021), esto supone que menos del 7% de la población española no ha usado Internet recientemente.

Si comparamos estos datos con los recogidos en el año 2018, vemos que en ese momento el porcentaje de usuarios de Internet, en el mismo periodo de tiempo, era del 86.6%.

Población que ha usado Internet en los últimos tres meses por edad. Serie 2016-2021 (%)

	2021	2020	2019	2018	2017	2016
Hombres						
Total (de 16 a 74)	**93,9**	**93,2**	**90,7**	**86,6**	**85,5**	**82,5**
De 10 a 15	..	..	..	..	..	..
De 16 a 24	99,7	99,9	99,2	98,3	98,1	98,6
De 25 a 34	99,4	99,6	98,2	97,0	96,0	96,3
De 35 a 44	98,1	98,8	97,0	96,4	95,9	93,3
De 45 a 54	97,4	96,5	92,6	89,6	89,7	85,4
De 55 a 64	90,0	88,6	87,2	78,1	74,5	68,4
De 65 a 74	74,6	70,5	63,7	51,2	47,6	40,6
Mujeres						
Total (de 16 a 74)	**93,9**	**93,2**	**90,7**	**85,6**	**83,7**	**78,6**
De 10 a 15	..	..	..	..	..	..
De 16 a 24	99,6	99,6	99,0	98,7	97,9	98,2
De 25 a 34	99,2	99,7	97,6	98,4	96,5	95,7
De 35 a 44	98,6	99,2	97,8	96,8	95,8	93,3
De 45 a 54	98,7	97,7	96,2	92,5	90,9	84,3
De 55 a 64	92,1	90,3	85,8	74,1	73,2	61,4
De 65 a 74	72,0	68,9	63,5	47,1	40,2	29,4
Brecha de género (hombres - mujeres)						
Total (de 16 a 74)	**0,0**	**0,0**	**0,0**	**1,0**	**1,8**	**3,9**
De 10 a 15	..	..	..	..	..	..
De 16 a 24	0,1	0,3	0,2	-0,4	0,2	0,4
De 25 a 34	0,2	-0,1	0,6	-1,4	-0,5	0,6
De 35 a 44	-0,5	-0,4	-0,8	-0,4	0,1	0,0
De 45 a 54	-1,3	-1,2	-3,6	-2,9	-1,2	1,1
De 55 a 64	-2,1	-1,7	1,4	4,0	1,3	7,0
De 65 a 74	2,6	1,6	0,2	4,1	7,4	11,2

Ilustración 1. Tabla con datos por edad de población que ha usado Internet en los últimos tres meses

Fuente: Instituto Nacional de estadística (2021)

Otro dato relevante que refleja este estudio es que, entre los usos más destacados que se le da a Internet, podemos encontrar la realización de videollamadas a través de internet y el intercambio de mensajes a través de aplicaciones de mensajería instantánea.

Porcentaje de usuarios de Internet en los últimos 3 meses por tipo de actividad realizada. 2021
(% de personas de 16 a 74 años)

	Mujeres	Hombres
Recibir o enviar correo electrónico	82,3	86,2
Telefonear o realizar videollamadas a través de internet	82,3	79,0
Participar en redes sociales	70,9	66,8
Usar mensajería instantánea	97,0	95,1
Leer noticias, periódicos o revistas de actualidad online	79,7	83,4
Buscar información sobre temas de salud	77,8	69,9
Buscar información sobre bienes o servicios	77,5	80,1
Emitir opiniones sobre asuntos de tipo cívico o político en sitios web o en redes sociales	16,4	19,9
Tomar parte en consultas online o votaciones sobre asuntos cívicos o políticos	11,1	12,5
Buscar empleo o enviar una solicitud a un puesto de trabajo	22,2	17,8
Realizar algún curso on line (o parcialmente on line)	32,0	27,1
Utilizar material de aprendizaje on line que no sea un curso completo on line	41,2	40,5
Vender bienes o servicios	17,3	22,2
Banca por internet	66,6	72,2
Concertar una cita con un médico a través de una página web o de una app de móvil	57,5	51,8

Ilustración 2. Tabla con porcentaje de usuarios que han usado internet en los últimos tres meses por tipo de actividad realizada

Fuente: Instituto Nacional de estadística (2021)

Esto quiere decir que la población española con acceso a Internet tiene costumbre de usar medios de comunicación online que involucren audio y video.

2.2 Uso de aplicaciones de videoconferencia

La aparición del coronavirus ha propiciado que el uso de aplicaciones de videoconferencia se dispare. Esto es algo lógico si consideramos que muchos trabajadores han tenido que trasladar sus trabajos a casa, y, que muchos centros de estudio han tenido que adaptar parte de sus clases a este tipo de plataformas.

Por ejemplo, si observamos el top de aplicaciones más descargadas en *Android*, a fecha lunes 15 junio de 2021, podemos ver que entre las primeras 10 aplicaciones más descargadas se encuentran aplicaciones de redes sociales y comunicación.

Es interesante observar que *Google Meet* y *Zoom* se encuentran incluso por encima de *WhatsApp,* la tercera aplicación de mensajería instantánea más descargada el año pasado en este mismo mes [Rub19].

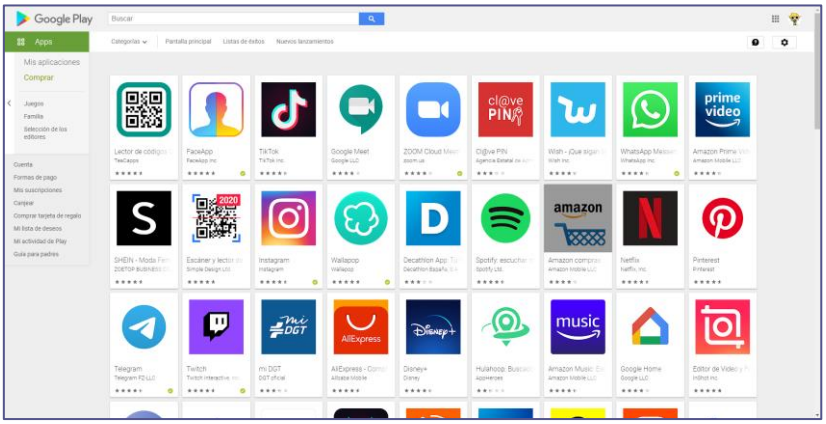

Ilustración 3. Lista de aplicaciones más descargadas en la tienda de Android (15 de junio de 2021)

En el caso de la tienda de aplicaciones de *Apple,* vemos unos patrones más pronunciados. Aquí las tres aplicaciones que encabezan la lista de apps gratis más descargadas son *Google Meet*,

WhatsApp y *Zoom,* en ese orden. A diferencia de lo que vemos en la tienda de *Android,* aquí aparece en el puesto 11 otra alternativa para videoconferencia, *Microsoft Teams*, orientada al teletrabajo.

Ilustración 4. Lista de aplicaciones más descargadas en la tienda de Apple (15 de junio de 2021)

Estos datos se vuelven más significativos si vemos cómo han evolucionado las tendencias de los usuarios a la hora de usar Internet. Una forma fácil de comprobar esto es utilizando los datos brindados por *Google Trends,* una herramienta que nos permite ver las tendencias de búsqueda de los usuarios a lo largo del tiempo, en concreto la gráfica mide el interés de la gente, comprendido en valores de 0 a 100, en un periodo de tiempo determinado.

Con *Google Trends* podemos ver fácilmente el momento del "boom" de las aplicaciones de videoconferencia, concretamente podemos notar como la búsqueda de estos programas (*Skype, Zoom, Microsoft Teams y Google Meet*) se dispara el día 14 de marzo del 2020 a nivel global [Goo20].

Un dato interesante a destacar es que la aplicación más buscada, con diferencia, es *Zoom,* alcanzando su pico máximo de interés, por parte de los usuarios del buscador, el día 2 de abril de 2020, seguido por *Microsoft Teams*, *Google Meets* y *Skype*.

En la gráfica también podemos apreciar claramente unos picos de desescalada bastante pronunciados en la búsqueda de estas aplicaciones, si vemos estos datos más de cerca podemos comprobar que ocurren cíclicamente durante 7 días, concretamente en los días no laborables (sábados y domingos).

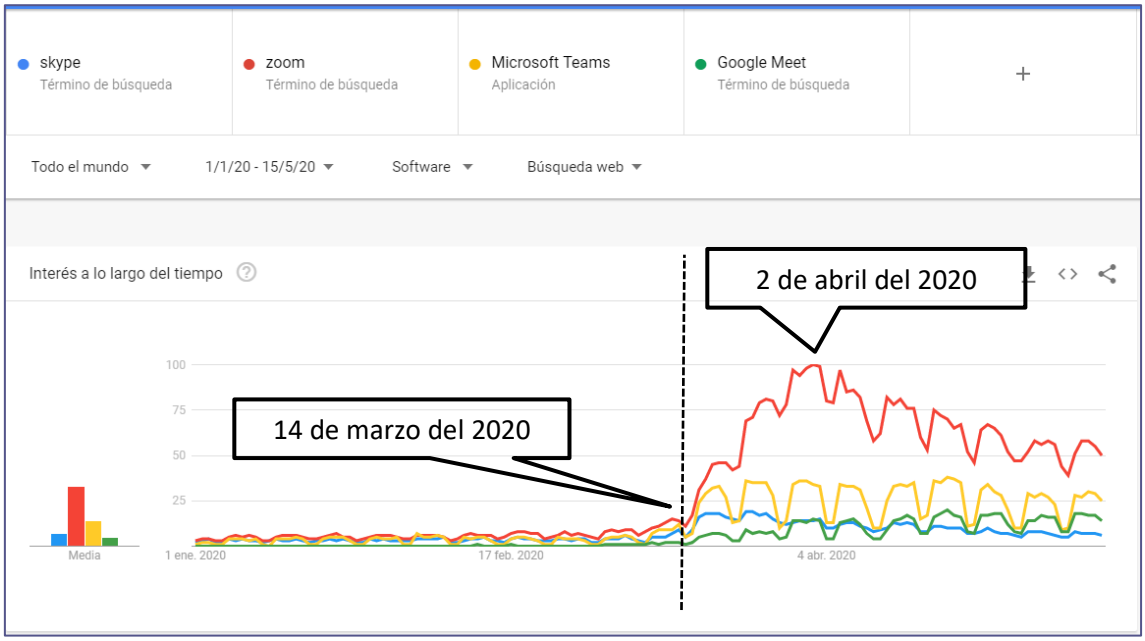

Ilustración 5. Gráfico relación interés/tiempo de las aplicaciones Skype, Zoom, Microsoft Teams y Google Meets

Fuente: Google Trends (2021)

2.3 Características de las aplicaciones de videoconferencia

Ya hemos visto, en el punto anterior, la escalada en el interés por las aplicaciones de videoconferencia por parte de los usuarios, por esta razón, pasaremos a continuación a describir las características de las aplicaciones más populares.

Zoom: es una aplicación multiplataforma, diseñada con el fin de realizar videoconferencia y reuniones virtuales en la nube. En este caso las conexiones a las salas de reuniones se hacen utilizando el protocolo de conexión H.323/SIP [Zoo1].

Este protocolo está compuesto por una serie de estándares que definen como han de comprimirse y enviarse los datos de audio y video, aunque no garantizan su entrega [Pac].

Microsoft Teams: es una aplicación de comunicación en tiempo real, basada en la nube, que integra la posibilidad de realizar videoconferencias, crear salas de reuniones, comunicarse mediante mensajería instantánea, compartir archivos, etc. [Mic].

Esta aplicación utiliza protocolos propios de Microsoft, y otros conocidos. Entre los protocolos propietarios de Microsoft destacaremos el protocolo MNP24, un protocolo para la realización de videoconferencias, y el protocolo H.264 que define el modo en que se envía los datos de video [Dje18].

***Skype*:** es una aplicación multiplataforma distribuida por Microsoft, utilizada para comunicarse a través de Internet por medio de llamadas de voz y videollamadas individuales o grupales, con la posibilidad de enviar mensajes instantáneos y compartir archivos [Sky]. En este caso los

protocolos internos que utiliza Skype son propietarios, por tanto, aunque puede conocerse parte de su funcionalidad, no hay documentación detallada sobre cada uno de ellos.

Google Meet: es un servicio de videoconferencia multiplataforma ofrecido por Google. Al igual que ocurre con el caso de *Skype* el transporte de audio, video y codificación de datos se realiza con protocolos privativos de la misma Google [Wik] [Goo].

2.4 Interpretación de los datos

Atendiendo al estudio *6.4 Población que usa Internet (en los últimos tres meses). Tipo de actividades realizadas por Internet* (Instituto Nacional de Estadistica, 2021) podemos interpretar los datos de la siguiente manera:

Primero analizaremos el caso de los hombres dentro de la población que ha usado Internet en los últimos tres meses. Según los datos que se nos proporcionan vemos que en el año 2021 las personas con edades comprendidas entre 16 a 24 años habían usado Internet recientemente en una proporción del 99,7%, las personas con edades comprendidas entre 25 a 34 años usaron Internet recientemente en el 99,4% de los casos, y las personas con edades comprendidas entre los 35 y 44 años usaron Internet en el 98,1% de los casos. Luego de esto, con edades más avanzadas, se genera una desescalada en el uso de Internet cada vez mayor.

Por tanto, en el caso de los hombres, consideraremos al conjunto de la población con edades comprendidas entre 16 a 44 años (por ser valores con un porcentaje cercano al 100% de uso).

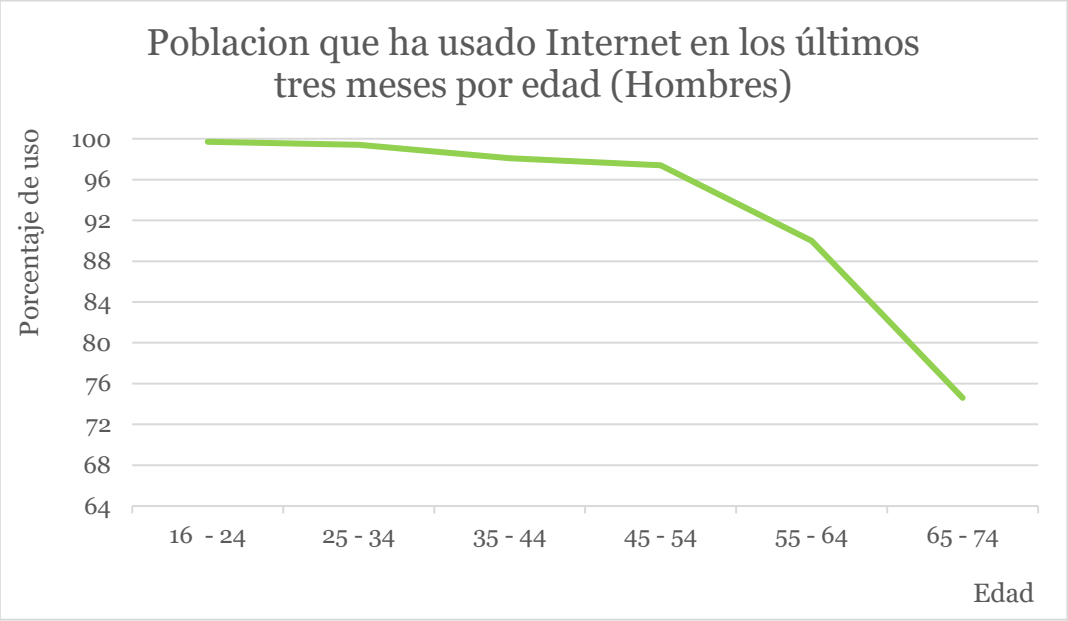

Ilustración 6. Población que ha usado Internet en los últimos tres meses (Hombres)

Seguidamente, interpretaremos los datos con respecto al porcentaje de uso de Internet en los últimos tres meses en el caso de las mujeres. Aquí vemos que en el rango de 16 a 24 años el porcentaje de uso de internet es del 99,6%, en el rango de edades de entre 25 a 34 años el porcentaje de uso disminuye muy poco, siendo este del 99,2 %. En edades comprendidas entre los 35 a los 44 años el porcentaje disminuye ligeramente, siendo este del 98,6%. Seguidamente,

vemos como en el rango de edades de entre 45 y 54 años el porcentaje se mantiene prácticamente igual, siendo este del 98,7 %. Por último, observamos como en edades más avanzadas el porcentaje de uso disminuye de manera más notoria llegando al 72,6 %.

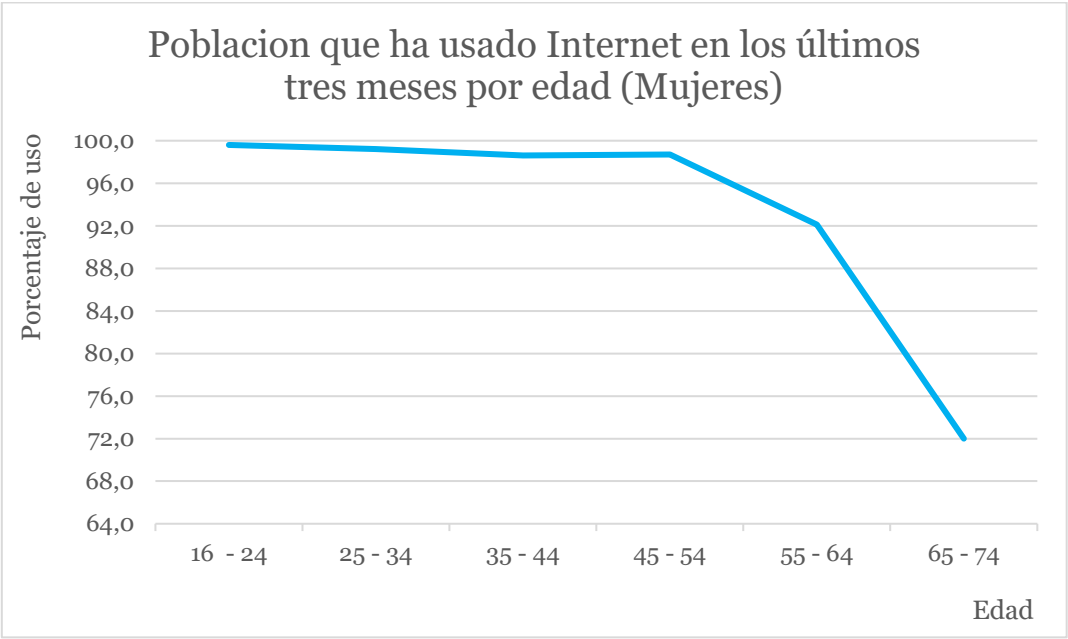

Ilustración 7. Población que ha usado Internet en los últimos tres meses (Mujeres)

Si comparamos los datos del uso de Internet entre hombres y mujeres vemos que la diferencia más significativa, aunque no especialmente crítica, aparece en el rango de edades comprendidas entre los 44 y 64 años, fuera de este rango los valores son muy similares.

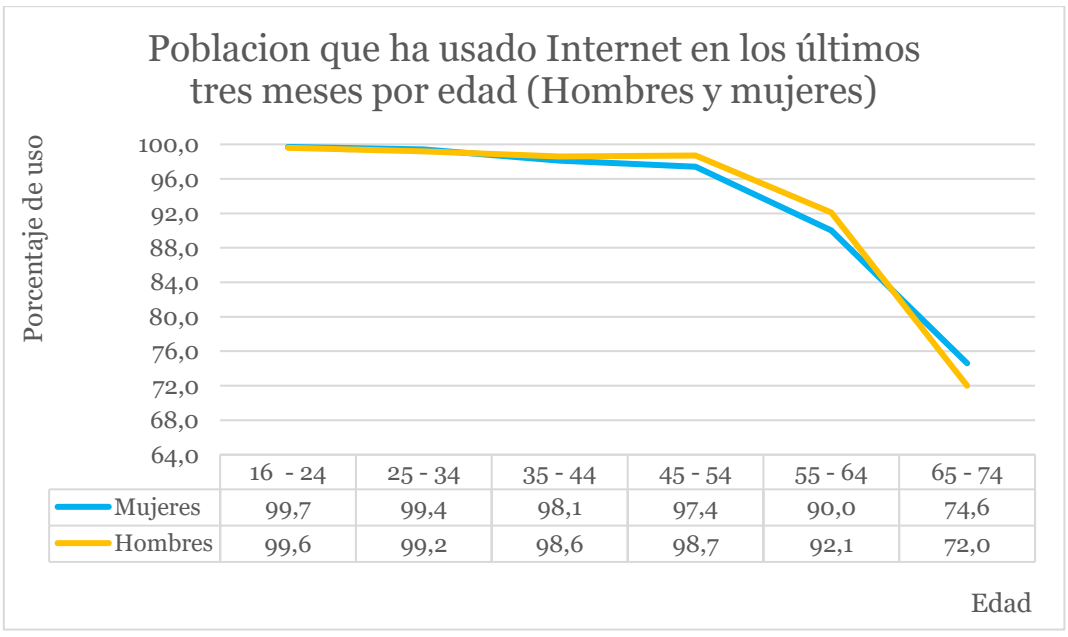

Ilustración 8. Población que ha usado Internet en los últimos tres meses (Hombres y Mujeres)

En este gráfico se presentan los datos recogidos del uso de Internet en España en los últimos tres meses, tanto en el caso de hombres y mujeres. Las diferencias en el porcentaje de uso son las que se exponen a continuación:

- *16 a 24 años: 0,1 puntos de diferencia a favor en el caso de las mujeres.*
- *25 a 34 años: 0,2 puntos de diferencia a favor en el caso de las mujeres.*
- *35 a 44 años: 0,5 puntos de diferencia a favor de los hombres.*
- *45 a 54: 2,3 puntos a de diferencia a favor de los hombres.*
- *55 a 64 años: 2,1 puntos de diferencia a favor de las mujeres.*
- *65 a 74 años: 2,6 puntos de diferencia a favor de las mujeres.*

2.5 Conclusión

En vista de los datos anteriormente citados podemos concluir: por un lado, que el uso y acceso a Internet por parte de la sociedad española está en crecimiento constante, por tanto, en un futuro próximo, será extraño que alguien no haya usado Internet en un plazo superior de tres meses.

Por otro lado, vemos como la costumbre y familiaridad en el uso de aplicaciones destinadas a la comunicación online es cada vez más común, en consecuencia, los usuarios están preparados para desenvolverse con diferentes opciones, ya que la estética de estas aplicaciones suele ser muy similares.

Hay que destacar que cada aplicación utiliza diferentes protocolos y formas de transportar los datos, pero todo buscan el mismo fin, aunque sea por medios diferentes, que es transportar los datos de manera rápida y efectiva de un punto al otro.

3. Análisis del problema

En esta sección analizaremos los datos del apartado *2. Estado de la cuestión* y plantearemos las oportunidades existentes en torno al tema de este trabajo.

3.1 Soluciones actuales

Como hemos visto existen diferentes soluciones en el mercado para cubrir la necesidad de comunicarse a través de internet usando medios multimedia, sin embargo, la mayoría de estas utilizan protocolos propietarios de la misma compañía.

A pesar de no poder conocer exactamente como están construidos dichos protocolos, sí que podemos listar las principales aplicaciones de comunicación y analizar sus características desde el punto de vista del usuario final

3.1.1 Análisis de las aplicaciones de videoconferencia

Comencemos por *Zoom*[1], un vistazo a su página principal nos indica que la aplicación es de descarga gratuita, pero tiene varios planes de uso, uno gratuito y otros de pagos.

El plan gratuito permite organizar reuniones de manera ilimitada siempre que sean conversaciones solo entre dos personas, en el caso de realizar una videoconferencia con más personas la conexión se mantendrá un máximo de 40 minutos.

En el caso de que optemos por pagar uno de los planes disponibles, tenemos diferentes opciones dependiendo de las necesidades que queramos cubrir (*Zoom Pro, Zoom Business, Zoom Enterprise y Zoom Rooms*), pero, en general, lo que conseguimos es eliminar las restricciones que nos impone el plan gratuito y agregar alguna función extra, como, por ejemplo, la posibilidad de grabar reuniones, atención al cliente personalizada, etc. Estas son opciones que quienes más provecho le sacarían serían las empresas.

Dentro de la aplicación tenemos la opción de utilizar o no la webcam, encender o silenciar el micrófono, escribir en un chat en tiempo real, y compartir archivos.

Otro punto interesante que destacar de esta aplicación es que es multiplataforma, existe versiones compatibles con los diferentes sistemas operativos más utilizados (*Windows, Linux, MacOS*, etc.). También dispone de una versión web, por tanto, no hace falta instalarla para poder utilizarla, con la condición de que nos creemos una cuenta en su plataforma.

[1] https://zoom.us/

Ilustración 9. Aplicación de escritorio Zoom

Fuente: *https://zoom.us/*

Ahora pasaremos a recopilar información de *Microsoft Teams*[2]. Nuevamente consultaremos la página oficial de la aplicación que estamos analizando para obtener los datos que nos interesan. Si vamos al apartado de opciones disponibles vemos que al igual que ocurre con otras aplicaciones de este tipo, tenemos la opción de usarlo de manera gratuita u optar por diferentes planes de pago.

La versión gratuita de la aplicación nos permite realizar videollamadas grupales de hasta 50 participantes, compartir nuestra pantalla, programar reuniones a horas determinadas, comunicarnos par un chat escrito y compartir archivos.

Los planes de pago de esta aplicación consisten en agregar a su descarga todo el paquete de *Microsoft 365* y características y servicios extra (Microsoft 365 Empresa Básico, Microsoft 365 Empresa Estándar y Office 365 E3). Entre estas características podemos destacar la posibilidad de grabar las reuniones, aumentar el número máximo de participantes hasta 10.000 personas si pagamos el plan más completo ($20 al mes por usuario), realizar llamadas telefónicas, etc.

A pesar de tratarse de una aplicación propietaria desarrollada por *Microsoft*, se encuentra disponible en diferentes sistemas operativos además del *Windows*, como son *MacOS, Linux, Android y iOS*.

[2] https://www.microsoft.com/es-es/microsoft-365/microsoft-teams/teams-for-home

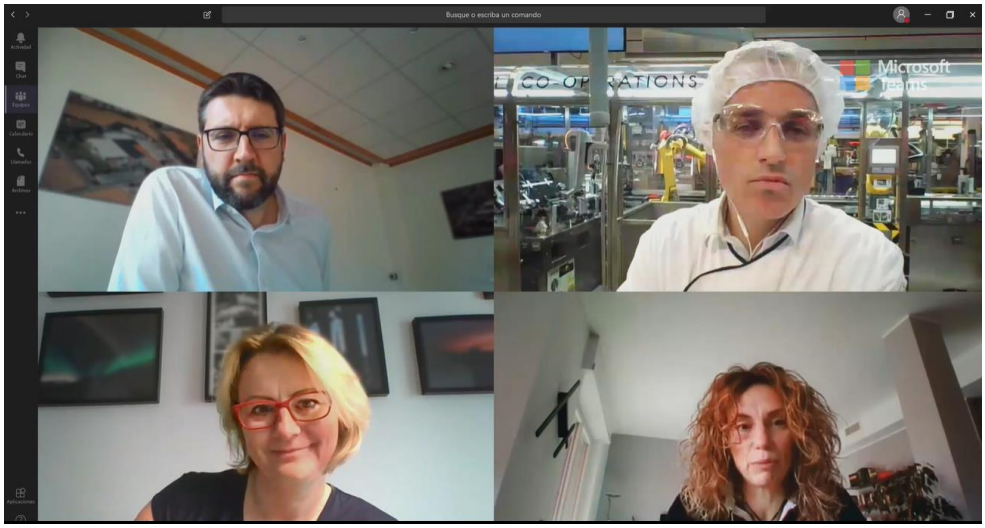

Ilustración 10. Aplicación de escritorio Microsoft Teams

Fuente: *https://www.microsoft.com/es-es/microsoft-365/microsoft-teams/group-chat-software*

La tercera aplicación que veremos es *Skype*[3]*,* explorando su página oficial podemos ver que esta es una aplicación de descarga y uso gratuito. Esta aplicación nos permite realizar videollamadas grupales con hasta 50 participantes con una duración máxima de 4 horas por reunión, con opción de utilizar o no la webcam o el micrófono. Además, tenemos la posibilidad de comunicarnos con un chat escrito, compartir archivos y poder enseñar nuestro escritorio al resto de participantes de la reunión.

Antiguamente existía una versión de *Skype* llamada *Skype Bussiness* pero fue fusionada con *Microsoft Teams* en marzo de 2020.

Existe la posibilidad de pagar una tarifa, con precios diferentes dependiendo del país, para poder tener asignado un número de Skype, este número funciona como un número de teléfono, por tanto, tendremos la posibilidad de utilizar la aplicación con si de un teléfono convencional se tratara.

Skipe se encuentra disponible en *Windows, Linux, macOS, Android, iOS, Xbox* y dispone de una versión web.

[3] https://www.skype.com/es/

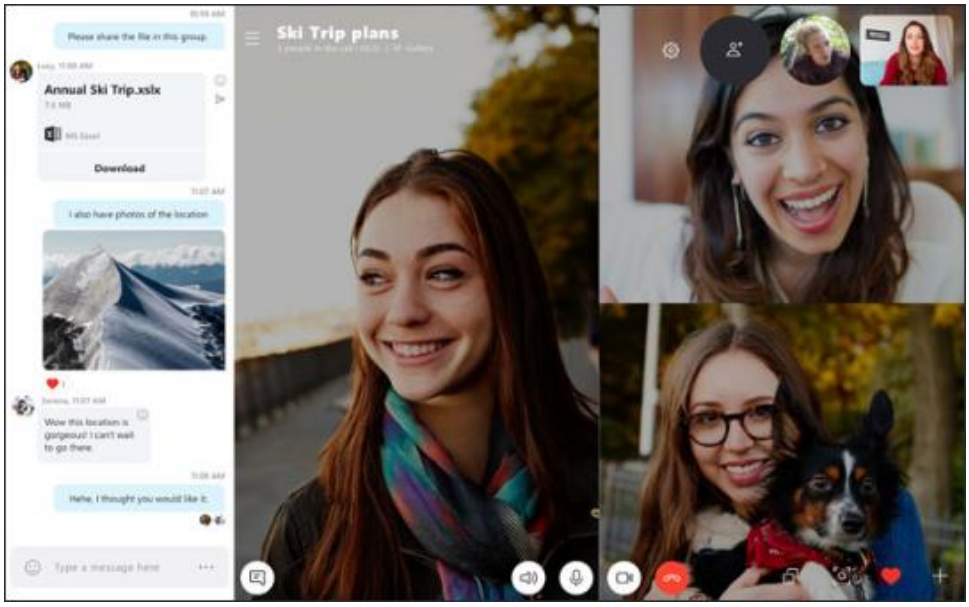

Ilustración 11. Aplicación de escritorio Skype

Fuente: https://www.skype.com/es/

Por último, veremos la aplicación *Google Meet*[4], su página oficial nos indica que existen tres planes distintos, uno de ellos gratis y dos de pago (*G Suite Essentials* y *G Suite Enterprise Essentials*).

En el caso de el plan gratis, podemos realizar videollamadas grupales de hasta 100 personas en sesiones de hasta 1 hora de duración, cada uno de los participantes ha de tener una cuenta de Google. Tenemos la posibilidad de compartir nuestra pantalla, generar subtítulos automáticos a medida que hablamos (solo en inglés), compartir archivos y la posibilidad de comunicarnos en un chat escrito.

Los planes de pago nos ofrecen una serie de características extra, entre ellas destacaremos la ampliación en la duración máxima de las reuniones hasta 300 horas con un máximo de 250 participantes. También nos brindan la posibilidad de grabar las reuniones y guardarlas en *Google Drive*, recibir asistencia a clientes en línea y la opción de compartir entre los integrantes de la reunión una memoria de Google Drive de 100 GB por usuario, o 1 TB dependiendo del plan por el que optemos.

El caso de esta aplicación es un poco diferente que las anteriores, si queremos descargarla solamente podemos hacerlo en los dispositivos móviles con sistemas operativos *Android o iOS*. En el caso de querer usarlo en un ordenador no es necesario descargar nada, lo único que necesitamos es un navegador web y una cuenta de *Google*, por tanto, el sistema operativo en el que intentemos ejecutarlo es indiferente, ya sea *Linux, Windows o macOS*, la aplicación siempre se verá y comportará de la misma manera, manteniendo la consistencia indistintamente de donde la ejecutemos.

[4] https://meet.google.com/

Ilustración 12. Aplicación de móvil Google Meet

Fuente: *https://meet.google.com/*

Si analizamos las características de estas aplicaciones podemos extraer una serie de patrones que se repiten:

- Todas las aplicaciones pueden usarse de forma gratuita.
- El uso del plan gratuito del que disponen las aplicaciones supone una serie de restricciones.
- Las restricciones no suponen necesariamente grandes limitaciones para un usuario básico.
- El pago de planes más avanzados agrega características que pueden resultar útiles para un uso empresarial o educativo.
- Las aplicaciones están diseñadas para poder funcionar en la mayor cantidad de dispositivos posibles.
- Existen opciones para no tener que descargar nada adicional en el dispositivo que se quiera ejecutar la aplicación, estando para este fin las versiones web.
- Permiten compartir archivos entre los participantes del chat.
- Existen diferentes formas de comunicarse dentro de la aplicación y una no es excluyente de otra (micrófono, cámara, chat escrito, etc.).

3.2 Solución propuesta

La solución que proponemos es la de ofrecer una nueva alternativa a las aplicaciones existentes en el mercado. Se propone la construcción y documentación de esta nueva solución desde sus bases hasta su desarrollo final.

La principal ventaja de la que disponemos con respecto a las alternativas ya existentes es que, al tratarse de una aplicación cuya construcción estará totalmente documentada, cualquier persona podrá acceder a esta documentación, entender su funcionamiento y mejorarla o modificarla como dese o crea conveniente.

3.3 Plan de Trabajo

Consideraremos que disponemos de una media de 5 a 6 horas al día para dedicar exclusivamente al desarrollo de la aplicación, con esto en mente presentamos a continuación una estimación de tiempo de las diferentes fases del desarrollo de la aplicación:

Fase de desarrollo	Duración (días)
Análisis de datos	10
Construcción de la persona y escenario de uso	2
Especificación de requisitos	3
Diseño de prototipado	10
Evaluación del diseño	2
Desarrollo	25
Pruebas	7

Si sumamos el tiempo de cada fase nos da un total de 64 días, a estos días le sumaremos un 10% de imprevistos, como pueden ser enfermedad, hospitalización, cortes de electricidad, etc.

Por tanto, el tiempo total de desarrollo sería aproximadamente de 71 días.

4. Tecnología utilizada

En este apartado describiremos las herramientas que se usarán a lo largo de este proyecto, con el fin de poder introducir al lector en los fundamentos teóricos y tecnológicos necesarios para entender la totalidad del trabajo.

4.1 Aplicaciones y herramientas

NetBeans IDE 1.8: es un entorno de desarrollo libre programado en Java, su finalidad es escribir, depurar y compilar código. A pesar de estar escrito en Java, puede ser usado para programar en otros lenguajes de programación [The].

JavaFX SceneBuilder 8.5.0**:** es una herramienta de diseño visual para crear interfaces de aplicaciones *JavaFX.* Las interfaces creadas con esta aplicación se guardan en archivos con extensión FXML, los cuales son vinculables a proyectos Java que definan la lógica de dicha interfaz [Ora].

Justinmind 8.7.8: es una herramienta orientada a la creación de prototipos de páginas web, aplicaciones y diseño de interfaces de usuario, destinados tanto a ordenadores como a dispositivos móviles [Jus20].

API *Webcam Capture v0.3.12:* se trata de una librería creada por el usuario Sarxos que nos permite integrar fácilmente una webcam a proyectos Java [Sar21].

LogMeIn Hamachi v2.3.0.78[5]: es un programa para generar VPNs de manera simple que permite conectar ordenadores remotos como si pertenecieran a una misma red.

4.2 Lenguajes, plataformas y modelos de programación

Java 1.8**:** es un lenguaje de programación multiplataforma, los programas escritos con este lenguaje de programación son ejecutables en cualquier entorno que sea compatible con la máquina virtual de *Java*, por tanto, no se requiere crear versiones específicas para diferentes máquinas con sistemas operativos distintos [Ora1].

[5] https://www.vpn.net/

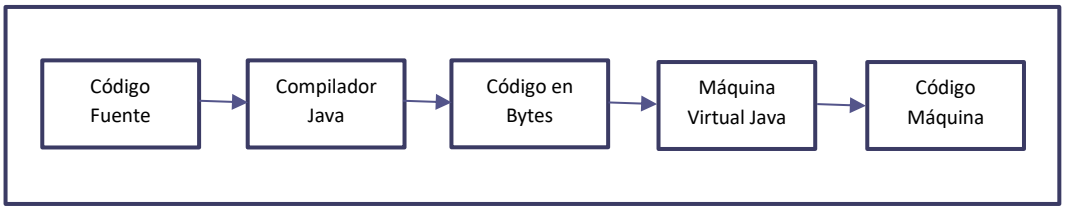

Ilustración 13. Flujo de compilación y ejecución de código Java

Aquí se ilustra el flujo que sigue un código en Java desde su escritura hasta pasar a ser un código entendible y ejecutable por el dispositivo donde se está ejecutando. Partimos de un código escrito siguiendo las reglas de programación de Java, estas reglas serán comprobadas en el momento de compilar el código, si estas reglas se cumplen se genera un archivo con extensión .class, el cual contiene una representación en bytes, entendible por la máquina virtual de Java, del código compilado. Al intentar ejecutar este archivo compilado .class, primero, será interpretado por la máquina virtual de Java, la cual se encargará, a continuación, de traducir este programa en código máquina [Rod19]

***JavaFX 1.8*:** es una plataforma de programación basada y dependiente de Java, destinada a la creación de programas compatibles con contenido audiovisual [Ora2].

***Programación paralela*:** es un tipo de programación en la cual las diferentes actividades que se pretenden ejecutar avanzan de manera simultánea. Las diferentes actividades que conforman el programa son secuenciales, pero cada una de ellas se ejecutará en distintas secuencias de manera simultánea [Muñ13].

4.3 Protocolo TCP

Es un protocolo de comunicación de la capa de transporte encargado del envío de datos. Es un protocolo orientado a la conexón, tanto el cliente como el servidor deben anunciarse y aceptar la conexión entre ellos antes de comenzar a transmitir paquetes.

Alguna de sus características principales son las siguientes:

- Ordena los segmentos provenientes del protocolo IP.
- Evita la congestión de la red a través del monitoreo del flujo de datos.
- Permite la circulación de información de forma simultánea proveniente de diferentes fuentes.
- Cuenta con un sistema de verificación de transmisión de información entre dispositivos.
- Este protocolo utiliza el número de puerto para identificar la aplicación emisora y transmisora.

Podemos diferenciar entre tres tipos de puertos distintos: los puertos bien conocidos, que son los contenidos en el rango del 0 al 1023 y son usados por el sistema, los puertos registrados, que van del rango del 1024 al 4951 y son usados por aplicaciones de usuario de manera temporal, y los puertos dinámicos o privados, que están contenidos dentro del rango del 4952 al 65535, estos son usados por aplicaciones de usuarios [Wik22].

4.4 Principios de Jakob Nielsen

En 1995 Jakob Nielsen citó diez principios de usabilidad, una guía de buenas praxis con el objetivo de que las páginas web fuesen lo más intuitivas de usar posible para los usuarios. Estos principios son los que se citan a continuación:

1) **Visibilidad del sistema**: el sistema debe mantener informado al usuario sobre lo que está ocurriendo en todo momento.
2) **Relación entre el sistema y el mundo real**: los procesos tienen que ser comprensibles por los usuarios, para ello se intentará relacionar las acciones realizadas por los usuarios en la interfaz, con acciones que realizaría un usuario en la vida real.
3) **Control y libertad del usuario**: el usuario mantiene siempre el control sobre las acciones que realiza, si el usuario se arrepiente o equivoca en una de sus acciones puede rehacerlas.
4) **Consistencia y estándares**: la aplicación ha de estar construida utilizando estándares y lenguajes conocidos y/o familiares para el usuario.
5) **Prevención de errores**: la aplicación ha de implementar métodos para que el usuario cometa la menor cantidad de errores posibles.
6) **Reconocimiento antes que recuerdo**: la interfaz tiene que estar dispuesta de tal forma que resulte sencillo para el usuario llegar a las diferentes opciones disponibles sin tener que memorizar los pasos necesarios para llegar a las mismas.
7) **Flexibilidad y eficiencia de uso**: el nivel de experiencia del usuario no debe suponer un impedimento para el uso eficiente de la interfaz.
8) **Estética y diseño minimalista**: la interfaz debe ser minimalista y fácil de leer e interpretar.
9) **Ayudar a los usuarios a reconocer, diagnosticar y recuperarse de errores**: si se produce un error, este será comunicado al usuario, de manera que pueda ser resuelto.
10) **Ayuda y documentación**: se trata de proporcionar información al usuario para que este tenga conocimiento de cómo usar correctamente la interfaz.

Aunque originalmente estos 10 principios fueron propuestos como una serie de buenas prácticas para el desarrollo de web, también pueden ser usadas para el desarrollo de aplicaciones [Ura19].

4.5 Video digital

Un video no es más que una secuencia de imágenes, reproducidas una tras otra en un orden determinado para generar la ilusión de movimiento. Para capturar estas imágenes e "introducirlas" en el ordenador es necesario un dispositivo digitalizador, que será el encargado de capturar dicha información, por ejemplo, una *webcam*.
La *webcam* se encargará de capturará una secuencia de imágenes en forma de onda analógica, para posteriormente pasar a ser convertida a una onda digital, interpretable para un ordenador. Este dispositivo de entrada analizará la imagen que entra por su lente, punto a punto, para posteriormente pasar a calcular el color de esos puntos, codificándolos, cada uno, en una secuencia de bits que representará tres valores: Rojo, verde y azul (RGB) con un valor comprendido entre 0 y 255, esto será transmitido del dispositivo digitalizador al ordenador.

Ilustración 14. Valor RGB de un punto en una fotografía

En esta fotografía de Lenna (1972) podemos ver el valor expresado en RGB de uno de los puntos que componen la imagen, en este punto concreto el valor de rojo es de 173, el de verde 69 y el de azul 77. Estos valores pueden estar comprendidos entre un rango de 0 a 255. Este rango se debe a que cada componente (rojo, verde y azul) está representado con un byte (1 byte = 8 bits = 2^8 = 256) por tanto cada punto tendrá una representación de 24 bits

Un video digital, como su nombre indica, no es más que la representación digital de un video analógico, es decir representado en forma de bits. Para conseguir esta representación digital se siguen tres pasos diferentes:

1. **Muestreo**: en este paso se toman muestras del valor de una representación analógica del vídeo a una frecuencia constante, a esto se le conoce como muestreo uniforme, ya que entre una muestra y otra hay exactamente el mismo espacio de tiempo, esta frecuencia se expresa con la fórmula $\mathbf{f_s = 1/T_m}$, siendo T_m el periodo de muestreo expresado en segundos.
   Una vez tenemos esto se debe cumplir que el ancho de banda de la señal sea más pequeño que la mitad de la frecuencia de muestreo (criterio de Nyquist) para que la señal pueda ser reconstruida sin sufrir pérdidas.

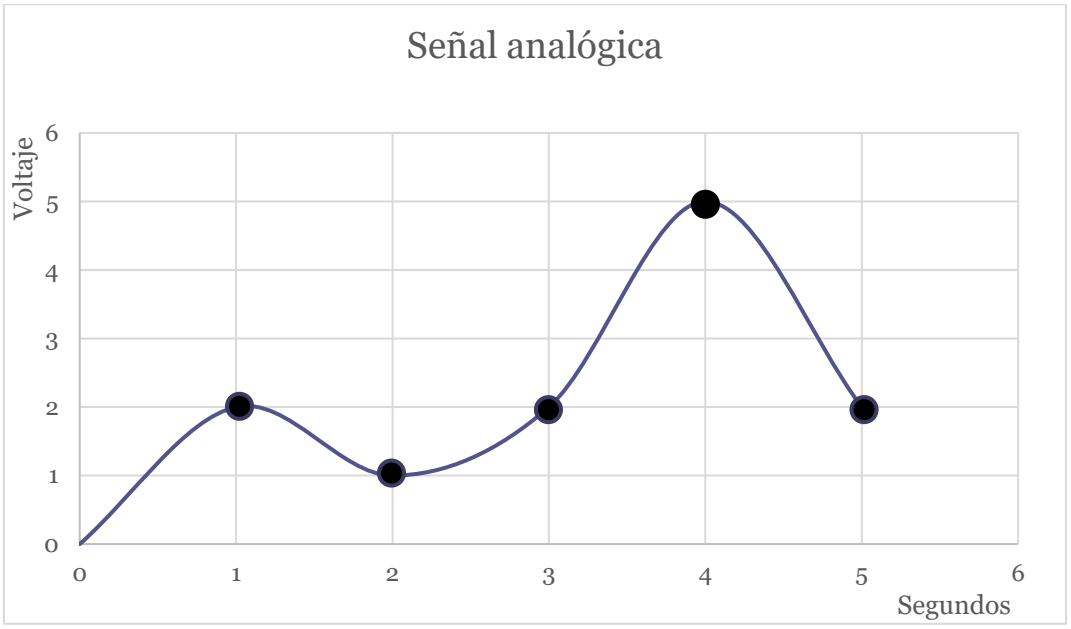

Ilustración 15. Muestreo de una señal analógica

En este ejemplo tenemos una señal analógica donde se toman muestras, representados con los puntos negros, a una frecuencia de un segundo entre muestra y muestra. Lo que nos interesa es el valor que adquiere el voltaje en ese punto concreto de tiempo. En este caso los valores observados son 2, 1, 5 y 2 respectivamente

2. **Cuantificación**: la cuantificación consiste en atribuir a cada muestra tomada de la señal analógica un valor dentro de un margen determinado, estos valores son conocidos con el nombre de niveles.
   El número de niveles empleado será determinante a la hora de obtener una imagen con la menor cantidad de ruido posible, por ello podemos decir que, a mayor cantidad de niveles, menor será el ruido generado. Este ruido es cuantificable, basta con comprobar la diferencia entre el valor en un punto de la señal analógica con el valor de la señal cuantificada, a esta diferencia se le conoce como error de cuantificación.

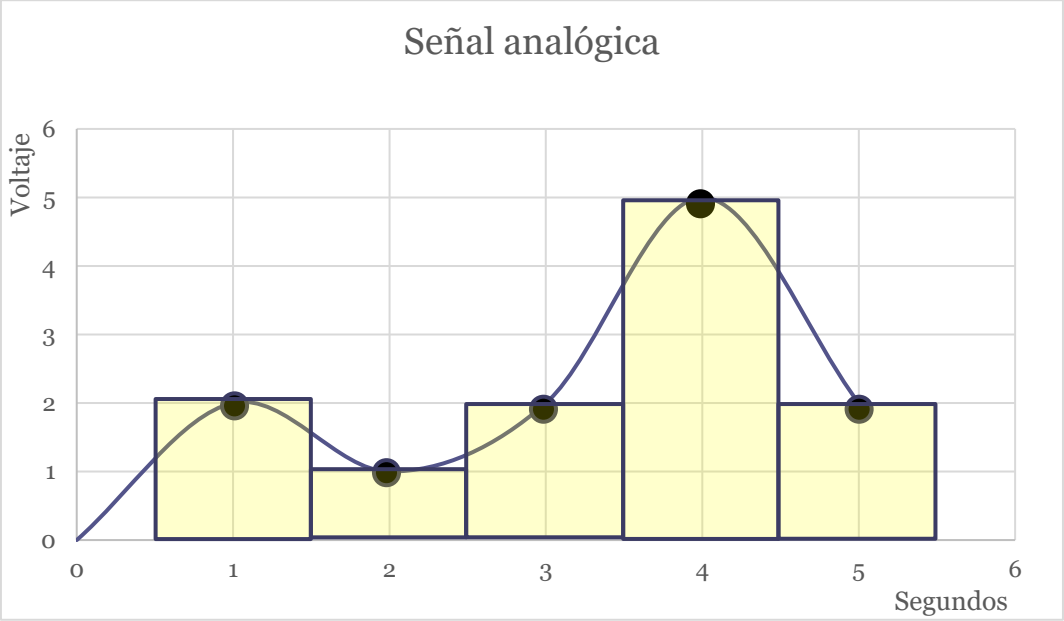

Ilustración 16. Cuantización de una señal analógica

En esta ilustración podemos ver una señal cuantificada a un único nivel por muestra, en la realidad no se utilizan tan pocos niveles, esto está hecho así en esta ilustración para poder verlo de la manera más gráfica posible el error de cuantificación producido

3. **Codificación:** consiste en convertir los valores obtenidos en el proceso de cuantificación a código binario.
   Un detalle importante que tener en cuanta en este punto es la incompatibilidad entre distintas regiones a la hora interpretar estos datos codificados, a no ser que el dispositivo que reproduzca esta señal de vídeo sea de región libre se tendrá que tomar en cuanta, a la hora de realizar la codificación, la región del dispositivo donde serán interpretados estos datos (PAL, NTSC, NTSC-J, etc.) [Gar15] [Ped11].

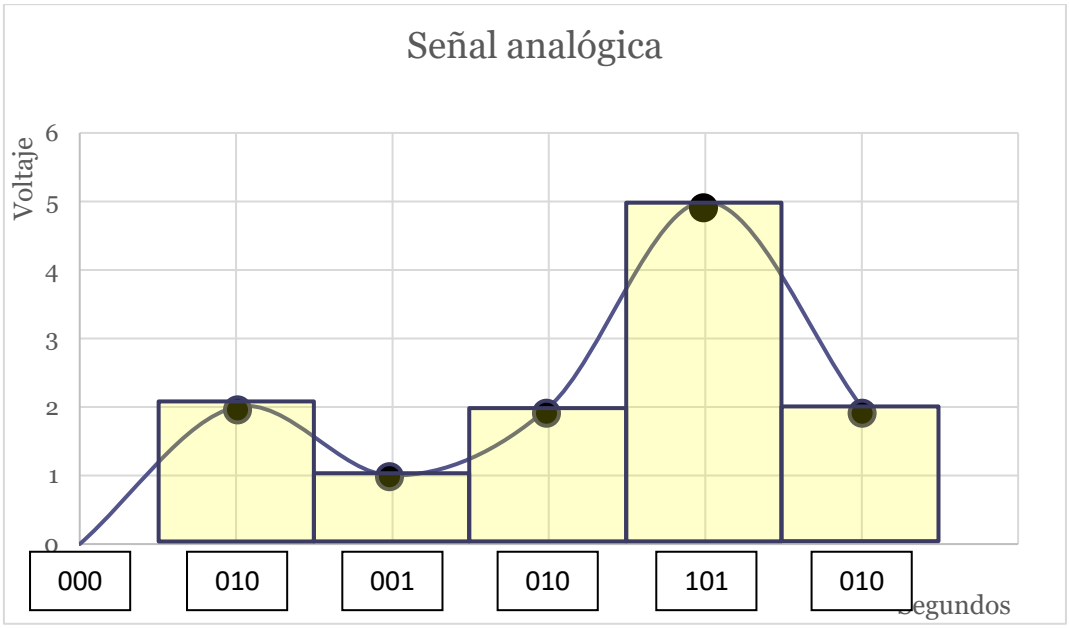

Ilustración 17. Codificación de una señal analógica

En este ejemplo, luego de que la señal esté muestreada y cuantificada, se procede a asignar el valor que adquiere cada muestra, representado en binario. Aquí la representación final de la señal será 000010001010101010

Existen varios formatos de codificación de video diferentes, dependiendo lo que se quiera conseguir, se empleará uno u otro, dos de los más habituales son los citado a continuación:

H.264: se trata de uno de los métodos de codificación de video más utilizados actualmente por su capacidad de compresión, poca pérdida de calidad de imagen y compatibilidad con dispositivos.
Utiliza métodos de compresión cuando se presenta redundancia temporal y espacial utilizando un algoritmo de predicción de bloques que aprovecha la información de los bloques previamente codificados.
La redundancia temporal se da cuando de un fotograma a otro se comparten píxeles idénticos, por el contrario, la redundancia espacial se da cuando en un mismo fotograma se repite una serie de pixeles idénticos [Och07].

MPEG-2: este Método de codificación de video está diseñado para la transmisión de audio y video en vivo (TDT, satélite, etc.). MPEG-2 utiliza una compresión de video con pérdidas, su finalidad es crear paquetes ligeros para conseguir un envío rápido de la información.

Estos dos protocolos utilizan el método de compresión GOP (*Group Of Pictures)*.
Cada grupo de imágenes está formado por tres tipos de imágenes, el primer son las imágenes de tipo I (*I-frame*), estas imágenes son aquellas que van a ser tomadas como referencia para comprimir el resto de las imágenes que componen la secuencia, el segundo son las imágenes de tipo P (*P-frame*), estas son imágenes generadas mediante predicción en base a la información de los fotogramas anteriores, por último tenemos las imágenes de tipo B (*B-frame*), estas imágenes son, al igual que el caso de las imágenes de tipo P, imágenes generadas mediante información de otros fotogramas, con la diferencia que estas toman como referencia una imagen anteriores y otra posteriores.
Los fotogramas de tipo I y P, pueden ser utilizados como referencia para generar otros fotogramas, cosa que no ocurre con los de tipo B.
La imagen de tipo I, en contraposición a los otros dos tipos, no elimina redundancia temporal, pero si espacial. [Man].

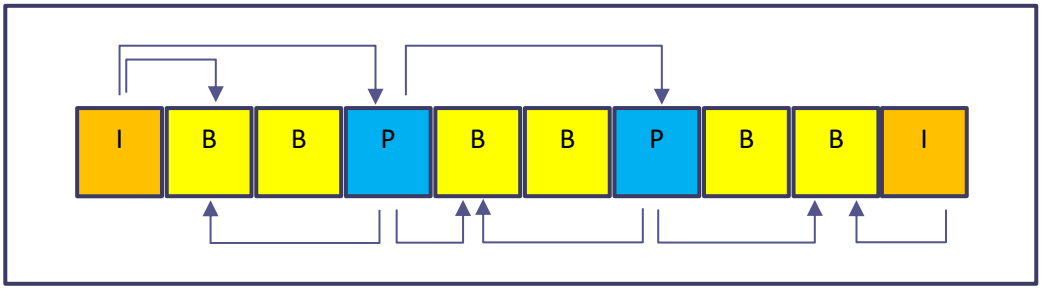

Ilustración 18. Group of Pictures, estructura típica del formato PAL

En este ejemplo se presenta la estructura GOP típica empleada en el formatio PAL. Cada GOP esta contenido entre dos I-frames. En este caso tenemos que el segundo frame se codifica como tipo B tomado como base el fotograma anterior I y el fotograma posterior P. A continuación, el curto fotograma se codifica como tipo P tomando la información del fotograma inicial I, luego de este encontramos un fotograma codificado como tipo B tomando como referencia el fotograma anteriór P y septimo fotograma que es, también, de tipo P. Despues vemos como el septimo fotograma se codifica como tipo P utilizando como referencia el cuarto fotograma P de la secuencia. Para finalizar, tenemos como el penúltmo fotograma se codifica como tipo B tomando como referencia el septimo fotograma de la secuencia, de tipo P, y el último fotograma, de tipo I.

4.6 Audio digital

Un sonido no es más que una onda acústica que se transporta a través del aire y es interpretado por nuestros oídos. Los humanos podemos oír ondas que tengan una representación comprendida entre los 20 Hz y 20 kHz, pero dentro de esta representación, los sonidos que mejor oímos son aquellos que se encuentren en el rango de 1 a 5 kHz, aunque esto depende también de la edad de la persona. Por poner un ejemplo, la voz humana en un tono normal tiene una amplitud entorno a los 3,5 kHz.

Como hemos dicho, el sonido, y por tanto el audio, es una onda, y como tal su digitalización es similar a lo visto en el apartado *4.5 Video digital*:

El sonido es captado por el dispositivo digitalizador, en este caso el micrófono, en forma de onda analógica, este es procesado por la tarjeta de sonido del dispositivo, en donde se convierte de

onda analógica a onda digital siguiendo los mismos pasos y parámetros mencionados en el apartado anterior (muestreo de la onda, cuantización y codificación.

De la misma manera que ocurre con el caso de los videos, existe diferentes formas de codificar un audio para conseguir que ocupe menos y/o conservar una buena calidad de audio, entre estos métodos destacaremos alguno de los dirigidos a la compresión de audio en diferentes calidades:

***PCM*:** se trata de una forma de codificar el audio sin comprimir, con niveles de cuantización equidistantes, compatible con hasta 8 canales de audio diferentes. Como se encuentra sin comprimir, la equivalencia de esta conversión de onda analógica a digital no presenta omisiones, por tanto, tiene la mayor calidad de audio, pero también los archivos más pesados. Los formatos que soportan este tipo de codificación son WAV, AIFF, SU, AU y RAW [Wik20].

***DPCM*:** Aquí se codifica la primera muestra de la onda en PCM, esta muestra codificada se utiliza como referencia para codificar solo las diferencias en el resto de los datos, es decir que cuantiza la diferencia entre la muestra de mayor calidad y el valor predicho.

MP3: Es uno de los formatos de audio comprimido con pérdidas más populares usados actualmente. Aquí se codifican las muestras de la onda aprovechando las frecuencias que puede captar nuestro oído. Cuando escuchamos un sonido más alto que el resto, se produce lo que se conoce como sonido enmascararte, como su nombre indica este sonido enmascarará a los sonidos más bajos que él, haciendo que nuestros oídos no puedan escucharlos. Tomando esto en cuenta, se eliminan o disminuyen los niveles de estas redundancias sonora (dependiendo de la calidad que queremos obtener) para comprimir los audios. [Ena22]

AAC: Se trata de un formato de audio comprimido con pérdidas. Es muy similar a *mp3,* al igual que este, aprovecha las frecuencias que nuestros oídos no escuchan para eliminar esta redundancia sonora. Su principal característica reside en el tamaño y calidad de los archivos resultantes, un archivo codificado en *aac* a 128 kbps puede tener la misma calidad que un *mp3* codificado a 256 kbps, además de poder gestionar frecuencias de sonido más altas y bajas que un *mp3* [Wik221].

5. Definición de requisitos

En este apartado definiremos los requisitos funcionales y no funcionales que deberá tener nuestra aplicación software. Esta definición es necesaria para especificar una guía de ruta sobre los aspectos a desarrollar en la aplicación final.

5.1 Requisitos funcionales

En este punto se definirán los requisitos funcionales de la aplicación, es decir, aquello que podrá hacer el usuario, y tendrá a su disposición, una vez tenga la aplicación delante:

- Cualquier usuario podrá iniciar una videollamada con otro siempre que los dos se encuentren disponibles.
- Pantalla inicial con introducción de datos obligatorios: para saber con quién se va a establecer la conexión en la videollamada es indispensable que se introduzca, por un lado, un nombre de usuario para identificarnos dentro de la conversación y por otro el destinatario con el que nos vamos a conectar, el establecimiento de conexión con el destinatario se realizará con un número o clave que lo identifique (como puede ser la IP).
- Imagen de los participantes: la visión de lo que está captando la webcam, y compartiéndolo con el resto de los participantes de la videoconferencia, ha de estar clara en todo momento.
- Silenciar audio: tiene que presentarse de manera clara y visible una opción para silenciar el audio de la conversación.
- Apagar cámara: tiene que presentarse de manera clara y visible una opción para cancelar el envío de imágenes desde la webcam.
- Terminar llamada: ha de existir una opción clara para poder terminar la llamada entre los participantes en cualquier momento.

5.2 Requisitos no funcionales

Los requisitos no funcionales son características generales y restricciones que tendrá la aplicación a desarrollar:

- Eficiencia:
    - La aplicación ha de ser capaz de enviar información a cada extremo de la conexión.
    - Los estados de los periféricos usados por el usuario han de mantenerse hasta que este decida cambiarlos de estado (encenderlos o apagarlos).
    - Una vez establecida la comunicación se ha de poder mantener una conversación entre los participantes por algún medio.
- Usabilidad:
    - La interfaz ha de ser fácil de aprender a usar.

  - Se ha de mantener un mismo lenguaje en cada ventana que integre la aplicación.
  - Las opciones tienen que estar a la vista o ser accesibles con pocos clics.
  - Las opciones han de evitar realizar modificaciones que puedan inducir a errores.
- Compatibilidad:
  - Esta aplicación será desarrollada usando el lenguaje de programación Java, por tanto, los requisitos mínimos que necesitaremos para ejecutarla, una vez se termine su desarrollo, serán:
    - Un ordenador con la máquina virtual de Java instalada en él.
    - Disponer interfaz de entrada y salida de red en el ordenador donde se vaya a ejecutar.
    - Los dispositivos que se van a comunicar deberán estar conectados a la misma red.

5.3 Otros requisitos

Existen también otros requisitos que no consideraremos tan importantes pero que tendremos en cuenta a la hora de construir el prototipado, estos serán requisitos extras que no resultarán críticos en su implementación para completar la aplicación final:

- Función de chat escrito: se considerará la implementación de un envió de texto escrito entre los participantes de la reunión, para cubrir el caso de que tanto cámara como micrófono no se encuentren disponibles.
- Nombre de los participantes: es necesario que el nombre de los participantes de la videoconferencia sea visible en el caso de establecer una comunicación escrita, por no disponer o no querer usar la webcam, de esta manera evitaremos confusiones y sabremos quien dice que en cada momento.
- Implementación de foto personalizada: en el caso de que la cámara no esté disponible se podrá enseñar a los participantes de la videollamada, una foto escogida por el usuario.

6. Diseño de la aplicación

En este apartado utilizaremos los datos anteriormente documentados para realizar el diseño de la aplicación. Para ello definiremos el perfil de la persona a la que irá dirigida la aplicación, describiremos un hipotético escenario de uso y presentaremos el diseño del proyecto en base a los requisitos que consideremos básicos para el buen funcionamiento de la aplicación.

6.1 Construcción de la persona

Para la definición del perfil de la persona utilizaremos los datos del apartado *2.4 Interpretación de los datos*. Con este fin montaremos una ficha de un usuario potencial donde definiremos su identidad, biografía, características y actitudes generales. El objetivo de hacer esto es tener siempre en mente los objetivos y necesidades del usuario para el que desarrollemos la aplicación.

Biografía:

- Nombre: Marta.
- Edad: 30 años.
- Ocupación: Empresaria.
- Es muy activa en redes sociales.
- Se comunica con sus amigos y compañeros de trabajo a través de internet.
- Le gusta ver videos y películas en YouTube y Netflix.
- Prefiere usar aplicaciones gratis, pero no tiene problema en invertir dinero si esto le proporciona un beneficio en la forma de organizarse.
- Le gustaría aprovechar mejor su tiempo.
- Prefiere los medios audiovisuales a los escritos.

Objetivos:

- Comunicarse de manera rápida con sus compañeros de trabajo.
- Tener todo lo más centralizado posible.
- Aprovechar el tiempo de trabajo al máximo.
- Tener más tiempo libre para desarrollar sus hobbies.

Tecnología:

- Utiliza su ordenador y móvil a diario.
- Le resulta sencillo navegar por internet.
- Ha tenido tanto móviles *Android* como *iOS*.
- Le resulta familiar comunicarse con aplicaciones de videoconferencia.

6.2 Escenario de uso

En este apartado describiremos un hipotético escenario de uso de la aplicación basándonos en el perfil de usuario que hicimos en el apartado anterior. Para esto plantearemos una situación lo más realista posible donde nuestro usuario utilizara la aplicación para cumplir un objetivo determinado.

Marta está teletrabajando en su casa desde hace unas semanas, no le ha resultado difícil adaptarse a esta nueva forma de organizarse ya que en su vida laboral le era muy normal realizar algunas reuniones por videoconferencia cuando no era necesario desplazarse. Ha recibido esa mañana un email para concertar una reunión con un cliente que se celebrará a las 4 de la tarde.

A la hora acordada Marta accede a la aplicación de videoconferencia, coloca su nombre en la casilla correspondiente y los datos necesarios para establecer la conexión, en este caso un número que le comunico su cliente por email. Una vez ha hecho esto hace clic en "Conectarse" y espera a que la aplicación le redirija a la sala de chat. aquí dentro ve, por un lado, su imagen procedente de su webcam y la webcam de la persona con la que va a tener la reunión. Avanzada la reunión, Marta necesita consultar sus papeles un momento, para no hacer ruido, aprovecha que su cliente está hablando para silenciar su propio micrófono y que no se escuche el ruido de papeles, para ello hace clic en el botón con el símbolo del micrófono. Antes de finalizar la conferencia el cliente se excusa que tiene que retirarse un momento, así que el apaga el micrófono un momento, haciendo clic en el botón con el símbolo del micrófono, y la cámara, haciendo clic en el botón con el símbolo de una cámara. A los minutos el cliente vuelve y enciende la cámara y el micrófono con los mismos botones que tocó para apagarla, de esta manera continua la reunión. Una vez finalizada la reunión Marta y su cliente se despiden y hacen clic en "Desconectar" para finalizar la videoconferencia.

6.3 Interfaces gráficas de las aplicaciones existentes

Antes de volcarnos en el diseño de la interfaz gráfica de nuestra aplicación, es conveniente ver el aspecto que presentan las aplicaciones existentes más populares, con esto nos aseguraremos de diseñar una interfaz que resulte familiar al usuario final.

6.3.1 Zoom

Al abrir la aplicación lo primero que se nos presenta es una ventana para ingresar nuestros datos e iniciar sesión, pudiendo usar una cuenta ya creada o utilizando los datos de cuentas ya existentes en otras plataformas, como pueden ser las cuentas de *Google* o *Facebook*.

Ilustración 19. Ventana de inicio de Zoom

Una vez iniciamos sesión nos aparece una nueva ventana con diferentes opciones separadas por pestañas.

La primera pestaña está etiquetada con el nombre de *Empezar*, en ella encontramos las opciones: iniciar una nueva reunión, unirse a una sala creada por otra persona, revisar una agenda virtual donde aparecen las reuniones programadas y compartir la imagen de nuestra pantalla.

La segunda pestaña se llama *Chat,* como su nombre indica aquí tenemos la opción de enviarnos mensajes escritos con otros usuarios, no tiene por qué ser con usuarios de *Zoom* ya que se nos permite vincular el chat de otras aplicaciones compatibles.

La tercera pestaña se llama *Reuniones*, y nos permite programar las próximas videoconferencias que vamos a realizar, con que usuarios y a qué hora, también tenemos un registro de las reuniones que ya hemos realizado y sus grabaciones.

La última de estas pestañas corresponde a *Contactos,* aquí tenemos un registro de los contactos vinculados a nuestra cuenta, además de poder realizar canales personalizados para agruparlos e interactuar con estos a la vez.

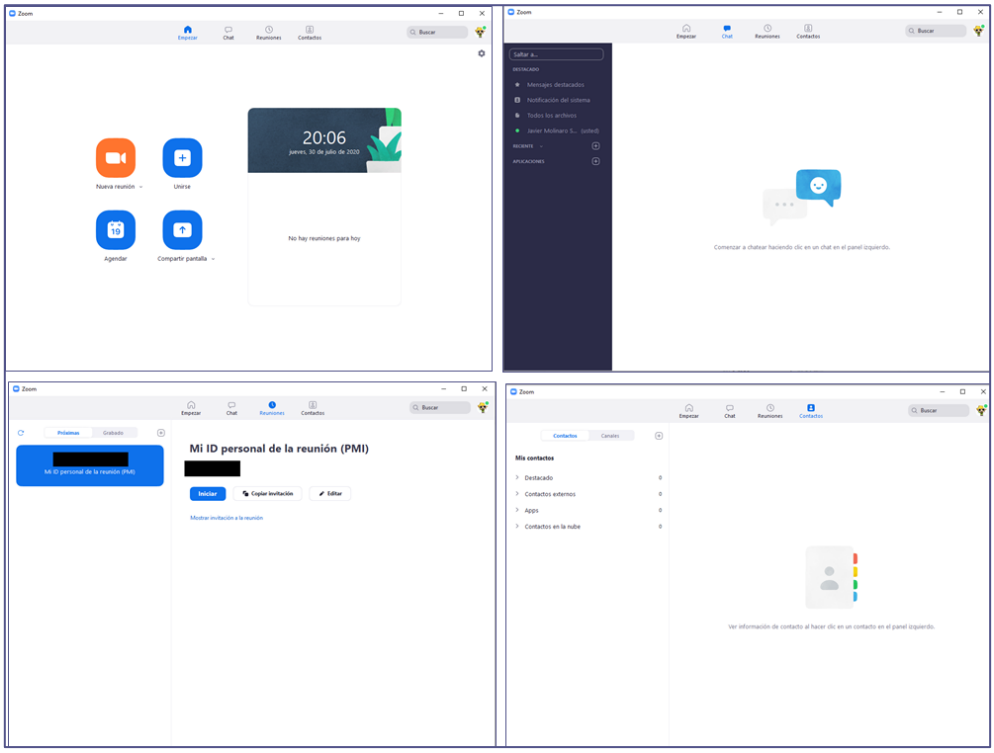

Ilustración 20. Ventanas de Zoom

Ventana superior izquierda: pestaña "Empezar", ventana superior derecha: pestaña "Chat", ventana inferior izquierda: pestaña "Reuniones", ventana inferior derecha: pestaña "Contactos".

Si iniciamos una videollamada nueva, se nos presenta una nueva ventana donde lo que predomina es la imagen que captura la webcam y, en la esquina inferior izquierda de esta imagen, el nombre del participante.

A lo largo de la parte inferior de esta ventana se nos presenta una barra con diferentes botones:

- Botones de control del estado del micrófono y la cámara.
- *Seguridad*: donde podemos controlar lo que pueden y no pueden hacer los invitados de la videoconferencia.
- *Participantes*: donde tenemos una lista con los participantes de la videoconferencia.
- *Chat*: el cual activa una ventana de chat para enviar texto.
- *Compartir pantalla:* para enseñar nuestra pantalla al resto de participantes.
- *Grabar*: el cual permite grabar la conversación.
- *Reacciones*: para compartir emojis.
- *Finalizar*: para terminar la videoconferencia.

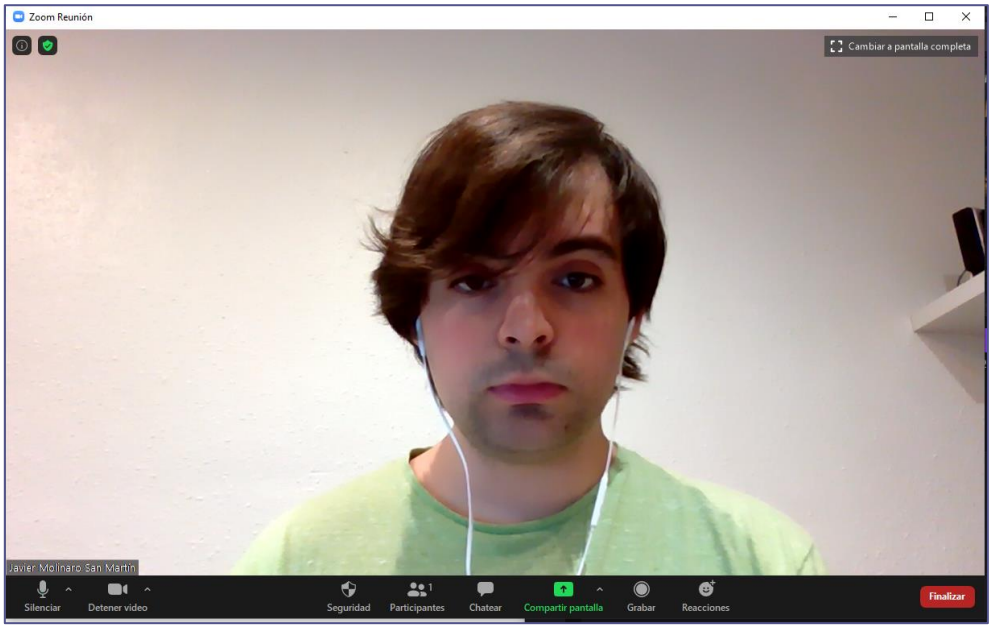

Ilustración 21. Ventana de videoconferencia en curso

6.3.2 Skipe

La siguiente aplicación que analizaremos será *Skype*, al ser tanto esta aplicación y *Microsoft Teams* propiedad de la misma compañía comparten muchas similitudes, por tanto, optaremos por analizar solo una de las dos.

Al igual que ocurre con *Zoom* lo primero que vemos es una ventana donde ingresar nuestros datos de inicio de sesión, esta cuenta puede ser con los datos de un correo de *Microsoft (Outlook, Hotmail, etc.)*, una cuenta de *Skype* o nuestro número de teléfono.

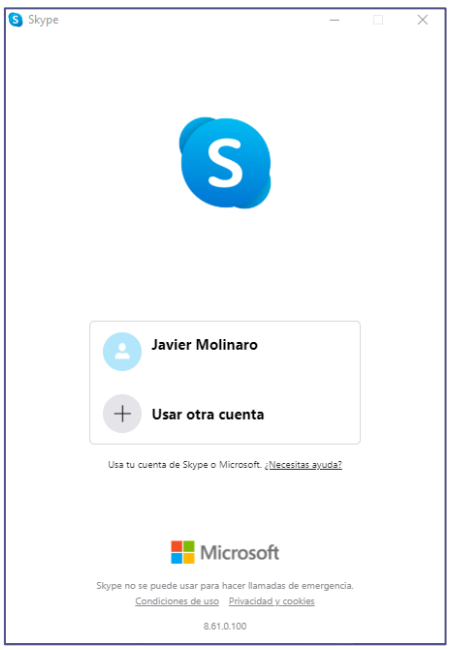

Ilustración 22. Ventana de inicio de sesión de Skype

Una vez iniciada la sesión, se nos presenta una nueva ventana dividida en dos partes principales. La primera de estas partes es una columna vertical, ubicada del lado derecho, que nos presenta

las opciones para interactuar con el resto de nuestros contactos. Estas opciones se encuentran divididas en diferentes apartados, accesibles desde unos botones:

- El primero de estos apartados es *Chats*, al tocar aquí se actualiza la información de la columna enseñándonos los contactos con los que recientemente hemos mantenido una conversación escrita y la opción de iniciar una nueva conversación con otros usuarios.
- El segundo apartado es *Llamadas,* al presionar en esta opción el aspecto de la ventana no se modifica significativamente, el único cambio destacable es que ahora los contactos que se nos muestran son aquellos con los que hemos mantenido una llamada recientemente, y se nos habilita la opción de iniciar una llamada nueva con el resto de los contactos.
- El tercer apartado es *Contactos,* como su nombre indica aquí se nos mostrará la lista de todos nuestros contactos.
- Por último, se nos presenta la opción *Notificaciones,* aquí se nos mostrará todas las notificaciones relacionadas con nuestro usuario, como pueden ser menciones, reacciones, mensajes directos, etc.

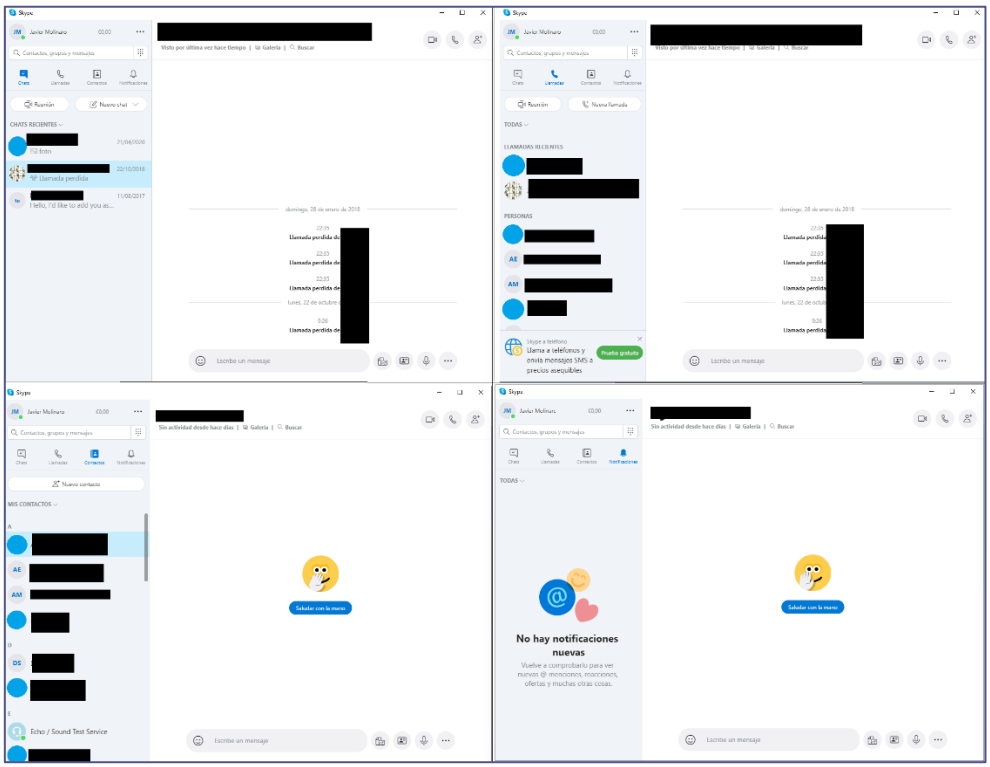

Ilustración 23. Ventanas de Skype

Ventana superior izquierda: pestaña "Chats", ventana superior derecha: pestaña "Llamadas", ventana inferior izquierda: pestaña "Contactos", ventana inferior derecha: pestaña "Notificaciones".

Al iniciar una videollamada se nos presenta una nueva ventana, donde lo que predomina es la imagen capturada por la cámara de la persona con la que estamos hablando, mientras que la imagen de nuestra cámara se presenta como una ventana flotante, más pequeña.

En la parte superior izquierda de esta ventana aparece el tiempo que ha transcurrido de la videollamada y el nombre del contacto con el que estamos manteniendo la conversación.

Observando la parte superior derecha de la ventana vemos dos botones, el primero nos permite integrar la ventana flotante de nuestra cámara a la ventana principal, junto a este botón existe otro que nos permite agregar nuevos participantes a la misma llamada.

En la parte inferior central se nos presentan tres botones principales, con los que podemos bloquear el envió de audio desde micrófono, apagar la cámara o terminar la llamada.

Por último, en la parte inferior derecha de esta ventana tenemos otros tres botones, con ellos podemos comunicarnos con un chat escrito, compartir nuestro escritorio con el resto de los participantes, y enviar reacciones (emoticonos de corazones, aplausos, etc.).

Ilustración 24. Videollamada en curso en la aplicación Skype

6.3.3 Google Meet

Por último, analizaremos la aplicación web *Google Meet*. Para utilizar esta aplicación online necesitamos tener una cuenta de Google iniciada en nuestro navegador web, lo primero que vemos al ingresar es una ventana muy simple con únicamente dos botones, *Nueva reunión* e *Introducir un código o enlace,* las dos opciones tienen nombres autoexplicativos, el primero es para crear una nueva sala para reuniones y la segunda para unirse a una sala ya creada por otro usuario.

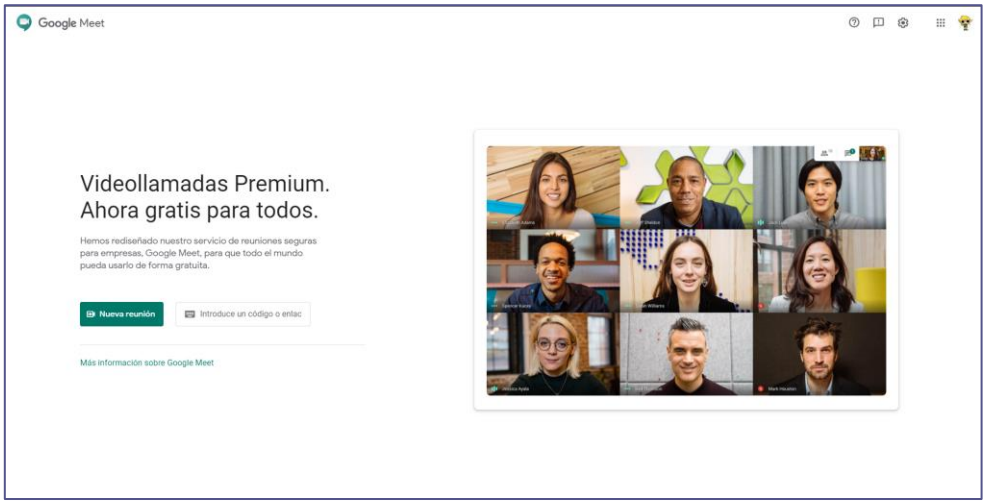

Ilustración 25. Ventana inicial de Google Meet

Al iniciar una reunión nueva se nos proporcionará un enlace que tendremos que compartir al resto de personas para que puedan unirse a esta llamada.

Al iniciar la videollamada vemos que lo que predomina es la imagen captada por la webcam de la persona con la que estamos comunicándonos.

En la parte superior derecha de la ventana tenemos tres botones, con el primero de ellos desplegamos una barra vertical conde se nos muestran todos los participantes de la videollamada con sus nombres y una miniatura con la imagen que capta sus respectivas webcams. El segundo botón despliega otra barra vertical donde podemos leer y participar en un chat escrito. El tercer botón no es interactivo, solamente nos indica la hora. El ultimo botón nos enseña una miniatura con la imagen de nuestra propia webcam, si lo tocamos podemos fijar la imagen.

Centrando nuestra atención en la parte inferior de la ventana veremos una barra horizontal con diferentes opciones, la primera de ellas es *Detalles de la reunión,* si presionamos aquí se nos mostrará el enlace que tenemos que compartir para agregar más gente a la llamada, y los archivos adjuntos que se nos han compartido. A continuación, tenemos en el centro de esta barra tres botones con funciones para controlar el estado del micrófono, la cámara y terminar la llamada.

En la parte derecha se nos muestra dos botones más, el primero de ellos nos permite compartir nuestro escritorio, o una ventana concreta, con el resto de los participantes. El segundo botón es para controlar opciones más específicas de la conversación (activar subtítulos, pantalla completa, cambiar diseño, etc.).

Ilustración 26. Videollamada en curso en la aplicación web Google Meet

6.4 Prototipado de interfaz gráfica de la aplicación

Procederemos ahora a crear el prototipo considerando los datos de las aplicaciones descritos hasta ahora y los requisitos planteados en el apartado *5. Definición de requisitos*.

Nuestra prioridad actual es plantear un diseño simple, minimalista y con accesos rápido a las diferentes opciones. Necesitamos que el uso de la aplicación sea sencillo ya estamos intentando que nuestro usuario ahorre tiempo para poder aprovechar sus horas de trabajo al máximo.

Cada una de las ventanas que compongan al programa tendrán un diseño claramente diferenciado, aunque manteniendo la coherencia, considerando todas las ventanas como parte de la aplicación en conjunto.

Recordemos que estamos planteando un prototipo, por tanto, no debe considerarse el diseño descrito como algo inamovible, es normal que durante la implementación de la aplicación final puedan surgir cambio o mejoras.

6.4.1 Ventana de inicio

Lo primero que se verá al iniciar la aplicación será una ventana sencilla con la menor cantidad de botones posibles: uno para iniciar la videollamada y otro para cambiar la foto del usuario.

En esta misma ventana tendremos dos cuadros de dialogo obligatorios que rellenar: uno correspondiente al nombre que mostrará nuestro usuario en el apartado de chat una vez se haya iniciado la videollamada, y el otro al valor que tendremos que ingresar para conectarnos con el otro usuario (en principio la IP de este).

Si el usuario se olvida de ingresar alguno de los datos obligatorios para establecer la conexión, se le hará saber por medio de un mensaje en rojo.

Ilustración 27. Ventana inicial de la aplicación prototipo

Al completar los datos y presionar el botón *Iniciar Videollamada* se nos redirigirá a una nueva ventana donde tendrá lugar la conversación.

6.4.2 Ventana de videollamada en curso

Cuando la videollamada esté en curso se indicar con un mensaje ubicado en el ángulo superior izquierdo. En esta ventana se le dará prioridad a la imagen procedente de la webcam del otro participante, nuestra imagen estará también siempre a la vista, pero con un tamaño menor, ubicándola en el ángulo superior derecho.

El nombre de los participantes de la videollamada estará visible en el ángulo inferior izquierdo de la ventana donde se muestre las correspondientes imágenes de la webcam.

Por otro lado, los diferentes parámetros que controlar en la conversación, así como la opción de finalizar la videollamada, se encontrará ubicada en una barra horizontal, ubicada a lo largo de la parte inferior de la ventana de la aplicación.

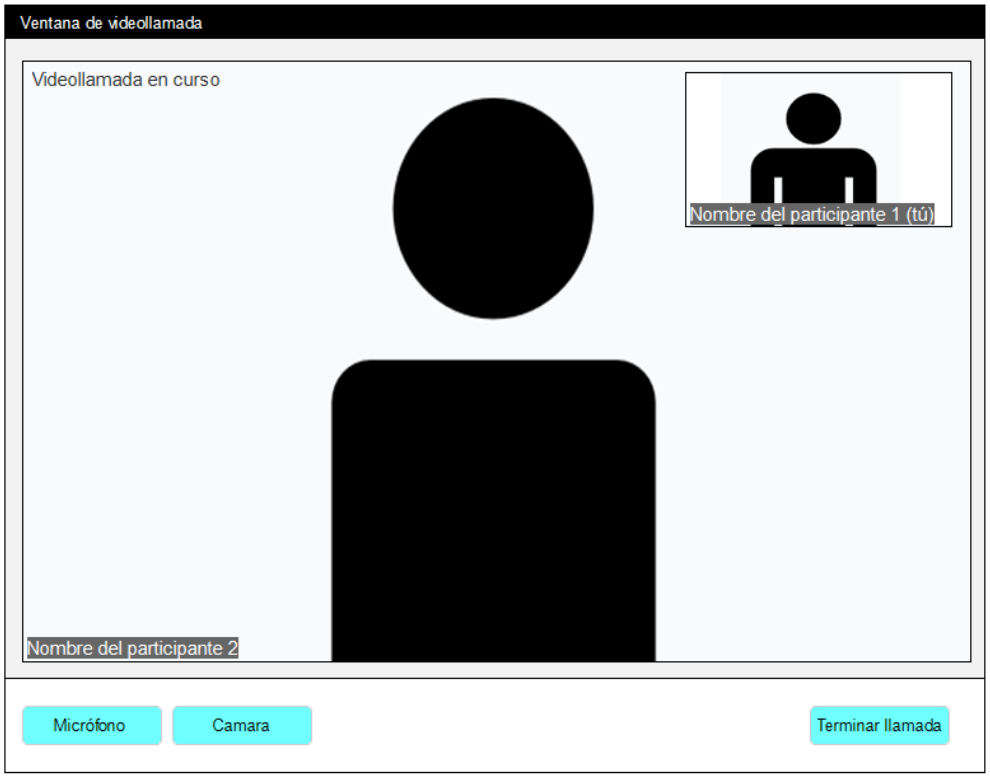

Ilustración 28. Ventana de videollamada de la aplicación prototipo

6.4.3 Ventana confirmación finalizar llamada

Al intentar terminar la videollamada, un mensaje de confirmación nos preguntará si estamos seguros de querer terminarla, en caso afirmativo se finalizará la videollamada, en caso negativo se cerrará solo este nuevo cuadro de dialogo y se continuará con la comunicación como estábamos haciendo hasta ahora.

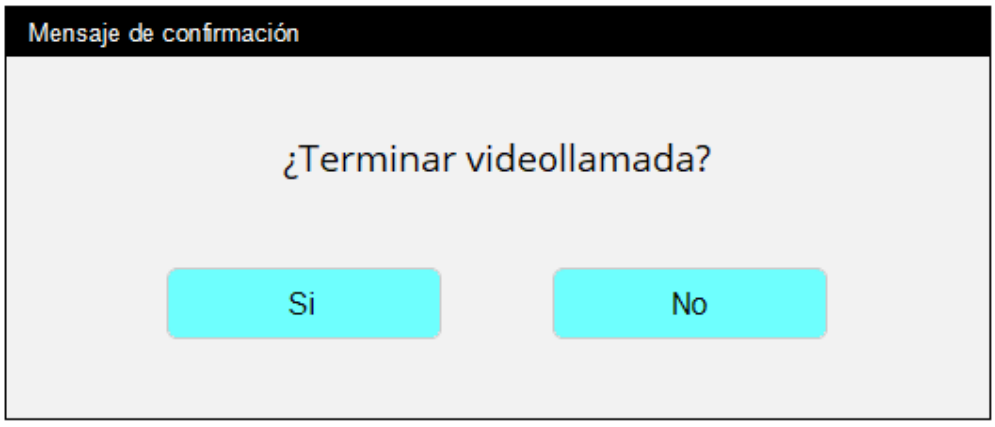

Ilustración 29. Cuadro de dialogo, confirmación de continuar o terminar con la videollamada

6.5 Diagrama de flujo

En este apartado se expone el diagrama de flujo del prototipo propuesto, donde describimos de manera gráfica la evolución de los diferentes pasos que realiza nuestro sistema a lo largo de su ejecución.

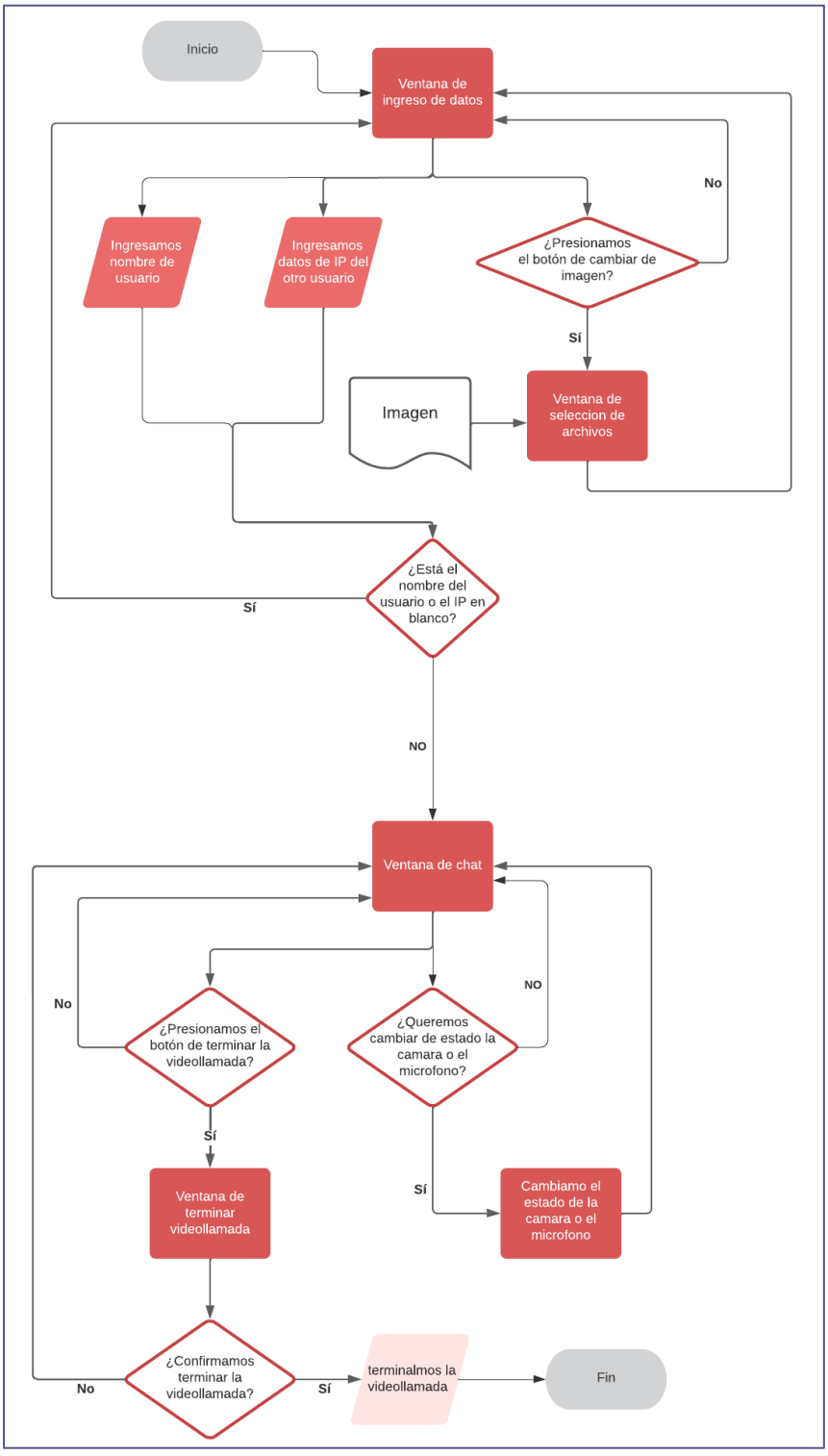

Ilustración 30. Diagrama de flujo del prototipo propuesto

6.6 Evaluación del diseño

Para la evaluación del diseño propuesto, nos basaremos en los 10 principios de usabilidad de Nielsen, descritos ya en el apartado *4.4 Principios de Jakob Nielsen*:

Visibilidad del sistema

Para garantizar el primer principio, donde se deberá mantener informado al usuario sobre el estado del sistema, hemos optado por incluir en cada ventana los elementos mínimos necesarios.

Además de esto se han incluido mensajes que indican al usuario lo que está ocurriendo en cada momento, como pueden ser el mensaje que indica si ha ocurrido algún error a la hora de intentar iniciar sesión, o el mensaje de videollamada en curso.

Relación entre el sistema y el mundo real

Este principio ha sido fácil de garantizar, ya que el sistema de comunicación de la aplicación tiene el mismo orden que una llamada telefónica:

Primero se debe ingresar el numero con el que queremos comunicarnos, luego de que se establezca la conexión se procede a realizar la comunicación, y una vez terminada esta se cuelga la llamada.

Además de esto, cuando la llamada está en curso, hemos planteado que se vea la cámara y se escuche el micrófono entre los participantes, es como una conversación convencional entre personas, por tanto, la relación con el mundo real está garantizada.

Control y libertad del usuario

El tercer principio de usabilidad de Nielsen es el control y libertad del usuario. Para abordarlo, hemos incluido en cada ventana una serie de opciones que permite a quien este usando el programa controlar los parámetros que desee durante la videollamada, de manera que sea el usuario quien decida que opciones usar y cuando. Además, cada pantalla tendrá las opciones correspondientes para poder deshacer los pasos que haya realizado hasta el momento.

Consistencia y estándares

Para garantizar el cuarto principio de usabilidad, se ha estudiado el aspecto y organización que presentan diferentes aplicaciones de videoconferencia, de esta manera pretendemos conseguir que la aplicación resultante refleje familiar a la vista de nuevos usuarios.

La consistencia se ve reflejada en el aspecto de las ventanas, todas comparten entre si un diseño similar, tanto en color como en estructura, así mismo, se ha puesto empeño en emplear un lenguaje claro y conciso en los diferentes menús, ventanas y botones, para conseguir que el usuario sepa en todo momento exactamente qué es lo que está viendo.

Prevención de errores

Respecto al quinto principio, se han incluido en los espacios rellenables de la aplicación ejemplos de texto para evitar que el usuario ingrese datos incorrectos. En el caso de ingresar datos incorrectos se indicará en con texto rojo donde está el error y como corregirlos.

Reconocimiento antes que recuerdo

El sexto principio de Nielsen se ve cubierto con la cabecera de las ventanas, donde se indica el nombre de la ventana donde nos encontramos en cada momento. En el caso de que esto no resultase suficiente se ha diseñado cada ventana con un aspecto claramente diferenciado de la anterior.

Flexibilidad y eficiencia de uso

El séptimo principio de usabilidad se ve reflejado en la cantidad de opciones que exponemos al usuario en cada ventana. Planteamos un número limitado de botones para no hacer la navegación complicada y no saturar con información a los usuarios. De esta manera podemos conseguir que, sin importar la experiencia de quien este usando la aplicación, todos puedan entenderla y aprovechar todas sus opciones.

Estética y diseño minimalista

Para cumplir el octavo principio se ha llevado a cabo un análisis de diferentes aplicaciones como material de referencia, y se ha desarrollado un prototipo con un diseño que refleje un aspecto limpio, geométrico y organizado.

Ayudar a los usuarios a reconocer, diagnosticar y recuperarse de errores

Para cubrir el noveno principio de usabilidad hemos incluido, como se ha mencionado anteriormente, mensajes con un color llamativo, en este caso rojos, que indica donde está el error y como solucionarlo.

Ayuda y documentación

El décimo principio de usabilidad de Nielsen, ayuda y documentación, lo estamos abordando en este momento. Se trata de un documento detallado donde se incluye cómo funciona, y como usar correctamente la aplicación, sin embargo, es lógico pensar que un usuario corriente no se detenga a leer dicho documento en su totalidad, por tanto, se considerará incluir algún botón de ayuda en la versión final de la aplicación que indique las acciones básicas para llevar a cabo un correcto uso de la aplicación. Aun así, confiamos en que el diseño minimalista de la aplicación ayude a evitar la consulta continua de dicho manual de uso.

7. Desarrollo de la aplicación

En este apartado describiremos el proceso de desarrollo de la aplicación final, basándonos en la información y el prototipo aportados en anteriores capítulos.

Comenzaremos describiendo las bases, primero construiremos un “esqueleto” en el que apoyar la parte funcional de la aplicación, una aproximación a las ventanas finales que sostendrán y mostrarán la información compartida entre dispositivos. Luego de esto nos preocuparemos de crear la parte funcional, como es la interpretación de las ondas sonido captadas por el micrófono, el procesamiento de imagen de la webcam y el envió de esta información a través de una conexión establecida entre los distintos dispositivos.

7.1 Construcción inicial de la interfaz de la aplicación

En este apartado mostraremos el proceso llevado a cabo para construir la parte visual de la aplicación, basándonos en lo planteado en el apartado *6.3 Prototipado de la interfaz gráfica de la aplicación,* utilizando *JavaFX* y *SceneBuilder*.

7.1.1 Ventana de inicio

Siguiendo como guía el prototipo planteado, se proceda a diseñar la ventana de inicio de la siguiente manera:

Ilustración 31. Ventana de inicio

Como puede apreciarse se han realizado ligeras variaciones con respecto al prototipo planteado inicialmente, dentro de estos cambios destacar la inclusión de un texto que se actualizará indicando nuestra IP, un desplegable que describe nuestro rol y un botón de configuración.

Cada componente de esta ventana, y de las siguientes que mencionemos, debe ser accesible desde el código de su controlador correspondiente (los cuales serán descritos más adelante), por tanto, es necesario que nombremos cada uno de estos componentes y definamos la forma que tendremos para interactuar con él.

A continuación, procederemos a enumerar los diferentes componentes detallando sus características:

- Imagen principal: este componente está formado por un *ImageView* y será reconocido por el código con el nombre de *myPhoto.*
  Esta imagen tendrá una relación de aspecto 16:9, para estar en concordancia con la relación de aspecto de las webcams actuales, además, se ha implementado una opción para poder interactuar con ella haciéndole clic, esto ejecutará la función *changePhoto*, la cual describiremos más adelante.
- Mi IP: este componente es un elemento *Text* y será reconocido en código con el nombre de *myIP*.
- Mi nombre: se trata de un *TextField* editable y será reconocido en el código con el nombre *myName*.
- Desplegable: este desplegable está formado por un componente *ComboBox* y será accesibles con el nombre *whoIAm*.
- A quien me conecto: este es un *TextField* editable donde introduciremos la IP del usuario con el que queremos conectarnos, su nombre es *ipPartner*.
- Mensaje de error: es un componente *Text* con tipografía roja*,* será reconocido en el código con el nombre *caution.*
- Botón Configuración: se trata de un componente *Button,* definido con el nombre *configurationButton*, al presionarlo se llamará a la función *openConfigurationButton*.
- Botón Conectar: se trata de un componente *Button* con nombre *connectionButton,* al presionarlo se llamará a la función *openConfigurationButton*.

7.1.2 Ventana de configuración

La siguiente ventana que describiremos es la de configuración. Esta ventana fue planteada posteriormente a la generación del diseño del prototipo, ya que se planteó la posibilidad que el usuario tuviera más de una cámara conectada a un mismo equipo o tuviera su puerto de comunicación por defecto ocupado, además de esto se añadió un desplegable de cara a una posible modificación futura en cuanto al protocolo de comunicación empleado.

Ilustración 32. Ventana de configuración

Las diferentes partes que componen esta ventana son las siguientes:

- Protocolo: se trata de un *ComboBox* el cual llamaremos *protocol,* este contendrá una lista con los diferentes protocolos con los que se podría establecer la comunicación.
- Puerto: se trata de un *TextField* modificable, por defecto contendrá el valor 5000 y será reconocido en el código con el nombre *port.*
- Cámara: es un *ComboBox* el cual contendrá una lista de las webcams disponibles en el ordenador, será reconocido con el nombre de *camera*.
- Botones Cancelar y Aceptar: son elementos *Button* que, al presionarlos, llaman a las funciones c*ancelChanges* y *applyChanges* respectivamente. Sus nombres son *cancelButton* y *acceptButton*.

7.1.3 Ventana principal de chat

Esta es la ventana principal de la aplicación, en esta se desarrollará todo aquello relacionado con la comunicación.

Esta ventana presenta dos cuadros de imágenes distintos, en el más pequeño se mostrará la imagen de nuestra cámara y en el más grande la imagen que nos envíe el otro usuario. En la parte inferior dispondremos de tres botones, uno que controla el estado del micrófono, otro que controla el estado de la cámara y el ultimo para finalizar la llamada.

Se ha realizado un cambio en la implementación final del diseño con respecto a lo visto en el prototipo inicial: para intentar solventar los posibles errores que pudiesen surgir a la hora de entablar la comunicación, hemos incluido, por seguridad, un medio más de comunicación, un chat escrito en la parte derecha de la ventana.

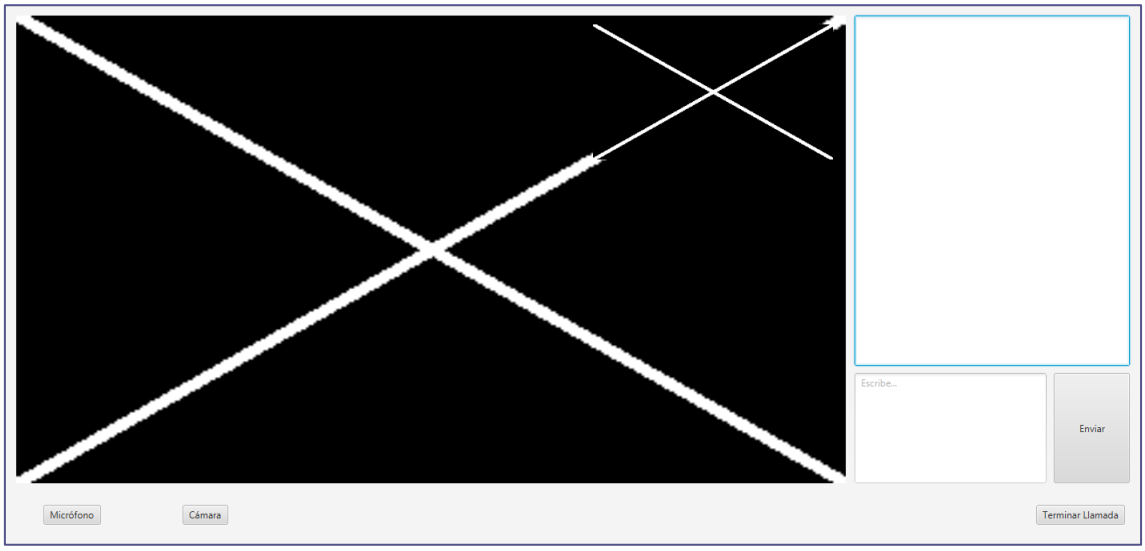

Ilustración 33. Ventana principal de la aplicación

Pasemos ahora a describir las partes que componen esta ventana:

- Imagen de la cámara: tanto nuestra cámara como la del otro participante serán de tipo *ImageView,* para que no sufran alteraciones en sus dimensiones se ha bloqueado sus tamaños respetando el formato 16:9, los nombres de cada uno serán *cameraImagePartner* para la imagen de la cámara del otro participante y *cameraImage* para la nuestra.
- Área de chat: el área de chat es un área creada con un *TextArea* no editable, lo hacemos no editable para que el usuario no pueda escribir directamente en él. El nombre que usaremos para identificarlo será *chatArea*.
- Área de texto: el área de texto está definido con un *TextArea* editable, este será el espacio asignado donde escribiremos los mensajes de texto que queramos compartir, este elemento será designado con el nombre de *textAreaUser*.
- Botón enviar: se trata de un *Button* definido con el nombre *sendButton,* al presionarlo se llama al método *sendText* que se encargará de enviar el contenido de *textAreaUser.*
- Botón micrófono: se trata de un *Button* con nombre *microphoneButton,* al presionarlo se llamará a la función *microphoneChangeStat.*
- Botón Cámara: se trata de un *Button* con nombre *cameraButton,* al presionarlo se llama a la función *cameraChangeStat*.
- Botón terminar llamada: se trata de un componente *Button* al que llamaremos *endCallButton,* al presionarlo se llamará a la función *endCall.*

7.1.4 Ventana finalizar llamada

Como su nombre indica, esta es la ventana de confirmación que se mostrará si el usuario decide finalizar la llamada.

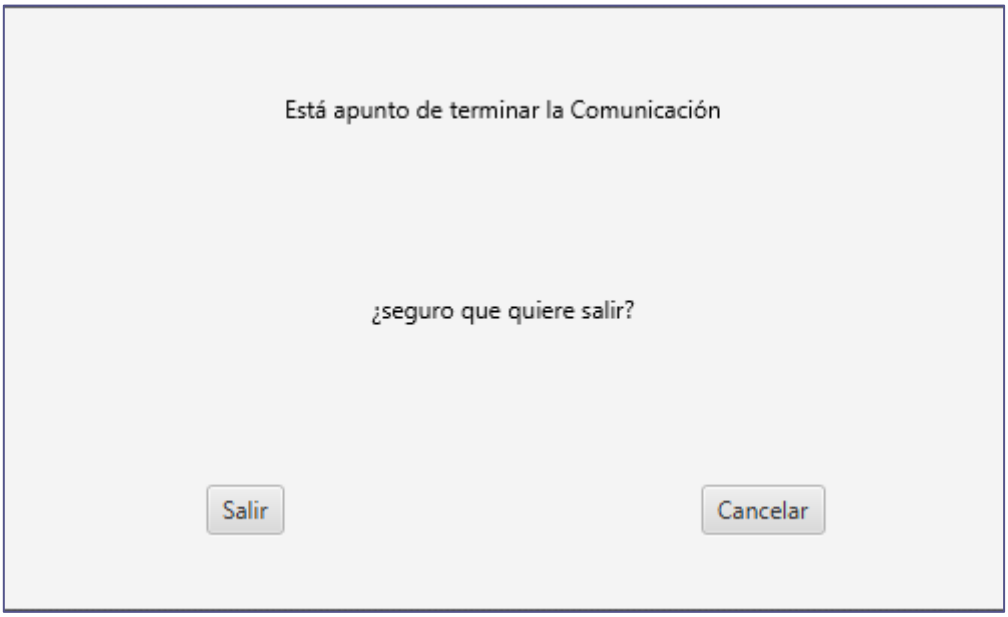

Ilustración 34. Ventana finalizar llamada

Esta ventana se compone de tres elementos:

- Un texto con un mensaje estático: al no variar este ni tener que interactuar con él, no hace falta que definamos su referenciación en el código.
- Un botón Salir: el cual será referenciado como *buttonExit* y estará vinculado a la función *endComunication*.
- Un botón Cancelar: el cual será referenciado con el nombre de *buttonCancel* y estará vinculado a la función *cancel*.

7.2. Funcionamiento de la aplicación

Antes de continuar definiremos los pasos que se tendrán que realizar para iniciar una comunicación, de esta manera tendremos una base más clara a la hora conectar el controlador de la ventana con su interfaz.

Primero, los usuarios intercambiarán entre ellos las direcciones IP de cada uno, ya que los mensajes se enviarán directamente de un dispositivo al otro. Esto es así porque se ha planteado un sistema de comunicación de extremo a extremo, sin servidores intermedios.

Lo siguiente que se hará será iniciar la comunicación entre los intervinientes a través del protocolo TCP en el puerto 5000, con esto hecho se procederá a generar mensajes que enviarán la información de un usuario a otro.

El tipo de mensaje enviado será interpretado de manera diferente dependiendo de su clase, pudiendo ser estos mensajes de texto, imágenes o audio.

El envío de estos tipos diferentes de datos se realizará en hilos de ejecución distintos, de esta manera, el fallo de uno de los canales de comunicación no perjudicará el funcionamiento de los otros.

Tanto el anfitrión como el invitado realizarán las funciones de servidor y cliente a la vez. La parte de cliente será el encargado de el envió de datos utilizando el protocolo TCP, y la parte de servidor se encargará de recoger la información.

Por tanto, y a modo de resumen, los pasos a seguir para el envió de información entre usuarios son los que siguen:

1. Se rellenará el apartado *Nombre*, se intercambiarán las direcciones IP de cada cual y se presionara el botón *Iniciar Videollamada*.
2. Una vez dentro de la ventana principal tendremos iniciado, en un hilo de ejecución independiente, un servidor TCP el cual será el encargado de recibir los paquetes enviados por nuestro interlocutor.

3. Por otro lado, dependiendo que tipo de comunicación multimedia que queramos establecer, se nos presentarán diferentes opciones:
    a. Si queremos intercambiar imágenes de audio presionamos en el botón *Cámara,* esto iniciará un proceso, en un hilo diferente de ejecución, donde se obtendrán imágenes de nuestra cámara y se enviarán por medio del protocolo TCP como cliente al otro usuario en la IP y puerto predeterminado.
    b. Si queremos intercambiar audio presionaremos en botón *Micrófono*, este iniciará un hilo de ejecución independiente que obtendrá muestreos de audio de nuestro micrófono y los enviará encapsulados en paquetes, utilizando el protocolo TCP como cliente, al otro usuario en la IP y puerto predeterminado.
    c. Si por el contrario queremos enviar un mensaje escrito, y por tanto prescindir o complementar a la información multimedia, se escribirá el texto en el cuadro correspondiente y se enviará usando el protocolo TCP como cliente al otro usuario.

7.3 Definición de los controladores

En el presente apartado definiremos los diferentes controladores que componen la aplicación. El código contenido en estos controladores será el encargado de definir la lógica de la aplicación.

7.3.1 Controlador de la ventana de inicio

El controlador de esta ventana será *StartWindowsController.* Lo primero que vamos a definir es el contenido de la función *initialize*, esta función se llama automáticamente si la ventana cargo correctamente.

Dentro de *initialize* realizaremos las siguientes funciones:

1 Colocar el mensaje de error con opacidad cero.
2 Rellenaremos *whoIAm* con dos opciones diferentes *Anfitrión* e *Invitado,* cuando el usuario cambie el contenido seleccionado de este desplegable, esta variación será recogida por un oyente que llamará a la función se*IsServer*, la cual modificara una variable global booleana que indica si somos el anfitrión o no.
3 Guardar en variables globales nuestra IP y foto seleccionada.

Lo último que tenemos que mencionar de esta función es que crea un objeto *ConfigurationWindowsController*, el cual explicaremos más adelante, pero básicamente es un objeto que contiene la configuración por defecto del protocolo TCP, y la webcam que tengamos en nuestro ordenador.

```
public void initialize(URL url, ResourceBundle rb) {
       //colocamos el mensaje de error con opacidad cero
       caution.setOpacity(0);

       //Rellenamos el combobox con las opciones de anfitrion o invitado
       //seleccionamos por defecto la opcion de anfitrión
       //colocamos el isServer a true
       whoIAm.getItems().addAll("Anfitrión", "Invitado");
       whoIAm.getSelectionModel().select(0);
       isServer = true;

       //creamos un ConfigurationWindowsController para cargar la configuración por defecto
       cwc = new ConfigurationWindowsController();

       //agregamos un oyente que detecte el cambio de anfitrión o invitado
       whoIAm.valueProperty().addListener(c -> setIsServer(whoIAm.getValue()));


       //al iniciar el programa obtenemos nuestra ip y la guardamos en una variable estática
       ip = getMyIP();
       myIP.setText(getMyIP());

       //guardamos nuestra imagen el una variable estática
       photo = myPhoto.getImage();

}



private void setIsServer(String i){
       if(i.equals("Anfitrión")){
           isServer = true;
       }
       else {
           isServer = false;
       }
       System.out.println("valor isServer: " + isServer);
}

public static String getMyIP() {
       if (ip == null) {
           String res = "";
           try {
               res = InetAddress.getLocalHost().getHostAddress();
           } catch (Exception e) {
               System.err.println("e");
           }
           return res;
       } else {
           return ip;
       }
}
```

Ilustración 35. Código de initialize, setIsServer y getMyIP

El siguiente método que definiremos del controlador es *changePhoto.* Como su nombre indica, sirve para cambiar la imagen que mostraremos al otro usuario. Para conseguir esto abre una ventana de navegación de archivos donde filtramos solo los archivos de imagen (.jpg, .jpeg, .png, .gif), utilizando el método *getExtensionFilters* de *FileChoose.*

```
@FXML
private void changePhoto(MouseEvent event) throws IOException {

    //creamos una ventana de selección para cambiar la imagen de usuario
    FileChooser fileChooser = new FileChooser();

    //filtramos los tipos de archivo para que solo aparescan imágenes
    List<String> fileExtensions = new ArrayList<>();
    fileExtensions.addAll(Arrays.asList("*.jpg", "*.jpeg", "*.png", "*.gif"));
    fileChooser.getExtensionFilters().addAll(new FileChooser.ExtensionFilter("Imagenes", fileExtensions));

    //abrimos la ventana de seleccion de archivos en la misma escena donde tenemos myPhoto
    //el archivo que seleccionemos se guarda en la variable newImage
    File newImage = fileChooser.showOpenDialog(myPhoto.getScene().getWindow());

    //comprobamos que la nueva imagen no sea nula
    if (newImage != null) {
        //asigamos la newImage a nuestra imagen Actual
        //aprovechamos la clase SwingFXUtils para convertir el tipo de la imagen de BufferedImage a Image
        myPhoto.setImage(SwingFXUtils.toFXImage(ImageIO.read(newImage), null));
        photo = myPhoto.getImage();
        photoChange = true;
    }

}
```

Ilustración 36. Código de changePhoto

Los últimos métodos a destacar son: *initConection* y *openConfigurationButton.* Como sus nombres indican, uno se encargará de redirigirnos a la ventana de configuración y el otro a la ventana principal de comunicación.

Para comprobar que los datos introducidos son correctos se ha creado una función *checkData* la cual comprueba que no existan campos obligatorios en blanco. En caso de existir un error, este se indicará en un texto en rojo en la parte inferior de la ventana.

```
private boolean checkData() {
    //comprobamos que nuestro nombre y la ip de nuestro receptor no esten en blanco
    if (myName.getText().equals("")) {
        caution.setOpacity(1.0);
        caution.setText("El campo Mi nombre no puede estar vacío");
        return false;
    }
    if (ipPartner.getText().equals("")) {
        caution.setOpacity(1.0);
        caution.setText("Debes indicar con quien vas a establecer la conexión");
        return false;
    }

    //si todo es correcto guardamos estos datos en varaibles estaticas y retornamos true
    name = myName.getText();
    ip_two = ipPartner.getText();
    //colocamos el mensaje de precaución con opacidad cero en caso de que tenga algo escrito por un error anterior
    caution.setOpacity(0);

    return true;
}
```

Ilustración 37. Código de checkData

7.3.2 Controlador de la ventana de configuración

El controlador de esta ventana es *ConfigurationWindowsController.* En este caso, al ejecutarse el método *initialize* se rellenarán los distintos *ComboBox:*

- En el caso de *protocol* con las opciones TCP y UDP, siendo la primera la opción seleccionada por defecto. Esta aplicación se comunicará utilizando TCP, incluimos UDP en la lista para posibles futuras ampliaciones de la aplicación.
- En el caso de *camera* con una lista de las webcams disponibles en el ordenador, seleccionado por defecto la primera de la lista.

Para conseguir esta lista de cámaras nos serviremos de la función *getWebcams* de la API *Webcam Capture v0.3.12*, dicha función devuelve un objeto *List* con cada una de las webcams disponibles en el ordenador.

El motivo por el que usar esta librería es simple, la construcción de una nueva librería por nuestra parte acarrearía un trabajo que iría más allá de lo que queremos abarcar en el proyecto, recordemos que el objetivo final es la construcción de una aplicación que transmita audio y video a través de una red, por tanto, nos serviremos de dicha librería y sus métodos para todo aquello que tenga relación con el manejo de la webcam.

```
@Override
public void initialize(URL url, ResourceBundle rb) {
    //añadimos a los desplegables los protocolos y camaras de nuestro equipo
    protocol.getItems().addAll("TCP", "UDP");
    camera.getItems().addAll(Webcam.getWebcams());

    //seleccionamos por defecto el protocolo TCP, el puerto 5000 y la camara que este primero en la lista en caso de que no hayamos seleccionado nada aún
    //en caso contrario mostramos el protocolo y la camara seleccionados
    if (selectedCamera == null && selectedProtocol == null && selectedPort == 0) {
        protocol.getSelectionModel().select(0);
        port.setText(4000+"");
        camera.getSelectionModel().select(0);
    } else {
        protocol.getSelectionModel().select(selectedProtocol);
        port.setText(selectedPort + "");
        camera.getSelectionModel().select(selectedCamera);
    }

    //los guardamos en variables estaticas
    selectedProtocol = protocol.getValue();
    selectedPort = Integer.parseInt(port.getText());
    selectedCamera = camera.getValue();


}
```

Ilustración 38. Código initialize de la ventana de configuración

Esta clase dispondrá de un constructor *ConfigurationWindowsController* en el que se establecerá por defecto el protocolo TCP, la primera webcam de la lista y el puerto 5000. En el caso de realizar cambios en las opciones de la ventana estos serán recogidos y modificados al lanzar el método *applyChanges* con el botón de *Aceptar*.

```
public ConfigurationWindowsController(){
    selectedProtocol = "TCP";
    selectedPort = 4000;
    selectedCamera =  Webcam.getWebcams().get(0);
}

@FXML
private void applyChanges(ActionEvent event) {
    setProtocol(protocol.getValue());
    setPort(Integer.parseInt(port.getText()));
    setCamera(camera.getValue());

    //imprimimos la configuración actual
    System.out.println("Configuracion actual:\n Protocolo: " + selectedProtocol + "\n Puerto: " + selectedPort + "\n Camara: " + selectedCamera);

    //cerramos la ventana una vez aplicado los cambios
    Node source = (Node) event.getSource();
    Stage stage = (Stage) source.getScene().getWindow();
    stage.close();
}
```

Ilustración 39. Código del constructor y applyChanges

7.3.3 Controlador de la ventana principal de chat

El controlador de esta ventana es *ChatApplicationWindowsController*. En este caso, al ejecutarse el método *initialize* se realizarán las siguientes acciones principales*:*

1 Se creará un Servidor TCP con el puerto que hayamos definido en la ventana de configuración (el código del cliente y servidor serán explicados más adelantes en el presente documento)
2 Agregamos un observador al servidor creado con el método *addObserver*, por tanto, nuestro *ChatApplicationWindowsController* implementará la interfaz *Observer.*
3 Se colorará dicho servidor TCP en un hilo de ejecución independiente, de esta manera no bloquemos la ejecución del resto del código del controlador mientras el servidor espera y lee los mensajes que le llegan.
4 Se guardará una referencia de nuestra cámara seleccionada por defecto, que se encuentra, en un principio, desactivada.
5 Forzaremos la resolución máxima de la cámara a 640x360, esto lo hacemos para garantizar que las imágenes no sean muy pesadas y por tanto ralenticen el flujo de datos de manera innecesaria.

```
public void initialize(URL url, ResourceBundle rb) {

     try {

         //si el protocolo es TCP...
         if (ConfigurationWindowsController.getProtocol().equals("TCP")) {
             //comprovamos si somos el servidor
             if (StartWindowsController.getIsServer()) {
                 stcp = new ServerTCP(ConfigurationWindowsController.getPort());

             } else {
                 stcp = new ServerTCP(ConfigurationWindowsController.getPort() + 1000);

             }
             //agregamos un observador al servidor
             stcp.addObserver(this);
             //colocamos el servidor guardado en un nuevo hilo de ejecución
             Thread ts = new Thread(stcp);
             //lo iniciamos
             ts.start();

         }

         //guardamos en default camera nuestra camara por defecto
         defaultCamera = ConfigurationWindowsController.getCamera();

         //definimos la resoluci´pn máxima de la cámara
         Dimension[] prefDimension ={ new Dimension(640,360)};
         defaultCamera.setCustomViewSizes(prefDimension);
         defaultCamera.setViewSize(prefDimension[0]);

         //indicamos que la camara esta apagada al iniciar
         isCameraOpen = false;


 }
```

Ilustración 40. Ilustración 38. Código initialize de la ventana de chat

En el caso de la comprobación de nuestro rol como anfitrión o invitado se ha realizado una modificación en el puerto, siendo este el valor introducido en la ventana configuración más mil, esto se ha hecho para evitar problemas de solapamiento de puertos.

Antes de continuar, realizaremos un pequeño paréntesis y expliquemos que es la interfaz *Observer.* La interfaz *Observer* nos es proporcionada por el propio Java y está relacionada con la clase *Observable.* La clase Observable producirá eventos cuando se detecte un cambio y lo comunicará con el método *notifyObservers* que recibe como argumento el *Object* del que queremos notificar el cambio. Por otra parte, es necesario que en la clase que queramos recibir

estas notificaciones implemente la interfaz *Observer,* a partir de aquí recibirá las notificaciones que envié la clase observada a través de la función *update*, esta función recibe dos argumentos: por un lado, un *Observable* que indica quien ha enviado la notificación, y por otro un *Object* con el objeto actualizado [Ora211] [Ora212].

Al presionar el botón *Micrófono* se llama al método *microphoneChangeStat,* este comprobará la variable global booleana *isAudioOpen*, si el resultado de comprobar el valor de esta variable es falso se cambiará a verdadero y llamará a la función *startMyMicrophone*, en caso contrario colocará la variable en falso.

La función *startMyMicrophone* será la encargada de obtener la información del audio captado por el micrófono, para ello lo primero que haremos será definir el formato en el que transmitir el audio, este formato estará definido por una clase *AudioFormat,* esta se encargará de estandarizar la manera en que los bits de sonido serán interpretados [Ora213], para construirlo se necesitan cinco argumentos diferentes:

- El primero de todos corresponde al *sampleRate:* se trata de un valor decimal que definirá el número de muestras por segundo
- El segundo argumento es *sampleSizeInBits:* es el tamaño en bits de cada muestra
- El tercer argumento es *chanels:* como su nombre indica definirá el número de canales de audio que serán tomados en cuenta
- El cuarto argumento corresponde a *signed:* un valor booleano que indica si los datos están firmados o no.
- El quinto argumento es *bigEndian:* este es un booleano que definirá el orden en el que se leerán los bits, si es *true* se interpretarán los bits del más significativo al menos significativo, en el caso de ser *false* se interpretarán del menos significativo al más significativo.

El formato de audio que definiremos constará de 8000 muestras por segundo, este valor no es casualidad, ya que es el valor mínimo recomendado para poder reproducir, de manera clara, la voz humana [Wik21]. Estas muestras tendrán un tamaño de 16 bits, cuanto más grandes sean las muestras, mayor calidad tendrán, pero eso también implicará que ocupen más, recordemos que nos interesa que el envío de información sea lo más fluido posible. Definiremos un único canal de audio, con información sin firmar, y en formato *big endian*.

Para enviar la información capturada por el micrófono iniciaremos un *TargetDataLine* con el formato definido anteriormente, este se encargará de iniciar la captura de sonido, con el método *start,* para posteriormente pasar a leer las secuencias de audio del micrófono y empaquetarlas en paquetes de 512 bytes (cuanto mayor sea el tamaño de los paquetes, mayor será la latencia).

Una vez tengamos estos fragmentos de audio empaquetados se enviarán a nuestro interlocutor, por medio de la creación de un cliente TCP en el puerto y dirección IP correspondiente, el cual lo recibirá en su método *update.*

```
private void startMyMicrophone() throws LineUnavailableException {

        byte m[] = new byte[512];

        AudioFormat format = new AudioFormat(8000.0f, 16, 1, true, false);
        DataLine.Info info = new DataLine.Info(TargetDataLine.class, format);
        audioSend = (TargetDataLine) AudioSystem.getLine(info);
        audioSend.open(format);
        audioSend.start();
        try {
            threadMicrophone = new Thread(() -> {
                while (isAudioOpen) {
                    try {
                        audioSend.read(m, 0, m.length);
                        if (StartWindowsController.getIsServer()) {

                            ctcp = new ClientTCP(StartWindowsController.getIPPartner(), ConfigurationWindowsController.getPort() + 1000, m);

                        } else {

                            ctcp = new ClientTCP(StartWindowsController.getIPPartner(), ConfigurationWindowsController.getPort(), m);

                        }
                    } catch (IOException ex) {
                        Logger.getLogger(ChatApplicationWindowsController.class.getName()).log(Level.SEVERE, null, ex);
                    }
                }
                audioSend.close();
                audioSend.drain();
            });
            threadMicrophone.start();
        } catch (Exception e) {
            System.err.print(e);
        }
    }
```

Ilustración 41. Código startMyMicrophone

Para capturar las imágenes de la cámara utilizaremos el método *startMyCamera,* este creará un nuevo hilo de ejecución donde se comprobará en bucle el valor booleano *isCameraOpen,* en caso de que este valor sea verdadero, se harán dos tareas:

1 Se modificará el contenido de *cameraImage*, donde veremos las imágenes recogidas por nuestra propia cámara, estas imágenes captadas tomarán la configuración estándar de la webcam, estando configurada en el sistema para capturar imágenes en formato jpeg. A pesar de que el formato jpeg no elimina redundancia temporal, es el que menor latencia introduce por cómputo del algoritmo de compresión.

2 Enviaremos esta imagen al otro usuario con el que estamos estableciendo la comunicación, en el puerto y dirección correspondiente.

```
private void startMyCamera() throws IOException {

    //enviamos un mensaje al cuadro de texto indicando que se ha encendido la cámara
     sendText("A ENCENDIDO SU CÁMARA");

    //creamos un hilo independiente que controle la actualización de la imagen de nuestra cámara
    //este hilo no envía datos, solo nos lo actualiza a nosotros mismos
    try {
        threadCamera = new Thread(() -> {

            while (isCameraOpen) {
                try {
                    //modificamos nuestra propia imagen
                    cameraImage.setImage(SwingFXUtils.toFXImage(defaultCamera.getImage(), null));

                    //Utilizamos ImageIcon en lugar de Image porque es serializable y enviamos la imagen al otro interlocutor
                    if(defaultCamera.isOpen()){
                        ImageIcon m = new ImageIcon(defaultCamera.getImage());

                        if (StartWindowsController.getIsServer()) {

                            ctcp = new ClientTCP(StartWindowsController.getIPPartner(), ConfigurationWindowsController.getPort() + 1000, m);

                        } else {

                            ctcp = new ClientTCP(StartWindowsController.getIPPartner(), ConfigurationWindowsController.getPort(), m);

                        }
                    }

                } catch (IOException ex) {
                    Logger.getLogger(ChatApplicationWindowsController.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
        });
        //iniciamos el hilo de la actualización de imagen de camara
        threadCamera.start();
    } catch (Exception e) {
        System.err.print(e);
    }

}
```

Ilustración 42. Código startMyCamera

Como ya se ha mencionado, el estado de la cámara se verá controlado por el valor de la variable booleana *isCameraOpen*, este valor se modificará con el método *cameraChangeStat*, el cual está vinculada al botón *Camara* de la interfaz gráfica.

Al presionar el botón *Camara* se realizarán las siguientes acciones:

- Caso verdadero:

  1 Se comprueba el valor del booleano *isCameraOpen*, si este es verdadero se cambia el estado a falso.
  2 Se apagará la cámara que tengamos por defecto.
  3 Se llamará al método *stopMyCamera* que se encargará de enviar nuestra imagen por defecto (aquella que seleccionamos en la ventana anterior) a nuestro interlocutor.

- Caso falso:

En caso de que *isCameraOpen* sea falso se procederá a realizar el opuesto a las acciones anteriormente mencionadas. Por un lado, se colocará la variable a verdadero, luego se encenderá la cámara que tengamos seleccionada por defecto, y por último se llamará a la función *startMyCamera.*

```
private void cameraChangeStat(ActionEvent event) throws InterruptedException {
    if (isCameraOpen) {
        cameraOffButton.setVisible(true);
        cameraOnButton.setVisible(false);
        isCameraOpen = false;
        defaultCamera.close();
        stopMyCamera();
        System.out.println("Camara Desactivada");

    } else {
        cameraOffButton.setVisible(false);
        cameraOnButton.setVisible(true);
        isCameraOpen = true;
        defaultCamera.open();
        startMyCamera();
        System.out.println("Camara Activada");

    }
}
```

Ilustración 43. Código cameraChangeStat

Para el envío de texto en la comunicación emplearemos el método *sendText*, asociada al botón *Enviar.* Este método se encargará de tomar el texto contenido en el área de escritura *textAreaUser*, agregarlo al área de chat común y enviarlo al usuario con el que estemos estableciendo la conexión. Agregaremos al comienzo de esta cadena de texto el nombre que hayamos definido en la ventana inicial.

```
@FXML
private void sendText(MouseEvent event) throws IOException {
    //Colocamos el cuadro de texto donde escribir en blanco
    if (ConfigurationWindowsController.getProtocol().equals("TCP")) {

        //tomamos el texto escrito y lo guardamos en la variable m
        String m = StartWindowsController.getName() + ": " + textAreaUser.getText() + "\n";
        //lo mostramos en el area de chat
        chatArea.appendText(m);
        //cremos un cliente en el puerto y le pasamos el mensaje guardado
        //ip de la otra máquina
        //al estar enviando un texto agregamos las letras tt delante del mensaje
        if (StartWindowsController.getIsServer()) {
            ctcp = new ClientTCP(StartWindowsController.getIPPartner(), ConfigurationWindowsController.getPort() + 1000, m);
        } else {
            ctcp = new ClientTCP(StartWindowsController.getIPPartner(), ConfigurationWindowsController.getPort(), m);
        }
        //colocamos en blanco el cuadro de texto
        textAreaUser.setText("");

    }
}
```

Ilustración 44. Código sendText

Para poder recibir e interpretar toda la información que nos llega desde los observables disponemos del método *update.* Aquí recolectaremos todos los mensajes que nos lleguen desde nuestro interlocutor, ya sea texto, imagen o audio, y lo interpretaremos de la manera que corresponda. Para ello nos serviremos del método *getClass,* este método devolverá la clase del objeto que nos está llegando, dependiendo del tipo que sea (*String, ImageIcon* o *array de byte*) se interpretarán de una manera o de otra:

- Si recibimos un *String*: lo que haremos será agregarlo al área de chat escrito.
- Si recibimos un *ImageIcon:* se trata de la imagen que nos está enviando la webcam de nuestro interlocutor, por tanto, tendremos que convertirlo a un formato compatible para poder verlo en nuestra interfaz gráfica.
  Recordemos que la imagen que vemos es de la clase *ImageView*, ya que es como trabaja *JavaFX*, por tanto, tenemos que convertirlo primero en un *BufferImage*, con el ancho y alto correspondiente, para poder utilizar la función *toFXImage* de la clase *SwingFXUtils*, de esta manera podremos actualizar la imagen de nuestra interfaz gráfica [Ora22][Wer22].
- Si recibimos un *array de bytes:* se trata de un audio que nos ha enviado nuestro interlocutor. El proceso para reproducirlo es muy similar al que realizamos a la hora de enviarlo, lo que tenemos que hacer es definir el formato de audio con las mismas características con las que se envió el paquete, una vez definido el formato, se procederá a recibir la información en un array de byte del mismo tamaño que el recibido, por último, escribimos esta información en el *SourceDataLine audioReceived*, con la función *write*, para que se reproduzca en nuestra salida de audio predeterminada.

```
//MÉTODO AL QUE LE LLEGAN LOS CAMBIOS DE NOTIFYOBSERVER
@Override
public void update(Observable o, Object arg) {
    try {

            System.out.println("se ha recibido:" + arg.getClass());

            //Si lo que llega es un String lo colocamos en el chat
            if (arg.getClass() == String.class) {
                chatArea.appendText(arg.toString());
                textMyPartner.setText(arg.toString().substring(0, arg.toString().indexOf(":")));
            }
            //Si lo que llega es una imagen la colocamos en el visor de imagen de la camara de nuestro compañero
            if (arg.getClass() == ImageIcon.class) {

                ImageIcon aux = (ImageIcon) arg;

                //convertimos la  ImageIcon en un bufferImage
                BufferedImage bi = new BufferedImage(aux.getIconWidth(),aux.getIconHeight(), BufferedImage.SCALE_FAST);
                Graphics g = bi.createGraphics();
                aux.paintIcon(null, g, 0, 0);
                g.dispose();

                //actualizamos la imagen de la cámara
                cameraImagePartner.setImage(SwingFXUtils.toFXImage(bi, null));
            } else {
                //en el caso que llegue un audio
                if (arg.getClass() != String.class && arg.getClass() != ImageIcon.class) {
                    byte[] buffer = new byte[512];
                    buffer = (byte[]) arg;
                    audioReceived.write(buffer, 0, buffer.length);

                }
            }
        });
        threadUpdate.start();

    } catch (Exception e) {
        System.err.print(e);
    }
}
```

Ilustración 45. Código update

Una de las ventajas que tenemos por gestionar los paquetes que recibimos a través de esta misma función *update* es que si, de cara al futuro, quisiésemos agregar funciones de intercambio de otro tipo de datos entre usuarios, solo tendríamos que preocuparnos de interpretar los *Objects* recibidos a través de esta función y gestionarlos de manera correcta, facilitando así la escalabilidad.

En los diferentes códigos que hemos comentado se han mencionado la comunicación entre dos interlocutores y el envío de información entre ellos, para conseguir esto se han definimos dos clases diferentes: *ClientTCP* y *ServerTCP*. Si observamos los anteriores códigos descritos relacionados con el envío y recepción de paquetes podremos comprobar que tanto el anfitrión como el invitado realizarán las funciones de servidor y cliente a la vez.

Como ya se ha mencionado en el apartado *7.1 Funcionamiento de la aplicación:* la parte cliente será el encargado de el envió de datos, y la parte de servidor se encargará de recoger la información que le llegue desde el cliente para notificarlo a los observadores.

```
public class ClientTCP {

        private String host;
        private int port;
        private Object message;

        private static int i = 0;
        Socket sc;
        ObjectOutput out;

        //el constructor de cliente recibe como argumentos el host un puerto y un mensaje
        public ClientTCP(String h, int p, Object o) throws IOException {
            host = h;
            port = p;
            message = o;
            //cremos un socket para comunicarnos con el servidor
            sc = new Socket(host, port);

            //preparamos todo para enviar el flujo de datos al servidor
            out = new ObjectOutputStream(sc.getOutputStream());
            //escribimos el objeto a enviar
            out.writeObject(message);
            //lo enviamos
            out.flush();

        }
}
```

Ilustración 46. Código ClientTCP

```
public class ServerTCP extends Observable implements Runnable{

    private int port;

    public ServerTCP(int p){
        port = p;
    }

    @Override
    public void run(){

        ServerSocket server = null;
        Socket sc = null;
        ObjectInputStream in;

        try {

            //iniciamos el servidor recibiendo mensajes por el puerto
            server = new ServerSocket(port);

            //creamos un bucle que escuche constantemerne las entradas del servidor
            while (true) {

                //el cliente se conecta al servidor
                sc = server.accept();

                //flujo de datos que llega desde el cliente
                in = new ObjectInputStream(sc.getInputStream());

                //leemos el mensaje que llega
                Object mensaje = in.readObject();

                //informamos que va a comabiar algo
                this.setChanged();
                //notificamos lo que ha cambiado a los observadores
                this.notifyObservers(mensaje);
                //limpiamos los cambios y continuamos con la ejecución
                this.clearChanged();

            }
        } catch (Exception e) {
            System.err.print(e);
        }

    }
}
```

Ilustración 47. Código ServerTCP

7.3.4 Controlador de la ventana de finalizar llamada

Para gestionar esta ventana y las funciones asociadas a sus botones se definirá el controlador *EndCallController,* en este tendremos:

- La función *endComunication*: como su nombre indica servirá para terminar la comunicación. Como el objetivo de la aplicación es establecer una comunicación, el terminarla supondrá la finalización de la ejecución del programa, por ello se hará uso de la función *exit* de la clase *System.* Utilizamos la función *exit* en particular porque con ella terminamos la ejecución de la máquina virtual de java, de esta manera nos aseguramos de que no quede ninguno hilo o código suelto ejecutándose en segundo plano.
- La función *cancel:* esta será llamada a la hora de presionar el botón *Cancelar*, aquí lo único que haremos es cerra la ventana que acabamos de abrir, por tanto, no influirá en la ejecución del programa de ninguna manera.

```
public class EndCallController implements Initializable {

    @FXML
    private Button buttonExit;
    @FXML
    private Button buttonCancel;

    /**
     * Initializes the controller class.
     */
    @Override
    public void initialize(URL url, ResourceBundle rb) {
        // TODO
    }

    @FXML
    private void endComunication(ActionEvent event) {
        System.exit(0);
    }

    @FXML
    private void cancel(ActionEvent event) {
        Stage stage = (Stage) this.buttonExit.getScene().getWindow();
        stage.close();
    }

}
```

Ilustración 48. Código EndCallController

8. Implantación

En este apartado se presentará el proceso de implementación de la solución para realizar las pruebas de funcionamiento pertinentes.

8.1 Implantación en máquina local

La implantación de la aplicación en una máquina local se trata de la configuración más sencilla de todas. En este caso bastara con ejecutar el archivo .jar resultante de la exportación del programa en un ordenador que disponga de micrófono, cámara y *Java* instalado.

Las características de esta implementación será la base para el resto de las implementaciones.

8.2 Implantación en entorno de red local

Para implantar la aplicación en un entorno de red local necesitaremos dos ordenadores conectados a la misma red. Dentro de esta red desconectaremos el resto de los dispositivos, solo estarán nuestros dos ordenadores con sus relojes internos correctamente sincronizados.

Estos ordenadores estarán conectados por cable Ethernet, también fijaremos sus direcciones IPs como estáticas para asegurarnos que no haya ninguna variación o error a la hora de establecer la conexión con el dispositivo correcto. Además de esto desactivaremos el firewall de ambos dispositivos.

La configuración de esta red será la presentada en la siguiente ilustración:

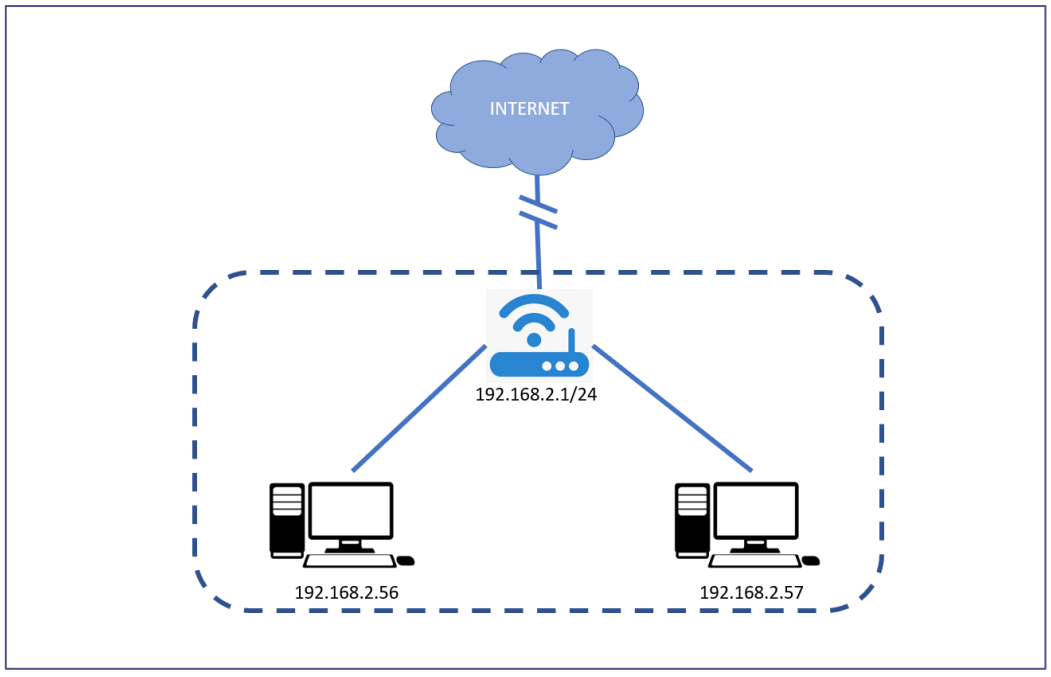

Ilustración 49. Ilustración 51. Esquema de las red entorno local

8.3 Implantación en entorno de redes independientes

Para esta implantación se realizará una configuración muy parecida a la vista en el apartado anterior. En este caso los ordenadores pertenecerán a redes distintas, para permitir su comunicación a través de internet se hará uso del programa *LogMeIn Hamachi*, con el que crearemos una LAN virtual compuesta por dos ordenadores distintos.

Para evitar retrasos en los envíos de paquetes desactivaremos en las opciones de *LogMeIn Hamachi* el cifrado de datos y compresión.

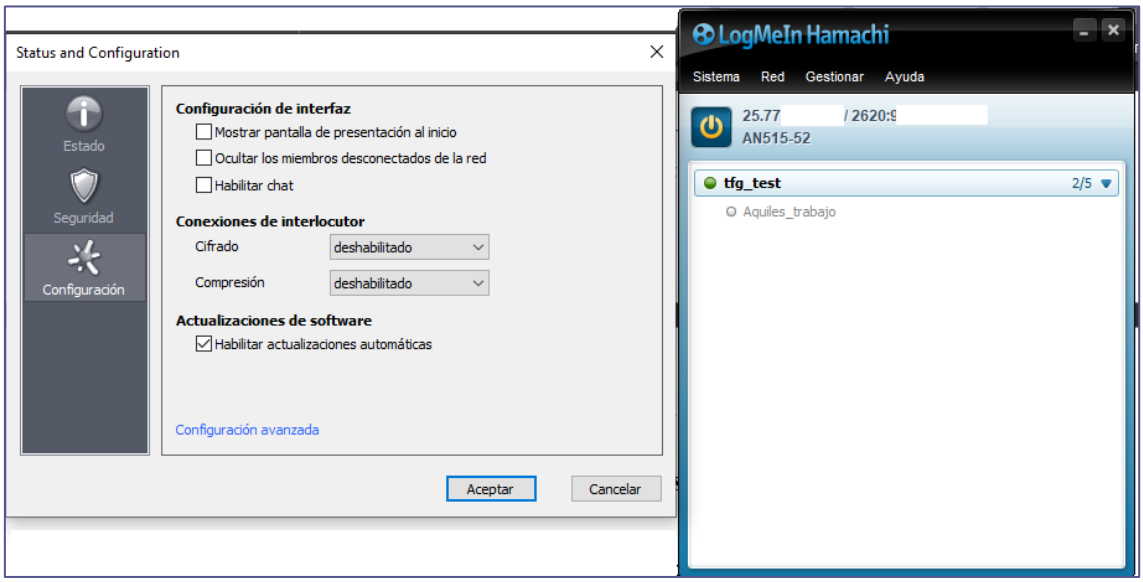

Ilustración 50. Configuración básica de LogMeIn Hamachi

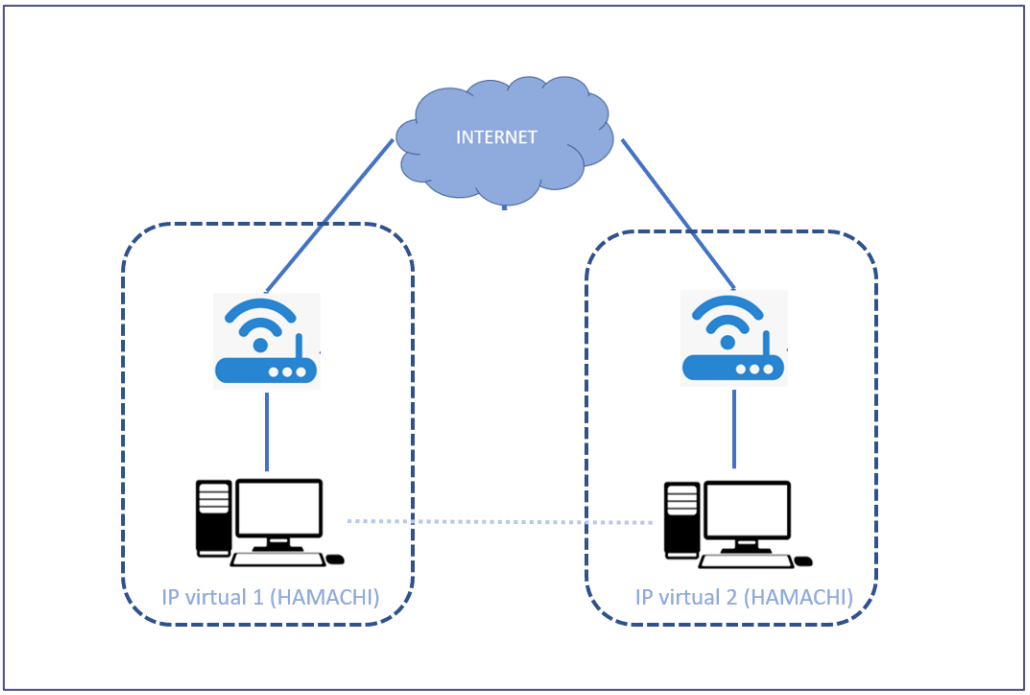

Ilustración 51. Esquema de red entorno LAN

Adicionalmente a esto conectaremos nuestros ordenadores a dos salidas de internet distintas. Uno de nuestros ordenadores obtendrá su salida a red por un adaptador wifi 5G de una compañía telefónica, el otro ordenador obtendrá su salida a red desde un rúter con conexión Ethernet.

9. Pruebas

En este apartado se describirán las pruebas realizadas a la solución desarrollada. Realizaremos tres pruebas de funcionalidad distintas. La primera prueba se ralizará dentro de un mismo ordenador a través de su comunicación local. La segunda prueba se realizará en dos ordenadores distintos conectados a una misma red, la idea de realizar esta prueba es simular un entorno de funcionamiento cercano a lo planteado en el apartado *6.2 Escenario de uso*. La tercera prueba se hará con ordenadores conectado a internet pertenecientes a dos redes distintas.

9.1 Prueba en máquina local

Como hemos dicho, para la primera prueba de funcionalidad se iniciará el programa en el mismo ordenador con dos ejecuciones de la aplicación a la vez, escogiendo quien sería el anfitrión y quien el invitado.

Una vez iniciada la comunicación se comprobará cada medio de transmisión de datos por partes:

1 Se intercambiarán mensajes de texto.
2 Se procederá a encender los micrófonos para comprobar el envío de audio.
3 Se iniciará la transmisión de la cámara web para comprobar el correcto envió de imágenes.

Como medida adicional, antes de iniciar las pruebas, se modificó los códigos de *ClientTCP* y *ServerTCP*. La modificación realizada corresponde a la adicción de la línea de código *System.out.println("recibido " + i + " : "+message.toString() + " de tipo: " + message.getClass()+ " a las: "+ZonedDateTime.now()),* en el caso de *ServerTCP ,* y *System.out.println("enviado " + i + " : "+ message.toString()+ " de tipo: " + message.getClass()+ " a las: "+ZonedDateTime.now()),* en el caso de *ClientTCP*. En ambos casos la variable entera *i* se incrementará en uno a medida que se vayan enviando y recibiendo mensajes. El hecho de imprimir estos mensajes por la salida del sistema, y no en algún apartado visible de la aplicación principal, no es otro que no sobrecargar la vista de la aplicación de cara al usuario final, consideramos que mostrar por pantalla textos con información relativa al intercambio de paquetes no resultaría relevante ni esclarecedor para la mayoría de los usuarios.

Esta modificación se ha realizado para poder tener un control del tipo de datos que estamos enviando y el tipo que llega, así como la coincidencia en el número y orden que llega la información en ambos extremos de la comunicación, la inclusión del *ZonedDateTime* se hizo para comprobar si existe algún retraso considerable en la recepción de paquetes.

El lanzado de la aplicación se realizará desde el terminal de *Windows con el comando java -jar .\ChatAplicaction.jar*, para poder leer y tener un registro de los mensajes del sistema anteriormente mencionados.

La primera prueba se realizó iniciando solamente una conversación escrita, para comprobar la eficiencia de este canal se enviaron, en principio, textos cortos y luego secuencias de texto más largas, concretamente los tres primeros párrafos del capítulo 1 de *El conde de Montecristo*.

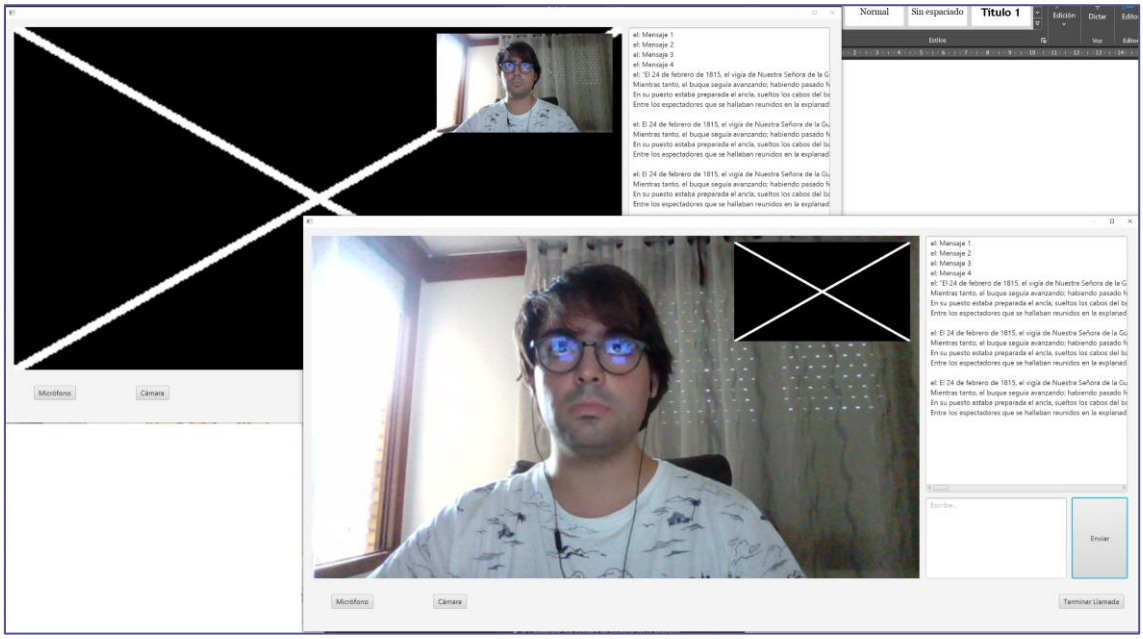

Ilustración 52. Aplicación ejecutándose de manera local

Los paquetes correspondientes a los mensajes escritos son los que van del 0 al 7, de estos los primeros cuatro son los mensajes cortos y los siguientes la copia del fragmento de *El conde de Montecristo*. La diferencia de tiempos desde el origen al destino son los que se ilustran en la tabla siguiente:

	Salida de anfitrión	Recepción de invitado	Diferencia
Envió 0	15:59:16.150	15:59:16.151	00:00:00.001
Envió 1	15:59:22.402	15:59:22.402	00:00:00.000
Envió 2	15:59:30.336	15:59:30.337	00:00:00.001
Envió 3	15:59:39.251	15:59:39.251	00:00:00.000
Envió 4	16:00:01.931	16:00:01.931	00:00:00.000
Envió 5	16:00:13.220	16:00:13.220	00:00:00.000
Envió 6	16:00:15.872	16:00:15.873	00:00:00.001
Envío 7	16:00:18.731	16:00:18.731	00:00:00.000

Como podemos ver la diferencia de tiempo entre el envío y recepción es casi inexistente, hasta el punto de que en ocasiones el rango de representación de los milisegundos no es capaz de registrar la diferencia de tiempos, independientemente de que tan largo sea la cadena de texto enviada.

La siguiente prueba que se realizó fue activar el micrófono para comprobar el envío de audios entre dispositivos. Los paquetes correspondientes al envió de audio son los comprendidos entre el 8 y el 206, para el análisis de estos tomaremos pequeños conjuntos de paquetes enviados:

	Salida de anfitrión	Recepción de invitado	Diferencia
Envió 8	16:00:29.612	16:00:29.617	00:00:00.005
Envió 9	16:00:29.612	16:00:29.713	00:00:00.001
Envió 10	16:00:29.612	16:00:29.713	00:00:00.100
Envió 11	16:00:29.735	16:00:29.735	00:00:00.000
Envió 12	16:00:29.735	16:00:29.736	00:00:00.001
Envió 13	16:00:29.736	16:00:29.736	00:00:00.000

Envió 14	16:00:29.862	16:00:29.863	00:00:00.001
…	…	…	…
Envío 58	16:00:31.262	16:00:31.262	00:00:00.000
Envío 59	16:00:31.263	16:00:31.263	00:00:00.000
Envío 60	16:00:31.263	16:00:31.264	00:00:00.001
Envío 61	16:00:31.264	16:00:31.265	00:00:00.001
Envío 62	16:00:31.390	16:00:31.391	00:00:00.001
Envío 63	16:00:31.391	16:00:31.392	00:00:00.001
Envío 64	16:00:31.391	16:00:31.393	00:00:00.002
…	…	…	…
Envío 113	16:00:32.925	16:00:32.927	00:00:00.002
Envío 114	16:00:33.051	16:00:33.052	00:00:00.001
Envío 115	16:00:33.052	16:00:33.053	00:00:00.001
Envío 116	16:00:33.053	16:00:33.055	00:00:00.002
Envío 117	16:00:33.053	16:00:33.055	00:00:00.002
Envío 118	16:00:33.054	16:00:33.056	00:00:00.002
Envío 119	16:00:33.180	16:00:33.180	00:00:00.000
…	…	…	…
Envío 199	16:00:35.728	16:00:35.728	00:00:00.000
Envío 200	16:00:35.728	16:00:35.729	00:00:00.001
Envío 201	16:00:35.729	16:00:35.730	00:00:00.001
Envío 202	16:00:35.854	16:00:35.855	00:00:00.001
Envío 203	16:00:35.855	16:00:35.856	00:00:00.001
Envío 204	16:00:35.855	16:00:35.857	00:00:00.002
Envío 205	16:00:35.856	16:00:35.858	00:00:00.002
Envío 206	16:00:35.984	16:00:35.984	00:00:00.000

Desde el momento que iniciamos el micrófono hasta que lo detuvimos, el número de paquetes enviados y recibidos fueron 198 en los dos extremos, por tanto, no se observaron perdidas de datos, a la hora de interpretar los audios por los dos extremos tampoco notamos una falta de información, siendo perfectamente interpretable el audio en ambos puntos.

Atendiendo a el retraso entre el envío y recepción de paquetes, tomando como referencia las muestras representadas en la tabla anterior, vemos que apenas hay una diferencia notable entre el momento que sale un paquete y se recibe. La diferencia más alta la notamos en el paquete 10, con un retraso en la entrega de 100ms, aunque esto solo es notable desde el punto de vista numérico al ver los datos representado en la tabla, en el funcionamiento normal de la aplicación no notamos dicho retraso.

Por último, activamos la cámara para comprobar el envío y recepción de los paquetes de imágenes, los paquetes correspondientes a las imágenes de la cámara son los comprendidos entre el 207 y el 310, tanto en la parte del anfitrión como en la parte del invitado, por tanto, no contemplamos ninguna perdida de paquetes.

Al igual que antes tomaremos muestras de estos paquetes para comprobar sus valores de envío y recepción en los dos extremos:

	Salida de anfitrión	Recepción de invitado	Diferencia
Envió 207	16:00:42.315	16:00:42.346	00:00:00.031
Envió 208	16:00:42.503	16:00:42.504	00:00:00.001

Envió 209	16:00:42.643	16:00:42.644	00:00:00.001
Envió 210	16:00:42.914	16:00:42.917	00:00:00.003
Envió 211	16:00:43.091	16:00:43.092	00:00:00.001
Envió 212	16:00:43.301	16:00:43.311	00:00:00.010
Envió 213	16:00:43.491	16:00:43.492	00:00:00.001
Envío 214	16:00:43.704	16:00:43.705	00:00:00.001
Envío 215	16:00:43.895	16:00:43.896	00:00:00.001
…	…	…	…
Envío 260	16:00:51.831	16:00:51.832	00:00:00.002
Envío 261	16:00:52.040	16:00:52.041	00:00:00.001
Envío 262	16:00:52.229	16:00:52.230	00:00:00.001
Envío 263	16:00:52.435	16:00:52.436	00:00:00.001
Envío 264	16:00:52.593	16:00:52.594	00:00:00.000
Envío 265	16:00:52.826	16:00:52.827	00:00:00.001
Envío 266	16:00:53.043	16:00:53.044	00:00:00.001
Envío 267	16:00:53.237	16:00:53.238	00:00:00.001
…	…	…	…
Envío 299	16:00:58.565	16:00:58.566	00:00:00.001
Envío 300	16:00:58.771	16:00:58.772	00:00:00.001
Envío 301	16:00:58.973	16:00:58.975	00:00:00.002
Envío 302	16:00:59.124	16:00:59.125	00:00:00.001
Envío 303	16:00:59.366	16:00:59.366	00:00:00.000
Envío 304	16:00:59.574	16:00:59.575	00:00:00.001
Envío 305	16:00:59.756	16:00:59.757	00:00:00.001
Envío 306	16:00:59.970	16:00:59.970	00:00:00.000
Envío 307	16:01:00.162	16:01:00.163	00:00:00.001
Envío 308	16:01:00.379	16:01:00.379	00:00:00.000
Envío 309	16:01:00.566	16:01:00.567	00:00:00.001
Envío 310	16:01:00.772	16:01:00.773	00:00:00.001

En este caso los resultados obtenidos son muy similares a los vistos en los envíos de paquetes anteriores, siendo el retraso más relevante el del paquete inicial, aunque no es un retraso crítico, al no llegar ni siquiera a un segundo resulta intrascendente, por tanto, no afecta al correcto desarrollo de la comunicación.

Estos resultados están dentro de lo predecible, ya que al estar ejecutando todo dentro de una misma máquina local se esperaba que la latencia de datos fuera mínima o nula.

9.2 Prueba en red local

Para la segunda prueba de funcionalidad ejecutaremos la aplicación en dos ordenadores diferentes conectado a la misma red. En esta comunicación se hará uso del envío de texto, audio e imágenes de la cámara para comprobar la correcta transmisión de datos, al igual que hicimos con las pruebas de ejecución local.

El orden de activación de los canales de comunicación serán los mismos que los realizados en la configuración anterior, por tanto, iniciaremos la ejecución enviando mensajes de texto (los textos enviados serán los mismos vistos en el ejemplo anterior).

	Salida de anfitrión	Recepción de invitado	Diferencia
Envió 0	15:26:40.247	15:26:40.252	00:00:00.005
Envió 1	15:26:51.323	15:26:51.327	00:00:00.004
Envió 2	16:26:55.644	16:26:55.649	00:00:00.005
Envió 3	16:27:02.495	16:27:02.500	00:00:00.005
Envió 4	16:27:21.298	16:27:21.303	00:00:00.005
Envió 5	16:27:23.390	16:27:23.396	00:00:00.006
Envió 6	16:27:25.842	16:27:25.847	00:00:00.005
Envío 7	16:28:02.731	16:28:02.737	00:00:00.006

Aquí vemos que, sin importar el tamaño de la cadena de texto que enviamos, la latencia entre el envío y recepción de los mensajes se mantiene casi constante, obteniendo mayoritariamente valores comprendidos, entre el envío y recepción, de unos 5-6 ms.

Lo siguiente que hemos hecho fue activar el micrófono. Los paquetes correspondientes al envió de audio son los comprendidos entre el 8 y el 206. Al igual que hemos hecho en la prueba anterior tomaremos una muestra de estos paquetes para analizar la latencia entre ellos:

	Salida de anfitrión	Recepción de invitado	Diferencia
Envió 8	21:13:46.556	21:13:46.612	00:00:00.056
Envió 9	21:13:46.558	21:13:46.613	00:00:00.055
Envió 10	21:13:46.685	21:13:46.740	00:00:00.055
Envió 11	21:13:46.689	21:13:46.741	00:00:00.037
Envió 12	21:13:46.692	21:13:46.742	00:00:00.050
Envió 13	21:13:46.694	21:13:46.743	00:00:00.049
Envió 14	21:13:46.697	21:13:46.744	00:00:00.047
…	…	…	…
Envío 58	21:37:47.789	21:37:47.846	00:00:00.057
Envío 59	21:37:47.791	21:37:47.847	00:00:00.056
Envío 60	21:37:47.793	21:37:47.848	00:00:00.055
Envío 61	21:37:47.796	21:37:47.849	00:00:00.053
Envío 62	21:37:47.798	21:37:47.850	00:00:00.052
Envío 63	21:37:47.802	21:37:47.851	00:00:00.049
Envío 64	21:37:47.805	21:37:47.852	00:00:00.047
…	…	…	…
Envío 113	21:37:49.519	21:37:49.544	00:00:00.025
Envío 114	21:37:49.521	21:37:49.545	00:00:00.024
Envío 115	21:37:49.524	21:37:49.549	00:00:00.025
Envío 116	21:37:49.526	21:37:49.552	00:00:00.050
Envío 117	21:37:49.528	21:37:49.578	00:00:00.050
Envío 118	21:37:49.531	21:37:49.582	00:00:00.051
Envío 119	21:37:49.533	21:37:49.584	00:00:00.051
…	…	…	…
Envío 199	21:37:53.386	21:37:53.436	00:00:00.050
Envío 200	21:37:53.388	21:37:53.437	00:00:00.049
Envío 201	21:37:53.390	21:37:53.439	00:00:00.049
Envío 202	21:37:53.393	21:37:53.440	00:00:00.047
Envío 203	21:37:53.395	21:37:53.442	00:00:00.047

Envío 204	21:37:53.397	21:37:53.443	00:00:00.046
Envío 205	21:37:53.399	21:37:53.444	00:00:00.045
Envío 206	21:37:53.400	21:37:53.444	00:00:00.044

Aquí podemos ver una latencia mayor que en el caso de los mensajes de texto, en este caso la latencia se ha mantenido siempre por debajo de 60 ms, siendo esta indetectable desde el punto de vista del usuario.

Por último, activamos la cámara y capturamos los tiempos de envío y recepción de estos paquetes:

	Salida de anfitrión	Recepción de invitado	Diferencia
Envió 207	00:30:33.192	00:30:33.250	00:00:00.058
Envió 208	00:30:33.350	00:30:33.422	00:00:00.072
Envió 209	00:30:33.506	00:30:33.549	00:00:00.043
Envió 210	00:30:33.694	00:30:33.795	00:00:00.101
Envió 211	00:30:33.936	00:30:34.001	00:00:00.064
Envió 212	00:30:34.034	00:30:34.083	00:00:00.049
Envió 213	00:30:34.198	00:30:34.248	00:00:00.050
Envío 214	00:30:34.354	00:30:34.401	00:00:00.050
Envío 215	00:30:34.474	00:30:34.534	00:00:00.060
…	…	…	…
Envío 480	00:30:35.121	00:30:35.171	00:00:00.051
Envío 481	00:30:35.277	00:30:35.332	00:00:00.055
Envío 482	00:30:35.433	00:30:35.492	00:00:00.059
Envío 483	00:30:35.588	00:30:35.615	00:00:00.027
Envío 484	00:30:35.741	00:30:35.793	00:00:00.053
Envío 485	00:30:35.897	00:30:35.947	00:00:00.050
Envío 486	00:30:36.050	00:30:36.122	00:00:00.062
Envío 487	00:30:36.170	00:30:36.220	00:00:00.050
…	…	…	…
Envío 580	00:30:41.410	00:30:41.461	00:00:00.051
Envío 581	00:30:41.530	00:30:41.582	00:00:00.052
Envío 582	00:30:41.651	00:30:41.701	00:00:00.050
Envío 583	00:30:41.771	00:30:41.822	00:00:00.051
Envío 584	00:30:41.891	00:30:42.941	00:00:00.050
Envío 585	00:30:42.061	00:30:42.116	00:00:00.055
Envío 586	00:30:42.181	00:30:42.237	00:00:00.056
Envío 587	00:30:42.301	00:30:42.355	00:00:00.054
Envío 588	00:30:42.418	00:30:42.470	00:00:00.052
Envío 589	00:30:43.538	00:30:42.591	00:00:00.053
Envío 590	00:30:42.657	00:30:42.603	00:00:00.054
Envío 591	00:30:42.776	00:30:42.829	00:00:00.053

Como podemos observar, la latencia obtenida en este caso es muy similar a la que encontramos a la hora de enviar audio (unos 54 ms entre emisión y recepción).

9.3 Prueba en redes independientes

Para la tercera prueba de funcionalidad ejecutaremos la aplicación en dos ordenadores diferentes, pertenecientes a redes distintas y conectados a internet, el puente entre ellos será posible con el uso de la aplicación *LogMeIn Hamachi*. En esta comunicación se hará uso del envío de texto, audio e imágenes de la cámara.

Los paquetes 0 al 7 se corresponden con el texto enviado, los resultados obtenidos fueron los siguientes:

	Salida de anfitrión	Recepción de invitado	Diferencia
Envió 0	01:07:45.591	01:07:45.623	00:00:00.032
Envió 1	01:07:47.630	01:07:47.672	00:00:00.042
Envió 2	01:07:50:155	01:07:50.178	00:00:00.023
Envió 3	01:07:51.779	01:07:51.812	00:00:00.033
Envió 4	01:07:53.616	01:07:53.655	00:00:00.039
Envió 5	01:07:55.059	01:07:55.089	00:00:00.039
Envió 6	01:08:59.115	01:08:59.155	00:00:00.040
Envío 7	01:09:00.059	01:09:00.097	00:00:00.037

El tiempo de envío de texto es relativamente estable en este entorno, siendo la latencia media de envío de unos 43 ms.

La siguiente prueba realizada se corresponde con el envío y recepción de paquetes de audio (paquetes de 8 al 95):

	Salida de anfitrión	Recepción de invitado	Diferencia
Envió 8	01:12:06.598	01:12:06.653	00:00:00.055
Envió 9	01:12:06.599	01:12:06.655	00:00:00.056
Envió 10	01:12:06.601	01:12:06.657	00:00:00.056
Envió 11	01:12:06.602	01:12:06.659	00:00:00.057
Envió 12	01:12:06.603	01:12:06.662	00:00:00.059
Envió 13	01:12:06.604	01:12:06.664	00:00:00.060
Envió 14	01:12:06.605	01:12:06.666	00:00:00.061
…	…	…	…
Envío 58	01:12:09.131	01:12:09.186	00:00:00.055
Envío 59	01:12:09.132	01:12:09.188	00:00:00.056
Envío 60	01:12:09.134	01:12:09.190	00:00:00.056
Envío 61	01:12:09.136	01:12:09.193	00:00:00.055
Envío 62	01:12:09.138	01:12:09.195	00:00:00.057
Envío 63	01:12:09.140	01:12:09.197	00:00:00.051
Envío 64	01:12:09.142	01:12:09.199	00:00:00.057
…	…	…	…
Envío 91	01:12:10.279	01:12:10.334	00:00:00.057
Envío 92	01:12:10.281	01:12:10.338	00:00:00.057
Envío 93	01:12:10.284	01:12:10.340	00:00:00.056
Envío 94	01:12:10.286	01:12:10.337	00:00:00.051
Envío 95	01:12:10.288	01:12:10.348	00:00:00.060

En este caso la latencia media registrada es de 56 ms, siendo estas muy similares a las vistas en las pruebas de red local. Podemos observar también que los paquetes llegan con muy poca diferencia de tiempo entre ellos, haciendo que la comunicación hablada sea posible.

Por último, realizaremos la prueba de envío de imagen (paquetes 96 a 195):

	Salida de anfitrión	Recepción de invitado	Diferencia
Envió 96	01:13:58.503	01:13:58.990	00:00:00.057
Envió 97	01:13:58.650	01:13:59.708	00:00:00.058
Envió 98	01:13:58.723	01:13:59.852	00:00:00.057
Envió 99	01:13:58.810	01:13:59.755	00:00:00.055
Envió 100	01:13:58.895	01:13:59.950	00:00:00.055
Envió 101	01:13:59.019	01:13:59.075	00:00:00.056
Envió 102	01:13:59.100	01:13:59.157	00:00:00.057
Envío 103	01:13:59.336	01:13:59.392	00:00:00.056
Envío 104	01:13:59.427	01:13:59.482	00:00:00.055
…	…	…	…
Envío 130	01:14:02.939	01:14:03.002	00:00:00.063
Envío 131	01:14:03.044	01:14:03.099	00:00:00.055
Envío 132	01:14:03.189	01:14:03.243	00:00:00.054
Envío 133	01:14:03.330	01:14:03.388	00:00:00.058
Envío 134	01:14:03.438	01:14:03.494	00:00:00.056
Envío 135	01:14:03.589	01:14:03.659	00:00:00.070
Envío 136	01:14:03.717	01:14:03.771	00:00:00.054
Envío 137	01:14:03.852	01:14:03.908	00:00:00.056
…	…	…	…
Envío 184	01:14:10.236	01:14:10.290	00:00:00.054
Envío 185	01:14:10.386	01:14:10.440	00:00:00.054
Envío 186	01:14:10.497	01:14:10.552	00:00:00.055
Envío 187	01:14:10.701	01:14:11.764	00:00:00.063
Envío 188	01:14:10.784	01:14:11.845	00:00:00.061
Envío 189	01:14:10.887	01:14:10.945	00:00:00.058
Envío 190	01:14:11.008	01:14:110.67	00:00:00.059
Envío 191	01:14:11.143	01:14:11.213	00:00:00.070
Envío 192	01:14:11.280	01:14:11.336	00:00:00.056
Envío 193	01:14:11.445	01:14:11.504	00:00:00.059
Envío 194	01:14:11.533	01:14:11.590	00:00:00.057
Envío 195	01:14:11.690	01:14:12.748	00:00:00.058

Al igual que ocurre con el caso del envío de audio, los tiempos son muy similares a los vistos en el caso de red local, aquí la latencia ente envió y recepción es de 55 ms. También podemos ver que la diferencia entre el envío y recepción de paquetes es muy uniforme, dependiendo de la complejidad de la imagen que se procese.

Por tanto, podemos establecer que es posible mantener una conversación entre los dos ordenadores, a pesar de encontrarse en redes distintas.

10. Actualización de la interfaz

Como se puede apreciar, la interfaz actual de la aplicación no invita a su uso, aunque el programa funciona, no es agradable desde el punto de vista estético. Esto es así porque las pruebas se realizaron en un programa con un diseño muy influenciado por el prototipo inicial.

No tendría sentido haber desarrollado el aspecto gráfico final de la aplicación en su totalidad solo para realizar pruebas de funcionamiento, por tanto, el siguiente paso lógico es diseñar un aspecto gráfico más acorde a lo visto en el apartado *6.3 Interfaces graficas de las aplicaciones existentes*.

La comodidad de uso de un programa viene determinada tanto por la posición de sus elementos como por la facilidad para identificar de un simple vistazo para que sirve cada cosa. Para conseguir que nuestra aplicación se ajuste a estos estándares se contactó con un diseñador de gráfico titulado, Gabriel Molinaro, para un rediseñó de la interfaz de usuario.

10.1 Directrices para el diseñador

Primero se le explico para que servía y cómo funcionaba la aplicación, una vez explicado este punto se presentó al diseñador cada ventana y como estaban distribuidos actualmente los elementos.

Con esto explicado se procedió a pactar con el diseñador los requisitos y condiciones con los que tendría que trabajar a la hora de redefinir el diseño gráfico:

- El número de ventanas existentes tiene que respetarse.
- Cada ventana presentará la misma cantidad de botones.
- Los elementos contenidos en cada ventana son suprimibles si se plantea una alternativa para mostrar la misma información de manera más sencilla.
- El estilo visual deberá estar unificando, todo el programa debe formar parte de un conjunto y presentar el mismo lenguaje visual.
- Las dimensiones de las ventanas pueden ser modificables, las establecidas actualmente son meramente orientativas.
- Se tiene libertad total para redistribuir y redimensionar los elementos contenidos en cada ventana.

10.2 Actualización ventana de inicio

La primera interfaz en ser rediseñada fue la correspondiente a la ventana de inicio, en esta se establecerán las bases del lenguaje visual que se utilizará en el aspecto final de la aplicación.

Las sugerencias del diseñador gráfico fueron las siguientes:

- Uso de la tipografía *Roboto Light* con tamaño 12 pt para los textos fijos.
- Uso de la tipografía *Roboto Regular* con tamaño 14 pt para los textos variables.
- Uso de la tipografía *Roboto Regular* con tamaño 12 pt para los textos rellenables y desplegables.
- Uso de la tipografía *Roboto Medium* con tamaño 14 pt para los botones con texto en su interior.
- Se plantea que los botones tengan unas dimensiones de 109 x 38 px y una coloración azul.
- Para el botón conectar se ha sugerido que se cambie su funcionalidad para que solo sea seleccionable cuando se rellenen los apartados de nombre y IP del destino, de esta manera podemos obviar el mensaje de error por falta de datos, ya que resultaría irrelevante.
- El botón de configuración se remplazará por un ícono de rueda dentada con dimensiones 31 x 31 px
- La coloración del fondo de la ventana pasará a ser blanco.
- La imagen de perfil se cambiará por una imagen de estilo circular con la silueta de una persona.
- La distribución de los elementos no presenta cambios significativos.

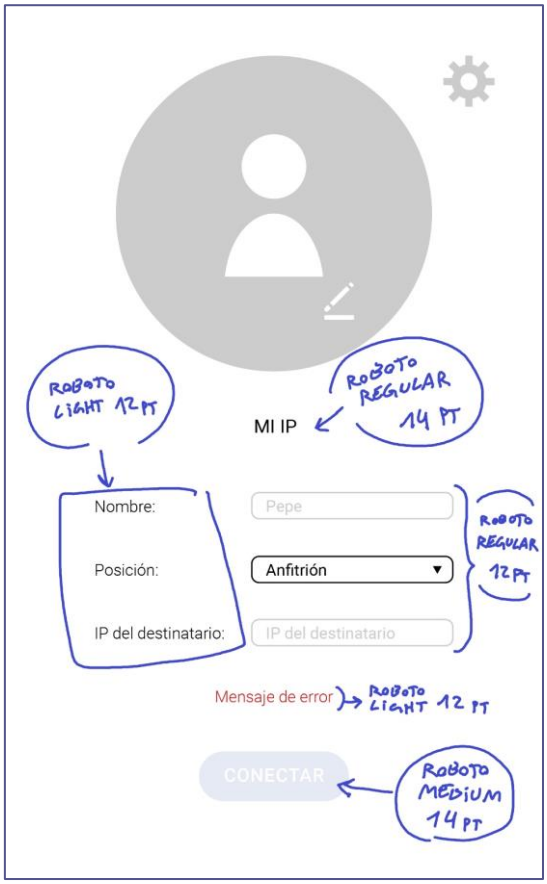

Ilustración 53. Croquis del nuevo diseño de la ventana de inicio con apuntes del diseñador

Para la realización de estos cambios, se procedió a crear un archivo CSS de estilos para aplicar la configuración correspondiente a cada elemento. En este archivo de estilos creamos tres configuraciones de estilos diferentes uno correspondiente a los botones con texto, otro para definir el color blanco de la ventana de fondo, y por último el que definirá el aspecto de las cajas de texto modificables.

El identificador de estilo para los botones con texto se llamará *#button-with-text* e incluirá las especificaciones necesarias para definir:

- El color del botón en #6c98d1 con el texto en blanco
- Tipo de letra *Roboto*, con ancho 500 (para definir que se trata del tipo *medium*)
- Tamaño de letra de 14 px
- Un radio de curvatura de 30, para conseguir esos bordes redondeados que vemos en el ejemplo proporcionado por el diseñador.

Para conseguir usar las fuentes de Google se importó la fuente por código directamente desde su web con la línea:

@import url('https://fonts.googleapis.com/css2?family=Roboto&display=swap').

Para las cajas de texto rellenables definimos el estilo *#box-text-fields,* en este se establece:

- Una curvatura de 30 para generar los bordes redondeados de estas cajas.
- Tipografía *Roboto* con un ancho de 400 (para precisar que se trata de tipo *regular*).
- Tamaño de fuente de 12 px.

Para las cajas de texto rellenables definimos el estilo #box-text-fields, en este se establece:

- Una curvatura de 30 para generar los bordes redondeados de estas cajas.
- Tipografía *Roboto* con un ancho de 400 (para precisar que se trata de tipo regular).
- Un tamaño de fuente de 12 px.

Para el color de fondo de la aplicación solamente tenemos que definir el comando *-fx-background-color: #ffffff* para que quede en tono blanco.

```
@import url('https://fonts.googleapis.com/css2?family=Roboto&display=swap');


#button-with-text{
 -fx-background-color:#6c98d1;
-fx-text-fill: #ffffff;
-fx-font-family: 'Roboto', sans-serif;
-fx-font-weight: 500;
-fx-font-size: 14px;
-fx-background-radius: 30;
}



#box-text-fields{
-fx-font-family: 'Roboto', sans-serif;
-fx-font-weight: 400;
-fx-font-size: 12px;
-fx-background-radius: 30;


}

#background-windows{
-fx-background-color: #ffffff;
}
```

Ilustración 54. CSS con la definición de #button-with-text, #box-text-fields y #background-windows

Para conseguir un aspecto circular en la imagen de perfil se realizó una modificación en el código de *initialize* del controlador *StartWindowsController.* En este se ha agregado un objeto *Circle* de radio igual a la mitad del ancho de la imagen que usamos de foto, la posición del mismo se ha definido en base a la posición relativa al centro de la imagen dentro de la ventana, obteniendo las coordenadas *x* e *y* de la imagen y sumándoles la mitad de su ancho, por último, aplicamos esta nueva forma al aspecto de nuestra foto con la función *setClip* [Ora223].

Para conseguir que no se pueda pulsar el botón de conectar, hasta que estén los datos de la IP del otro participante y nuestro nombre rellenos, se procedió a definir dentro de *initialize* dos nuevos oyentes, estos oyentes se encargarán de detectar las modificaciones de las cajas de texto de *myName* y *ipPartner.* En el momento que se detecten modificaciones, se llamará a una nueva fusión *setButtonVisible,* esta función revisará que los contenidos de estas dos cajas de texto no sean texto en blanco, de ser alguna de las dos un texto en blanco se desactivará el botón de conectar, por el contrario, de estar las dos cajas de texto rellenas se activará el botón.

```
//modificamos la forma de la imagen
Circle clip = new Circle();
clip.setRadius(myPhoto.getFitWidth() / 2);
clip.setCenterX(myPhoto.getX() + myPhoto.getFitWidth() / 2);
clip.setCenterY(myPhoto.getY() + myPhoto.getFitHeight() / 2);
myPhoto.setClip(clip);

//agregamos un oyente que detecte si se han llenado los campos
myName.textProperty().addListener(c -> setButtonVisible());
ipPartner.textProperty().addListener(c -> setButtonVisible());


//funcion que comprobará que todo esté rellenado con texto
private void setButtonVisible(){
        if (!myName.getText().equals("") && !ipPartner.getText().equals("")) {
         connectionButton.setDisable(false);
        }
         else{
         connectionButton.setDisable(true);
        }
  }
```

Ilustración 55. Código agregado a StartWindowsController dentro de initialize y setButtonVisible

Para hacer que la experiencia del usuario sea más cómoda, se realizó un cambio más para que le usuario final reconozca que tanto su imagen de perfil como la rueda dentada de configuración son botones seleccionables. Se agregaron dos funciones nuevas para cada uno de estos elementos, estas realizarán una tarea muy sencilla, se encargará de detectar el momento que el ratón está encima de ellas o no. Cuando el ratón esté encima de uno de estos dos elementos la opacidad de este se colocará a la mitad, en el momento que el ratón no esté encima de estos elementos la opacidad regresará a ser del 100%

```
@FXML
private void setButtonOpacityFull(MouseEvent event) {
    imageConfig.setOpacity(1);

}

@FXML
private void setButtonOpacityLess(MouseEvent event) {
    imageConfig.setOpacity(0.5);
}

@FXML
private void setImageOpacityLess(MouseEvent event) {
    myPhoto.setOpacity(0.5);
}

@FXML
private void setImageOpacityFull(MouseEvent event) {
    myPhoto.setOpacity(1);
}
```

Ilustración 56. Definición de setButtonOpacityFull, setButtonOpacityLess, setImageOpacityLess y setImageOpacityFull.

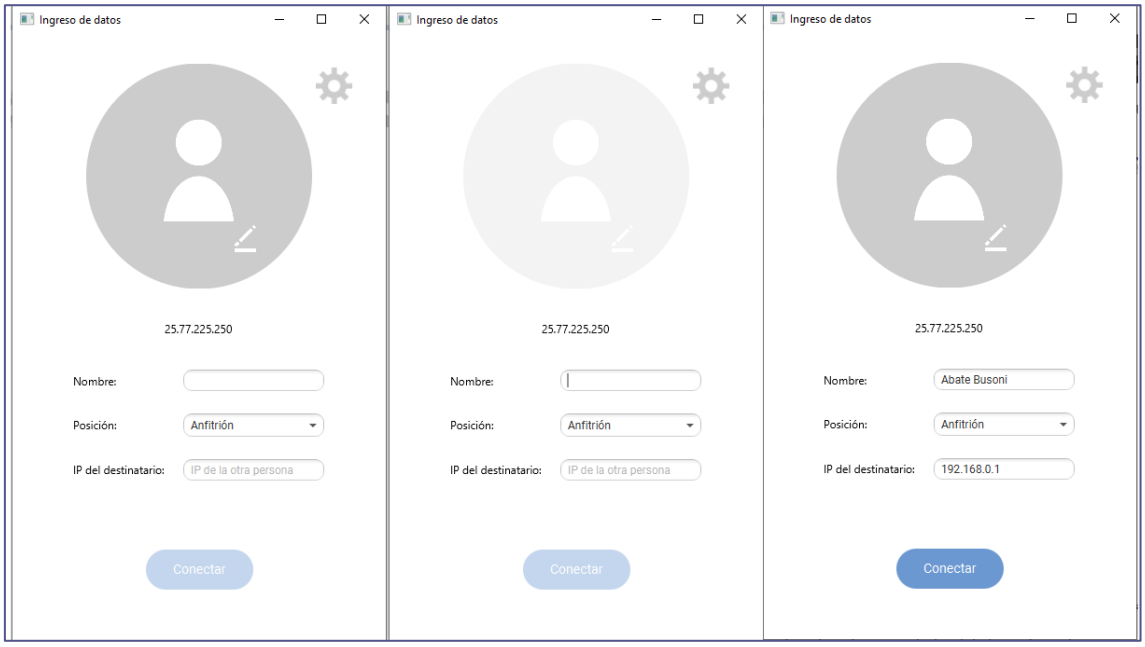

Ilustración 57. De izquierda a derecha: aspecto final de la ventana de inicio, aspecto de la ventana con el ratón encima de la imagen persona, ventana con los datos de Nombre y IP de destino relleno

10.3 Actualización ventana de configuración

En esta ventana se mantiene el lenguaje visual implementado en la ventana de inicio:

- Fondo blanco.
- Tipografía *Roboto Light* con tamaño 12 px para el texto no modificable
- Tipografía *Roboto Regular* con tamaño 12 px para el texto contenido en los recuadros de texto modificables.

Aquí se ha sugerido una redistribución de la posición del texto y los recuadros en la ventana, estando estos más centrados y con una sangría a ambos lados para que “respiren”. A su vez, los botones de aceptar y cancelar se han intercambiado por símbolo visuales de dimensiones 31 x 31 px.

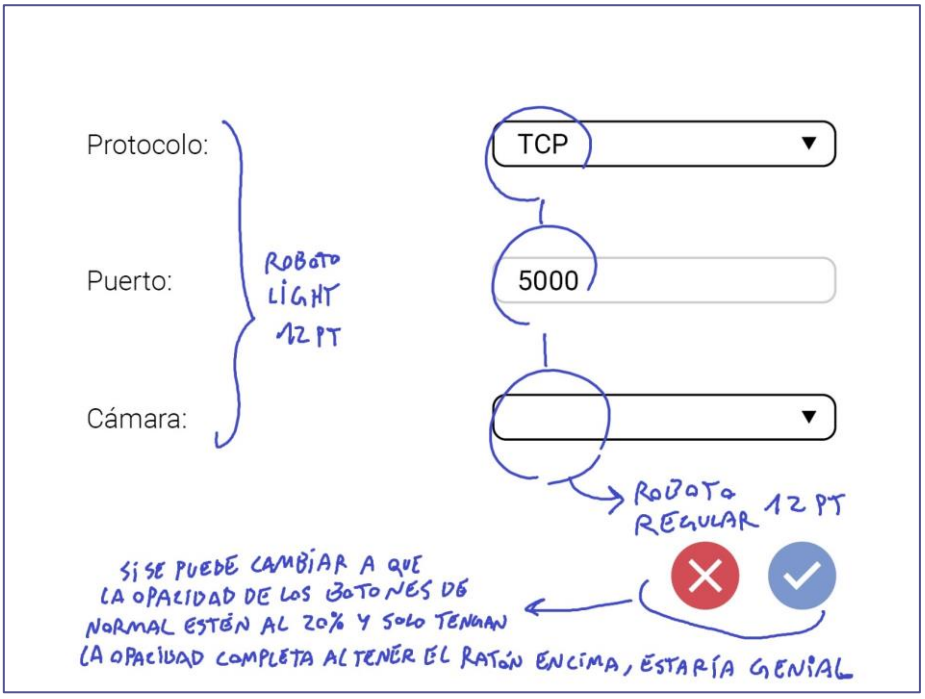

Ilustración 58. Croquis del nuevo diseño de la ventana de configuración con apuntes del diseñador.

Para implementar estos cambios se aprovecharon los estilos ya definidos (*#button-with-text, #background-windows* y *#box-text-fields*) ya que los elementos contenidos en esta ventana son similares.

En este caso el diseñador gráfico agregó una nota sugiriendo que los botones disminuyan su opacidad cuando se sitúe el ratón encima, como ya implementamos anteriormente, sin embargo, sus notas apuntan a una opacidad del 20%. Se realizó la prueba de hacer esto, pero el resultado no era agradable, se perciba demasiado claro, por tanto, se aplicó una opacidad del 50%, tal como ya habíamos hecho en la ventana anterior.

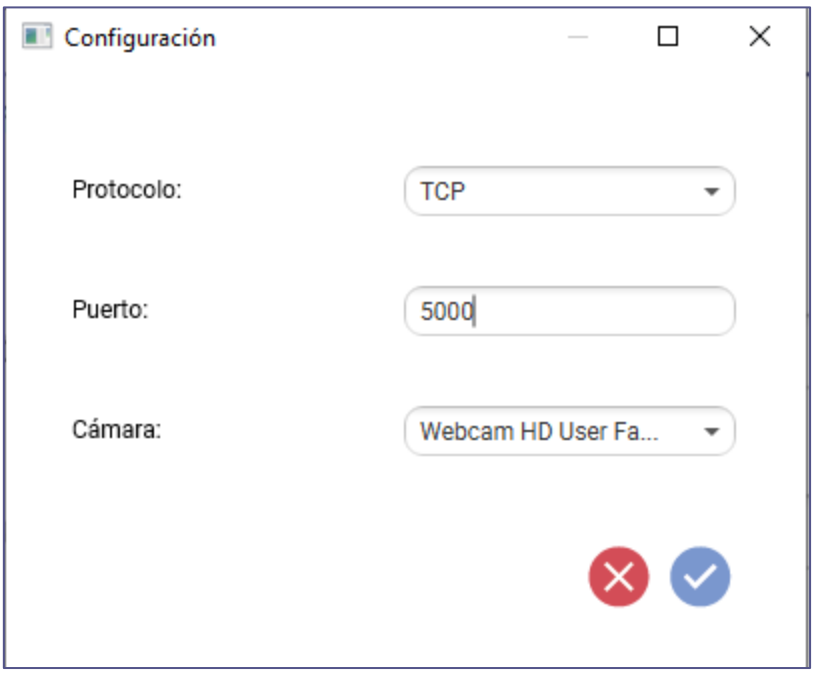

Ilustración 59. Aspecto final de la ventana de configuración

10.4 Actualización ventana principal de chat

La siguiente ventana en rediseñarse es la ventana principal de chat. Según las recomendaciones del diseñador gráfico se nos sugirió realizar los siguientes cambios:

- Cambiar el color de fondo de esta ventana a blanco.
- Intercambiar los botones de audio y cámara por dos íconos más gráficos, estos incluirían los símbolos de micrófono y cámara. El color de estos variaría dependiendo de la situación, azul al estar activados y rojos al estar desactivados.
- En el caso de las imágenes de la cámara, modificarla para darle un aspecto más integrado.
- Los cuadros que contienen el texto del chat presentarán un aspecto limpio con bordes redondeados.
- Tanto el botón de colgar como el de enviar texto deberán presentar siempre una transparencia de 20%, cambiando al 100%al pasar el ratón por encima.

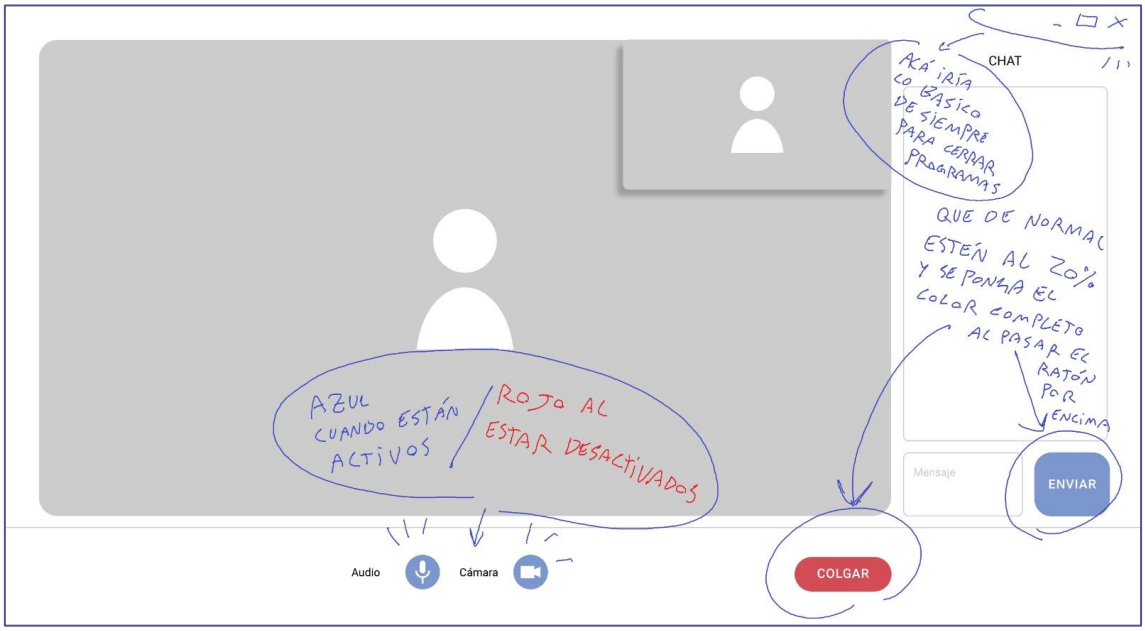

Ilustración 60. Croquis del nuevo diseño de la ventana de chat con apuntes del diseñador

Para mantener la consistencia con las otras ventanas ya modificadas, se consideró mantener el mismo lenguaje visual ya implementado, por tanto, los botones de colgar y enviar mantendrán su opacidad al 100% y solo disminuirá al pasar el ratón por encima.

Para conseguir replicar el aspecto propuesto por el diseñador gráfico comenzamos con lo más sencillo, al ya tener definidos el estilo para las cajas de texto (*#box-text-fields*) lo aplicamos a la caja de registro de chat y la caja donde escribimos mensajes.

El caso de los botones es un tanto diferente a lo visto en las otras ventanas. Aquí el botón de enviar tiene otras dimensiones y el botón colgar es de otro color, por tanto, definimos dos nuevos estilos, el primero de ellos es *#button-with-text-red*, este tendrá una definición muy similar al aplicado al resto de botones con texto, la única diferencia es que el color de fondo pasará a ser *#d34d57*.

Para el aspecto de del botón *Enviar* utilizamos como base el estilo creado para el botón *Colgar*, al que llamaremos *#button-send*. La diferencia radica en dos puntos, el primero de ello está en el tono del color, que pasará a ser el mismo tono de azul que el de los botones del micrófono y la cámara, el segundo de ellos se encuentra en la curva aplicada a los bordes, al presentar unas dimensiones más grandes que el resto de los botones, siendo las de este 84 x 86 px, la curvatura será menos pronunciada para que no quede tan redondeado, teniendo esta un valor de 15.

```
#button-with-text-red{
    -fx-background-color:#d34d57;
    -fx-text-fill: #ffffff;
    -fx-font-family: 'Roboto', sans-serif;
    -fx-font-weight: 500;
    -fx-font-size: 14px;
    -fx-background-radius: 30;
}

#button-sent{
    -fx-background-color:#7d96ce;
    -fx-text-fill: #ffffff;
    -fx-font-family: 'Roboto', sans-serif;
    -fx-font-weight: 500;
    -fx-font-size: 14px;
    -fx-background-radius: 15;
}
```

Ilustración 61. CSS con la definición de #button-with-text-red y #button-send

Para la imagen de nuestra cámara se aplicó el efecto de *InnerShadow* desde la misma interfaz de *Scene Builder*. Aquí definimos una ligera sombra para que separe la imagen de nuestra cámara con la del otro usuario. Esta sombra se ha definido con un grosor de 20, un ligero radio de 9.5, una posición inferior izquierda con las coordenadas (x,y)=(-5, -5), y una coloración grisácea *#979595.*

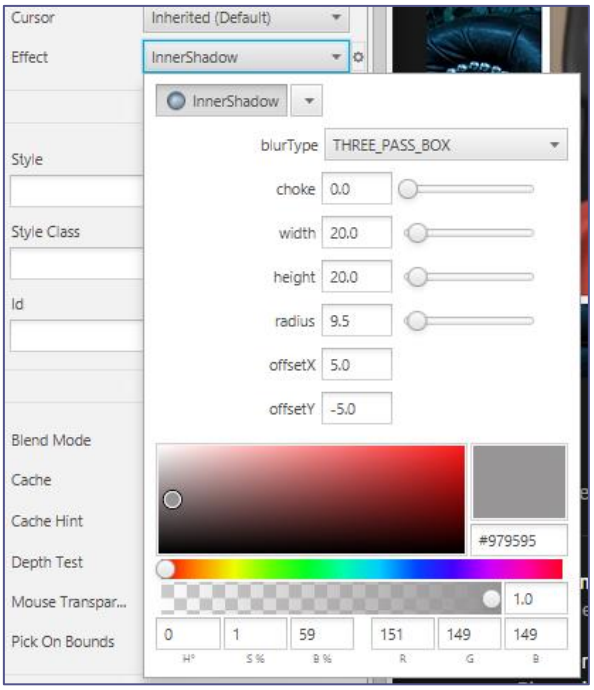

Ilustración 62. Definición de InnerShadow

En lo que respecta a código se realizaron cambios en las funciones *microphoneChangeStat* y *cameraChangeStat*, para que, en base al valor de las variables que indican el estado del micrófono y la cámara, se active o desactiva la visión del botón correspondiente.

```
@FXML
    private void microphoneChangeStat(ActionEvent event) throws LineUnavailableException {
        if(isAudioOpen){
            micOffButton.setVisible(true);
            micOffButton.setVisible(false);
            isAudioOpen = false;
            System.out.println("Micrófono Desactivado");

        }
        else{
           micOffButton.setVisible(false);
           micOffButton.setVisible(true);
           isAudioOpen = true;
           startMyMicrophone();
           System.out.println("Micrófono activado");
        }
    }

    @FXML
    private void cameraChangeStat(ActionEvent event) throws InterruptedException {
        if (isCameraOpen) {
            cameraOffButton.setVisible(true);
            cameraOnButton.setVisible(false);
            isCameraOpen = false;
            defaultCamera.close();
            stopMyCamera();
            System.out.println("Camara Desactivada");

        } else {
            cameraOffButton.setVisible(false);
            cameraOnButton.setVisible(true);
            isCameraOpen = true;
            defaultCamera.open();
            startMyCamera();
            System.out.println("Camara Activada");

        }

    }
```

Ilustración 63. Actualización de microphoneChangeStat y cameraChangeStat

En esta ventana implementamos unas funciones adicionales, las cuales se encargarán de detectar cuando estamos con el ratón encima de un botón para reducir su opacidad o aumentarla.

```
@FXML
  private void setSendOpacityFull(MouseEvent event) {
      sendButton.setOpacity(1);
  }

  @FXML
  private void setSendOpacityLess(MouseEvent event) {
      sendButton.setOpacity(0.5);
  }

  @FXML
  private void setEndOpacityFull(MouseEvent event) {
      endCallButton.setOpacity(1);
  }

  @FXML
  private void setEndOpacityLess(MouseEvent event) {
      endCallButton.setOpacity(0.5);
  }

  @FXML
  private void setMicOpacityFull(MouseEvent event) {
      micOffButton.setOpacity(1);
      micOnButton.setOpacity(1);
  }

  @FXML
  private void setMicOpacityLess(MouseEvent event) {
      micOffButton.setOpacity(0.5);
      micOnButton.setOpacity(0.5);
  }

  @FXML
  private void setCamOpacityFull(MouseEvent event) {
      cameraOffButton.setOpacity(1);
      cameraOnButton.setOpacity(1);
  }

  @FXML
  private void setCamOpacityLess(MouseEvent event) {
      cameraOffButton.setOpacity(0.5);
      cameraOnButton.setOpacity(0.5);
  }
```

Ilustración 64. Funciones para detectar en la ventana de chat si estamos encima o no con el ratón de un botón

Ilustración 65. Aspecto final de la ventana de chat

10.5 Actualización ventana finalizar llamada

La última ventana que queda por modificar es la correspondiente a finalizar llamada. Esta presenta los mismos elementos que vimos en la ventana de configuración, por tanto, su planteamiento será el mismo:

- Colocaremos el estilo *#background-windows,* para definir el color de fondo de la ventana.
- Tipografía *Roboto Light* de tamaño 12 para la frase inicial.
- Tipografía *Roboto Regular* de tamaño 14 en el caso de la pregunta que nos cuestiona si estamos seguros de terminar la llamada.
- Definición de dos botones, uno de confirmación y otro de negación, con las mismas dimensiones y características que ya vimos en la ventana de configuración.

En este caso las dimensiones de la ventana se han modificado para que guarden relación con la ventana de configuración

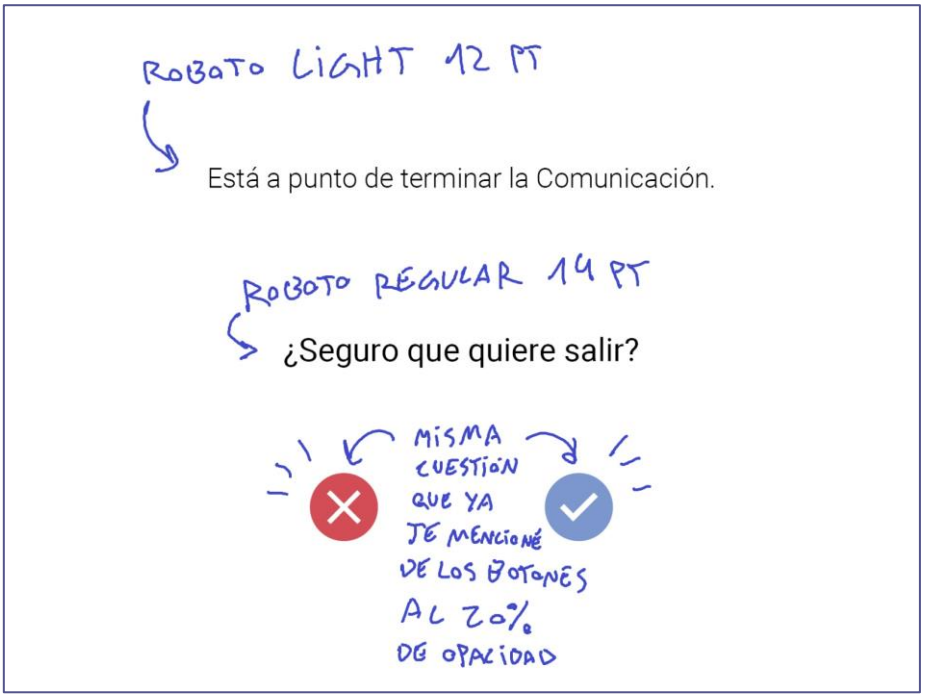

Ilustración 66. Croquis del nuevo diseño de la ventana de finalizar llamada con apuntes del diseñador

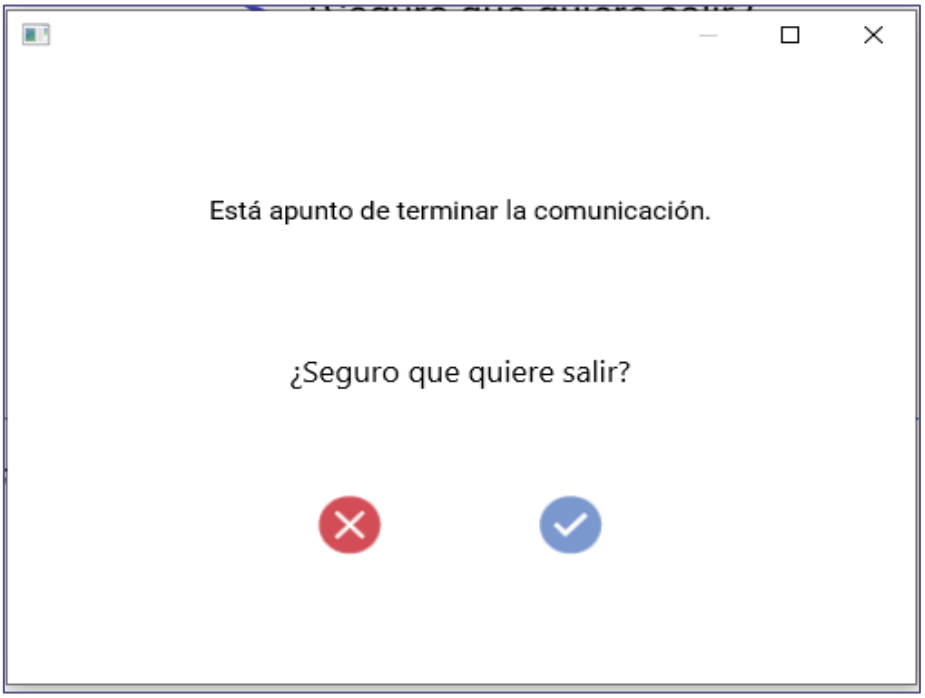

Ilustración 67. Aspecto final de la ventana de finalizar llamada

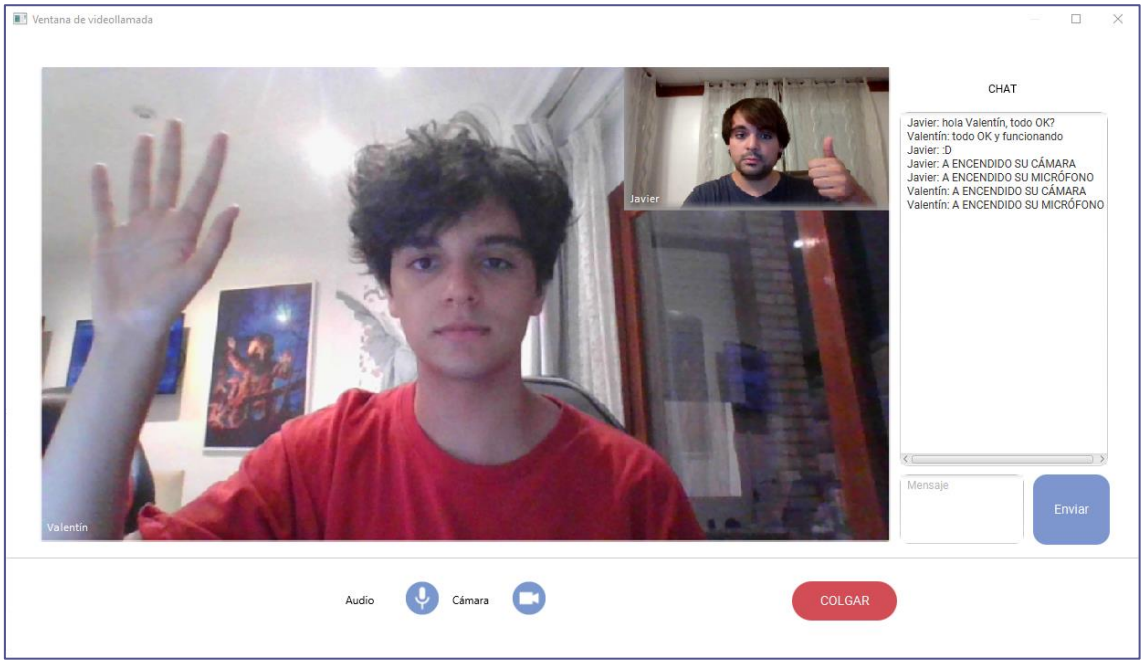

Ilustración 68. Aplicación final en funcionamiento

11. Conclusiones

11.1 Trabajo realizado

A modo de cierre de este trabajo, procederemos a analizar los resultados y datos obtenidos a lo largo de la realización de este proyecto.

La guía principal que seguimos, y los objeticos alcanzados se vieron siempre reflejados dentro del marco planteado en el apartado *1.2 Objetivos,* pero limitándonos, en la medida de lo posible, por lo planteado dentro del apartado *5. Definición de requisitos.*

El resultado de este trabajo de documentación y desarrollo fue la finalización de una aplicación de comunicación multimedia a través de internet con la que podemos comunicar dos ordenadores contenidos dentro de la misma red. Para conseguir el resultado más profesional posible se inició recopilando información sobre las diferentes alternativas existentes en el mercado. En base a ellas, se extrajeron sus características principales y se procedió a construir el perfil del usuario de estas aplicaciones, el posible escenario de uso y el prototipo de la aplicación. Con esto hecho, se finalizó con la construcción funcional de la aplicación, su testeo y una actualización del apartado gráfico de cara al usuario.

En el desarrollo de este proyecto se aplicaron diferentes habilidades y conocimientos adquiridos a lo largo de la carrera, esto se ve reflejado en la implementación del desarrollo funcional de la aplicación desde el punto de vista de la programación, la aplicación de conceptos de paralelismo, el diseño de un escenario de uso, planteamiento del desarrollo de la persona, y el posterior refinamiento del aspecto final del proyecto.

Con estas bases, también se profundizó e investigó más sobre los temas relacionados con la captura y la reproducción de audio, el tratamiento de imágenes digitales y la transmisión de paquetes a través de la red. No solo se han cubierto cuestiones meramente relacionadas con la informática, sino que se ha ido más allá consultado a un diseñador gráfico para cubrir aspectos que sobrepasaban nuestra competencia.

11.2 Resultados obtenidos y problemas encontrados

En este proyecto se realizó el desarrollo de la aplicación totalmente desde la base, desde el planteamiento hasta su implementación.

Desde la perspectiva que se planteó en el apartado 1.2 *Objetivos,* se considera que se ha logrado implementar una versión de la aplicación que reúne las características básicas para demostrar la viabilidad del desarrollo en solitario de esta, con características similares a las otras opciones existentes actualmente. Por supuesto, al decir con características similares, nos referimos siempre a la finalidad de la aplicación, ya que las grandes aplicaciones comerciales tienen un personal especializado en desarrollo y mantenimiento de las distintas partes que componen un programa similar, además de un capital asignado para el desarrollo de estas.

En este caso el desarrollo fue totalmente en solitario, de aquí podemos extraer la primera de las conclusiones: es posible realizar una aplicación funcional en solitario, aunque esto desembocará en tiempos de desarrollo más largos.

A pesar de querer abarcar todos los ámbitos que engloban el desarrollan la aplicación, se terminó por pedir ayuda a un diseñador gráfico para el pulido de la parte estética. Por más que queramos refinar al máximo posible el apartado visual de una aplicación, será muy difícil obtener los mismos resultados que los proporcionados por una persona especializada en el tema.

Uno de los principales objetivos a abarcar, y que supuso algunos problemas a la hora de implementar, fue la captura de imágenes desde la webcam. En principio se intentó desarrollar de manera nativa con las distintas opciones que nos brinda *Java*, pero luego de intentarlo de diferentes maneras, y comprobar que el problema estaba escalando a niveles que iban más allá de los objetivos del presente proyecto, se optó por tomar otra vía. Los resultados que obtuvimos al realizar diferentes búsquedas sobre la obtención de imágenes digitales decantaban casi en su totalidad en el uso de la API *Webcam Capture,* al existir una gran cantidad de información y ejemplos de uso de esta se optó por su uso.

La modificación del código del servidor y cliente nos resultó de gran ayuda a la hora de detectar el desempeño de la aplicación al enviar y recibir paquetes. Gracias a esto hemos podido comprobar fácilmente, por un lado, el correcto envío de información, dado que el número de paquetes enviados, el tipo, y el orden de estos coincidía en ambos extremos, y, por otro lado, también hemos podido averiguar uno de los principales problemas existentes, relacionado con la latencia entre el envío y recepción de los paquetes más grandes, como es en este caso los correspondientes al transporte de imágenes.

11.3 Modificaciones y ampliaciones de cara al futuro

Luego de la finalización de este proyecto, y las consecuentes pruebas realizadas de la aplicación, llegamos a la conclusión que nos encontramos ante un proyecto con gran capacidad de mejora y ampliación. En este apartado desarrollaremos recomendaciones y sugerencias para la mejora de este u otros futuros proyectos relacionados con el tema.

La generación de códigos de comunicación o identificadores diferentes a la dirección IP personal sería una gran mejora respecto al modelo actual, por dos motivos fundamentales:

1. Se facilitaría el uso si el mismo usuario pudiera definir su identificador personal.
2. Desde el punto de vista de la seguridad, el intercambiar las direcciones identificadoras de nuestros ordenadores, aunque sea dentro de una red privada, puede resultar riesgoso dependiendo del canal de comunicación.

Otro punto que se planteó implementar a lo largo del desarrollo del apartado de comunicación del proyecto fue la compatibilidad con el protocolo de comunicación UDP. A pesar de no disponer un control de flujo fiable, ni sistema de recuperación de errores, podría resultar interesante para contextos de comunicación donde lo que prime sea la velocidad y no la fiabilidad.

Atendiendo al envió de paquetes, otra mejora interesante sería la implementación de sistemas de cifrado de datos para aumentar la seguridad de la comunicación.

Otra de las posibles mejoras, de cara a facilitarle el acceso al usuario, sería crear una libreta de direcciones que almacene las credenciales de las personas con las que ya hemos establecido comunicación, en lugar del actual sistema de escribirlas en cada sesión. Esta libreta de direcciones podría ser implementada como una base de datos externa en un servidor independiente o interna en el mismo ordenador.

Desde el punto de vista de la comunicación online, se plantea la posibilidad de adaptar el programa para que los nodos funcionen detrás de NATs.

Por último, una mejora que podría dejar este proyecto a una escala más cercana al resto de aplicaciones comerciales sería la incursión de llamadas grupales y la implementación de envío de archivos.

En conclusión, el estado actual del proyecto es ampliamente escalable: se ha establecido una base funcional sobre la que trabajar para todo aquel que quiera investigar o experimentar con el envío de medios multimedia a través de la red. Al estar cada punto documentado, desde las bases teóricas a su puesta en funcionamiento, su adaptación y mejora no debería resultar un problema para futuros desarrolladores.

12. Bibliografía

[Dje18] Djeek. (28 de enero de 2018). Microsoft Teams and the protocols it uses, OPUS, MNP24, VBSS, ICE and WebRTC. Recuperado el 24 de mayo de 2020, de https://www.djeek.com/2018/01/microsoft-teams-and-the-protocols-it-uses-opus-and-mnp24/

[Dum45] Dumas, A. (1845). Capítulo 1: Marsella. La llegada. En A. Dumas, El Conde de Montecristo (pág. 1). Epublibre.

[Ena22] Ena, D. (s.f.). Cosas de Audio. Recuperado el 20 de agosto de 2022, de Como funciona la compresión MP3: http://cosasdeaudio.com/como-funciona-la-compresion-mp3/

[Gar15] García Álvarz, J. E. (septiembre de 2015). ASÍ FUNCIONA LA CONVERSIÓN ANALÓGICO DIGITAL. Recuperado el 04 de junio de 2020, de http://www.asifunciona.com/electronica/af_conv_ad/conv_ad_5.htm

[Goo] Google. (s.f.). G Suite, Google Meet. Recuperado el 25 de mayo de 2020, de https://gsuite.google.com/intl/es-419/products/meet/

[Goo20] Google. (15 de junio de 2020). Google Trends. Recuperado el 15 de junio de 2020, de https://trends.google.es/trends/explore?cat=32&date=2020-01-01%202020-05-15&q=skype,zoom,%2Fg%2F11csb2gq0p,Google%20Meet

[Ins19] Instituto Nacional de Estadística. (2 de diciembre de 2021). 6.4 Población que usa Internet (en los últimos tres meses). Tipo de actividades realizadas por Internet. Recuperado el 09 de junio de 2021, de Instituto Nacional de Estadística: https://www.ine.es/ss/Satellite?L=es_ES&c=INESeccion_C&cid=1259925528782&p=1254735110672&pagename=ProductosYServicios%2FPYSLayout#:~:text=En%20el%20a%C3%B1o%202021%20en,33%2C1%20millones%20de%20usuarios.

[Jus20] JustinMind. (s.f.). Recuperado el 28 de junio de 2020, de JustinMind: https://www.justinmind.com/

[Man] Enrici, M. C. (2009). MPEG-2. Recuperado el 24 de mayo de 2020, de En M. C. Enrici, La televisión digital - Fundamentos y teorías (págs. 125-134). Barcelona: Marcombo.

[Muñ13] Muñoz Escoí, F., Argente Villaplana, E., Espinosa Miguel, A., Galdámez Saiz, P., García Fornes, A., de Juan Martín, R., & Sendra Roig, J. (2013). Programación concurrente. En Concurrencia y sistemas distribuidos (págs. 1-2). Valencia: Universidad Politecnica de València.

[Och07] Ochoa Domínguez, H., & Mirales, G. J. (3 de julio de 2007). Descripción del nuevo estándar de video H.264 y comparación de su eficiencia de codificación con otros estándares. Recuperado el 10 de Julio de 2020, de http://www.scielo.org.mx/scielo.php?pid=s1405-77432007000300004&script=sci_arttext

[Ora] Oracle. (s.f.). Oracle JavaFX Scene Builder Information. Recuperado el 28 de mayo de 2020, de https://www.oracle.com/java/technologies/javase/javafxscenebuilder-info.html

[Ora1] Oracle. (s.f.). Conozca más sobre la tecnología Java. Recuperado el 28 de mayo de 2020, de https://www.java.com/es/about/

[Ora2] Oracle. (s.f.). Información general sobre JavaFX. Recuperado el 28 de mayo de 2020, de https://www.java.com/es/download/faq/javafx.xml

[Ora213] Oracle. (s.f.). Class AudioFormat. Recuperado el 16 de junio de 2021, de https://docs.oracle.com/javase/7/docs/api/javax/sound/sampled/AudioFormat.html

[Ora22] Oracle. (s.f.). Class SwingFXUtils. Recuperado el 14 de marzo de 2022, de https://docs.oracle.com/javafx/2/api/javafx/embed/swing/SwingFXUtils.html

[Pac] Packetizer, Inc. (s.f.). Why H.323? Recuperado el 23 de mayo de 2020, de https://www.packetizer.com/ipmc/h323/why_h323.html

[Ped11] Pedruelo, M. R. (23 de septiembre de 2011). Proceso de digitalización | | UPV. Valencia, España. Recuperado el 05 de junio de 2020, de https://www.youtube.com/watch?v=3yo7TLtuFqA

[Rod19] Rodríguez, A., & Sagástegui, W. (28 de marzo de 2019). La máquina virtual Java (JVM o Java Virtual Machine). Compilador e intérprete. Bytecode. (CU00611B). Recuperado el 28 de mayo de 2020, de https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=392:la-maquina-virtual-java-jvm-o-java-virtual-machine-compilador-e-interprete-bytecode-cu00611b&catid=68&Itemid=188

[Rub19] Romero, J. (s.f.). Las apps móviles más descargadas en el mundo en junio 2020. Recuperado el 15 de junio de 2021, de https://www.trecebits.com/2020/07/10/las-apps-moviles-mas-descargadas-en-el-mundo-en-junio-2020/

[Sar21] Sarxos. (s.f.). webcam-capture. Recuperado el 13 de marzo de 2021, de GitHub: https://github.com/sarxos/webcam-capture

[Sky] Skype. (s.f.). ¿Qué es Skype? Recuperado el 24 de mayo de 2020, de https://support.skype.com/es/faq/fa6/que-es-skype

[The] The Apache Software Fundation. (s.f.). ¿Qué es NetBeans? Recuperado el 28 de mayo de 2020, de https://netbeans.org/index_es.html

[Ura19] Urarte, E. (21 de octubre de 2019). Los 10 principios de usabilidad de Nielsen. Recuperado el 13 de agosto de 2020, de Welcom to UX: https://welcometoux.com/ux/10-principios-usabilidad-nielsen/

[Wer22] Werner Kvalem Vesterås. (s.f.). Stackoverflow. Recuperado el 14 de marzo de 2022, de https://stackoverflow.com/questions/15053214/converting-an-imageicon-to-a-bufferedimage

[Wik] Wikipedia. (s.f.). Google Meet. Recuperado el 25 de mayo de 2020, de https://en.wikipedia.org/wiki/Google_Meet

[Wik20] Wikipedia. (s.f.). Audio Digital. Recuperado el 13 de julio de 2020, de Wikipedia: https://es.wikipedia.org/wiki/Audio_digitalPCM

[Wik21] Wikipedia. (s.f.). Frecuencia de muestreo. Recuperado el 14 de junio de 2021, de https://es.wikipedia.org/wiki/Frecuencia_de_muestreo

[Wik22] Wikipedia. (s.f.). Protocolo de control de transmisión. Recuperado el 5 de mayo de 2022, de https://es.wikipedia.org/wiki/Protocolo_de_control_de_transmisi%C3%B3n

[Zoo1] Zoom Video Communications, Inc. (s.f.). Introducción al conector de sala H.323/SIP. Recuperado el 23 de mayo de 2020, de https://support.zoom.us/hc/es/articles/201363273-Introducci%C3%B3n-al-conector-de-sala-H-323-SIP

Anexo 1: Textos para las pruebas

Mensaje 1

Mensaje 2

Mensaje 3

Mensaje 4

"El 24 de febrero de 1815, el vigía de Nuestra Señora de la Guarda dio la señal de que se hallaba a la vista el bergantín El Faraón procedente de Esmirna, Trieste y Nápoles. Como suele hacerse en tales casos, salió inmediatamente en su busca un práctico, que pasó por delante del castillo de If y subió a bordo del buque entre la isla de Rión y el cabo Mongión. En un instante, y también como de costumbre, se llenó de curiosos la plataforma del castillo de San Juan, porque en Marsella se daba gran importancia a la llegada de un buque y sobre todo si le sucedía lo que al Faraón, cuyo casco había salido de los astilleros de la antigua Focia y pertenecía a un naviero de la ciudad.

Mientras tanto, el buque seguía avanzando; habiendo pasado felizmente el estrecho producido por alguna erupción volcánica entre las islas de Calasapeigne y de Jaros, dobló la punta de Pomegue hendiendo las olas bajo sus tres gavias, su gran foque y la mesana. Lo hacía con tanta lentitud y tan penosos movimientos, que los curiosos, que por instinto presienten la desgracia, preguntábanse unos a otros qué accidente podía haber sobrevenido al buque. Los más peritos en navegación reconocieron al punto que, de haber sucedido alguna desgracia, no debía de haber sido al buque, puesto que, aun cuando con mucha lentitud, seguía éste avanzando con todas las condiciones de los buques bien gobernados.

En su puesto estaba preparada el ancla, sueltos los cabos del bauprés, y al lado del piloto, que se disponía a hacer que El Faraón enfilase la estrecha boca del puerto de Marsella, hallábase un joven de fisonomía inteligente que, con mirada muy viva, observaba cada uno de los movimientos del buque y repetí a las órdenes del piloto.

Entre los espectadores que se hallaban reunidos en la explanada de San Juan, había uno que parecía más inquieto que los demás y que, no pudiendo contenerse y esperar a que el buque fondeara, saltó a un bote y ordenó que le llevasen al Faraón, al que alcanzó frente al muelle de la Reserva." [Dum45]

Anexo 2: Objetivos de desarrollo sostenible

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	**Alto**	**Medio**	**Bajo**	**No Procede**
ODS 1. **Fin de la pobreza.**				**X**
ODS 2. **Hambre cero.**				**X**
ODS 3. **Salud y bienestar.**				**X**
ODS 4. **Educación de calidad.**	**X**			
ODS 5. **Igualdad de género.**				**X**
ODS 6. **Agua limpia y saneamiento.**				**X**
ODS 7. **Energía asequible y no contaminante.**				**X**
ODS 8. **Trabajo decente y crecimiento económico.**		**X**		
ODS 9. **Industria, innovación e infraestructuras.**	**X**			
ODS 10. **Reducción de las desigualdades**.		**X**		
ODS 11. **Ciudades y comunidades sostenibles**.				**X**
ODS 12. **Producción y consumo responsables.**				**X**
ODS 13. **Acción por el clima.**			**X**	
ODS 14. **Vida submarina.**				**X**
ODS 15. **Vida de ecosistemas terrestres.**				**X**
ODS 16. **Paz, justicia e instituciones sólidas.**				**X**
ODS 17. **Alianzas para lograr objetivos.**				**X**

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

El presente trabajo puede relacionarse con los siguientes objetivos de desarrollo sostenible:

- **Educación de calidad:** como hemos visto con la presente pandemia, muchos centros de estudio tuvieron que adaptar rápidamente su sistema de impartir clase para trasladarlas a plataformas de videoconferencia online, por tanto, el disponer de estas herramientas contribuyó a que los alumnos pudieran seguir con su formación a pesar de no asistir físicamente a clase. Por tanto, el desarrollo y estudio de herramientas como la planteada en este trabajo promueve la posibilidad de recibir una educación de calidad.
- **Trabajo decente y crecimiento económico:** el recibir una educación de calidad es un factor determinante a la hora de encontrar un trabajo decente, por tanto, no puede entenderse un punto sin el otro. Como puede verse en este mismo trabajo, siempre se tuvo en mente el perfil de un trabajador y se buscó como mejorar sus condiciones facilitado el acceso a otra alternativa de comunicación. Además de esto, los beneficios obtenidos por las tecnologías digitales son cada vez mayores, no solo por su venta, sino también por su implantación en los ámbitos laborales, lo cual repercute de manera directa en el crecimiento económico.
- **Industria, innovación e infraestructuras:** el objetivo principal de este trabajo es la construcción de una aplicación de comunicación multimedia a través de la red, que, a día de hoy, representa un medio técnico esencial sobre el que soportar la infraestructura técnica y logística de cualquier empresa. Por ello, considero que podemos encontrar una relación directa entre el trabajo presentado y este ODS.
- **Reducción de las desigualdades:** considero que el presente trabajo está relacionado con este ODS, por potenciar la transformación digital de los trabajos presenciales. El estudio y desarrollo de este tipo de tecnologías de comunicación a distancia facilitan el acceso al empleo a personas con residencias alejadas de sus puestos de trabajo. Pensemos, por ejemplo, en trabajos como telefonistas, oficinistas, programadores, traductores, etc. no es necesario que vivan en la misma ciudad donde está la oficina central, con tener acceso a internet y un ordenador donde ejecutar un programa de comunicación a distancia ya podrían realizar sus trabajos sin desplazarse. Por tanto, se reduce la desigualdad al permitir a diferentes personas acceder a puestos de trabajo a distancia.
- **Acción por el clima:** el reducir los desplazamientos por realizar reuniones o mantener conversaciones a distancia, reduce el consumo de combustible, y, por tanto, repercute directamente en el cambio climático.