UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

EXTENSIÓN DE UN SISTEMA DE APOYO A LA ACTIVIDAD SOCIAL DIGITAL DE ADULTOS MAYORES

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERA CIVIL EN COMPUTACIÓN

LORETO VALENTINA PALMA DONOSO

PROFESOR GUÍA:
SERGIO OCHOA DELORENZI

MIEMBROS DE LA COMISIÓN:
NELSON BALOIAN TATARYAN
RODRIGO ARENAS ANDRADE

SANTIAGO DE CHILE
2022

Resumen

Es ampliamente conocido el hecho de que la población de adultos mayores está aumentando de forma vertiginosa en todo el mundo, principalmente debido a los avances de la medicina. También es sabido que en las últimas dos décadas las interacciones sociales entre las personas han ido cambiando de un escenario cara-a-cara, a uno digital, cada vez más enfocado a interacciones asincrónicas basadas en herramientas de software (por ejemplo, Facebook, Instagram o Whatsapp). En ese escenario los adultos mayores, es decir, inmigrantes o iletrados digitales, llevan la peor parte pues deben afrontar el dilema de adoptar herramientas de software para mantener un vínculo social fluido con su red familiar, o bien, ir alejándose socialmente más y más con el paso del tiempo.

Para intentar paliar esta situación en el DCC se desarrolló el software llamado SocialConnector, el cual abstrae a los adultos mayores de gran parte de los desafíos de usar estas aplicaciones, entregándoles una interfaz simple que ellos pueden usar. A pesar de los buenos resultados reportados por este software, se identificó la necesidad de que el adulto mayor pudiera compartir fotos de manera simple, mientras hacía una videoconferencia con algún amigo o miembro de su familia. Se espera que esta funcionalidad pueda enriquecer y extender las interacciones con el adulto mayor, y así impactar positivamente en su calidad de vida.

La implementación de este servicio representó un desafío importante, no sólo porque éste debía ofrecerse a través de una interfaz simple e intuitiva, sino también porque Android cambió la forma de manejar eventos sobre la interfaz de usuario, lo cual llevó a reestructurar elementos de la interfaz de la aplicación, además de implementar el nuevo servicio. Debido a eso, se extendió tanto el front-end como el back-end del sistema.

La solución implementada permite cumplir con los objetivos inicialmente planteados, y su funcionalidad fue evaluada a través de dos mecanismos distintos. Por un lado, se evaluó la estabilidad de la herramienta, dejándola inactiva durante distintos períodos de tiempo, y retomando el control luego de ellos. Por otra parte, se evaluó la usabilidad del servicio con dos adultos mayores. Los resultados obtenidos muestran que la funcionalidad implementada permite cumplir con los objetivos planteados.

Como parte del trabajo a futuro, se requiere realizar más pruebas con los adultos mayores, para determinar si los resultados preliminares se mantienen con otros miembros de dicha población. También se debe evaluar qué es lo mejor para los adultos mayores, si compartir la pantalla de sus dispositivos o solo compartir fotos o imágenes, tal como fue implementado en este servicio. Hasta no saber eso, no se podrá determinar si el actual servicio necesita ser extendido a futuro.

Agradecimientos

Esta memoria está dedicada a mi mamá, Susana Donoso, por su apoyo incondicional.

En primer lugar, quiero agradecer a mi profesor, Sergio Ochoa, por toda la ayuda que me brindó y por estar constantemente presente en el desarrollo de esta memoria.

Además, hay muchas personas a quienes quiero agradecer, así que empezaré cronológicamente: con mi familia. A mi mamá, porque siempre me ha acompañado y me ha apoyado en todas mis decisiones, siempre preocupándose (a veces mucho). Mi abuela, mi mami, que siempre me ha cuidado y a quien admiro infinitamente. Mi tía Kika, que siempre está presente para una conversación o ver una película (sin dormirse como mi mamá). Mi tío Ponchi, quien siempre está dispuesto a llevarme a donde sea. Y mi tía Beña, que siempre me trae un engañito cuando estoy de visita.

Mis amigos, los del colegio, Javi y Aldo por aguantarme tantos años. También a mis amigos de la universidad, que me han apoyado en momentos de estrés, y también han estado en los tiempos de relajo. Una mención especial a Catalina, mi única e inigualable compañera computina, que hizo que la especialidad fuese una buena experiencia, a pesar de todas las dificultades.

A mi pololo Javier, que me ha acompañado, me ha dado ánimos incondicionalmente y me ha ayudado a dar lo mejor de mí durante este proceso. Sé que siempre puedo contar con él para todo.

Y finalmente, a todos los profesores y compañeros que me han acompañado en este proceso universitario.

Este trabajo de memoria ha sido parcialmente financiado por el Proyecto Fondecyt Nro: 1191516.

Tabla de Contenido

1. Introducción

La comunicación en tiempos actuales se ha diversificado gracias a los avances tecnológicos que se dan con el paso del tiempo. Si bien antes la comunicación interpersonal se basaba en interacción cara a cara o llamadas telefónicas, hoy en día los diversos medios de comunicación han añadido más complejidad a la interacción cotidiana. El compartir imágenes, videos o archivos, es una acción que ya es parte de cómo nos comunicamos actualmente.

Este constante intercambio de información puede ser abrumador para aquellos que no son usuarios ávidos de las tecnologías actuales, como lo son los adultos mayores. Sin embargo, la falta de esta comunicación con esas personas puede generar un aislamiento social de estos últimos. En un mundo en donde compartir elementos visuales es tan importante, entregar un canal de comunicación que facilite la transmisión de contenido de forma fácil para los adultos mayores, es de suma importancia.

Al tratarse específicamente de interacciones sociales con personas de la tercera edad, es importante destacar que típicamente una persona de la familia tendrá el rol de ser un puente entre el adulto mayor y el resto de la familia. A esta persona se la conoce como un *broker* [1]. Este rol es clave para integrar al adulto mayor en el contexto familiar, por lo que facilitar la comunicación, y permitir un intercambio fluido de información entre el broker y el adulto mayor es prioritario. Este intercambio puede ocurrir de diversas maneras, pero nos enfocaremos en una en específico.

Una de las formas de comunicación online que más se asemeja a una conversación cara a cara es la videollamada. Ésta permite tanto hablar, como ver a la otra persona, por lo que es una forma ideal para comunicarse con quienes no acostumbran a usar las tecnologías de hoy en día. Además, el compartir contenidos audiovisuales sigue siendo de suma importancia para la comunicación. Por lo tanto, facilitar este intercambio de información simultáneamente con las videollamadas, afectará positivamente la interacción con los adultos mayores. En particular, permitirá que el *broker* pueda entregar más información al adulto mayor, y de manera más clara, enriqueciendo así la interacción social.

1.1. Problema Abordado

El uso de las redes sociales se ha masificado con el paso de los años, y actualmente es una herramienta vital al momento de comunicarse; existen diversas aplicaciones que permiten mantenernos al tanto de lo que hacen nuestros amigos y familia, y cada vez se introducen nuevas alternativas con este fin. Sin embargo, el uso de estas tecnologías puede resultar confuso para

aquellas personas que no utilizan las redes sociales regularmente. En específico, los adultos mayores pueden tener problemas al enfrentarse al uso de estas aplicaciones para comunicarse con sus pares.

Una de las grandes ventajas del uso de redes sociales es la flexibilidad que éstas presentan, desde el envío de mensajes de texto, hasta el poder compartir lo que estamos haciendo con nuestros amigos, conocidos e incluso desconocidos. Sin embargo, esta abundancia de opciones es un obstáculo para aquellos menos familiarizados con el uso de dichas aplicaciones. Sumado a esto, algunas redes sociales prefieren interfaces estéticamente agradables, pero que no siempre son amigables para todos usuarios.

Para facilitar las interacciones de los adultos mayores con el resto de su comunidad familiar, en el DCC se ha desarrollado un sistema de software llamado SocialConnector [2], el cual permite la comunicación entre un adulto mayor y su familia a través de dispositivos móviles. Los familiares utilizan Telegram desde su lado, mientras que el adulto mayor dispone de la interfaz del SocialConnector en una tablet PC, la cual presenta una interfaz simplificada para facilitar su uso. La Figura 1 muestra la interfaz principal del SocialConnector.

Figura 1. Interfaz del sistema SocialConnector

Como se mencionó anteriormente, una videollamada es una forma de comunicación muy efectiva, especialmente cuando un familiar no vive con el adulto mayor. Además de entregar información verbal, el poder presentar fotos/videos a los adultos mayores o mostrar redes sociales de otros familiares, es una buena forma de mantener al día al adulto mayor con respecto al resto de la familia. Este tipo de interacción actualmente no es posible llevar a cabo en el SocialConnector, por lo tanto, se le dificulta la labor a aquellos que cumplen el rol de *broker*.

Como se puede observar en la Figura 1, el SocialConnector ya tiene la opción de iniciar una videollamada, pero no la opción de compartir la pantalla durante esta actividad. Para llevar a cabo todos sus servicios, el SocialConnector interactúa con Telegram, e implementa interacciones entre los usuarios de dicha aplicación, por lo que se deben tener en cuenta esto al momento de extender el SocialConnector.

Actualmente Telegram soporta las videollamadas y el compartir pantalla, pero aún no existe una forma de participar en una videollamada mientras se comparte pantalla simultáneamente con el SocialConnector. Además, el SocialConnector no tiene las herramientas para recibir una transmisión de pantalla desde Telegram. Este es el principal desafío abordado en este trabajo de memoria.

1.2. Objetivos de la Memoria

La motivación de este trabajo de memoria es expandir la aplicación SocialConnector, para permitir a los usuarios de esta compartir imágenes almacenadas en sus dispositivos, mientras participan en una videollamada. Esta funcionalidad permitirá mejorar la comunicación entre el usuario y sus contactos, afectando positivamente en la vida social de los adultos mayores que utilizan el SocialConnector. Para ello se definió un objetivo general y cuatro objetivos específicos, los cuales se indican a continuación.

1.2.1. Objetivo General

El objetivo general de este trabajo de título es extender el SocialConnector para que facilite el rol del broker, permitiendo que el envío de información entre este y el adulto mayor sea lo más fluido posible. Para ello, a la aplicación se le debe añadir la funcionalidad de compartir pantalla y archivos durante una videollamada. Esta funcionalidad debe estar disponible tanto para el adulto mayor (que es usuario del sistema), como para quienes son sus contactos en el sistema.

1.2.2. Objetivos Específicos

- Definir el protocolo de interacción entre el SocialConnector y la API de Telegram, el cual debe permitir compartir la pantalla de un dispositivo de manera fácil.

- Añadir un servicio para compartir pantalla, que esté accesible a través de la interfaz de usuario del SocialConnector. Dicho servicio debe ser fácil de entender para los usuarios finales.

- Desarrollar una interfaz que sea amigable para los usuarios del SocialConnector, en donde el flujo de la aplicación sea natural para un usuario sin experiencia.

- Generar un producto final estable, que funcione sin complicaciones técnicas y entregue una experiencia positiva al usuario.

1.3. Estructura del Documento

Este documento está estructurado en siete capítulos. En el capítulo 2 se presenta la investigación previa realizada a la creación de la solución, en donde se ahonda en el estado actual de las tecnologías que serán utilizadas para el desarrollo de la solución. En el capítulo 3 se presentan los requisitos, restricciones y la arquitectura del sistema. En el capítulo 4 se presenta en detalle el diseño de la solución propuesta, ahondando en lo que respecta tanto a *frontend* como a *backend*. En el capítulo 5 se presenta la implementación realizada; particularmente, en base a lo expuesto en el capítulo 4, se indica cuál fue la estructura utilizada para solución, y si existieron cambios en base al diseño original. En el capítulo 6 se detalla la evaluación realizada y los resultados obtenidos. El capítulo 7 presenta las conclusiones de esta memoria y el trabajo a futuro.

2. Marco Teórico

El trabajo de memoria comenzó por familiarizarse con el sistema actual del SocialConnector. Como se menciona en la sección 1.1, el sistema permite la comunicación por videollamada sin la posibilidad de transmitir pantalla. Con las herramientas usadas hasta ahora tampoco es posible implementar la funcionalidad, por lo que se pasa al siguiente punto en la planificación: buscar de qué forma realizar una transmisión desde el SocialConnector a Telegram.

Esta investigación fue de suma importancia para el desarrollo de la memoria, pues para introducir una nueva interacción en el SocialConnector, se debe considerar que la aplicación no interactúa con otro cliente de sí misma, sino con un cliente de Telegram, por lo que las alternativas deben ser compatibles con ese flujo de comunicación ya existente. Para esto se deben revisar diversas tecnologías que permitan la transmisión de la pantalla de un dispositivo. Esto se comenta en detalle en la sección 2.2 del documento.

Una vez elegida la tecnología a usar, se identificó la forma de utilizarla en la interacción del SocialConnector con Telegram, cómo realizarla para la conexión, y tener en cuenta su comportamiento en relación a las interacciones ya existentes en el SocialConnector.

2.1. Nuevo Escenario de Interacción del SocialConnector

Como ya se ha mencionado, la interacción en el sistema del SocialConnector implica compartir la pantalla durante las videollamadas. La investigación para determinar la forma de realizar la conexión entre el SocialConnector y Telegram está detallada en la sección 2.3, y como decisión final se optó por usar la API de Telegram directamente para llevar a cabo la nueva interacción.

De esta forma, el SocialConnector se comunica con la API de Telegram para iniciar una interacción de cualquier tipo: enviar mensaje, iniciar una videollamada o iniciar una transmisión de pantalla durante esta, tal como se observa en la Figura 2. La API de Telegram hace las llamadas directamente al servidor de Telegram, permitiendo la conexión con los familiares, quienes por su lado ocupan la aplicación Telegram para Android. Esto es posible gracias a que la API de Telegram permite la creación de clientes de Telegram, y ofrece una variedad de métodos para el manejo de acciones desde estos.

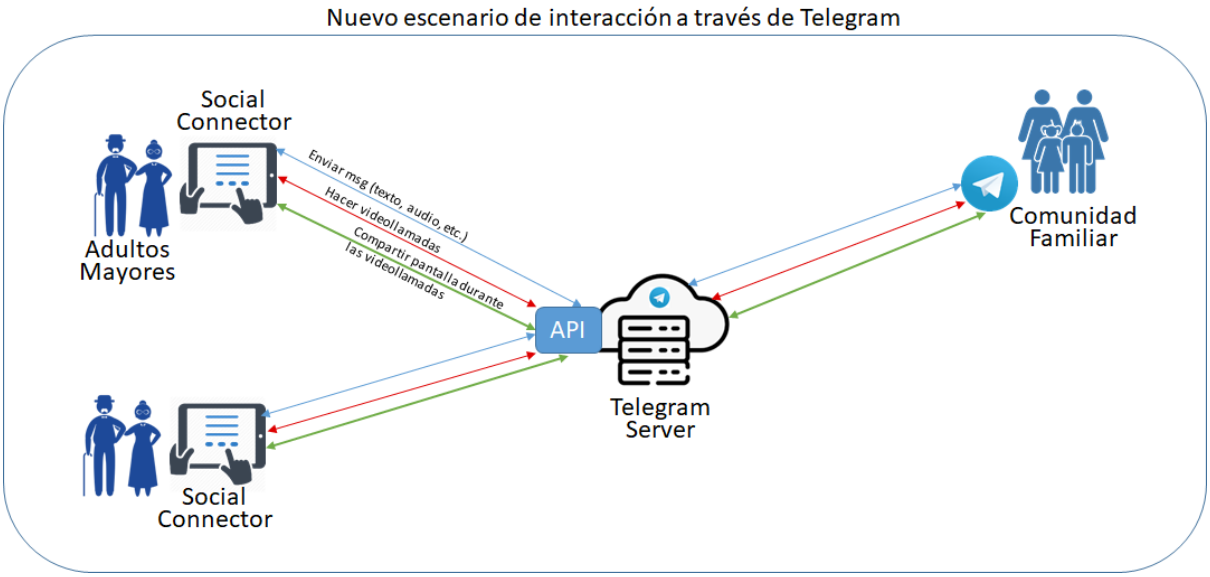

Figura 2. Nuevo escenario de interacción del SocialConnector

De esta forma, el nuevo escenario del SocialConnector incluye una conexión directa al servidor de Telegram a través de la API de Telegram, la cual provee los métodos necesarios para iniciar y terminar una transmisión de pantalla, que se utilizarán para llevar a cabo esas acciones. Esta nueva acción debe tener su correspondiente interfaz de usuario, y, además, se debe poder recibir una transmisión de manera eficiente.

La conexión con Telegram gracias a la API permite que el SocialConnector se comporte como un cliente de Telegram, lo que hace que la interacción sea más directa, permitiendo al *broker* comunicarse a través de la plataforma de Telegram, mientras que el usuario usa el SocialConnector.

2.2. Revisión de Tecnologías

Durante el desarrollo del trabajo, se investigó qué tecnologías están disponibles y permiten añadir la funcionalidad de compartir pantalla a una aplicación de Android. Para este caso, los requisitos de la tecnología a usar es que sea de código abierto y tenga soporte actualmente.

En primer lugar, se revisó el estado actual de la API de Telegram, la cual es de código abierto. Como el SocialConnector se comunica a través de Telegram con los familiares del adulto mayor, se busca si la API tiene disponible métodos que se puedan usar en la solución. Telegram implementó recientemente la funcionalidad de compartir pantalla mientras se está en una videollamada, pero la API aún no tenía los métodos disponibles para su uso al momento del inicio de la investigación.

Otra posible solución encontrada fue la plataforma Sendbird [3], que ofrece APIs que permiten añadir funcionalidades como chat, videollamada y compartir pantalla a una aplicación. Sin embargo, esta plataforma es de uso pagado. Este problema se repite con varias alternativas encontradas durante la investigación; plataformas como Vonage [4] o Agora [5] ofrecen herramientas para implementar la funcionalidad de compartir pantalla y tienen soporte, pero son pagadas, por lo que son descartadas como posibles soluciones.

Una alternativa de código abierto es WebRTC, un proyecto que proporciona comunicación en tiempo real a través de APIs, usando una comunicación entre pares (*peer-to-peer*) y que se especializa en la comunicación de audio y video. El proyecto es apoyado por Apple, Google, Microsoft y Mozilla, y está disponible en navegadores y dispositivos nativos. Considerando todo lo mencionado, WebRTC se posiciona como la mejor alternativa para la solución, por lo que se decide hacer una demostración de prueba para evaluar la factibilidad de esta tecnología.

WebRTC comenzó como un proyecto para la comunicación en tiempo real basada en navegador, siendo el soporte para aplicaciones nativas en Android una adición más reciente. El desarrollo de la solución se enfrenta a un obstáculo debido a esto; la documentación para el uso de WebRTC en navegadores es más extensa que para el caso de Android, lo que dificulta el desarrollo de un ejemplo de uso simple. Si bien el código fuente se encuentra actualizado, la documentación y ejemplos de uso son escasos u obsoletos.

En vista del obstáculo encontrado, se considera la opción de utilizar librerías de código abierto que permitan hacer *screen mirroring*, la tecnología que permite mostrar los contenidos de la pantalla entre dos dispositivos. Otra opción es utilizar las herramientas de Android para implementar una solución propia, empezando desde cero. Ambas alternativas involucran tareas que están fuera del alcance del trabajo considerado inicialmente para la memoria.

Durante la investigación de otras alternativas a WebRTC o documentación actualizada de esta, la API de Telegram, solución investigada inicialmente, es actualizada con los métodos para compartir pantalla durante una videoconferencia. Como se mencionó anteriormente, esta alternativa era la solución más directa considerando como trabaja el SocialConnector actualmente. Teniendo todo esto, se decide utilizar los métodos implementados en la API de Telegram para la solución de esta memoria.

2.3. API de Telegram

Telegram, la aplicación de mensajería que utilizará el *broker* para interactuar con el adulto mayor, tiene disponibles dos APIs para desarrolladores: Bot API y la API de Telegram. Para este trabajo se utilizó la última. La API de Telegram permite crear clientes de tipo Telegram, es decir, crear aplicaciones de Telegram en su plataforma. Estas APIs son de código abierto y permiten compilaciones verificables.

La API de Telegram tiene disponible 396 métodos actualmente, con su documentación correspondiente para su uso. Para el desarrollo de la memoria, por ahora se tiene considerado usar los métodos para compartir pantalla:

- *phone.joinGroupCallPresentation*: para iniciar la transmisión de pantalla durante una videoconferencia.

- *phone.leaveGroupCallPresentation*: para terminar una transmisión de pantalla durante una videoconferencia.

El uso de la API de Telegram debe seguir los términos de servicio determinados por Telegram, y para su uso el desarrollador se debe registrar con una cuenta de Telegram, para obtener una **api_id**. Además, se deben manejar las autenticaciones del usuario a través de la misma API.

De esta forma, la API de Telegram se presenta como una solución directa al problema de cómo realizar la transmisión de pantalla desde el SocialConnector, ya que, como se ha mencionado, los *brokers* de los usuarios estarán usando Telegram desde su lado.

3. Requisitos de la Solución

En este capítulo 3 se presentan los requisitos a considerar para que la solución sea apropiada, y efectivamente resuelva el problema planteado anteriormente, y se cumplan los objetivos ya mencionados en la sección 1.2. Para esto, en primer lugar, se plantea cuál es la funcionalidad mínima esperada de la solución. Luego, se distribuyen los requerimientos en distintos ámbitos.

A continuación, se define la forma que se espera debe tener la arquitectura de la solución, y las restricciones de esta. En las siguientes secciones se delimitan los comportamientos esperados respecto a la usabilidad, disponibilidad y rendimiento del resultado.

Respecto a la usabilidad, se especifican los parámetros que se espera cumpla la aplicación para satisfacer las necesidades del tipo de usuario esperado, es decir, de un adulto mayor. Cuando se habla de disponibilidad, se refiere a delimitar los casos en que se espera que la solución pueda ser usada, y en cuáles no. Mientras que, respecto al rendimiento, se definen los indicadores mínimos para considerar que la solución es eficiente.

3.1. Funcionalidad Mínima

En esta sección se especifica cuál es la funcionalidad mínima esperada para la solución del problema. Esto corresponde a que el usuario del SocialConnector pueda compartir elementos multimedia durante una videollamada. Como elemento multimedia se refiere a la pantalla del computador, videos o imágenes; consideraremos una imagen como el elemento multimedia más básico que se espera poder compartir.

La solución debe incluir la creación de una interfaz que muestre claramente los elementos que están siendo compartidos, mientras simultáneamente se pueda ver al contacto con quien se está realizando la videollamada.

El usuario debe tener la posibilidad de iniciar una transmisión de elementos en cualquier momento durante la videollamada, a través de un botón flotante que se encuentre presente durante esta, de manera que el usuario pueda ver claramente donde está la opción para comenzar una transmisión de pantalla. Esta operación debe poder realizarse como mínimo una vez durante la videollamada.

Además, el usuario debe poder interrumpir la transmisión y volver a la videollamada de forma fácil, y en cualquier momento de la transmisión. En caso de volver, se debe poder volver a la interfaz para compartir elementos.

3.2. Arquitectura de la Solución

Tal como se mencionó antes, el sistema permite compartir pantalla desde el SocialConnector, y también recibir una transmisión de pantalla desde los dispositivos de los contactos del adulto mayor, tal como se muestra en la Figura 3.

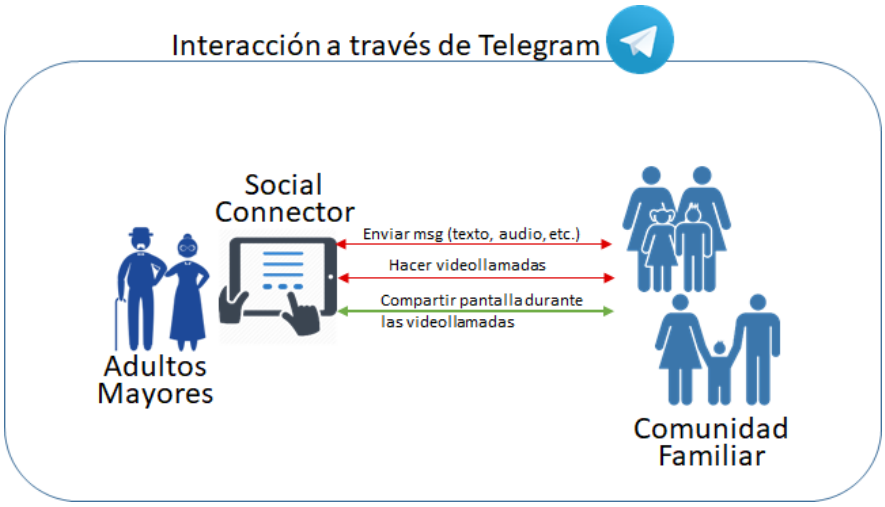

Figura 3. Escenario de operación del SocialConnector

En particular, la información de la pantalla compartida debe poder ser recibida en cualquier dispositivo que tenga Telegram, al igual que en el SocialConnector. Además, se debe tener la opción de compartir pantalla desde la interfaz de la videollamada, mostrando de forma clara la opción para evitar confusión de parte del usuario.

El diseño de la interfaz debe ser intuitivo para los usuarios del SocialConnector, además de ser consistente con la interfaz actual de la aplicación. Debe existir un botón para iniciar la transmisión de pantalla y para detenerla, así como también adaptar la interfaz para mostrar a los participantes de la videollamada mientras se está transmitiendo. Todo esto mientras el usuario interactúa con su dispositivo normalmente.

Para poder brindar el servicio antes mencionado, se debe tener una conexión con la API de Telegram, de modo que la transmisión de pantalla desde el SocialConnector se pueda realizar con los contactos del usuario. Esta interacción entre el sistema y la API debe ser lo más fluida posible.

El SocialConnector usará la API de Telegram [6], la cual es explicada en detalle en la sección 2.3, para compartir la pantalla del usuario, usando el método *joinGroupCallPresentation*, el cual inicia una transmisión de pantalla durante una videollamada. Para terminar la transmisión, se dispone del método *leaveGroupCallPresentation*. Se debe tener en cuenta el manejo de los posibles errores que tienen estos métodos e implementar elementos en la interfaz según sea necesario.

La API de Telegram utiliza WebRTC [6], tanto para las videoconferencias como para la transmisión de pantalla, por lo que primero se deben declarar los parámetros de WebRTC a usar. Estos parámetros están relacionados con el encoding de la transmisión.

Una vez realizada la conexión entre los dispositivos, se debe asegurar que el usuario del SocialConnector pueda navegar en la aplicación mientras comparte su pantalla. Además, debe poder ver claramente al otro participante de la videollamada, y tener la opción de terminar la transmisión de pantalla en cualquier momento.

3.3. Restricciones a la Solución

La solución por desarrollar estará limitada al sistema operativo Android, ya que se utilizará Android Studio para su implementación. En particular, la versión Android 11 es donde se realizan las pruebas de validación.

La aplicación SocialConnector, y en particular la funcionalidad para compartir elementos durante la videollamada, usan tanto la API de Telegram como la aplicación en sí, por lo que ésta debe estar instalada en el dispositivo. En caso de no estarlo, se debe indicar al usuario que la aplicación debe estar instalada previamente en el dispositivo.

3.4. Requisitos de Usabilidad

Como se ha mencionado anteriormente, la aplicación SocialConnector está diseñada específicamente para usuarios que son adultos mayores. Teniendo esto en consideración, uno de los requisitos más importantes es que sea de fácil uso para cualquier persona, en especial aquellos

con menos experiencia en el manejo de aplicaciones móviles. La solución planteada debe ser lo suficientemente entendible para que un usuario pueda realizar una transmisión de pantalla durante la videollamada, sin necesidad de que se le entreguen instrucciones directas de cómo hacerlo.

Cuando se habla de usabilidad, se debe tener en cuenta algunos principios básicos al momento de desarrollar una aplicación. En base a esto, se espera que la solución implementada cumpla con algunos estándares básicos de usabilidad. Estos se basan en principios planteados por diversos autores, tales como Schneiderman [8] y Nielsen [9].

- **Utilidad**: La solución debe satisfacer la necesidad del usuario de poder compartir elementos durante una videollamada, siendo éste un servicio útil para que los adultos mayores puedan socializar con su familia y amigos.

- **Consistencia**: La terminología usada en la aplicación debe ser consistente, para evitar generar confusión en el usuario, y haciendo que el flujo de uso de la aplicación sea fácil de seguir.

- **Simplicidad**: La interfaz debe ser sencilla, de forma que no necesite explicaciones externas para que los usuarios puedan usarla. La información necesaria debe estar expuesta de forma clara y concisa.

- **Comunicación**: Cada acción realizada por el usuario debe tener una respuesta apropiada, ya sea el cambio de interfaz correspondiente, o un aviso en caso de error. Se debe evitar que el usuario se pierda en el flujo de acciones de la solución.

- **Prevención y Manejo de Errores**: Es importante tener en cuenta los posibles errores que pueden cometer los usuarios, y adelantarse a estos. En caso de que se comentan errores, se debe permitir que los usuarios los aborden de manera fácil.

Todas estas variables se deben tener en cuenta al momento del diseño de la aplicación, el cual se presenta en detalle en el capítulo 4 de este documento.

3.5. Rendimiento

Es necesario fijar los parámetros de rendimiento que se esperan para la solución, pues existen diversos tipos de pruebas de rendimiento que se pueden aplicar a un software. Para este caso en específico, las más relevantes son las pruebas de carga y las de estabilidad de la solución.

Las pruebas de carga se centran en observar el comportamiento de la aplicación ante cierta cantidad de peticiones. Durante estas pruebas se registran los tiempos de respuesta para las transacciones de la aplicación. De esta forma, se puede identificar cuáles transacciones funcionan correctamente y cuáles no.

Por otro lado, las pruebas de estabilidad se enfocan en poner a prueba la aplicación bajo diferentes parámetros de ambiente. Se espera testear el software para determinar si éste tiene la capacidad de funcionar de forma estable a lo largo del tiempo.

Si bien es difícil definir parámetros para las pruebas de carga antes de tener bien definidas las transacciones que se espera tener en la aplicación, si podemos fijar algunos parámetros respecto a los tiempos máximos que se espera para un flujo de transacciones completo. Es decir, plantear algunos tiempos óptimos para ciertas acciones que deben estar en la solución. Estos se presentan en la tabla a continuación.

Tabla 1. Tiempos estimados de respuesta para un flujo óptimo.

Transacción	Tiempo Máximo
Pasar de la videollamada a la transmisión de pantalla.	5 seg.
Envío de datos desde el usuario a la persona con la que está haciendo la videollamada.	2 seg.
Salir de la transmisión de pantalla y volver a la videollamada normal.	5 seg.

Estos tiempos son estimados en base al tiempo de lauch que debería tener el iniciar una actividad. Para pasar desde la videollamada a la transmisión de pantalla, se lleva a cabo un *warm start* [10], el cual implica generar la vista de nuevo, aunque el proceso siga estando activo. Para enviar los datos a la persona, primero se realiza un *hot start* [11], pues mientras se está eligiendo la imagen,

la actividad solo pasa a estar en el *background*, y cuando se elige y se comparte, la vista solo pasa a ser visible nuevamente. Finalmente, para volver a la videollamada se está realizando otro *warm star*t, pero en la aplicación de Telegram. Según Android Developers [12], un *hot start* puede tomar 1.5 segundos o más, mientras que un *warm start* debería tomar 2 o más segundos. El iniciar una actividad desde cero, conocido como un *cold start*, toma 5 segundos o más, por lo que se considerará este como tiempo límite para los *warm start*, mientras que se considerará 2 segundos como el tiempo máximo de un *hot start*.

Lo ideal es que el paso de la videollamada a la transmisión no tome mucho tiempo, de manera de evitar que el usuario se sienta perdido y piense que sus acciones no generaron una respuesta en el software. En caso de sobrepasar estos tiempos, es importante mantener al usuario informado sobre lo que está ocurriendo.

Respecto a la estabilidad, se pueden establecer parámetros mínimos para la espera, que va desde que se inicia la videollamada hasta que se comienza la transmisión. Durante este periodo, la aplicación estará en segundo plano, por lo que existe la posibilidad de que el servicio no esté disponible después de cierto tiempo. Se puede plantear un periodo mínimo de disponibilidad de 10 minutos, pero es necesario realizar pruebas para verificar si este periodo es mayor. Además, hay que tener en cuenta el plazo máximo que se puede tener entre el inicio de la videollamada y la primera transmisión de pantalla.

4. Diseño de la Solución

En este capítulo se presenta en detalle el diseño elegido para la aplicación, y también los motivos para estas elecciones. Primero, se provee un poco de contexto respecto al diseño actual del SocialConnector, para luego ahondar en las decisiones tomadas para el diseño de la interfaz de la aplicación, la cual es de suma importancia en el caso de una aplicación para adultos mayores. Luego se profundizará en el diseño a nivel de software, la estructura esperada y los elementos más relevantes de éste.

4.1. Diseño de la Interfaz de Usuario

Tal como se mencionó en el capítulo 3, uno de los requisitos más importantes de la solución es la usabilidad de ésta, ya que el SocialConnector funciona como una herramienta para facilitar la comunicación de adultos mayores con sus pares. Es por esto que se debe poner particular atención al diseño de las interfaces que se presentarán al usuario, es decir, al *frontend* de la aplicación.

Se deben tener en cuenta diversos factores al momento de decidir cómo se mostrarán las interfaces en el SocialConnector. En primer lugar, que éstas efectivamente cumplan con su propósito: *permitir que el usuario pueda compartir pantalla con sus pares durante una videollamada dentro de la aplicación*. Este es un aspecto fundamental de la solución y debe tener una interfaz asociada capaz de lograr eso.

Para poder generar interfaces que cumplan el propósito de esta solución, debemos tener claro cuál es el flujo de acciones que los adultos mayores esperan para llevar a cabo esta acción. A continuación, se presenta el patrón de operación que llevaría a cabo un usuario del SocialConnector para lograr compartir pantalla:

- ➔ Se inicia una videollamada. Esta acción ya se puede realizar en el SocialConnector.
- ➔ Durante la videollamada, el usuario decide compartir pantalla.
- ➔ El usuario comparte su pantalla con la persona con la está en videollamada, sin dejar de verla (en caso de que tengan la cámara activada).
- ➔ El usuario deja de compartir pantalla y vuelve a la videollamada normal.

Estas tareas representan el caso de uso esperado del SocialConnector, y ahora se deben crear interfaces que permitan llevar a cabo cada una de estas acciones. Como referencia, se presenta la interfaz actual del SocialConnector en la Figura 4, para tener en cuenta las decisiones de diseño que están presentes y con las que se deben alinear las interfaces que se proponen en esta sección.

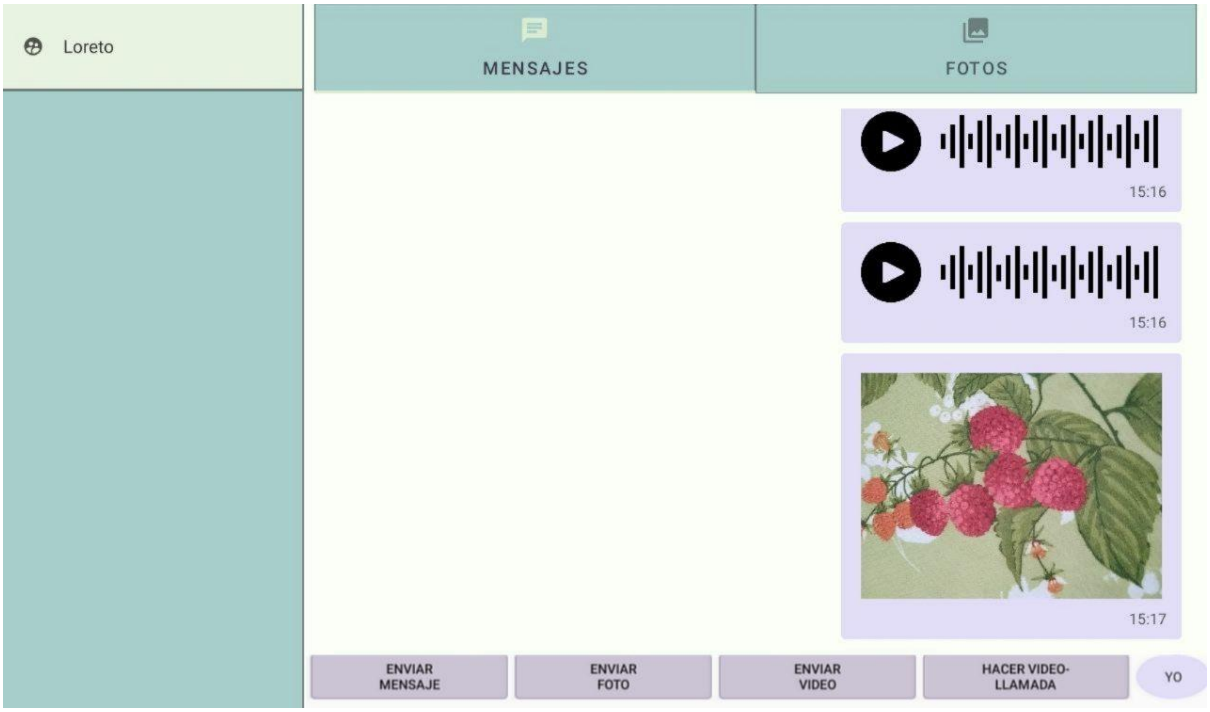

Figura 4: Interfaz actual del SocialConnector.

4.1.1. Iniciar la transmisión de pantalla

En primer lugar, tenemos la operación de compartir pantalla durante la videollamada. Actualmente, el SocialConnector utiliza la interfaz de Telegram para realizar esto, por lo que al momento de realizar la videollamada la persona estará fuera del SocialConnector. Debido a eso, la interfaz utilizada para satisfacer esta acción debe coexistir con la interfaz de la videollamada de Telegram.

Teniendo en cuenta esto, lo ideal es que la interfaz para comenzar a compartir pantalla sea lo más simple y poco intrusiva posible, de manera de evitar que se interfiera con la videollamada. Para esto existen algunas opciones:

- Indicar en una notificación que se puede comenzar a compartir pantalla, y dar la opción de comenzar a compartir dentro de ésta.

- Agregar un botón flotante que esté presente durante la videollamada y que permita iniciar la transmisión de pantalla.

La primera opción es viable, pero si consideramos el tipo de usuario que esperamos tenga el SocialConnector, debemos tener claro que una notificación fácilmente puede ser olvidada al no estar completamente visible en todo momento. Además, una notificación puede ser descartada, ya sea intencional o accidentalmente, lo que eliminaría la opción de compartir pantalla.

Es por esto, que se decide implementar una interfaz como la segunda opción, un botón que esté presente durante la videollamada, sobrepuesto a la interfaz de Telegram. A continuación, se muestra un bosquejo de esta interfaz.

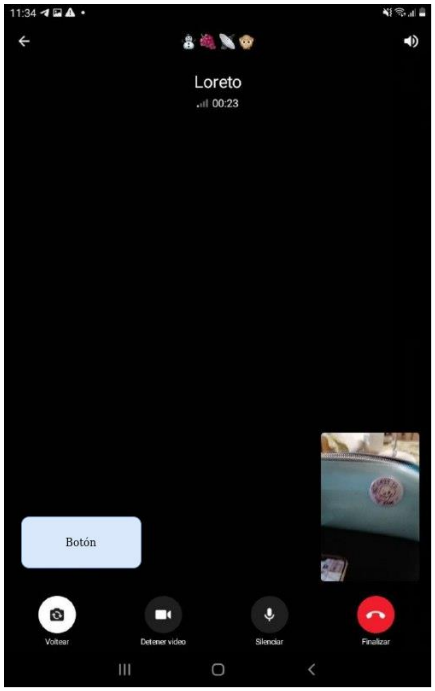

Figura 5: Bosquejo de cómo se presenta la opción de compartir pantalla durante la videollamada.

También se debe determinar la forma que tendrá este botón. La gran mayoría de los botones que ya existen en el SocialConnector no utilizan íconos, sino que usan texto para representar las acciones que gatillan. Para ser consistente con el resto de la aplicación, el botón para compartir pantalla también debe tener texto.

El texto a mostrar debe ser claro y conciso; no queremos saturar de información al usuario, ni tampoco obstruir la videollamada, por lo que es necesario que sea simple. “Compartir Mi Pantalla”

es una frase lo suficientemente corta, y deja en claro cuál es su funcionalidad: *compartir nuestra propia pantalla a otro usuario*. Según esto, el botón debería verse así.

Figura 6: Bosquejo de botón para iniciar la transmisión de pantalla.

4.1.2. Compartiendo pantalla

Teniendo claro cómo luce la interfaz para comenzar a compartir pantalla, se debe diseñar la interfaz para el siguiente paso: estar compartiendo los contenidos de nuestra pantalla con otro usuario durante la videollamada. Esta interfaz sí está dentro del SocialConnector, es decir, el diseño corresponde a una ventana completa de la aplicación.

Nuevamente hay que tener en cuenta los requisitos de usabilidad que se mencionaron en el capítulo anterior, pues esta interfaz es la más prevalente para la solución. También hay que considerar la siguiente acción del usuario (volver a la videollamada) en esta interfaz. Por lo tanto, en esta interfaz se debe incluir tanto el proceso de compartir la pantalla, como el de volver a la videollamada y cerrar esta ventana.

Como se ha mencionado, esta interfaz debe seguir los requerimientos de usabilidad planteados anteriormente, y para lograr esto, se requiere que incluya lo justo y necesario para cumplir las acciones que hemos mencionado. Con esto en cuenta, hay dos acciones importantes que deben estar en la interfaz hasta en su versión más simple: 1) mostrar los contenidos que se comparten, y 2) volver a la videollamada, cerrando la vista donde se transmite la pantalla.

Debemos recordar que el motivo de compartir la pantalla es poder mostrar fotos, videos o elementos para que adulto mayor se comunique con sus pares, es decir, cuando inicie la transmisión, el usuario va a navegar en su pantalla. Por lo tanto, la interfaz principal que verá será esta misma pantalla, mostrando exactamente lo mismo que el usuario hace.

Esto puede ser confuso para un usuario, de modo que hay que comunicarle claramente que se encuentra compartiendo pantalla, durante una videollamada, y no está navegando en el dispositivo

de forma común y corriente. Para dejar claro esto, se puede agregar una notificación flotante que indique claramente esto, sin obstruir en exceso la vista de la pantalla.

También es necesario indicar claramente la forma de dejar de compartir pantalla, en caso de que el usuario quiera volver a la videollamada normal, o haya iniciado la transmisión por error. Usar un color rojo para este botón es buena idea para que visualmente se entienda que este proceso puede ser detenido. Además, se debe ser consistente con el lenguaje utilizado al iniciar la transmisión; si el botón para iniciarla tenía la etiqueta “Compartir mi pantalla”, al momento de terminarla se debe utilizar un lenguaje similar, como, por ejemplo, “Dejar de compartir mi pantalla”.

Finalmente, se debe presentar también el video de la persona con la que se está hablando; éste se puede mostrar como una miniatura al costado de la pantalla. En caso de que la persona no esté utilizando su cámara, se omite esa vista. Considerando todo lo mencionado, la interfaz para compartir pantalla tendría la forma presentada en la Figura 7.

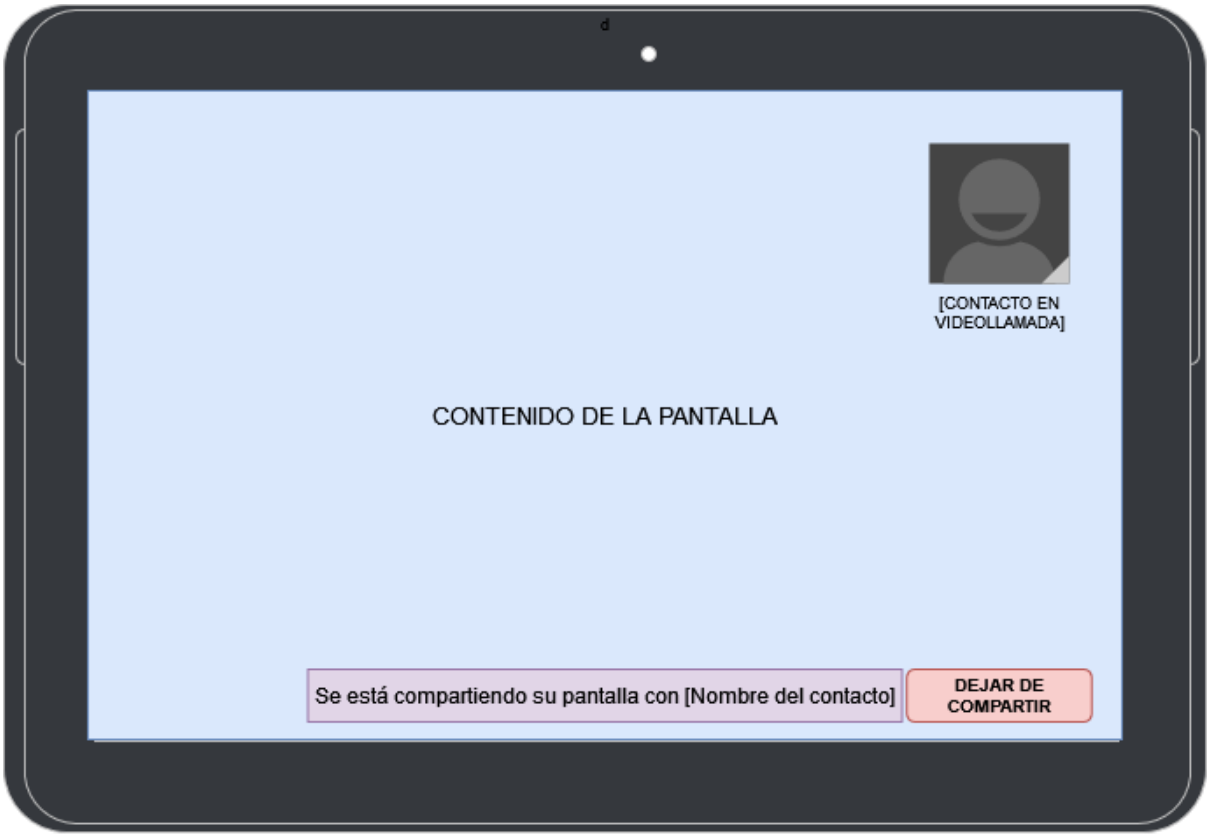

Figura 7: Diseño de interfaz para compartir pantalla.

Esta interfaz es consistente con el diseño del SocialConnector, manteniendo un lenguaje simple, pero entregando toda la información necesaria al usuario para saber qué está sucediendo. Permite al usuario salir en caso de haber cometido un error y, lo más importante, cumple el propósito de que comparta su pantalla.

4.2. Diseño de Software

A continuación, se presenta el diseño del *backend* de la solución. En esta sección, se comentará sobre las decisiones tomadas en el desarrollo del software, y qué características se espera de éste.

En primer lugar, se especifica que el código a desarrollar estará en lenguaje Java, para ser compatible con el desarrollo actual del SocialConnector. Se introducen términos de clases de Java utilizadas para el desarrollo en Android, los cuales serán brevemente explicados. También se provee contexto del diseño que actualmente tiene el SocialConnector.

Se presentan los elementos nuevos que se deben tener para desarrollar la aplicación, y se debe modelar provisionalmente cómo interactúan estos con el *backend* existente en el SocialConnector.

4.2.1. Contexto del Diseño Actual

Actualmente, el SocialConnector ya tiene implementadas diversas funcionalidades como el envío de mensajes de audio, creación de video llamada, y envío de fotos y videos. Para llevar a cabo estas, se han desarrollado actividades, fragmentos y entidades que son relevantes para el diseño de la solución. A continuación, se presentarán los aspectos más relevantes para entender el contexto de donde se encontrará eventualmente la implementación de la solución.

Es importante mencionar que, a grandes rasgos, la arquitectura actual del SocialConnector se asemeja bastante a una arquitectura MVC (Modelo-Vista-Controlador) [13]. Sin embargo, debido a la interacción que existe con Telegram, existen algunas diferencias.

En primer lugar, la pieza posiblemente más importante en la implementación del SocialConnector es la actividad principal, llamada *MainActivity* (*Activity* en la Figura 8), la cual actúa como el controlador de todas las acciones que se realizan en la aplicación.

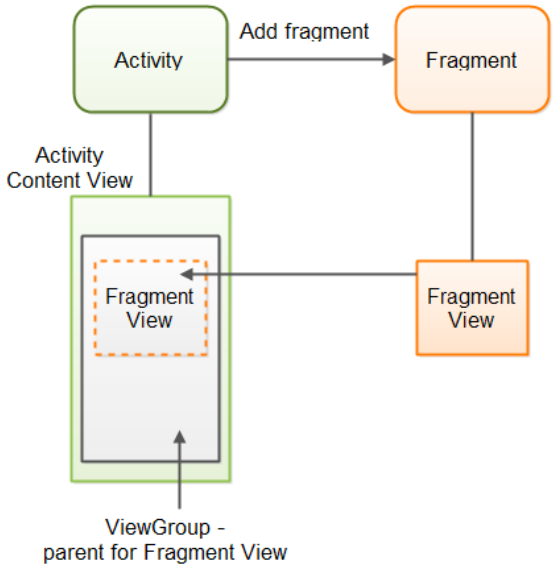

Figura 8: Interacción entre una actividad y un fragmento.

Una actividad de Android [14] corresponde a un componente clave, en donde se presenta una ventana que representa un proceso. En general, la *MainActivity* es la actividad que se ejecuta cuando se inicia la aplicación. En este caso, además se incluye la lógica del SocialConnector, la cual se distribuye en fragmentos.

Un fragmento [15] representa una sección reusable de la vista de la aplicación. En el caso del SocialConnector, los fragmentos se utilizan para manejar el comportamiento de los botones con los que interactúa el usuario. En *MainActivity* se enlaza cada vista con un fragmento; es decir, cada fragmento actúa como un controlador individual para diferentes acciones.

Las vistas [16] de la aplicación se representan con archivos XML, con diseños para distintas acciones que se presentan en la aplicación. Como ya mencionamos, cuando el usuario interactúa con éstas, los fragmentos asociados se encargan de aplicar la lógica para generar una respuesta y actualizar la vista según corresponda. Sin embargo, aún no se menciona cómo se representa el Modelo en la aplicación; esto es porque, las modificaciones que hacen los fragmentos no se hacen a un objeto en la aplicación, sino que se hace un llamado a la API de Telegram para enviar información, y obtener información actualizada.

En base a lo comentado, la interacción entre los componentes del SocialConnector podría plantearse como lo que está en la figura 9. Ahora que ya se conoce cómo se comporta la aplicación

actualmente, por lo tanto, se puede ahondar en el diseño de las nuevas funcionalidades, y en cómo se adaptarán a la arquitectura actual del SocialConnector.

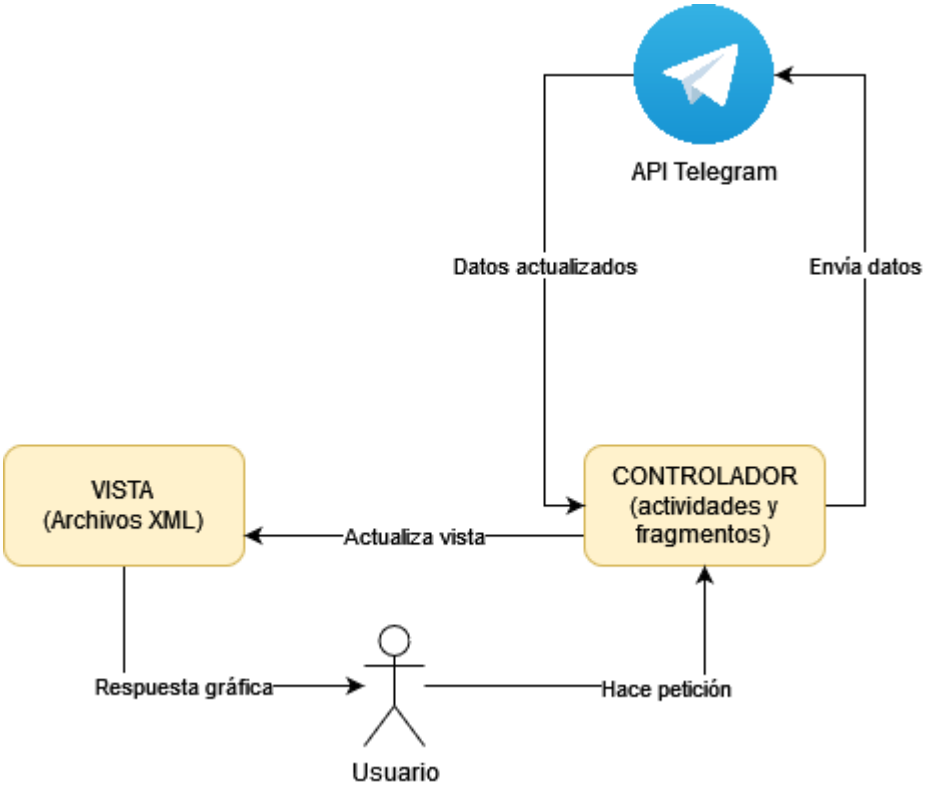

Figura 9: Arquitectura actual del SocialConnector.

4.2.2. Iniciar la transmisión de la pantalla

En primer lugar, se necesita representar el botón para iniciar la videollamada. Éste debe estar asociado a su propia vista con la cual interactúa con el usuario. Si existe una interacción, las acciones a realizar deberán estar contenidas dentro de una actividad para manejar este botón. Una actividad se define como una cosa en particular que el usuario puede hacer. Todo esto, conectado con la actividad principal de la aplicación. Sin embargo, se debe tener en cuenta que esté botón estará presente cuando la aplicación esté en segundo plano, por lo que existen dos opciones: crear una tarea en segundo plano [17] para éste, o bien, usar un servicio [18].

El servicio tiene el potencial de ser más estable, por lo que se considerará como la primera opción para llevar a cabo la solución. Debemos recordar que esta opción debe estar disponible por el mayor tiempo posible, por lo que un servicio es una alternativa sólida para esta tarea. Teniendo en cuenta todo lo mencionado, el esquema para este flujo debería ser de la forma presentada en la Figura 10.

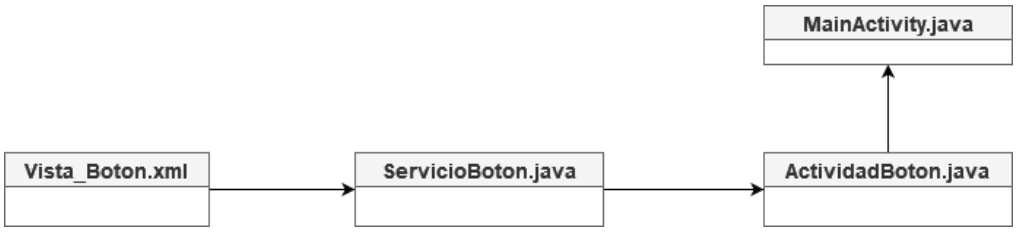

Figura 10: Diseño del *backend* para comenzar a compartir pantalla.

Este diseño es bastante genérico, pero representa de manera simple la dinámica que hay entre estos componentes. *Vista_Boton* es la vista con la que interactúa el usuario, y el servicio *ServicioBoton* maneja las respuestas a estas interacciones. Como se mencionó anteriormente, el servicio permite manejar operaciones en segundo plano, por lo que es esencial para la solución.

El servicio es creado por *ActividadBoton* al momento de iniciar la videollamada, pues debe pasar a segundo plano. A su vez *ActividadBoton*, es creado por la actividad principal *MainActivity* al momento de iniciar la videollamada.

4.2.2. Compartir pantalla

Nuevamente, se debe revisar cuáles son las acciones que se toman al momento de compartir pantalla, y diseñar el *backend* acorde a lo que se necesite:

1. La aplicación debe estar disponible mientras el usuario navega por el dispositivo, usando otras aplicaciones o el mismo SocialConnector.
2. Se puede volver a la videollamada de forma normal.
3. Se debe tener información visible sobre el contacto con quien se realiza la videollamada.
4. Se debe capturar y compartir el contenido de la pantalla.

Considerando el primer punto, es claro que, nuevamente, se debe usar un servicio para mantener a la aplicación activa en el fondo. El servicio puede encargarse de mostrar, tanto el botón para volver a la videollamada, como la información sobre el contacto, pues ambos deben estar presentes mientras el usuario se mueve en el dispositivo.

Dada la complejidad que puede tener cada tarea por separada, es preferible modularizar la aplicación y usar fragmentos para programar el comportamiento de cada tarea. Es decir, hacer un fragmento para manejar el comportamiento del botón y otro para mostrar la información, tal cómo se implementa actualmente en el SocialConnector.

Como se mencionó en el capítulo 2, en base a la investigación previa, para compartir la pantalla se utilizará el método *joinGroupCallPresentation* de la API de Telegram. Este método permitirá que la transmisión de pantalla se realice por la plataforma de Telegram; es por eso que el manejar la miniatura de la videollamada no está considerado dentro de las tareas a implementar, pues lo maneja Telegram directamente.

Pese a esto, de todas formas se debe crear una actividad que maneje la llamada a este método de la API, tanto como para iniciar la transmisión, como para terminarla. Estos casos se pueden manejar en fragmentos que se encarguen de la tarea de comenzar o terminar de compartir pantalla, pero es importante tener una actividad aparte de *MainActivity*. Esto se debe a que cuando se inicia la videollamada, no se quiere volver a la vista principal (con los contactos y el chat), sino que a una vista aparte que se usa al momento de transmitir pantalla.

En resumen, para agregar la funcionalidad de compartir pantalla es necesario crear una actividad nueva que se encargue de manejar la lógica de las vistas, usando al menos dos fragmentos para manejar el mensaje de información sobre la videollamada, y el retorno a ésta. Además, se debe crear un servicio que esté en segundo plano esperando el inicio de la transmisión. El proceso final es tal como se presenta en la Figura 11.

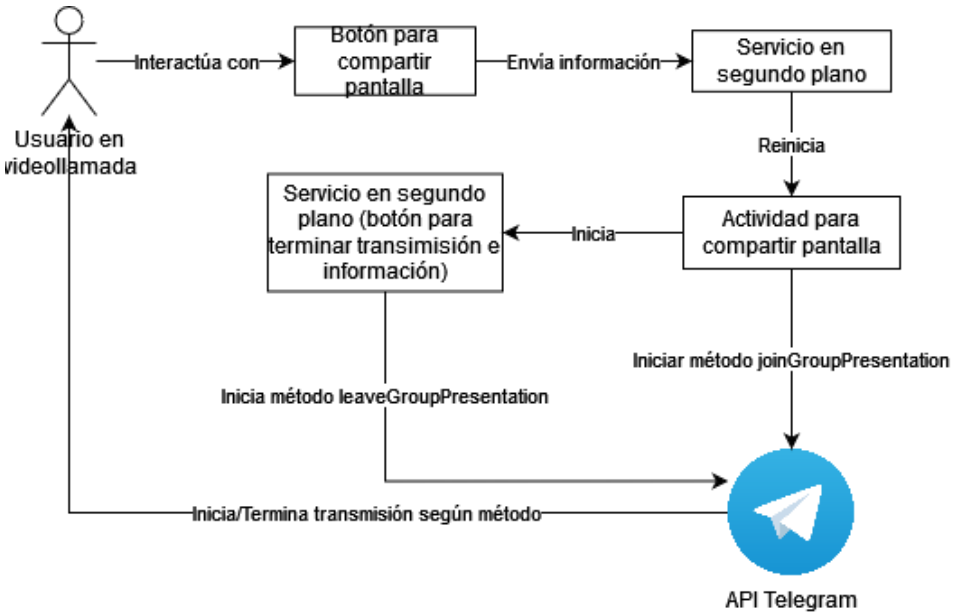

Figura 11: Proceso para compartir pantalla.

5. Implementación de la Solución

En este capítulo, se detalla la implementación final de la solución al problema planteado. Se comentan los cambios que hay respecto a lo planteado en el diseño previo presentando en el capítulo anterior, y las dificultades que se surgieron durante el desarrollo. Finalmente, se presenta el producto final.

5.1. Dificultades presentadas

Antes de comenzar a describir el proceso que se llevó a cabo para la implementación de la solución, se comentarán algunas dificultades que se presentaron al momento del desarrollo, lo que significó cambios en lo presentado en el capítulo anterior. Estas dificultades significaron cambios, tanto en el diseño de la solución, como en algunas interacciones que ya estaban presentes en el SocialConnector, las cuales se detallarán a continuación.

Al iniciar la implementación de la solución, surgieron errores para iniciar la videollamada en el SocialConnector. Este problema tenía que solucionarse de forma urgente, pues la opción para compartir pantalla es durante la videollamada. El problema estaba en la forma en que se iniciaba la videollamada usando Telegram; no se hacía uso de la API de Telegram, sino que se utilizaba un *intent* [19] para esto. Un *intent* en Java es una descripción de una operación para realizar, en general utilizado para iniciar una actividad.

El problema se presentó porque en la versión de Android donde se desarrolló la solución, Android 11, el *intent* para iniciar una videollamada en Telegram no funcionaba correctamente. En versiones anteriores de Android, el *intent* utilizado representaba la operación de iniciar la videollamada usando la opción desde la aplicación de contactos del dispositivo. Esta opción estaba disponible al sincronizar los contactos de Telegram con el aparato, junto con la opción de enviar un mensaje e iniciar una llamada de voz. Sin embargo, en Android 11, la opción para realizar llamadas (de voz o video) no estaban disponibles desde la aplicación de contactos, por lo que el *intent* que se estaba utilizando era inválido.

Se consideraron varias opciones para solucionar este problema; por ejemplo, usar la API de Telegram para realizar la videollamada, pero significaba crear una interfaz nueva para esto, pues la API solo provee los parámetros con la información necesaria, pero no inicia una videollamada en la aplicación. También se consideró utilizar servicios de accesibilidad [20], pero Telegram no disponía de estos más allá de herramientas para usuarios con problemas de visibilidad.

Finalmente, se utilizó el *deep link* [21] que utiliza Telegram para iniciar una videollamada; de esta forma se accede a la URI (Identificador de recursos uniforme) que accede directamente a Telegram. Esta opción tiene una desventaja pues se pide una confirmación dentro de la interfaz de Telegram antes de iniciar la videollamada, como se muestra en la Figura 12, pero debido a las complicaciones que presentan las otras opciones, usar el *deep link* es la opción más directa para solucionar el problema.

Figura 12: Confirmación para iniciar videollamada en Telegram.

Otro problema que se presentó fue el uso de la API de Telegram, para poder realizar la transmisión de pantalla. El método *phone.joinGroupCallPresentation* resultó presentar complicaciones al momento de ser implementado, debido a falta de documentación. Si bien la API presenta documentación de los parámetros de cada método, ciertos parámetros que debían ser generados no tenían ninguna documentación al respecto; tampoco existían ejemplos de uso para métodos relacionados a las llamadas. Luego de una extensa investigación sobre el tema, se optó por simplificar la solución a implementar.

Como consecuencia, la solución presentada no transmitirá la pantalla del usuario, pero permitirá que éste pueda compartir fotos durante la videollamada, y recibir fotos que envíe el *broker* a través de una interfaz que muestre las fotos en la pantalla, manteniendo la imagen de la videollamada en miniatura. De esta forma, la comunicación entre el adulto mayor y el *broker* durante la llamada es mejorada, pues se pueden compartir elementos visuales, aunque solo se limite a imágenes. Teniendo en cuenta los cambios al diseño original, presentamos la implementación de la solución en las siguientes secciones, especificando los cambios al diseño original.

5.2. Tecnologías utilizadas

La solución se desarrolló usando el lenguaje Java, para mantener la consistencia con el desarrollo ya realizado en el SocialConnector. Principalmente, se utiliza la librería de Android y métodos de la API de Telegram.

Respecto a la versión de Android, la versión Android 11 es utilizada para probar la aplicación, por lo que pueden existir discrepancias en el comportamiento del SocialConnector en un dispositivo con Android 12.

5.3. Botón para compartir imágenes

En primer lugar, es necesario implementar el botón para compartir imágenes durante la videollamada. A pesar del cambio respecto al diseño original de la solución, la funcionalidad que concierne a este botón no presenta mayores cambios a lo esperado, solo en detalles del diseño como el texto que se muestra al usuario.

En primer lugar, se debe crear la vista que se utilizará para este botón. Ya que la funcionalidad de compartir pantalla pasó a ser compartir imágenes; el texto que se muestra pasa a ser "compartir foto". A diferencia de compartir pantalla, término que puede resultar confuso si un usuario no sabe a qué se refiere, el compartir una foto es un término fácil de comprender, incluso para un adulto mayor.

Una vez generada la vista, es necesario implementar la arquitectura de software que maneja la lógica de esta operación. Cuando se inicia la videollamada, se inicia la actividad *PictureSharingActivity*, encargada de manejar toda la lógica necesaria para compartir imágenes durante ésta. Dentro de la actividad, inmediatamente se inicia el servicio que muestra el botón sobrepuesto a la videollamada, y permite volver a la actividad cuando sea necesario comenzar a compartir. Cuando se inicia *SharingService* se le envía la información del contacto, la hora actual (inicio de la videollamada) y un *boolean* que indica si la videollamada está iniciando o no.

A diferencia de lo planteado en el diseño original, se utiliza solo una actividad que maneja tanto las interacciones con el botón, como aquellas con la vista que se usa para compartir imágenes, y no dos actividades para cada acción. En gran parte se debe a cómo se estructuró el botón para iniciar la transmisión, pues la lógica de este se maneja en el servicio que tiene asociado.

El servicio *SharingService* se encarga de generar la vista del botón, posicionarlo en la pantalla, y especificar la lógica a seguir cuando se interactúa con éste. La posición del botón se fija en la parte inferior-izquierda de la pantalla, evitando que la vista oculte los botones que están en la videollamada de Telegram (Fig. 13).

Durante la videollamada, este servicio solo se preocupa de responder en caso de que el botón sea utilizado. Si esto ocurre, se utiliza la función *onClick* [22] de Android, en la cual definimos que al momento de hacer click en el botón, esta vista se cierra y se inicia la actividad *PictureSharingActivity* a través de *intent*, y además se detiene el servicio.

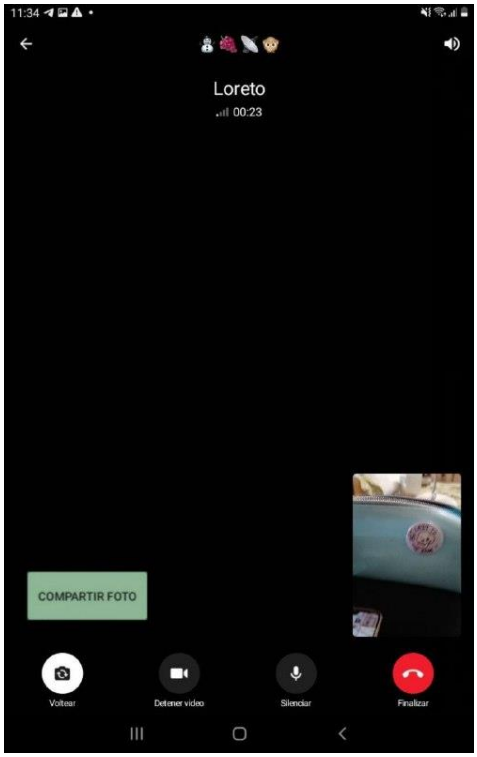

Figura 13: Vista del botón para compartir imágenes durante una videollamada.

Algo muy importante que debe ser mencionado es cómo se logra que la vista esté superpuesta a la videollamada. Para esto, se utiliza el parámetro de ventana *TYPE_APPLICATION_OVERLAY* [23], que permite que las ventanas se muestren sobre cualquier otra actividad. Usar este parámetro requiere también el permiso *SYSTEM_ALERT_WINDOW* [24], el cual debe ser otorgado por el usuario para poder situar el botón sobre otras actividades.

Una vez realizado todo esto, el botón para "Compartir foto" estará disponible durante la videollamada, listo para iniciar que el usuario envíe y transmita imágenes con el *broker* mientras se comunican por videoconferencia, tal como se observa en la Figura 13.

5.4. Compartiendo imágenes durante la videollamada

A continuación, se presenta en detalle la implementación realizada para compartir imágenes durante la videollamada. Nuevamente, se incluye tanto la implementación de interfaces como la de arquitectura del software.

Para transmitir fotos durante la videollamada, se crea una interfaz específica, la cual tiene tres componentes: 1) un botón para compartir una foto, 2) un botón para detener la transmisión, y 3) una vista para las imágenes a medida que se van enviando, tal como se observa en la Figura 14.

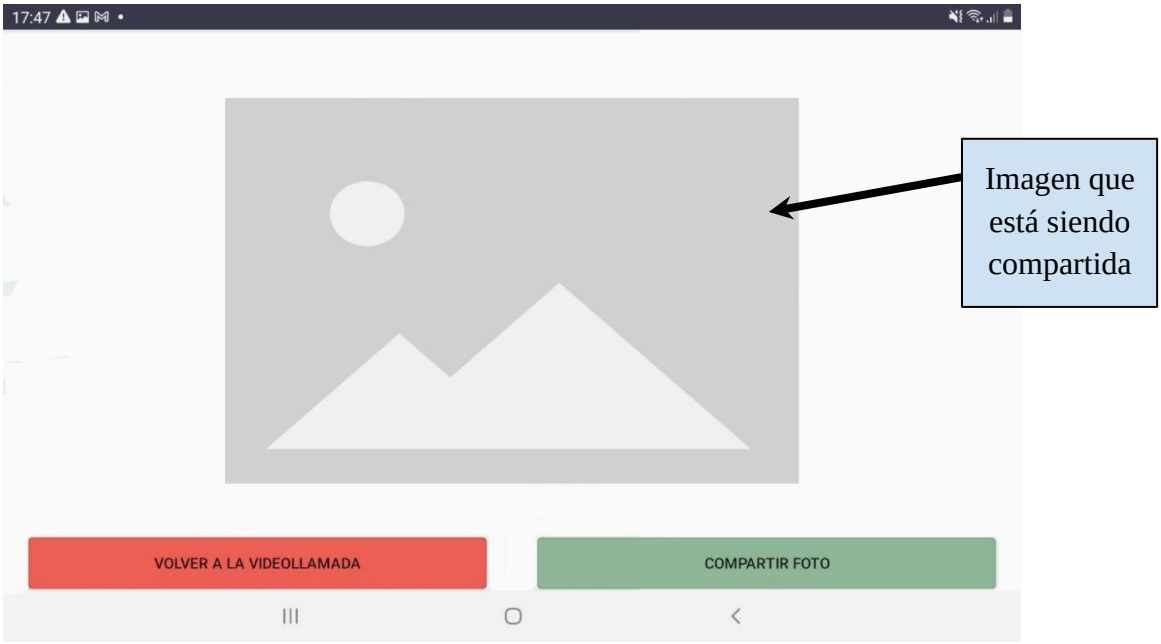

Figura 14: Interfaz de para compartir pantalla.

La interfaz se crea al momento de comenzar la actividad *PictureSharingActivity*. Al iniciar, también se establece el comportamiento de los dos botones, usando el método de Java *setOnClickListener*. Este método permite definir la función *onClick*, con el comportamiento al hacer click en el botón. El botón con la implementación para detener la transmisión y volver a la videollamada es el más simple, y se define la función onClick como un *intent* a la actividad de Telegram, activando nuevamente el servicio con el botón para compartir imágenes.

Por otro lado, el método *onClick* correspondiente al botón para compartir una foto inicia un fragmento *GalleryFragment*, el cual lanza una actividad que permite escoger una foto. Si existe más de una aplicación para visualizar fotos, el usuario deberá escoger una para abrir la galería de imágenes. Una vez en la galería, el usuario podrá escoger una foto y enviarla a través de Telegram.

La comunicación con la API de Telegram es fundamental para la implementación de la solución. Como mencionamos en el capítulo anterior, si se compara la implementación de esta solución a un patrón de arquitectura de software, el patrón Modelo-Vista-Controlador representa la arquitectura más similar. En ese caso, la comunicación entre el SocialConnector y la API de Telegram, corresponde a la comunicación con el modelo de datos. Sobre este modelo, hay 2 clases importantes que corresponden a los datos que se envían y obtienen desde la API.

La clase *Contact* representa, tal y cómo lo dice su nombre, a un contacto de Telegram. Cada contacto tiene los siguientes parámetros:

- ***uid***: identificación del contacto, la cual se obtiene directamente desde Telegram.
- ***chatId***: identifica al chat que existe entre el contacto con el usuario.
- ***name***: nombre del contacto.
- ***phone***: número de teléfono del contacto.
- ***photo***: foto de perfil en Telegram.

La segunda clase es *Message,* que representa un mensaje de Telegram, y contiene otros parámetros que se utilizan para representar información del mensaje. En total la clase Message tiene 15 parámetros, pero a continuación se mencionan algunos de los más relevantes para la implementación de la solución.

- ***uid***: identifica al usuario (contacto) que envía el mensaje.
- ***chatId***: identifica el chat al que pertenece el mensaje.
- ***content***: contenido del mensaje.
- ***date***: fecha del mensaje.
- ***fileType***: entero que corresponde al tipo de mensaje, donde 1 es audio, 2 es foto y 3 es video. Hay que recordar que el SocialConnector solo utiliza estos 3 tipos de mensajes.
- ***isOutgoing***: indica si el mensaje está siendo enviado (con 1) o si está siendo recibido (con 0).
- ***isDownloaded***: indica si el contenido del mensaje está descargado.
- ***newMessage***: 1 si el mensaje es nuevo, 0 si no lo es.

El fragmento *GalleryFragment* obtiene la instancia actual de *TelegramService* en la aplicación. *TelegramService* es una clase que implementa el cliente de Telegram, y que contiene métodos para comunicarse con la API. En particular, se utiliza el método de la API *SendMessage* [25], para el caso específico en que se envía una imagen. Una vez enviada la foto, se retorna a la vista donde se comparten las fotos; luego de eso, se debería mostrar la imagen que se compartió.

Cuando se llama al método *SendMessage*, se envía el parámetro *chatId* y la información sobre la imagen (dimensiones y ubicación). Una vez realizado esto, *GalleryFragment* retorna a la vista de donde se comparten las fotos, y donde el fragmento *PictureFragment* toma relevancia.

PictureFragment es el fragmento que muestra las imágenes al usuario, y se actualiza constantemente para mostrar la imagen más reciente que esté en el chat, de manera que se compartan imágenes mutuamente entre el adulto mayor y el *broker*. La actual implementación se limita a mostrar solo una imagen, y solo se incluyen imágenes enviadas durante la videollamada, es decir, con fecha mayor a la fecha de inicio de la llamada. Se hizo esto para simplificar el servicio de cara a que sea más fácilmente entendible y usable para los adultos mayores.

Este fragmento está asociado a una vista de imagen, y no necesita que el usuario interactúe directamente con ella, si no que se debe actualizar la vista según se actualice la base de datos de Telegram. Sin embargo, para uno de los casos sí se puede utilizar la interacción del usuario con el fragmento *GalleryFragment* como una indicación para actualizar la vista. Cuando el usuario del SocialConnector compare una foto, se llama al método *getChat* [26] de la API, para obtener la información del chat y actualizar los mensajes, mostrando la imagen más reciente en el chat, que corresponde a la foto recién enviada por el usuario.

Básicamente, cuando se envía una foto, la vista debe ser actualizada a través del fragmento *PictureFragment*. Cuando el contacto sea quien envía la foto, la vista debe actualizarse automáticamente al recibir el mensaje, sin ninguna indicación como en el caso anterior.
PictureFragment debe implementar la clase *TelegramClient.Callback*, para poder implementar los métodos de ésta, entre los cuales está el método *onResult* en el cual se maneja la respuesta de las peticiones a la API de Telegram. Para este caso en específico, el objeto que se recibe debe ser del tipo Messages, los cuales se obtienen y luego se procesan.

En el método *processMessages* se revisan los mensajes del chat, filtrando solo aquellos cuyo *fileType* sea igual a 2, es decir solo las imágenes. Luego se elige la que tenga fecha más reciente, siempre y cuando ésta tenga una fecha mayor a la del inicio de la videollamada. Esto es para evitar mostrar una foto que estaba en el chat anteriormente, en caso de que se reciba un mensaje durante la videollamada, y se llegue a la función *onResult* sin haber recibido una foto. Una vez que se identifica la imagen más reciente, se actualiza la vista de *PictureFragment*, mostrando la imagen recibida más recientemente.

Un elemento que no se ha mencionado sobre el proceso de compartir imágenes, es el realizado por el servicio *PictureSharingService*, el cual se mantiene en el foreground y notifica al usuario que el SocialConnector está listo para compartir imágenes. La notificación es necesaria para considerar un servicio como foreground [27]. El objetivo de este servicio es permitir volver al SocialConnector sin problemas, evitando que el sistema operativo termine el proceso de la actividad *PictureSharingActivity*.

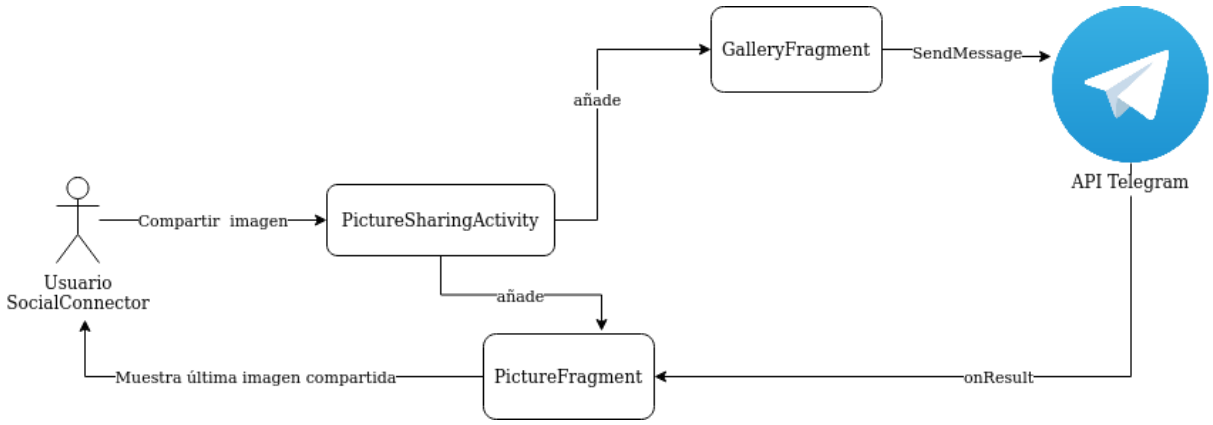

Figura 15: Representación de la implementación.

Considerando todo lo mencionado, *PictureSharingActivity* corresponde a la actividad donde se manejan todas las interacciones referentes a la acción de compartir imágenes durante la videollamada, y cuya lógica queda representada (de forma simplificada) en la Figura 15. Este esquema muestra que la actividad PictureSharingActivity es el controlador al que el usuario realiza sus peticiones. Sin embargo, son los fragmentos los que interactúan con la API de Telegram y actualizan las vistas al usuario.

6. Evaluación de la Solución

Para determinar la efectividad de la solución propuesta, se evaluará la utilidad y la usabilidad del servicio creado. En el caso de la utilidad, se debe asegurar que el sistema cumpla el objetivo principal, y que la funcionalidad de compartir pantalla se ejecute correctamente. Esto se puede evaluar sometiendo al sistema a los tests correspondientes.

Con respecto a la usabilidad del sistema, la cual es de suma importancia debido a que los usuarios son adultos mayores, ésta debe ser evaluada mediante el testeo con usuarios reales. Por ejemplo, utilizando técnicas como *thinking aloud* [28], en donde estos indican sus acciones en voz alta, mientras prueban funcionalidades específicas del sistema. También se puede utilizar la encuesta que propone la escala de usabilidad SUS (System Usability Scale) [29].

Para evitar que los adultos mayores se confundan con algunas de las terminologías que utiliza la encuesta SUS, se decide por realizar una evaluación del estilo *thinking aloud*, incluyendo una opinión abierta del adulto mayor sobre la funcionalidad para compartir imágenes durante la videollamada en el SocialConnector. Extractos de las opiniones dadas se incluyen en este capítulo.

6.1. Pruebas Realizadas

En esta sección se comentan las pruebas realizadas para probar diversos aspectos de la aplicación, y comprobar si el funcionamiento corresponde al mínimo esperado de la solución. En primer lugar, se realiza la prueba de estabilidad que se mencionó en el capítulo de requisitos, y luego las pruebas de usabilidad mencionadas.

6.1.1. Pruebas de estabilidad de la solución

Para realizar esta prueba, se comenzó una videollamada y se dejó pasar cierta cantidad de tiempo, para luego comenzar a compartir fotos. En cada prueba se registró si se encontraba disponible la actividad, y si la aplicación se caía al haber terminado el proceso. Esta prueba se realizó varias veces para observar qué tan consistente era el comportamiento del software. Los resultados obtenidos se muestran en la Tabla 2 (prueba vs tiempo de espera). Una “O” en la celda correspondiente indica que la aplicación cambió de actividad exitosamente. En cambio, cuando se indica una “X” significa que se presentaron problemas, y se registra el problema encontrado.

La idea de esta prueba es registrar los tiempos de disponibilidad mínimos y máximos, y registrar inconsistencias en el comportamiento de la aplicación cuando está inactiva por periodos extendidos de tiempo. Usando la clase *Timer* [30], se programa un toast que indica que la aplicación sigue estando activa en el *background*; si el proceso de la aplicación es terminado por el sistema operativo, el toast no será mostrado y se considera el último toast como el último lapso de tiempo disponible de la aplicación.

Tabla 2: Esquema para testear la disponibilidad del servicio para compartir fotos.

Prueba/ Tiempo	5 min.	10 min.	15 min.	20 min.	25 min.	30 min.
Prueba n° 1						
Prueba n° 2						
…						

Tal como se mencionó en el capítulo 3, se llevó a cabo también una prueba para registrar los tiempos de respuesta a las acciones más importantes que los usuarios realizan en la aplicación. Estos consideran los tiempos de respuesta en la interfaz de la aplicación, es decir, el instante en el que el usuario obtiene una respuesta visual a la operación realizada. Las operaciones a analizar son las siguientes: 1) pasar de la videollamada a la transmisión de imágenes, 2) envíar datos desde el usuario a la persona con la que está haciendo la videollamada, y 3) salir de la transmisión de imágenes y volver a la videollamada normal. El tiempo se calcula usando la diferencia entre el momento en que se llama a la función *onClick* de cada botón, y el momento en que se termina la función *onCreateView* de la vista que se procede a mostrar.

6.1.2 Prueba de usabilidad

Como se mencionó al comienzo de este capítulo, se realizaron pruebas de usabilidad con usuarios adultos mayores. Para la prueba, se les pidió a los adultos mayores voluntarios realizar una videollamada a través del SocialConnector, compartir una foto mientras realizaban dicha actividad, recibir una foto, y salir de la vista donde se comparten fotos y luego volver a la videollamada. Se pidió que ofrecieran comentarios en voz alta durante el uso de la aplicación. Una vez finalizada la prueba, se les preguntó por sus opiniones respecto a la experiencia de uso de este nuevo servicio.

En este caso, la prueba se realizó a seis adultos mayores, con las siguientes características:

- Usuario 1: mujer, 76 años.
- Usuario 2: mujer, 68 años.
- Usuario 3: mujer, 71 años.
- Usuario 4: hombre, 65 años.
- Usuario 5: hombre, 71 años.
- Usuario 6: hombre, 75 años.

El proceso desde iniciar la videollamada, hasta compartir una foto y retornar a ésta, se llevó a cabo de forma continua. Por lo tanto, no se observaron aspectos como el tiempo de disponibilidad del servicio al momento de hacer esta prueba.

6.2. Resultados Obtenidos

6.2.1. Resultados pruebas de estabilidad

En primer lugar, se presentan los resultados de la prueba de disponibilidad del servicio, cuyo objetivo es encontrar los tiempos mínimos y máximos en que se puede comenzar a compartir pantalla durante la videollamada, y la estabilidad de dicho servicio.

Tabla 3: Resultados de las pruebas de disponibilidad del servicio.

Prueba/ Tiempo	5 min.	10 min.	15 min.	20 min.
Prueba n° 1	O	O	O	X
Prueba n° 2	O	X		
Prueba n° 3	X			
Prueba n° 4	O	O	O	X
Prueba n° 5	O	X		

Como se puede apreciar en la Tabla 3, el sistema presenta algunos problemas para mantener consistentes los tiempos, lo que significa que existe un problema de disponibilidad luego que la aplicación queda inactiva por un cierto período. Es difícil determinar tiempos mínimos y máximos de disponibilidad si la funcionalidad no es consistente, pues estos tiempos estarán sujetos a variaciones.

Respecto a la prueba de tiempos de respuesta, estos se calculan cuando la respuesta visual está disponible, por lo que los tiempos capturados pueden tener un leve desfase al ser registrados manualmente. Las pruebas para cada acción se realizaron 20 veces, y los tiempos presentados en la Tabla 4 son el promedio de tiempos de todas las pruebas.

Tabla 4: Resultados de tiempos de respuesta.

Transacción	Tiempo Promedio de Respuesta (promedio para 20 pruebas)
Pasar de la videollamada a la transmisión de pantalla.	2.78 s.
Envío de datos desde el usuario a la persona con la que está haciendo la videollamada.	1.94 s.
Salir de la transmisión de pantalla y volver a la videollamada normal.	3.18 s.

6.2.2. Resultados pruebas de usabilidad

En esta sección del documento se presentan los resultados de las pruebas de usabilidad que se condujeron. Los usuarios que probaron la aplicación fueron libres de dar sus opiniones durante el uso del servicio, y además brindaron una opinión luego de haber utilizado el sistema. A continuación, se muestran los comentarios más relevantes que se expresaron durante cada operación, los cuales han sido transcritos de la forma más textual posible, sin distinción del usuario que los realizó y entregando el contexto en que se dijo.

Al comenzar a compartir imágenes:

- Inmediatamente después de apretar el botón para compartir fotos: “¿Ahora hay que esperar?”
- Luego de que se muestra la respuesta: “Ahí está.”
- Cuando se llega a la ventana para compartir imágenes: “Se dio vuelta la pantalla.”

- Al llegar a la ventana para compartir, en referencia al botón para compartir foto: “¿Tengo que apretar de nuevo esto?”
- “¿Toco este botón verde?” Luego de leer la etiqueta del botón. “Ah, sí.”
- Nuevamente, al cambiar a la pantalla para compartir imágenes. “Ay, se dio vuelta esta cosa”.
- Apretando el botón para compartir imágenes. “No sale.” Apreta el botón nuevamente, cambiando a la pantalla.
- Leyendo el botón. “Entonces, apretando esto empiezo.”
- Tocando el botón para iniciar la transmisión de imágenes y cambiando a la siguiente vista. “Ahí lo puse.”

Mientras se comparten fotos:

- Al ver la interfaz por primera vez: “Voy a buscar mis lentes.”
- Leyendo las etiquetas de los botones: “Compartir foto… ¿Este es? ¿lo aprieto?”
- Eligiendo botón: “El verde es para compartir”.
- Al entrar a la galería: “¿Aquí qué hago, tengo que apretar la foto?”
- Al elegir la foto: “Casi me equivoco.”
- Una vez compartida la imagen: “Ahí está la que elegí.”
- Al recibir una foto: “¿Y esta de donde salió?”
- Leyendo la etiqueta del botón. “Ya, ahora voy a ‘compartir foto’, así.” Se toca el botón para compartir imágenes. “Esta es.” Se toca la foto especificada.
- Eligiendo una foto equivocada. “¿Cómo la borro?”

Al volver a la videollamada:

- Observando los botones: “Ahora es el rojo.”
- Leyendo los botones. “Con el botón rojo cierro esto entonces.”
- Al hacer click en el botón para volver a la videollamada: “¿Sigue esta cosa?”
- Al volver a la videollamada. “Se dio vuelta (la pantalla) de nuevo.”
- Tocando el botón para terminar la videollamada. “¿Y ahora se acaba para siempre esto?”

Luego de haber utilizado la aplicación, se pide a los usuarios una breve opinión respecto a la experiencia usando el SocialConnector para compartir fotos. Éstas se presentan de forma anónima.

Opinión Usuario 1: “Me gustó la que era bien simple, y que podía leer lo que hacían las cosas (botones), pero no entendía cuando aparecía una foto que no era la que había apretado antes. Cuando empezó pensé que no estaba funcionando, porque se demoró en mostrarme algo. Y decía que iba a compartir foto, pero después decía lo mismo en la cosa donde estaban los dos botones. También me pasó que me confundí cuando apreté el rojo porque pensé que me había salido (se refiere a terminar la videollamada), pero pasó a otra cosa. Pero me gustó que explique todo”.

Opinión Usuario 2: “La letra era un poco chica. Se podía leer sin lentes, pero cuando apareció la encontré muy chica. Cuando apreté el botón verde (compartir foto) cuando estaba en la cámara (videollamada) se cambió para otro lado (la tablet cambió del modo vertical a horizontal), no me gustó eso. Me imaginé que el verde era para mandar las cosas, pero el rojo pensé que era para cerrar todo. Cuando salieron las fotos (abrir la galería), no sabía bien qué hacer”.

Opinión Usuario 3: “Me gusto que saliera ahí lo que hacía cada cosa, en vez de un monito (ícono) como los que salen en el Whatsapp. Lo que sí, encontré que era un poco lenta a veces, ya me acostumbré a Whatsapp que es un poco más rápido. Pero me gustó que se pueda compartir cosas cuando estoy llamando.”

Opinión Usuario 4: “Cuando le puse lo de compartir la foto, no salía nada aparte de los botones, se veía un poco vacío y no sabía si faltaba que saliera otra cosa o estaba cargando. Después salió la imagen y se veía bien. Me gustaría poder ver más de una foto.”

Opinión Usuario 5: “Los botones con colores me gustaron, pero la letra era un poco difícil de leer. A veces si apretaba algo no sabía si lo había apretado si es que se demoraba en pasar algo.”

Opinión Usuario 6: “Me gusto más que otras aplicaciones porque salía lo que hacen los botones. Me hubiese gustado que se pudiera cambiar la foto porque me equivoqué y se mandó no más. Pero lo otro era fácil de aprender.”

6.3 Discusión de los resultados

A partir de los resultados obtenidos, existen algunas observaciones relevantes que se pueden realizar para mejorar el diseño de las interfaces, e identificar errores en la arquitectura de software utilizada.

Sobre las pruebas de estabilidad, existen algunos problemas para mantener la aplicación estable cuando esta pasa al segundo plano al iniciar la videollamada. Esto significa que el servicio que se implementó para que el proceso de la aplicación pudiese seguir trabajando en segundo plano, no está funcionando como debería, pues la aplicación se reinicia de forma aleatoria, lo que genera problemas cuando se intenta volver desde la videollamada. Se debe revisar si es un problema del servicio, o si está relacionado a cómo se comporta el SocialConnector al momento de reiniciarse, y si es este proceso el que produce problemas al retornar a la aplicación. También podría deberse a los cambios en los manejos de eventos en las nuevas versiones de Android.

Respecto a los tiempos de respuesta, si bien estos no son los tiempos óptimos que se esperaba obtener, el margen de diferencia no es particularmente alto, por lo que se considera que son aceptables. Además, una buena forma de solucionar posibles incomodidades que genera la espera, es entregar feedback inmediato de manera visual; por ejemplo, indicadores de que un botón ha sido apretado, iconos de carga cuando se está actualizando una imagen, o un mensaje tipo *Toast* [31] que aparezca en la pantalla por unos segundos. Este tipo de feedback puede ser suficiente para que los tiempos de espera no representen un problema de uso.

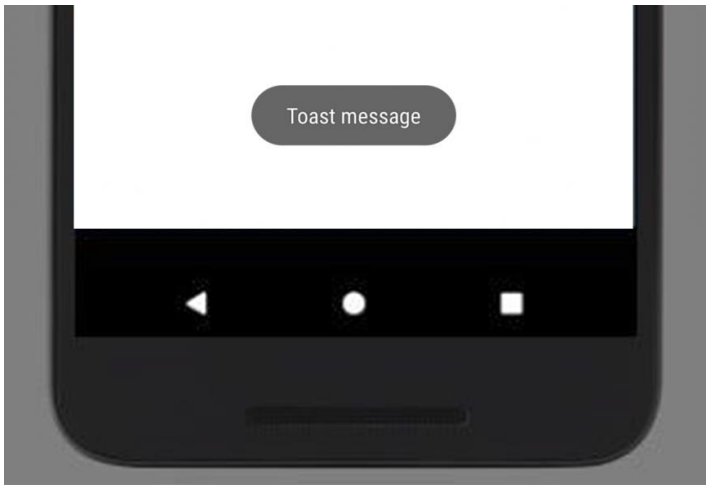

Figura 16: Ejemplo de un mensaje *Toast*.

Respecto a las pruebas de usabilidad, existen varios comentarios que resultan ser valiosos para tener claro qué aspectos del diseño de la solución se pueden mejorar, para así generar una solución que sea cómoda para los adultos mayores.

En primer lugar, el diseño simple que se presentó para la solución está bien ejecutado, pero el intento de mantener la interfaz extremadamente minimalista resultó ser perjudicial, al no entregar toda la información que un usuario con poca experiencia necesita. Se debe encontrar un balance entre mostrar toda la información que puede ser útil al usuario, sin sobrecargar la vista. Como se mencionó anteriormente, el uso de notificaciones tipo *Toast* puede ser útil para guiar al usuario, sobre todo cuando se mueve entre aplicaciones, en donde no siempre es posible mostrar un elemento dentro de la aplicación.

Otro elemento al que se debe poner atención es al uso de colores dentro de la aplicación. Los principales colores dentro del SocialConnector cuando se trata de botones son el lila, rojo y verde, como se presenta en la Figura 17. En general, el lila se utiliza para representar una acción, el verde para enviar y el rojo para descartar una acción.

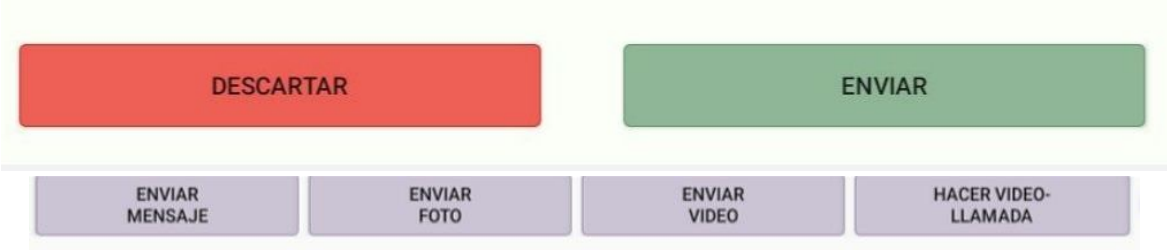

Figura 17: Ejemplos de botones presentes en el SocialConnector.

Teniendo esto en mente, se puede reconsiderar los colores utilizados en las interfaces para compartir pantalla. Si bien la combinación de rojo para cancelar y verde para enviar ya está presente en varias interfaces del SocialConnector (como por ejemplo, en las pantallas de confirmación de la Figura 17), es necesario preguntarse si estas acciones se corresponden con lo que se hace en la actividad para compartir imágenes.

Aunque el botón “Compartir foto” se corresponde con la acción de enviar algo, el botón rojo podría generar confusión pues no se está descartando ninguna información, sino que se está cambiando la ventana principal en donde se encuentra el usuario. Usar el color lila, que implica una acción resulta más intuitivo.

En relación con la consistencia de la aplicación, es importante evitar cambios que puedan desorientar al usuario, y la orientación de la pantalla es uno de ellos. Considerando que la videollamada de Telegram automáticamente usa la vista vertical, se debe considerar tener la opción de una vista vertical al momento de compartir las fotos en el SocialConnector.

El tamaño de la letra es de suma importancia para los adultos mayores, pues la mayoría tiene problemas de visión. Respecto a esto, también es importante entregar más información durante la transmisión de fotos, indicando el contacto con el que se comparten las fotos, y también quién envió la foto que se está proyectando.

La implementación de todas estas correcciones pueden ser una gran mejora para la experiencia de usuario con el SocialConnector, sin necesidad de perturbar el diseño simple y directo que tiene, sino que acercando la información al usuario para evitar confusiones e inseguridad al usar la aplicación.

7. Conclusiones y Trabajo a Futuro

A continuación, se presentan las conclusiones que se obtuvieron a partir del trabajo realizado, junto con ideas de trabajo a futuro que se puede realizar para mejorar los resultados del trabajo de memoria.

El SocialConnector se enfoca en facilitar la comunicación entre adultos mayores y el *broker* (persona que hace de puente entre el adulto mayor y el resto de la familia), a través de un sistema de mensajería que posee una interfaz mucho más amigable para usuarios con poca o nula experiencia tecnológica. En este trabajo, se agregó la funcionalidad para compartir imágenes durante una videollamada, debido a la necesidad de mantener al usuario (adulto mayor) informado, permitiendo que pueda ver imágenes al mismo tiempo que las comenta con el *broker*.

Es claro que, con esta funcionalidad, se agrega otro nivel en el que el usuario puede comunicarse con sus pares, cumpliendo el objetivo general que se plantea en la memoria. Sin embargo, es importante considerar no sólo se agregó una funcionalidad útil a la aplicación, sino que también se debió considerar qué herramientas se le entregan al usuario para que le ayuden a interactuar con otros, y no se vuelvan un obstáculo para dichas interacciones.

Una interfaz simple y directa fue uno de los puntos que más se enfatizó al momento del desarrollo, pero es importante considerar que no se puede sacrificar el mantener al usuario informado (entrega de feedback), en favor de mantener una interfaz simple. En una aplicación que se enfoca en usuarios, que muchas veces tienen problemas con la tecnología, no se puede entregar simplicidad a costa de desinformación. Un usuario poco informado tendrá dudas, y esas dudas pueden llevar a un miedo de usar las herramientas que se le entregan. No sirve de nada agregar nuevas funcionalidades si el usuario no se ve incentivado a utilizarlas.

Desarrollar aplicaciones para adultos mayores es un desafío, ya que existen muchos parámetros que pueden ser de suma importancia para ellos, pero que un desarrollador más joven puede pasar por alto con facilidad. Es por eso que aplicaciones que se enfoquen en ser accesibles para usuarios que salen de lo típico, son de suma importancia. En un mundo donde la tecnología se vuelve casi indispensable para la comunicación con otros, no se puede dejar a nadie atrás.

Como trabajo a futuro, se deben realizar más pruebas de usabilidad con adultos mayores, considerando a usuarios con la mayor diversidad posible. También se debe investigar si la

funcionalidad actual (es decir, el compartir imágenes), es mejor o no que compartir la pantalla del dispositivo como instrumento para fomentar la interacción social entre un adulto mayor y su familia y amigos.

Existen aspectos técnicos que también deben ser mejorados, como por ejemplo, el rendimiento de la aplicación y el tiempo en el que está disponible la opción de compartir las fotos. Es ideal reducir al mínimo la posibilidad de que los adultos mayores se enfrenten a problemas técnicos, como la caída de la aplicación, pues es muy difícil que puedan resolverlo, y es probable que esto les genere estrés y aversión a usar tecnología.

Además, creo que una aplicación como el SocialConnector se beneficiaría mucho de agregar otras opciones de accesibilidad, tales como traducción texto-a-voz para que usuarios con problemas de visión puedan usar todas las funcionalidades de la aplicación. También se puede agregar la opción de traducción de voz-a-texto, ya sea para que usuarios con capacidades motrices limitadas puedan entregar instrucciones a la aplicación, o para que aquellos adultos mayores con discapacidad auditiva puedan leer los contenidos enviados en un archivo de audio o un video.

Bibliografía

[1] Francisco J. Gutierrez, Sergio F. Ochoa. Mom, I Do Have a Family!: Attitudes, Agreements, and Expectations on the Interaction with Chilean Older Adults. Proc. of the ACM CSCW 2016: pp. 1400-1409, 2016.

[2] Diego Muñoz, Francisco J. Gutierrez, Sergio F. Ochoa, Nelson Baloian. *Social connector: a ubiquitous system to ease the social interaction among family community members*. Comput. Syst. Sci. Eng. 30(1), 2015.

[3] Rodriguez, W. (2021, 13 octubre). *How to add screen sharing to your Android app with Calls*. Sendbird. Fecha de recuperación: 16 de septiembre de 2022, URL: https://sendbird.com/developer/tutorials/screen-sharing-android

[4] Vonage. (s. f.). *Developer Center - Everything you need to build WebRTC apps | Vonage Video API Developer*. TokBox. Fecha de recuperación: 16 de septiembre de 2022, URL: https://tokbox.com/developer/

[5] Agora. (s. f.). *Share the Screen*. Fecha de recuperación: 16 de septiembre de 2022, URL: https://docs.agora.io/en/Video/screensharing_android?platform=Android

[6] Telegram APIs. (s. f.). Telegram. Fecha de recuperación: 16 de septiembre de 2022, URL: https://core.telegram.org/api

[7] WebRTC. (s. f.). *Getting started with*. Fecha de recuperación: 16 de septiembre de 2022, URL: https://webrtc.org/getting-started/overview

[8] Schneiderman, B. (1998). *Designing the user interface.* Third edition. Addison-Wesley. (First edition published 1987).

[9] Nielsen, J. (1994). *Heuristic evaluation*. Usability inspection methods. Nielsen, J., and Mack, R.L. (Eds.). John Wiley & Sons.

[10] Android Developers. *Warm start.* Fecha de recuperación: 01 de noviembre de 2022. URL: https://developer.android.com/topic/performance/vitals/launch-time#warm

[11] Android Developers. *Hot Start.* Fecha de recuperación: 01 de noviembre de 2022. URL: https://developer.android.com/topic/performance/vitals/launch-time#hot

[12] Android Developers. *App startup time.* Fecha de recuperación: 01 de noviembre de 2022. URL: https://developer.android.com/topic/performance/vitals/launch-time

[13] Wikipedia. *Model–view–controller*. Fecha de recuperación 22 de septiembre de 2022. URL: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

[14] Google Developers. *Introduction to activities.* Fecha de recuperación: 22 de septiembre de 2022. URL: https://developer.android.com/guide/components/activities/intro-activities

[15] Google Developers. *Fragments*. Fecha de recuperación: 22 de septiembre de 2022. URL: https://developer.android.com/guide/fragments

[16] Google Developers. *View*. Fecha de recuperación: 22 de septiembre de 2022. URL: https://developer.android.com/reference/android/view/View

[17] Google Developers. *Guide to background work*. Fecha de recuperación: 22 de septiembre de 2022. URL: https://developer.android.com/guide/background

[18] Google Developers. *Services overview*. Fecha de recuperación: 22 de septiembre de 2022. URL: https://developer.android.com/guide/components/services

[19] Google Developers. *Intent.* Fecha de recuperación: 23 de Septiembre. URL: https://developer.android.com/reference/android/content/Intent

[20] Google Developers. *AccessibilityService.* Fecha de recuperación: 23 de Septiembre. URL: https://developer.android.com/reference/android/accessibilityservice/AccessibilityService

[21] Google Developers. *Deep links.* Fecha de recuperación: 23 de Septiembre. URL: https://developer.android.com/training/app-links#deep-links

[22] Google Developers. *onClick.* Fecha de recuperación: 24 de Septiembre. URL: https://developer.android.com/reference/android/view/View.OnClickListener#onClick(android.view.View)

[23] Google Developers. *TYPE_APPLICATION_OVERLAY*. Fecha de recuperación 25 de Septiembre de 2022. URL: https://developer.android.com/reference/android/view/WindowManager.LayoutParams#TYPE_APPLICATION_OVERLAY

[24] Google Developers. *SYSTEM_ALERT_WINDOW*. Fecha de recuperación 25 de Septiembre de 2022. URL: https://developer.android.com/reference/android/Manifest.permission#SYSTEM_ALERT_WINDOW

[25] Google Developers. *sendMessage Class Reference.* Fecha de recuperación 25 de Septiembre de 2022. URL: https://core.telegram.org/tdlib/docs/classtd_1_1td__api_1_1send_message.html

[26] Telegram. *getChat Class Reference*. Fecha de recuperación 25 de Septiembre de 2022. URL: https://core.telegram.org/tdlib/docs/classtd_1_1td__api_1_1get_chat.html

[27] Google Developers. *Foreground Services.* Fecha de recuperación 25 de Septiembre de 2022. URL: https://developer.android.com/guide/components/foreground-services

[28] Sauro, J. (2016, 1 noviembre). *The Origins and Evolution of Thinking Aloud – MeasuringU*. MeasuringU. Fecha de recuperación: 25 de septiembre de 2022, URL: https://measuringu.com/thinking-aloud/

[29] Brooke, J. (1996). *SUS: A "quick and dirty" usability scale.* In P. W. Jordan, B. Thomas, B. A. Weerdmeester, & A. L. McClelland (Eds.), *Usability Evaluation in Industry*. London: Taylor and Francis.

[30] Android Developers. *Timer*. Fecha de recuperación: 31 de octubre de 2022. https://developer.android.com/reference/java/util/Timer

[31] Android Developers. *Toasts overview*. Fecha de recuperación: 26 de septiembre de 2022. URL: https://developer.android.com/guide/topics/ui/notifiers/toasts